EVERTON HIDEO NAKAHARADA DOS SANTOS

PROCESSO DE ENGENHARIA REVERSA PARA SOFTWARE LEGADO (ERSL)

São Paulo
2016

EVERTON HIDEO NAKAHARADA DOS SANTOS

PROCESSO DE ENGENHARIA REVERSA PARA SOFTWARE LEGADO (ERSL)

Monografia apresentada ao PECE – Programa de Educação Continuada em Engenharia da Escola Politécnica da Universidade de São Paulo como parte dos requisitos para conclusão do curso de MBA em Tecnologia de Software.

São Paulo
2016

EVERTON HIDEO NAKAHARADA DOS SANTOS

PROCESSO DE ENGENHARIA REVERSA PARA SOFTWARE LEGADO (ERSL)

Monografia apresentada ao PECE – Programa de Educação Continuada em Engenharia da Escola Politécnica da Universidade de São Paulo como parte dos requisitos para conclusão do curso de MBA em Tecnologia de Software.

Área de Concentração: Tecnologia de Software

Orientadora: Profa. Dra. Selma Shin Shimizu Melnikoff.

São Paulo
2016

Santos, Everton Hideo Nakaharada dos
Processo De Engenharia Reversa Para Software Legado (ERSL) / E. H. N. Santos -- São Paulo, 2016.
106 p.

Monografia (MBA em Tecnologia de Software) - Escola Politécnica da Universidade de São Paulo. PECE – Programa de Educação Continuada em Engenharia.

1.Engenharia Reversa de Software 2.Reengenharia de Software 3.Softwares I. Universidade de São Paulo. Escola Politécnica. PECE – Programa de Educação Continuada em Engenharia II.t.

Dedico este trabalho a minha família, pelo apoio e motivação nos momentos difíceis.

AGRADECIMENTOS

A Deus, pela minha vida e saúde;

À Profa. Dra. Selma Shin Shimizu Melnikoff, pelo apoio à realização do curso de Tecnologia de Software e pela orientação, paciência e incentivo para a conclusão deste trabalho;

Aos meus pais; que me educaram e sempre motivaram a buscar meus objetivos;

À minha namorada, que sempre me apoiou em todos os momentos;

À minha irmã, que sempre foi um exemplo em minha vida;

Aos meus amigos Tiago e William, que me ajudaram com atividades de meu emprego para poder me dedicar ao curso de Tecnologia de Software;

Ao meu amigo Fabiano Yoschitaki, do curso de Tecnologia de Software, pelo sacrifício em realizar diversos trabalhos;

E a todos que colaboraram direta ou indiretamente, na execução deste trabalho.

RESUMO

A constante evolução tecnológica e o surgimento de novos requisitos de negócio tornam a evolução do software inevitável. Essa evolução, no entanto, não é trivial, pois é comum encontrar diversos sistemas ativos nas organizações com Software Legado. Devido a esse cenário, surgem frequentes questionamentos sobre como evoluir ou migrar um Software Legado e também como elicitar os requisitos desse tipo de software. O objetivo desse trabalho é definir o processo de Engenharia Reversa para Software Legado (ERSL), para aplicar no contexto de migração de software legado. O processo consiste na integração das técnicas de três métodos selecionados da literatura. Esse processo realiza a elicitação de requisitos do software legado através da análise das interfaces de usuário e do código-fonte. A análise de código-fonte utiliza as técnicas de redocumentação e de modelo de metas, que permitem a criação de novos documentos e modelos. O processo ERSL foi aplicado em um módulo de um sistema real para avaliar a sua eficiência. Os resultados da aplicação mostram que cada fase do processo complementa as informações elicitadas na fase anterior. É um processo iterativo e incremental pois pode ser aplicado em um contexto de migração gradativa do software legado. O resultado foi positivo, pois gerou melhoras qualitativas dos requisitos elicitados. O processo pode ser melhorado, pois existem atividades que podem ser automatizadas através do desenvolvimento de ferramentas específicas.

Palavras-Chave: Engenharia Reversa. Software Legado. Migração de Software.

ABSTRACT

Constant technological evolution and emerging of new business requirement make software evolution unavoidable. However, such evolution is not trivial because finding various active systems in the organizations is something common when it comes to legacy software. Due to that scenery, questionings about how to evolve or migrate a legacy software and how to elicit the requirements of this kind of software are often raised. The goal of this paper is to define the reverse engineering for legacy software process so that it can be applied in the legacy software migration context. The process consists of integration of three techniques selected from the literature. This process elicits legacy software requirements through source-code and user interface analysis. The analysis of the source-code makes use of refiling techniques and goal model, which allow the creation of new files and models. The reverse engineering for legacy software process was applied in a real system module to assess its efficiency. The application results shoe each phase of the process complements the elicited information from the previous phase. It is an interactive and incremental process because it can be applied in a legacy software gradual migration context. The process created qualitative improvements of the elicited requirements, being a positive result. The process can be improved because developing specific tools can automatize some activities.

Key-Words: Reverse Engineering; Legacy Software; Software Migration.

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

BPMN	*Business Process Modeling Notation*
CRUD	*Create*, *Read*, *Update* e *Delete*
ER	Engenharia Reversa
ERSL	Engenharia Reversa para Software Legado
RE	Reengenharia
RREE	*Rigi Reverse Engineering Environment*
RSF	*Rigi Standard Format*
UML	*Unified Modeling Language*
XML	*eXtensible Markup Language*

SUMÁRIO

1 INTRODUÇÃO

Este capítulo tem como objetivo descrever a motivação para a realização do trabalho, seguido de seu objetivo, a justificativa para a sua realização, a metodologia de trabalho e a sua estrutura de organização.

1.1 MOTIVAÇÃO

O surgimento de novas tecnologias, novos requisitos de negócio e necessidade de correção de diversos erros existentes em um sistema de software torna a evolução de software inevitável. Atualmente existem diversos sistemas de software em funcionamento, os quais não possuem um alto índice de alteração em relação aos requisitos, estão implementados com tecnologias obsoletas, mas tem alto valor para as organizações, do ponto de vista de negócio. De uma forma simplificada, estes sistemas são considerados como Sistema Legado (SALVATIERRA et al., 2013).

Na engenharia de software, encontram-se diversas abordagens para realizar alterações em software; porém para software legado, existem fatores específicos que dificultam a manutenção. Fatores como: falta de documentação e/ou documentação desatualizada, escassez de profissionais especialistas em tecnologias que não são mais utilizadas e também falta de especialistas que saibam as características e requisitos que o software possui, pois, essas pessoas, em geral, não estão disponíveis, podendo estar aposentadas ou em outro emprego, dificultam o entendimento do software para realizar qualquer manutenção (GEET; EBRAERT; DEMEYER, 2010).

Baseado no contexto de software legado, é comum não se optar para realizar manutenção evolutiva; porém, com a evolução dos processos organizacionais surgem novas necessidades de negócio, em forma de novos requisitos, necessidade de integração com novos sistemas de software externos e/ou internos e também melhorias de desempenho visando o crescimento organizacional. Desta forma, uma das soluções é migrar o software legado para uma plataforma de tecnologia mais atual.

A migração de um software normalmente está ligada à necessidade de modernização ou substituição. A modernização pode ser tratada por duas estratégias diferentes: a migração direta, em que é criada uma camada com tecnologia moderna encapsulando o software legado e a migração indireta, que objetiva reimplementar o software legado com outra tecnologia e em outra plataforma (SALVATIERRA et al., 2013).

O processo de migração de um software legado não é o mesmo que o de criação de um novo software. Uma das diferenças está na fase de elicitação de requisitos, pois o processo não depende totalmente dos usuários e pessoas envolvidas com o contexto do software. Existe também uma grande dependência da realização de engenharia reversa, que apoia a elicitação dos requisitos com maior detalhamento, ajuda a identificar funcionalidades, que estão disponíveis no software e não são frequentemente utilizadas, e identifica as regras de negócio que estão implementadas no software (YU et al., 2005).

É comum relacionar os assuntos de migração de software, engenharia reversa e reengenharia, porém, apesar de serem assuntos afins, cada um possui um papel diferente. Para realizar a migração de um software é necessário entender como ele funciona e, para isso, são utilizadas técnicas de engenharia reversa; uma vez identificados os requisitos, é necessário projetar um novo software baseado no que foi analisado e esse processo é chamado de reengenharia.

A engenharia reversa tem como objetivo a extração de informações sobre o conhecimento incluído no software, suas características e seu projeto. Ela pode ser utilizada em qualquer nível de abstração e em qualquer fase do ciclo de vida do software, pois envolve a análise de um software que está em funcionamento. (CHIKOFSKY; CROSS, 1990).

Na literatura encontram-se diversos métodos para a engenharia reversa de software, porém a eficiência e a eficácia destes métodos estão relacionadas à experiência dos profissionais envolvidos, às condições em que o software legado se encontra e ao tipo da finalidade do software. Esses fatores podem influenciar de forma quantitativa e qualitativa na identificação dos requisitos do software estudado.

1.2 OBJETIVO

O objetivo deste trabalho é definir um processo de engenharia reversa para aplicar no contexto de migração de software legado.

Para isso, foram inicialmente estudados diversos métodos de engenharia reversa da literatura. Dentre eles, foram selecionados três pela finalidade apresentada por eles e pelo nível de detalhamento que permitiram a realização de experimentos.

Os métodos selecionados foram Engenharia Reversa através de Interfaces de Sistemas Legados (STROULIA et al., 1999), Redocumentação de Sistemas Legados (GEET; EBRAERT; DEMEYER, 2010) e Engenharia Reversa através Modelos de Meta (YU et al., 2005). Após sua análise, constatou-se que os métodos apresentavam superposição e complementação de características e concluiu-se que eles poderiam ser usados em conjunto dentro de um processo.

1.3 JUSTIFICATIVA

Frequentemente, a decisão da organização para realizar a migração de software legado está baseada na expectativa de recuperar os conhecimentos implícitos no software legado, juntamente com a evolução do sistema para utilizar tecnologias mais modernas. A migração é, portanto, motivada pela busca do crescimento de seus negócios.

A elicitação de requisitos é uma das etapas fundamentais e de grande importância no processo de migração de um software legado, pois é o momento em que são identificadas as funcionalidades do software. Em geral, no contexto de migração, a elicitação de requisitos é realizada através da engenharia reversa; é importante que os métodos utilizados resultem em requisitos de qualidade e que o sistema novo possa substituir o software legado com segurança.

Na literatura, é comum encontrar a engenharia reversa como uma atividade essencial do ciclo de vida de um software, podendo ser definida como um processo de análise de um sistema, que consiste da identificação dos componentes e suas inter-relações (CHIKOFSKY; CROSS, 1990).

De acordo com a engenharia de software, os erros cometidos na fase de elicitação de requisitos podem levar ao fracasso de um projeto. Ao comparar essa afirmação com o processo de migração de um software legado, entende-se que os erros cometidos durante a identificação de requisitos na engenharia reversa podem trazer grandes problemas futuros após o novo software ser implantado.

Dessa forma, a engenharia reversa é uma disciplina fundamental na migração de software legado, pois fornece base para construção de um novo sistema que implementará os requisitos do software legado.

1.4 METODOLOGIA DE TRABALHO

Para a realização deste trabalho, foram pesquisados artigos técnicos, anais de conferência, livros e revistas de tecnologia que abordam os assuntos de software legado, migração de software legado, engenharia de software, engenharia reversa, manutenção de software, reengenharia, engenharia de requisitos, entre outros temas que se relacionam ao software legado.

Dentre os trabalhos pesquisados, foram selecionados três artigos que tratam sobre métodos de engenharia reversa em software legado, sendo que cada um aborda a engenharia reversa de forma diferente, obtendo os requisitos sob focos diversos. Analisando-se as características desses métodos, concluiu-se que os três poderiam ser aplicados dentro de um mesmo processo de engenharia reversa, pois cobrem aspectos diferentes, enriquecendo a identificação de requisitos.

A elaboração do processo de engenharia reversa foi dividida da seguinte forma:

1) Comparar as diferenças e as semelhanças dos métodos;
2) Definir as fases e atividades do processo de engenharia reversa, de forma que o processo resultante seja coerente;
3) Compatibilizar os artefatos gerados pelos métodos, para que exista um fluxo consistente de artefatos durante a execução do processo definido.

Isso resultou no processo ERSL (Engenharia Reversa para Software Legado), descrito nesse trabalho.

Para avaliar o processo de engenharia reversa definido, decidiu-se aplicá-lo em um caso real. Isso não foi complicado, pois o autor desse trabalho lida com migração de software legado. Feita a seleção do software legado, o processo ERSL foi aplicado em uma das suas partes, escolhido através do próprio processo. Em seguida, foi feita a análise da aplicação e dos resultados obtidos.

1.5 ORGANIZAÇÃO DO TRABALHO

Este trabalho está estruturado em 6 (seis) capítulos, sendo que cada um tem um objetivo especifico.

O capítulo 1 (Introdução) apresenta e contextualiza o tema deste trabalho. Descreve a motivação para este estudo, o seu objetivo, a justificativa, a metodologia utilizada e a sua organização.

O capítulo 2 (Conceitos e Definições) apresenta os conceitos e as definições necessárias para compreensão do trabalho.

O capítulo 3 (Métodos para Engenharia Reversa) apresenta os três métodos selecionados da literatura para compor o processo ERSL.

O capítulo 4 (Processo de Engenharia Reversa para Software Legado - ERSL) apresenta o processo definido no trabalho, com base na compatibilização dos métodos descritos no capítulo anterior.

O capítulo 5 (Aplicação do Processo ERSL) apresenta a aplicação do processo em um software legado real e a discussão dos resultados obtidos.

O capítulo 6 (Considerações Finais) apresenta as conclusões obtidas pelo desenvolvimento do trabalho, as contribuições geradas pela a pesquisa e os trabalhos futuros que podem ser desenvolvidos.

2 CONCEITOS E DEFINIÇÕES

Este capítulo tem como objetivo definir os conceitos essenciais para este trabalho. Devido à variedade de definições de termos encontrados na literatura, são apresentadas as definições que possuem maior coerência com este trabalho.

2.1 SOFTWARE LEGADO

Esta seção objetiva apresentar os conceitos sobre o software legado, suas características e seus requisitos. Para isso, são apresentadas, inicialmente, as definições de software e sistema, pois vários autores referem-se a esses termos como sinônimos.

2.1.1 Software

De acordo com Pressman (2011), software é definido como um conjunto formado por três itens: programas computacionais, dados e documentação. Os programas computacionais são constituídos de instruções que, ao serem executadas, realizam funções. Os dados são manipulados pelo programa computacional com o intuito de gerar informações. A documentação é a descrição sobre as operações e uso do programa computacional.

O software possui algumas características específicas que o diferenciam de produtos da manufatura; uma das mais citadas é o fato de ser um produto desenvolvido que não se desgasta e é personalizável (PRESSMAN, 2011). Os produtos de software são desenvolvidos para um cliente especifico ou para um mercado e, desta forma, o desenvolvimento do software é em geral customizado para atender ao cliente especifico; existe também o desenvolvimento do software sem customizações, para a venda a vários clientes (SOMMERVILLE, 2011).

2.1.2 Sistema

Sistema (computacional), segundo Sommerville (2011), é um conjunto de: hardware, software, processo, pessoas, entre outros fatores que representam uma

solução para um dado problema. Desta forma, fica claro que software é um item que, em conjunto com outros, forma uma solução.

2.1.3 Características de Software Legado

O software legado é assim denominado, pois foi desenvolvido em décadas passadas e, em geral, não é expansível, possui códigos complexos, de difícil entendimento, pouca ou documentação inexistente (PRESSMAN, 2011). Stroulia et al. (1999) citam que um outro aspecto importante e comum de software legado é o fato de que ele ainda tem grande importância para as organizações; isso ocorre pois é considerado o único repositório de conhecimento sobre os seus negócios.

É fundamental que as organizações saibam qual é a importância do conteúdo do software legado do ponto de vista de seus processos e negócios, para poder tomar a decisão correta no momento em que esses processos de negócio serão mudados. Para realizar uma avaliação sobre isso, devem-se considerar dois fatores importantes: a Qualidade do Software e o Valor de Negócio (SOMMERVILLE, 2011).

2.1.4 Elicitação de Requisitos em Software Legado

Para poder construir um software é necessário realizar o processo de elicitação de requisitos, objetivando identificar as necessidades dos clientes e usuários.

É comum encontrar, na literatura, diversos tipos de processos de engenharia de requisitos voltada para a construção de um novo software. No entanto, quando se trata de trabalhar com software legado, também é necessário identificar os requisitos já implementados, seja para alterá-los ou evolui-los.

Para a realização da elicitação de requisitos em software legado é importante compreender e realizar os processos de Engenharia Reversa e Reengenharia, pois a busca dos requisitos através de pessoas que supostamente conheçam o software legado pode não ser suficiente.

2.2 MANUTENÇÃO DE SOFTWARE

A manutenção de software é o termo usado para um conjunto de atividades voltadas para a modificação em um software que já foi disponibilizado para o usuário. Seus objetivos podem ser correção de defeitos, adaptação para um novo ambiente ou até mesmo a necessidade de adicionar novas funcionalidades com o propósito em atender novos objetivos do usuário (SOMMERVILLE, 2011).

2.2.1 Conceitos sobre Manutenção de Software

Atualmente presencia-se um acelerado avanço nas áreas de tecnologia e um grande crescimento dos negócios nas organizações; isso é motivado pela competividade acirrada entre as organizações. Como o software é uma ferramenta que ultimamente está presente nas organizações e também é um grande aliado para a aceleração de novos negócios, é muito importante manter o software que está em funcionamento, evitando problemas que acabam afetando negativamente os negócios (PRESSMAN, 2011).

Devido aos diferentes objetivos existentes, a manutenção de software pode ser classificada em três tipos (SOMMERVILLE, 2011):

1) Correção de Defeitos: realizar a correção do software devido aos erros de codificação, projeto ou requisitos.
2) Adaptação de Ambiente: adaptar o software quando ocorrem mudanças de hardware, sistema operacional ou outro software que está relacionado.
3) Adição de Funcionalidade: adicionar nova funcionalidade no software, para atender às mudanças de requisitos do negócio.

2.2.2 Processo da Manutenção de Software

O processo de manutenção está incorporado no ciclo de vida de um software, iniciando quando a equipe de desenvolvimento entrega o software, para que o usuário possa utilizá-lo, e somente é finalizado quando ocorre a desativação deste software.

As atividades da manutenção de software normalmente são realizadas pelas equipes da própria organização (PRESSMAN, 2011). Em alguns casos existem

contratos que transferem as responsabilidades da manutenção para equipes terceiras (SOMMERVILLE, 2011).

Tem-se, a seguir, os exemplos de acionamento da manutenção:

1) Assim que se inicia a utilização do software pelo usuário, são identificados defeitos que, por sua vez, são reportados para a equipe de desenvolvimento e, a seguir, são iniciadas as atividades para a realização da manutenção classificada como correção de defeitos.
2) No decorrer da utilização do software, pode acontecer a necessidade de atualizar a versão do sistema operacional em que este software é hospedado, e dessa forma, é identificada a necessidade de realizar uma nova manutenção para adaptar o software ao novo ambiente.
3) Passado algum tempo a organização teve que alterar alguns processos de negócio para poder manter a competitividade do mercado e, desta forma, chega-se à conclusão de que novas funcionalidades devem ser adicionadas ao software, e assim se inicia uma nova manutenção para adição de funcionalidades.

Ao término de cada exemplo citado, a equipe de manutenção disponibiliza uma versão atualizada para o usuário final e essa versão do software é o produto das atividades de manutenção.

É comum que as equipes que realizaram o projeto e construção do software sejam diferentes das equipes responsáveis pela manutenção, pois as organizações montam equipes somente para o desenvolvimento do sistema e, assim que este projeto é encerrado, as equipes são alocadas em outros projetos (CHIKOFSKY; CROSS, 1990). Também é comum encontrar cenários em que a responsabilidade para o projeto ou para a manutenção é transferida para a equipe do usuário ou para equipes terceirizadas e, desta forma, sempre haverá dificuldades relacionadas com a falta de conhecimento sobre o software por parte da equipe de manutenção. Outro ponto que dificulta a manutenção de software é o fato da não utilização de boas práticas de desenvolvimento o que torna o software não manutenível (SOMMERVILLE, 2011).

2.2.3 Manutenção para Software Legado

Quando se trata da manutenção de software legado, devem-se considerar ocorrências do tipo: documentação desatualizada, falta de documentação, não disponibilidade de pessoas que possuem o conhecimento do software, entre outras características de software desse tipo (GEET; EBRAERT; DEMEYER, 2010). Devido a esse tipo de dificuldades, o software legado encontrado nas organizações, acaba passando por pouca manutenção.

Devido a essas características, a manutenção de um software legado exige que avaliações sejam feitas para tomar a melhor decisão sobre como ou o que realizar na sua manutenção. Essas avaliações são divididas em duas partes: a primeira consiste em analisar a importância e necessidade do software legado para o negócio; a segunda é voltada para os aspectos técnicos que consistem em avaliar a sua qualidade (SOMMERVILLE, 2011).

As organizações que utilizam o software legado ainda continuam passando por mudanças de negócio e de processos de negócio e, desta forma, necessitam de equipes responsáveis para realizar manutenção desse tipo de software. Sommerville (2011) cita quatro tipos de estratégias para a manutenção de software legado:

1) Descartar: essa estratégia é embasada nos resultados da análise dos processos de negócio, quando se chega à conclusão de que o software não é mais importante para a organização, devido às mudanças que ocorreram nos processos.
2) Não alterar: essa estratégia está relacionada ao fato de o software legado ainda ser uma ferramenta importante para os processos de negócio e ele se encontra estável com poucas solicitações de mudança.
3) Reestruturar: essa estratégia também está relacionada à importância que o software legado tem para os processos de negócio; porém devido a diversas mudanças que ocorreram anteriormente o software ficou degradado, porém ainda existem muitas novas solicitações de mudanças. Para executar essa estratégia são criados componentes no software legado objetivando criar interfaces com o novo software a ser desenvolvido.
4) Substituir: estratégia pode ser feita de forma total ou parcial, sendo motivada semelhantemente à estratégia de restruturação; a diferença é que

o custo para continuar mantendo o software restruturado é semelhante ao custo de substituir por um novo software, ou a manutenção para adaptação de ambiente não é mais possível.

2.3 ENGENHARIA REVERSA DE SOFTWARE

A engenharia reversa consiste em analisar um produto já construído com a finalidade de identificação de seus componentes e interdependências, e como resultado, criar formas de representação que descrevem as características desse produto (CHIKOFSKY; CROSS, 1990).

Segundo Pressman (2011) a engenharia reversa teve origem na competitividade de organizações fabricantes de hardware. Com o objetivo de conhecer os possíveis segredos do produto e sem a possibilidade de obter a sua documentação, as organizações desmontavam o produto de seu concorrente para identificar como é o funcionamento que estava implícito em seus componentes internos.

Essa seção objetiva definir os conceitos da engenharia reversa e introduzir esse conceito no contexto de software legado.

2.3.1 Conceitos sobre Engenharia Reversa

A engenharia reversa de software é o processo para criar uma representação que possa descrever as suas características, através da identificação do funcionamento do software (PRESSMAN, 2011).

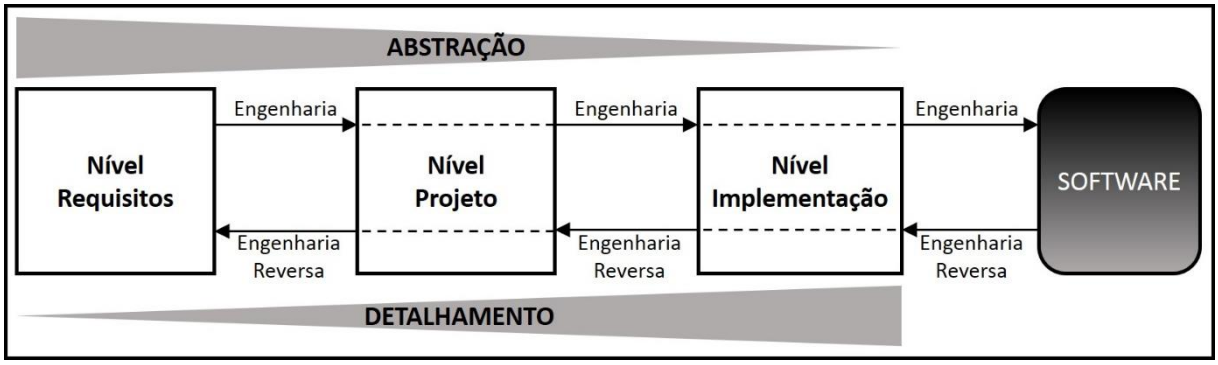
Segundo Chikofsky e Cross (1990), a engenharia reversa é somente um processo com o objetivo em analisar o software, sendo assim o termo não contempla um processo que realiza mudanças nesse software.

Para poder realizar o processo da engenharia reversa de software, é necessário definir os níveis de abstração em uma representação do software. Conforme a definição de Pressman (2011) a divisão da abstração em diferentes níveis auxilia o entendimento da solução que o software implementa, sendo que nos níveis mais altos de abstração a solução é representada em forma de linguagem de domínio;

em níveis medianos a solução possui maior detalhamento e em níveis mais baixos o detalhamento está próximo à implementação.

Conforme Chikofsky e Cross (1990), os níveis de abstração podem ser classificados em três grupos: Nível de Requisitos, Nível de Projeto e Nível de Implementação. A Figura 1 é uma representação desses três níveis.

Figura 1 - Níveis de Abstração em Engenharia e Engenharia Reversa



Baseado em: Chikofsky e Cross (1990)

As setas superiores representam a direção em que a engenharia de software evolui, até chegar ao software em funcionamento, as setas inferiores representam a direção em que a engenharia reversa trabalha até chegar ao nível dos requisitos. Entende-se que a engenharia possui um foco em aumentar o detalhamento das informações até chegar ao software em funcionamento; a engenharia reversa possui a finalidade de reduzir o detalhamento do código e, consequentemente, aumentar o nível de abstração. Desta forma, conclui-se que o nível de detalhamento é inversamente proporcional ao nível de abstração.

Pressman (2011) define três tipos de especificações que são tratadas no processo de engenharia reversa:

1) Dados: voltada a identificar dois tipos: Estruturas Internas de Dados que identifica variáveis internas, estruturas de registros, arquivos, listas, entre outras estruturas de dados; e Estrutura de Base de Dados que identifica o relacionamento entre os dados.

2) Processamento: voltada ao entendimento procedural contido no código-fonte.
3) Interfaces de Usuário: voltada ao entendimento do comportamento do software perante as ações do usuário.

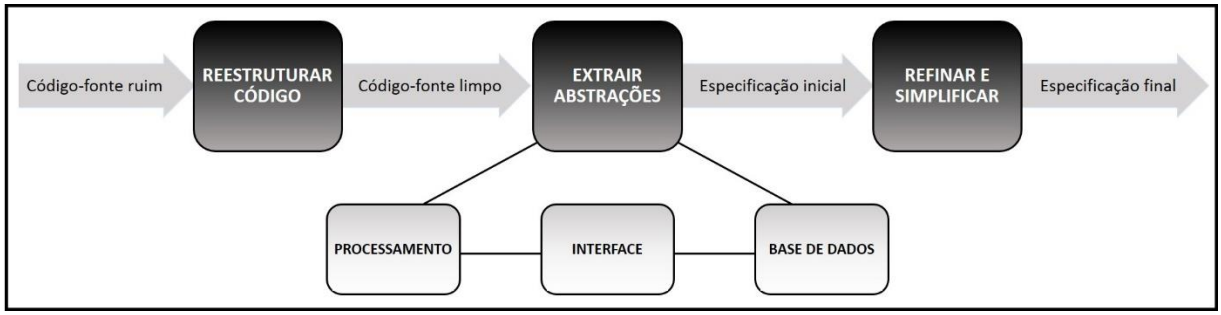
Os três tipos de especificação citados compõem as informações necessárias para poder desenvolver a especificação ao executar a elicitação de requisitos no processo de engenharia reversa de software (PRESSMAN, 2011).

2.3.2 Processo da Engenharia Reversa

Em linhas gerais, o processo de engenharia reversa pode ser definido através de três atividades: reestruturar código, extrair abstrações, refinar e simplificar (PRESSMAN, 2011).

A Figura 2 exemplifica um processo de engenharia reversa de software segundo Pressman (2011). Pode-se observar que a entrada para o processo é o código-fonte do software sobre o qual será realizada a engenharia reversa, o qual em geral é considerado de baixa qualidade. Esse artefato é compreendido e reestruturado, gerando o código-fonte limpo, com módulos organizados, para melhor compreensão. A partir do código-fonte limpo são extraídas as abstrações de processamento; interface e base de dados. O resultado é a especificação inicial, que é refinada e simplificada para obter a especificação final, que é o artefato de saída do processo de engenharia reversa (PRESSMAN, 2011).

Figura 2 - Processo de Engenharia Reversa de Software



Fonte: Pressman (2011)

2.3.3 Engenharia Reversa para Software Legado

Em geral, o termo de engenharia reversa de software está relacionado com a aplicação em software legado, devido à falta de documentação, obrigando as equipes a analisar o código-fonte para obter o conhecimento sobre as suas características e suas funções.

Para realizar a manutenção em software, é necessário conhecer sua arquitetura e seus detalhes de implementação, porém como essas informações raramente estão presentes na documentação e os especialistas que conhecem o software legado não estão mais presentes, torna-se necessária a execução do processo de engenharia reversa (GEET; EBRAERT; DEMEYER, 2010).

Além da necessidade de alterar ou corrigir as funções do sistema, um outro fator que também impulsionam a aplicação da engenharia reversa no software legado, é o avanço tecnológico. Dessa forma, devido ao desenvolvimento de novas plataformas de hardware, que aumentam o desempenho de software, à necessidade de modernização de software e também à diversidade de software de prateleira, as organizações estão substituindo o software legado por novo software (STROULIA et al., 1999).

A Tabela 1 apresenta a utilização da engenharia reversa nas estratégias de manutenção de software legado, segundo Sommerville (2011).

Tabela 1 - Estratégias de Manutenção de Software Legado X Engenharia Reversa

ESTRATÉGIA	ENGENHARIA REVERSA
Descartar	Não utiliza
Não alterar	Utiliza pouco
Reestruturar	Utiliza
Substituir	Utiliza

Baseado em: Sommerville (2011)

A utilização da engenharia reversa em software legado está fortemente relacionada às estratégias para a sua manutenção pois, para poder aplicar qualquer

estratégia, é necessário conhecer este software (SOMMERVILLE, 2011). A partir da Tabela 1, pode-se constatar que a utilização da engenharia reversa na estratégia de manutenção de software legado aumenta conforme aumenta a necessidade do entendimento do software.

Observa-se, portanto, que a utilização da engenharia reversa para software legado é comum para resolver os problemas da falta de conhecimento, documentação e especialistas que conhecem o software durante a sua manutenção. Desta forma, a engenharia reversa pode recuperar informações para a criação de representações em níveis mais abstratos.

2.4 REENGENHARIA DE SOFTWARE

Essa seção objetiva definir e esclarecer os conceitos da reengenharia e relacionar esse conceito no contexto de software legado.

2.4.1 Conceitos sobre Reengenharia

A reengenharia consiste na realização de análises e alterações de um sistema com a finalidade de reconstruir o produto (CHIKOFSKY; CROSS, 1990).

Para Sommerville (2011), a reengenharia de software tem o objetivo de analisar um software existente e criar uma nova versão melhorada e reestruturada, sem alterar as funcionalidades. A reengenharia pode ser aplicada no software de várias formas. Através da redocumentação, da refatoração da arquitetura, da mudança da linguagem de programação, da alteração ou modificação das estruturas do software.

2.4.2 Processo de Reengenharia

Na literatura, encontram-se diversos modelos de processo de reengenharia e cada processo possui suas particularidades. Mesmo com essas diferenças, existem pontos em comum nas atividades principais da reengenharia, tais como a engenharia reversa, a reestruturação e engenharia direta.

Pressman (2011) define um modelo de processo de reengenharia dividido em 6 atividades:

1) Análise de Inventário: criar uma ordem de priorização para a execução da reengenharia;
2) Reestruturação dos Documentos: identificar a existência de documentos sobre o software e decidir se é necessário redocumentar;
3) Engenharia Reversa: conhecer o software e como está implementado;
4) Reestruturação do Código: realizar as devidas alterações no código-fonte do software, tais como a refatoração e mudança de linguagem;
5) Reestruturação dos Dados: melhorar o modelo e arquitetura dos dados;
6) Engenharia Direta: reconstruir o software.

2.4.3 Reengenharia para Software Legado

A reengenharia é, em geral aplicada ao software legado, para viabilizar a sua manutenção.

A Tabela 2 apresenta o relacionamento entre as estratégias de manutenção de software legado e a aplicabilidade da reengenharia, segundo Pressman (2011) e Sommerville (2011).

Tabela 2 - Estratégias de Manutenção de Software Legado X Reengenharia

ESTRATÉGIA	REENGENHARIA
Descartar	Não utiliza
Não alterar	Não utiliza
Reestruturar	Utiliza
Substituir	Utiliza

Baseado em: Pressman (2011) e Sommerville (2011)

Pode-se observar que a reengenharia está diretamente ligada as estratégias de reestruturação e substituição do software.

2.5 MIGRAÇÃO DE SOFTWARE LEGADO

A migração de um software normalmente é motivada por necessidades de evolução tecnológica e de negócios.

2.5.1 Conceitos sobre Migração de Software Legado

Conceitualmente, a migração de software consiste na substituição total ou parcial de um software por outro software, sendo, que para Sommerville (2011), a substituição de um software é realizada através da implementação de componentes principais em um novo software.

A decisão pela migração de software ocorre quando a manutenção do software passa a não atender mais os requisitos de negócio, devido a seus fatores tecnológicos e estruturais. Desta forma, é necessário proporcionar melhores condições que somente podem ser obtidas com a migração deste software (SALVATIERRA et al., 2013).

Salvatierra et al. (2013) descrevem dois tipos de estratégias para a migração de um software legado, a Migração Direta e a Migração Indireta.

A migração direta, também chamada de *black-box* (caixa-preta) consiste em criar uma espécie de capa envoltória para o software legado, a qual é um novo software que serve como uma interface para outros softwares e usuários do software legado. As funcionalidades do software legado ficam encapsuladas e as interações passam a ser realizadas através do novo software.

A migração indireta, também chamada de *white-box* (caixa-branca) objetiva reimplementar completamente o software legado e, nesse caso, torna-se necessária a aplicação de engenharia reversa e reengenharia para obter o conhecimento e os requisitos do software legado e, assim, criar ou adaptar esses requisitos através de um novo software.

Considerando o conceito de Pressman (2011) em que o software é um conjunto de programas computacionais, dados e documentação, a migração pode ser feita considerando cada um destes itens.

A migração dos programas computacionais e a documentação são realizadas em conjunto, pois para poder abstrair os requisitos do software legado, é necessário documentar em níveis adequados de abstração. A migração dos dados necessita de estratégias específicas para o tratamento de dados que envolvem extrações, transformações e cargas dos dados entre o software legado e o software destino (WU et al., 1997).

2.5.2 Métodos de Migração de Software Legado

O processo para realizar a migração de um software legado possui variações, dependendo se é uma migração total ou parcial e existem métodos mais apropriados para cada cenário. Salvatierra et al. (2013) e Wu et al. (1997) descrevem três tipos de métodos para a migração de um software legado: *Cold Turkey*, *Chicken Little* e *Butterfly*.

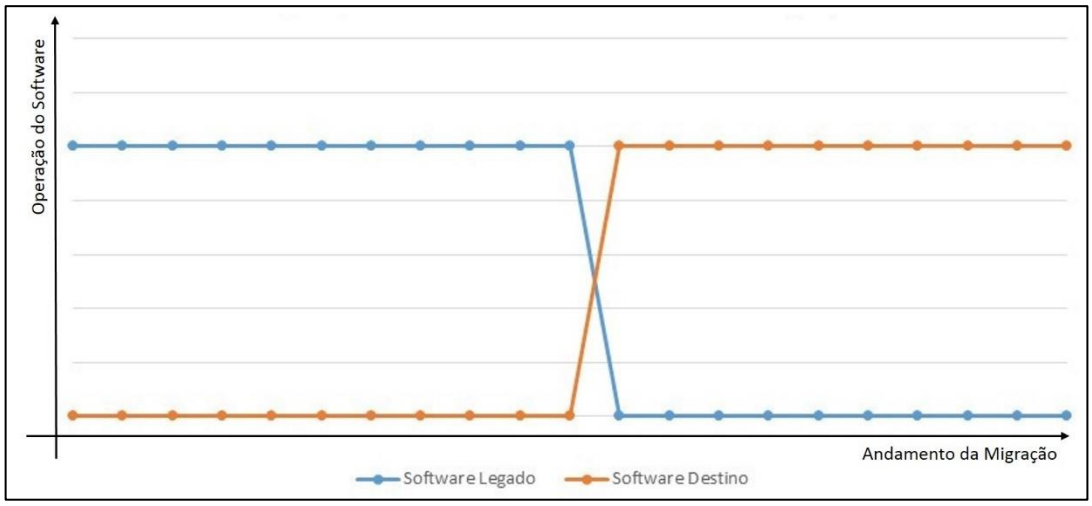
Cada método citado possui uma estratégia para a migração do software legado. O método *Cold Turkey* apresenta a migração total do software legado de forma que o novo software é implantado no ambiente produtivo de uma vez só. O método *Chiken Little* apresenta uma migração gradativa do software legado, permanecendo com as duas versões em produção até que a migração total ocorra. O método *Butterfly* se diferencia dos anteriores pois implanta o software destino em um ambiente de teste para depois mover esse novo software para o ambiente produtivo.

A Figura 3, a Figura 4 e a Figura 5 apresentam graficamente a operação do software legado e o software destino em um ambiente produtivo no decorrer da migração. Em todas as figuras, o eixo vertical representa o percentual em que o software está em operação no ambiente produtivo, e o eixo horizontal representa o tempo.

A Figura 3 apresenta a migração com método *Cold Turkey.* Neste caso, o software legado fica 100% em operação até que o software destino seja implantado, substituindo a sua operação no ambiente produtivo e, assim, o software legado é desativado. A Figura 4 apresenta a migração com método *Chicken Little*, neste caso software legado fica em operação enquanto o software destino é implantado

gradativamente; em um determinado instante, o software legado e o software destino ficam 100% em operação até que o software legado é desativado gradativamente.

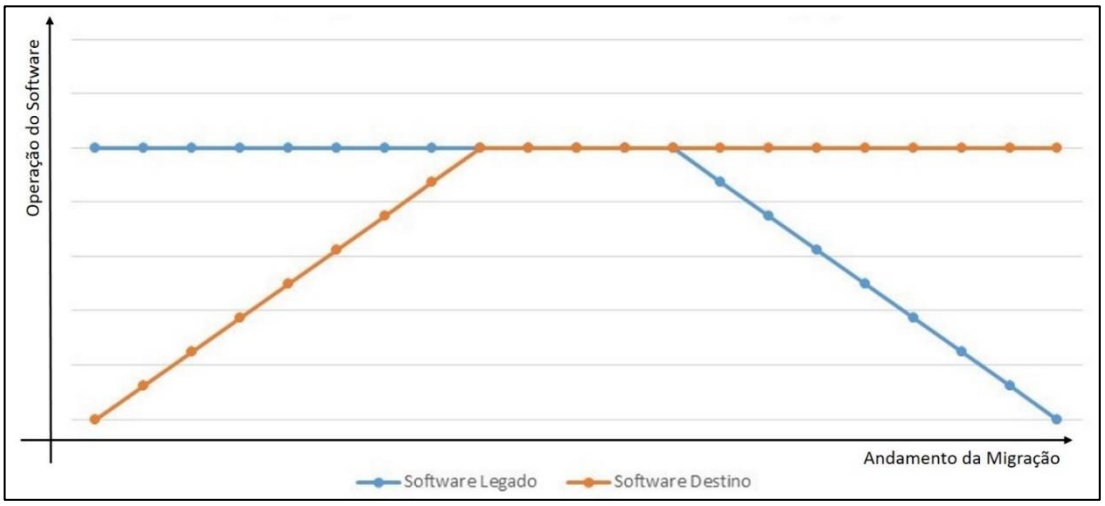
Figura 3 - Operação do Software X Andamento da Migração com *Cold Turkey*



Baseado em: Salvatierra et al. (2013)

A Figura 4 apresenta a migração com método *Chicken Little.* Neste caso, o software legado fica em operação enquanto o software destino é implantado gradativamente; em um determinado instante, o software legado e o software destino ficam 100% em operação até que o software legado é desativado gradativamente.

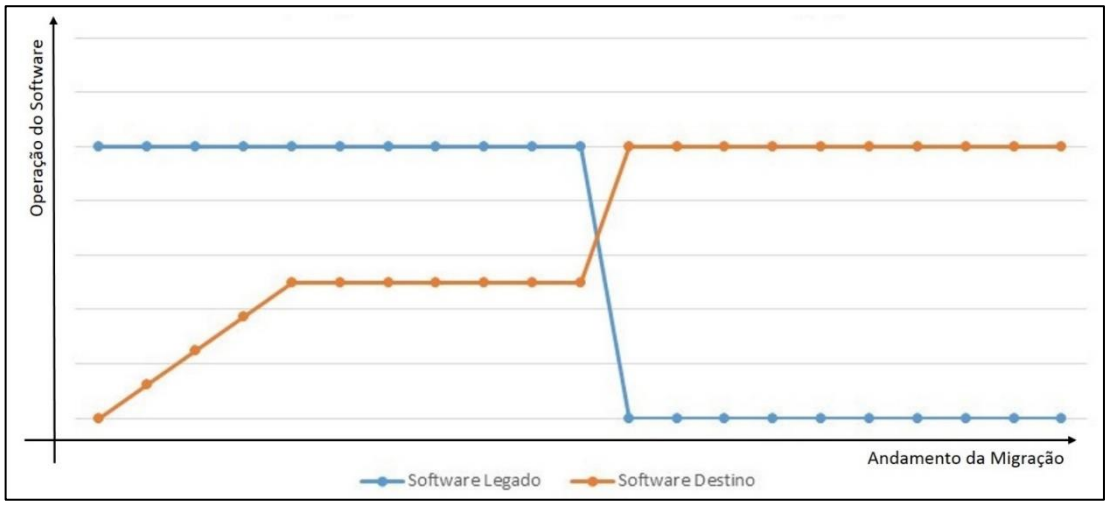
Figura 4 - Operação do Software X Andamento da Migração com *Chicken Little*



Baseado em: Salvatierra et al. (2013) e Wu et al. (1997)

A Figura 5 apresenta a migração com o método *Buttlerfly*. Neste caso, o software legado fica em operação enquanto o software destino é implantado gradativamente em um ambiente de testes; quando o software destino estiver estável no ambiente de testes, ele é implantado no ambiente produtivo e o software legado é desativado.

Figura 5 - Operação do Software X Andamento da Migração com *Butterfly*



Baseado em: Salvatierra et al. (2013) e Wu et al. (1997)

Cada um dos métodos possui vantagens e desvantagens, porém é necessário sempre avaliar o contexto em que será realizada a migração do software legado para fazer a melhor escolha.

2.6 CONSIDERAÇÕES DO CAPÍTULO

Neste capítulo foram definidos os conceitos essenciais para este trabalho.

Apesar dos termos sistema legado e software legado serem usados indistintamente por diversos autores, no contexto deste trabalho é utilizado o termo software legado.

O entendimento do software e dos novos requisitos é importante na manutenção de qualquer tipo de software. No entanto, no contexto de software legado, é necessário técnicas de engenharia reversa e reengenharia, pois muitas vezes só se

tem o código como documento. Através da engenharia reversa obtém-se o entendimento do software e, se for necessário reconstruir o software, usam-se as técnicas de reengenharia, para reconstruir ou reestruturar o software legado.

O conceito de migração de software legado é importante para o entendimento do contexto em que este trabalho está inserido, pois a engenharia reversa e a reengenharia são aplicadas iterativamente para as partes do software legado a ser migrado.

Dentre as quatro estratégias de manutenção de software propostas por Sommerville (2011), este trabalho foca na estratégia de substituição, que visa realizar a migração total ou parcial do software legado para um novo software. Esse assunto será tratado e detalhado nas próximas seções.

No contexto deste trabalho, considera-se reengenharia, as atividades que ocorrem após a engenharia reversa, ou seja, depois de ter os requisitos gerados pela engenharia reversa. Desta forma, o modelo de processo de reengenharia proposto por Pressman (2011) foi dividido como:

1) Engenharia Reversa:
    a. Análise de Inventário
    b. Reestruturação dos Documentos
    c. Engenharia Reversa
2) Reengenharia:
    a. Reestruturação do Código
    b. Reestruturação dos Dados
    c. Engenharia Direta

Esta definição consiste em tornar clara a utilização do termo de reengenharia neste trabalho. Desta forma, quando ocorrer referências à reengenharia, deve-se somente considerar as atividades de reestruturação do código e dos dados e a engenharia direta.

3 MÉTODOS PARA ENGENHARIA REVERSA

Este capítulo tem como objetivo descrever os principais métodos de engenharia reversa de software legado, selecionadas durante a pesquisa bibliográfica realizada. Eles apresentam diferentes formas de abordagem, com focos diversos.

Inicialmente é apresentado o objetivo geral que esses métodos possuem em comum para realizar a engenharia reversa e, a seguir, é feita a descrição de cada um deles, citando as vantagens e as desvantagens.

3.1 MÉTODOS DE ENGENHARIA REVERSA PARA SOFTWARE LEGADO

Existem diversos tipos de métodos para realizar a engenharia reversa de software legado, pois a sua eficácia depende de cada software legado, uma vez que pode ser de diversos tipos e, normalmente, possui pouca documentação e/ou a documentação existente está desatualizada, dificultando o entendimento sobre os requisitos que este software implementa.

Ao analisar os métodos encontrados na literatura, observou-se que os autores tratam o tema de engenharia reversa com base no contexto específico em que o software legado está inserido e propõem, por causa disso, formas diferentes de abordagem, que podem até ter objetivos diferentes. Por exemplo, existem autores que propõem automatizar os processos de engenharia reversa, enquanto outros buscam melhorar o seu processo de forma quantitativa e qualitativa. Porém, é perceptível que eles têm, como objetivo, a manutenção do software legado, independentemente de ser apenas manutenção evolutiva, manutenção corretiva ou até mesmo migração total ou parcial do software em questão.

Conforme mencionado, a diversidade de tipos de software legado influencia a seleção do método de engenharia reversa a ser utilizado, pois o estado em que o software ou documentação se encontra requer um método mais apropriado. Nem sempre um método existente é adequado, sendo necessário adaptá-lo para uma dada situação; a tarefa de engenharia reversa não é trivial, não existindo procedimentos padrões para realizar a elicitação de requisitos. Para isso, é necessário entender

inicialmente o cenário em que o software legado se encontra, para então selecionar e aplicar o método que melhor atenda esse cenário.

Durante o desenvolvimento desse trabalho, houve dificuldade de se encontrar um método ideal para o cenário de interesse do autor. Dentre os trabalhos analisados, foram elencados três métodos com abordagens diferentes que pareceram promissores para apoiar a Engenharia Reversa de Software Legado. Nas próximas seções deste capítulo são apresentados cada um destes métodos.

3.2 ENGENHARIA REVERSA ATRAVÉS DE INTERFACES DE SISTEMAS LEGADOS

O método de Engenharia Reversa proposto por Stroulia et al. (1999), tem como objetivo o mapeamento de requisitos com base na análise da interface do usuário. Esse método é composto por duas tarefas principais:

1) Mapeamento de Interface, que objetiva construir um mapa que relaciona as interfaces de interesse do software, através das transições entre elas;
2) Modelagem de Domínio, que consiste em analisar o mapa da interface resultante e criar o gráfico de interface. Este diagrama apresenta as tarefas que o usuário precisa realizar, para que o software atinja o seu objetivo, e passa a ser uma especificação para a criação de uma nova interface gráfica.

3.2.1 Abordagem

Stroulia et al. (1999) defendem que o software legado é um grande repositório de conhecimento, porém raramente possui documentação e apresenta grande complexidade para utilização. Com a evolução da tecnologia e também dos negócios, as empresas encontram dificuldades em manter um software legado em funcionamento. Dentre essas dificuldades estão: a realização de treinamento de pessoas para que possam utilizar o software legado e a necessidade de oferecer acesso ao seu software legado a parceiros e clientes para acompanhar a evolução do mercado. Isto nem sempre é viável, pois é necessário implementar também requisitos voltados a segurança.

Com base nessas dificuldades, o método de Engenharia Reversa através de Interface de Sistemas Legados aborda o entendimento da lógica do processo no contexto em que o software legado está inserido. O método propõe analisar como os usuários interagem com o software e, assim, identificar as atividades que são realizadas através dele, pois o uso da interface fornece muitas informações sobre o software.

Este método não considera as informações internas do software, tais como estrutura de dados e algoritmos, pois seu foco é realizar a migração somente da interface de usuário do software. Entende-se também que toda interface é uma unidade de apresentação de informações para o usuário e que as ações que o usuário realiza refletem as transições entre cada interface e modificam um estado interno do software.

3.2.2 Aplicação do Método

O processo para a aplicação do método de Engenharia Reversa através de Interface de Sistemas Legados utiliza software de apoio (CELLEST e URGENT) para a realização das tarefas de Mapeamento de Interface e Modelagem de Domínio.

O ambiente CELLEST é composto por dois *middlewares* (Recorder e Pilot) que auxiliam na automatização de coleta de informações. O Recorder é responsável por coletar as informações referentes à interação do usuário com o software legado, tais como: dados preenchidos em um formulário, cliques em botões, informações apresentadas em tela, entre outros. O Pilot é um software responsável por simular um usuário utilizando o software legado.

O ambiente URGENT é utilizado em conjunto com o CELLEST, porém possui uma função diferente. Ele utiliza as informações que foram coletadas pelo Recorder, identifica possíveis constantes e variáveis que o software legado utiliza e constrói um modelo em forma de diagrama que apresenta a navegação entre cada interface de usuário. Esse diagrama é base para uma nova interface gráfica gerada pelo URGENT e que é operada pelo software Pilot. Desta forma o Pilot automatiza os testes na nova interface, simulando um usuário real.

Tem-se, a seguir, a descrição das tarefas de Mapeamento de Interface e Modelagem de Domínio.

3.2.2.1 Mapeamento de Interface

O mapeamento de interface é realizado com o usuário utilizando o software legado, para que ele identifique cada interface relacionada ao contexto relevante às suas atividades no software. Durante a utilização, o software Recorder do ambiente CELLEST coleta as informações que estão relacionadas a cada interface acionada pelo usuário.

Ao termino da utilização do usuário, as informações coletadas são agrupadas em três tipos de conjuntos:

1) Família de Itens Comuns: composta por itens comuns da interface de usuário, tais como códigos de identificação de tela, títulos de tela, data e hora.
2) Família de Padrões Geométricos: composta por itens que possuem forma geométrica, tais como códigos e localização dos campos, símbolos e caracteres especiais.
3) Família de Características Específicas: composta por itens que auxiliam na identificação da finalidade da interface de usuário, tais como palavras chave e posicionamento das informações.

Após o agrupamento das informações coletadas, através do acionamento de cada interface, são identificadas as transições entre cada interface, com base também nas informações coletadas pelo Recorder. Essas informações são classificadas como: eventos que geram a transição e pré-condição para a transição.

3.2.2.2 Modelagem de Domínio

O software URGENT é responsável por realizar a Modelagem de Domínio e utiliza as informações geradas através da tarefa de Mapeamento de Interface para elaborar, de forma automática, um diagrama chamado de Gráfico de Interface.

O Gráfico de Interface é constituído por nós que representam individualmente cada interface do software legado e por conectores que ligam os nós, correspondendo às transições entre cada interface de usuário.

Após a geração do Gráfico de Interface, o software URGENT cria uma nova interface gráfica relativa à interface do software legado, utilizando as informações coletadas e o diagrama gerado. Essa nova interface é submetida a testes pelo software Pilot do ambiente CELLEST como uma forma de validar se a tarefa de Modelagem de Domínio foi realizada corretamente, ou seja, se o Pilot realiza na nova interface as mesmas ações que o usuário fez no software legado.

3.2.3 Vantagens e Desvantagens

O método de Engenharia Reversa através de Interface de Sistemas Legados apresenta vantagens, porém isso dependerá do objetivo final pretendido com a aplicação da engenharia reversa.

Pode-se citar como uma vantagem, o fato deste método considerar a preocupação em entender o processo em que o software legado está inserido, identificando as ações que o usuário realiza e, assim, apresentando representações do software em um nível maior de abstração. Além disso, o Gráfico de Interface gerado pelo método é uma consolidação dos requisitos que são visíveis para os usuários.

Também se pode citar que este método fornece suporte para uma migração da interface de usuário do software legado para uma nova interface gráfica gerada automaticamente com base nas informações coletadas durante a utilização do software pelo usuário.

No entanto, conforme mencionado anteriormente, a eficiência deste método depende do objetivo final da sua aplicação. Se o objetivo for uma engenharia reversa da interface e/ou um entendimento das interações entre usuário e software, o resultado pode ser considerado satisfatório. Mas o resultado deste método pode não ser apropriado, se o objetivo for a identificação de aspectos internos do software, tais como identificação de seus componentes e suas inter-relações, ou ainda, identificação de maiores detalhes sobre as regras de negócio que estão implícitas no software.

Conclui-se que as vantagens e desvantagens dependem do objetivo final, independente do contexto em que o método for aplicado.

3.3 REDOCUMENTAÇÃO DE SISTEMAS LEGADOS

O método de Engenharia Reversa, proposto por Geet, Ebraert e Demeyer (2010), tem como objetivo redocumentar o software legado, para recuperar o conhecimento que está embutido de forma implícita no software e explicitá-lo na nova documentação. Dessa forma, a análise dos novos documentos possibilita o entendimento das funções do software. Este método apresenta pontos essenciais relacionados sobre o que deve e o que não deve ser documentado e também, sobre o que pode e o que não pode ser automatizado na sua aplicação.

3.3.1 Abordagem

Devido ao software legado ter como características a falta de documentação e/ou documentação desatualizada, o trabalho para a realização de sua manutenção ou sua migração torna-se muito difícil. É comum que o conhecimento sobre o software não seja facilmente acessível, pois não foi devidamente documentado pelos especialistas que o criaram e, em geral, eles não estão mais disponíveis para auxiliar neste processo, pelo fato de estarem aposentados ou em outras ocupações.

Com base nessas dificuldades, os autores definiram um método que aborda a redocumentação do software legado com o foco na realização de um tipo de engenharia reversa que consiste em extrair informações disponíveis no código-fonte recriando a documentação baseada em diagramas da UML.

A realização manual do trabalho torna a redocumentação um trabalho tedioso, mas a tentativa de automatizar o processo também não é trivial, pois existem atividades que necessitam de interpretação humana, como por exemplo, a obtenção dos passos lógicos que o software executa. Desta forma, o método de Redocumentação de Sistema Legado possui partes automáticas e partes manuais, objetivando redocumentar o software legado para uma futura migração de plataforma.

Diferentemente do método de Engenharia Reversa através da Interface de Sistemas Legados proposto por Stroulia et al. (1999), esse método propõe gerar

partes da documentação técnica, incluindo informações internas de software, tais como estrutura de dados e algoritmos. Estas informações são utilizadas como base para a documentação funcional e/ou a documentação de domínio.

3.3.2 Aplicação do Método

O método de Redocumentação de Sistema Legado utiliza o conceito de que o código-fonte possui informações implícitas. Essas informações são chamadas de fatos, que podem ser estruturais ou semânticos. Os fatos estruturais são: nomes de programas, dependências funcionais, dependências de dados e estruturas de dados. Os fatos semânticos são: efeito de uma função e passos lógicos dentro da função.

Com o objetivo de organizar os fatos e automatizar tarefas do método, é utilizado o ambiente Rigi Reverse Engineering Environment (RREE) para a representação dos fatos identificados e o ambiente MEGA, para gerar os documentos usando a representação UML.

O ambiente RREE é utilizado para importar um arquivo texto que contém os fatos identificados e gerar um arquivo do tipo Rigi Standard Format (RSF). O formato RSF é semelhante ao do XML, pois consiste de uma sequência de tuplas. É utilizado este formato como pré-requisito para convertê-lo novamente em um padrão de importação no ambiente MEGA.

O ambiente MEGA é composto por ferramentas que convertem os arquivos RSF em diagramas UML e geram código-fonte com base nos modelos criados utilizando um repositório que armazena os dados correspondentes.

O processo para a aplicação deste método consiste de três tarefas: Extração dos Fatos, Representação dos Fatos e Geração da Documentação.

3.3.2.1 Extração dos Fatos

A tarefa de extração dos fatos possui atividades automáticas e manuais. Os fatos são extraídos a partir de três tipos de artefatos: representações estáticas do software (código-fonte), documentos disponíveis e dados obtidos a partir de pessoas,

que trabalham com o software, tais como: usuários, equipes que projetaram o software e equipes de manutenção.

A extração de fatos do código-fonte é feita automaticamente através de *scripts* que realizam a coleta de informações técnicas, tais como constantes, variáveis, algoritmos e relacionamento de dados. Essas informações denominadas de fatos estruturais são extraídas para um relatório.

A extração manual de fatos é feita através dos documentos existentes e das pessoas que trabalham com o software. O entendimento do funcionamento do software é obtido através dos diversos documentos (incluindo o código-fonte) e através do conhecimento extraído das pessoas que trabalham com o software. O resultado obtido também é armazenado em um relatório.

Após as extrações dos fatos, os dois relatórios gerados são agrupados em um novo relatório que deve estar em formato do tipo texto, utilizando o caractere (;) como separador de informações.

3.3.2.2 Representação dos Fatos

Para gerar a representação dos fatos, são executados os *scripts* que importam os fatos contidos no arquivo texto gerado, durante a tarefa de extração, para o ambiente RREE. Após a importação, os fatos são convertidos e exportados resultando um arquivo do tipo RSF em que as tuplas contém a informação extraída e a sua classificação.

O arquivo RSF é uma representação padrão dos fatos e esse formato é necessário para unificar todas as informações obtidas, servindo como um pré-requisito para gerar a documentação.

3.3.2.3 Geração de Documentação

Para gerar a documentação, é necessário converter o arquivo RSF em um formato padrão que permita a importação no ambiente MEGA através da ferramenta CrocoPat. Os fatos são importados automaticamente para uma biblioteca de objetos que fica disponível para a modelagem da documentação. Com os objetos disponíveis

na biblioteca, o projetista constrói, de forma manual, a documentação em formato de diagramas da UML no próprio ambiente MEGA.

Como resultado final, tem-se a documentação do software em diversos diagramas da UML, que ficam disponíveis nos repositórios de modelos do ambiente MEGA.

3.3.3 Vantagens e Desvantagens

O método de Redocumentação de Sistemas Legados apresenta uma série de vantagens, porém como é um método híbrido, é necessário que os resultados das partes automáticas sejam interpretados corretamente pelos projetistas, para que os resultados das atividades manuais estejam corretos.

Pode-se descrever como principal vantagem, a automação das atividades tediosas da extração de informações do código-fonte, poupando esforços dos projetistas e também melhorando a qualidade do resultado. Outro ponto importante é o tratamento referente aos fatos. Sua representação apresenta uma padronização, ou seja, os fatos são representados de forma única, independentemente do seu tipo e da sua origem, tornando mais fácil a sua manipulação na geração do novo documento que consistem em diagramas da UML.

Além das vantagens mencionadas, é importante que a equipe técnica esteja ciente em selecionar apenas os documentos relevantes, pois apesar das informações terem sido disponibilizadas no repositório através das tarefas automatizadas de extração, a documentação em excesso pode dificultar a manutenção futura dos documentos gerados.

Um ponto importante a ser definido antes da aplicação do método é definir a finalidade da documentação a ser gerada. Se os documentos forem focados na engenharia reversa do software, não serão adequados para os usuários de software e vice-versa. A natureza e a profundidade da documentação devem ser, portanto, definidas em função do objetivo da aplicação do método.

Conclui-se que as vantagens e desvantagens dependem do cenário, em que o método é aplicado, pois as ferramentas podem não estar disponíveis durante a

aplicação do método, necessitando de uma customização nas atividades deste método.

3.4 ENGENHARIA REVERSA ATRAVÉS DE MODELOS DE META

O método de Engenharia Reversa através de Modelos de Metas, proposto por Yu et al. (2005), tem como objetivo a criação de modelos para representar as metas desejadas pelo usuário e as funções de software necessárias para atingir essas metas. Este método propõe a elicitação de requisitos a partir da análise e interpretação de informações contidas no código-fonte, transformando-as em modelos. Seu produto final é o Modelo de Metas, que representa os requisitos encontrados no código-fonte, para servir como base da realização da Reengenharia do Software Legado.

3.4.1 Abordagem

Os autores propõem um método para a Engenharia Reversa de Software Legado, com base em informações que não seja apenas código fonte. Esse método tem por objetivo:

- Compreender, refatorar e estruturar o código-fonte;
- Identificar os requisitos funcionais para extrair as metas do usuário;
- Identificar os requisitos não funcionais implícitos no código-fonte e no entendimento da funcionalidade do software.

Quando a engenharia reversa é realizada apenas pela análise de código-fonte, é comum que se identifiquem detalhes técnicos de implementação; porém os resultados importantes são as suas funções e não esses detalhes técnicos. Então, como em geral tem-se somente o código-fonte como artefato inicial, esse método procura dividir o programa em partes que façam sentido, ou seja, que possam ser analisadas separadamente para poder organizar o entendimento do software.

Para um melhor entendimento, é apresentada inicialmente os padrões do código para a extração do Modelo de Metas e, a seguir, a aplicação do método.

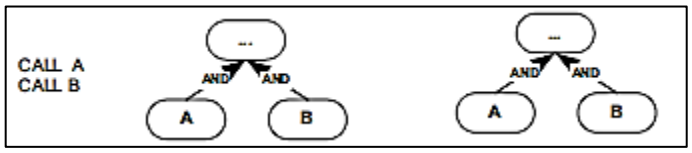
3.4.2 Padrões do código para Extração do Modelo de Metas

O Modelo de Metas representa a lógica do código-fonte através de uma árvore em que as funções são representadas pelos nós e a dependência entre elas, através de arestas.

Para a construção do modelo, devem-se identificar os padrões do código-fonte, através de palavras-chave; os mais relevantes são invocações de outros métodos ou funções (CALL), estrutura de decisão (IF/ELSE) e estrutura de repetição (REPEAT).

A Figura 6 representa o padrão de código-fonte e seu respectivo padrão gráfico para a invocação de um método ou função.

Figura 6 - Padrão de invocação de método ou função

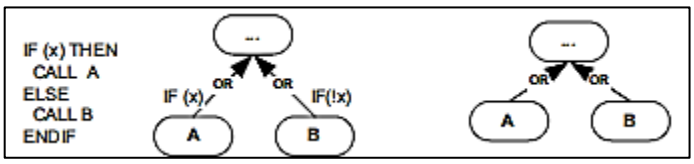


Fonte: Yu et al. (2005)

A parte da esquerda apresenta o padrão de código da chamada sequencial de A e B. A parte da direita apresenta o padrão gráfico correspondente às chamadas sequenciais.

A Figura 7 representa o padrão de código-fonte e seu respectivo padrão gráfico para a estrutura de decisão.

Figura 7 - Padrão de estrutura de decisão

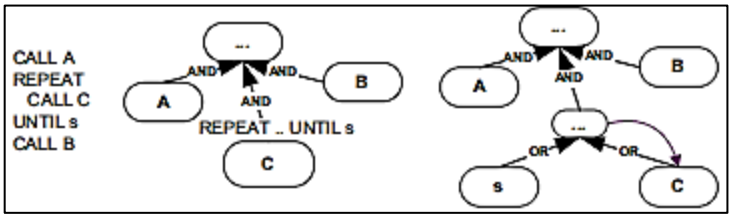


Fonte: Yu et al. (2005)

A parte da esquerda apresenta o padrão de código da estrutura de decisão para a chamada de A e B. A parte da direita apresenta o padrão gráfico correspondente à estrutura de decisão.

A Figura 8 representa o padrão de código-fonte e seu respectivo padrão gráfico para a estrutura de repetição.

Figura 8 - Padrão de estrutura de repetição



Fonte: Yu et al. (2005)

A parte da esquerda apresenta o padrão de código da estrutura de repetição em que consiste de uma sequência de chamada de A, repetição de C até que s seja verdadeiro e chamada de B. A parte da direita apresenta o padrão gráfico correspondente a essa estrutura de repetição.

3.4.3 Aplicação do Método

O método de Engenharia Reversa através de Modelo de Metas baseia-se nas atividades manuais de análise, transformação e modelagem do conteúdo do código-fonte:

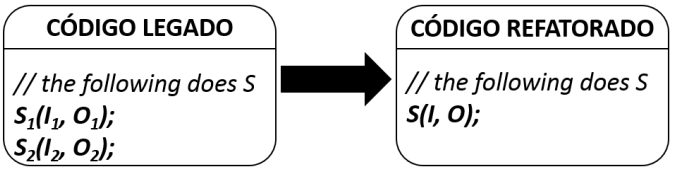
1) Análise: consiste em entender o código-fonte e aplicar uma refatoração;
2) Transformação: permite criar gráficos de estados que permitem estruturar a lógica do código refatorado;
3) Modelagem: é a extração do Modelo de Metas do código-fonte estruturado.

3.4.3.1 Refatoração do Código-Fonte

A refatoração do código-fonte é realizada para melhorar o seu entendimento, através da abstração do conteúdo das linhas de código e assim identificar a função correspondente. É comum encontrar, em códigos-fonte de software legado, a presença de diversas linhas de código que realizam vários cálculos, conversões e até mesmo acesso a dados, para somente obter um valor específico. Quando esse tipo de software foi construído, não existia a preocupação pela organização e distribuição de responsabilidades em funções e, portanto, diversas linhas de código precisavam ser analisadas para identificar a implementação das funções. Portanto, o objetivo de refatoração do código-fonte é definir funções ou métodos correspondentes a essas linhas de código, substituindo-as pela invocação às respectivas funções ou métodos.

Para realizar a refatoração, é proposta a utilização da técnica de análise de comentários para entendimento do trecho do código-fonte. A partir da informação obtida, cria-se uma função ou um método correspondente e substituem-se as linhas de código pela sua invocação. A Figura 9 exemplifica uma refatoração.

Figura 9 - Refatoração baseado na análise de comentários



Baseado em: Yu et al. (2005)

A Figura 9 representa a transformação do código legado em um código refatorado. A partir do comentário, foi observado que a função S é realizada a partir da chamada de S1 e S2. Foi então, definida a função S e as chamadas S1 e S2 foram substituídas pela chamada de S.

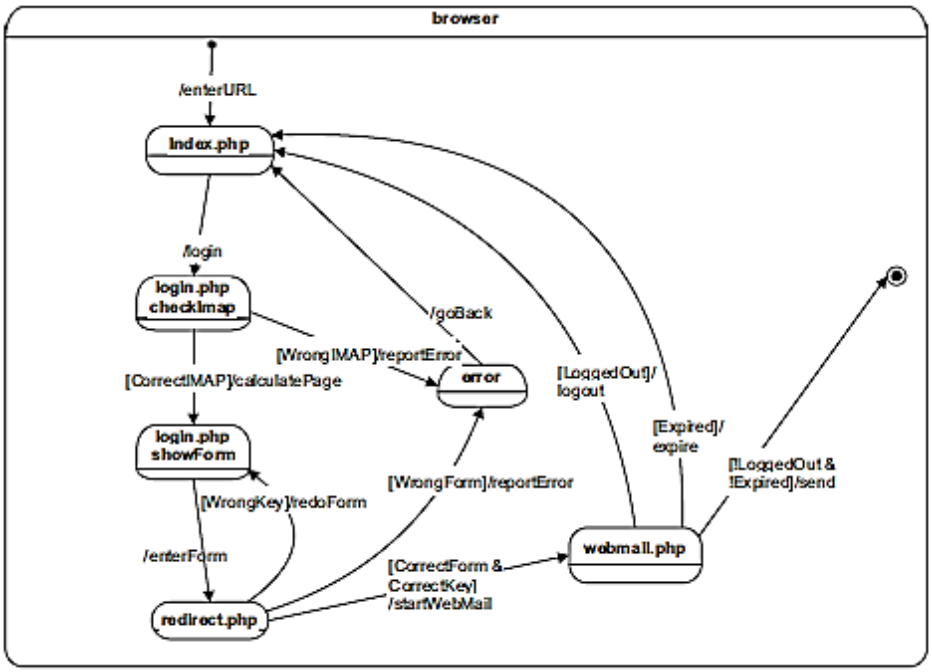
3.4.3.2 Estruturação do Código-Fonte

A estruturação do código-fonte somente é realizada quando o código não segue a programação estruturada. Para a sua realização é utilizado o gráfico de estados que representa os estados do software, o qual é construído a partir dos dados obtidos do código refatorado.

A construção do gráfico de estados é feita representando-se as funções ou os métodos do código-fonte como estados em que eles são executados, ligados pelas transições que ocorrem após a execução de cada função ou método.

Após a construção do gráfico de estados é realizada a refatoração deste gráfico para retirar os elementos que não trazem informação relevante. Por exemplo, se ocorrer apenas três transições de estado sem caminhos alternativos ou repetições, os três nós podem ser transformados em apenas um, com o nome da função correspondente à execução dos três métodos. A Figura 10 apresenta um exemplo de gráfico de estados que foi refatorado.

Figura 10 - Gráfico de Estados refatorado



Fonte: Yu et al. (2005)

Tem-se na Tabela 3 a legenda do diagrama da Figura 10.

Tabela 3 - Legenda do Gráfico de Estados

FIGURA	DESCRIÇÃO
	Estado inicial
	Estado final
	Estado
	Transição

O gráfico de estados refatorado é utilizado para criar um trecho de código-fonte, seguindo-se a sequência que pode ser levantada através desse gráfico. A Figura 11 representa o código-fonte estruturado extraído do gráfico de estados da Figura 10.

Figura 11 - Código-Fonte estruturado extraído do gráfico de estados

```
      call EnterURL
10    call Login
    if (wrongIMAP) goto 30
20    call ShowForm
      if (wrongKey) goto 20
      call EnterForm
      if (wrongForm) goto 30
    call StartWebMail
    if (loggedOut) goto 10
    if (expired) goto 10
    call Send
    Stop
30  call ReportError
      call GoBack
    goto 10
    end
```

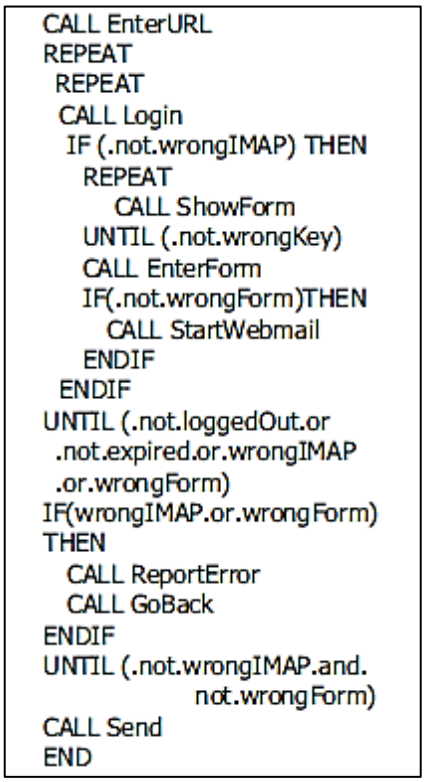
Fonte: Yu et al. (2005)

Pode-se notar que os conectores do gráfico de estados se transformaram em invocações de funções ou métodos e as condições em estruturas de decisão em desvios (goto).

Mesmo após extrair o código-fonte estruturado do gráfico de estados, pode-se notar que o código ficou com instruções de desvio (goto). Para extrair o Modelo de Metas do gráfico de estados é necessário eliminar as instruções de desvio, pois essas instruções não possuem um padrão gráfico e dificultam a extração do Modelo de Metas.

A Figura 12 representa o novo código-fonte estruturado, após a eliminação das instruções de desvios; pode-se notar que, para eliminar as instruções de desvio (goto), foi necessário implementar estruturas de repetição.

Figura 12 - Código-Fonte estruturado após eliminação de desvios



```
CALL EnterURL
REPEAT
 REPEAT
 CALL Login
  IF (.not.wrongIMAP) THEN
    REPEAT
       CALL ShowForm
    UNTIL (.not.wrongKey)
    CALL EnterForm
    IF(.not.wrongForm)THEN
      CALL StartWebmail
    ENDIF
 ENDIF
UNTIL (.not.loggedOut.or
 .not.expired.or.wrongIMAP
 .or.wrongForm)
IF(wrongIMAP.or.wrongForm)
THEN
  CALL ReportError
  CALL GoBack
ENDIF
UNTIL (.not.wrongIMAP.and.
                not.wrongForm)
CALL Send
END
```

Fonte: Yu et al. (2005)

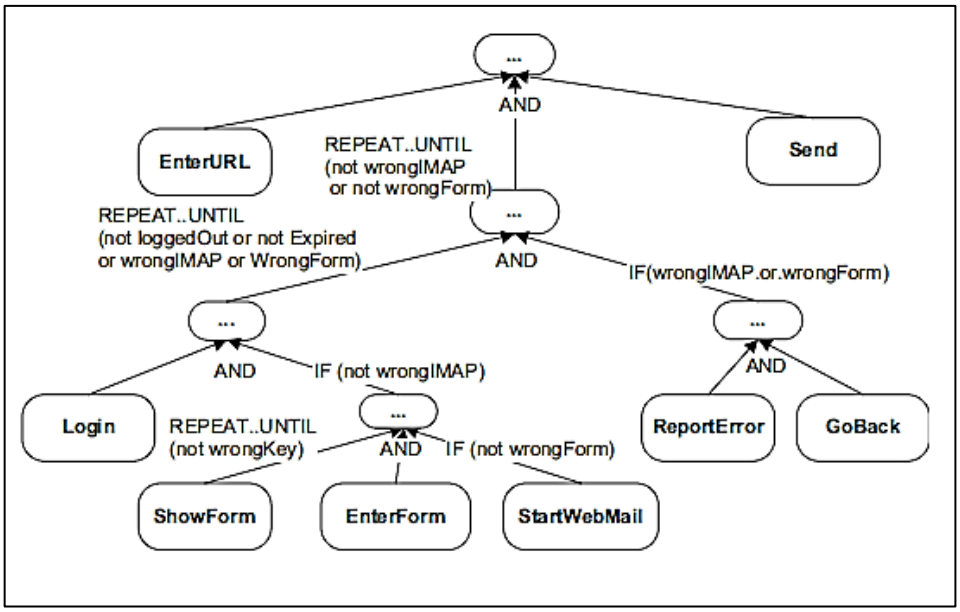
3.4.3.3 Extração do Modelo de Metas

O Modelo de Metas é constituído por um conjunto de árvores, sendo que cada uma representa uma meta do usuário. Os nós representam as funções do software, necessárias para atender uma meta do usuário.

A extração da árvore do Modelo de Metas é realizada a partir do código-fonte estruturado obtido a partir da tarefa Estruturação do Código-Fonte. Para isso, os padrões de extração descritos da seção 3.4.2 devem ser identificados no código fonte e o Modelo de Metas pode ser construído.

Com a aplicação dos padrões de extração sobre o código-fonte estruturado da Figura 12, obtém-se o Modelo de Metas da Figura 13.

Figura 13 - Modelo de Metas extraído do código-fonte estruturado



Fonte: Yu et al. (2005)

A Figura 13 representa o Modelo de Metas de um software para envio de e-mail. O nó que está na parte superior representa a meta de usuário. O restante dos nós representa, de forma estruturada, as funções necessárias para cumprir a meta de usuário. Primeiro é necessário executar a função EnterURL e, em seguida, devem-se executar as funções Login, enquanto a autenticação não for válida para acessar o

software, ShowForm para abrir o formulário do e-mail, EnterForm para preencher o formulário e StartWebMail para completar a ação de escrita do e-mail. Se não ocorrer nenhum erro na execução dessas funções, as funções ReporError e GoBack não são executadas. Após todas as funções descritas serem executadas, deve-se executar a função Send para concluir o envio do e-mail e assim atender a meta de usuário.

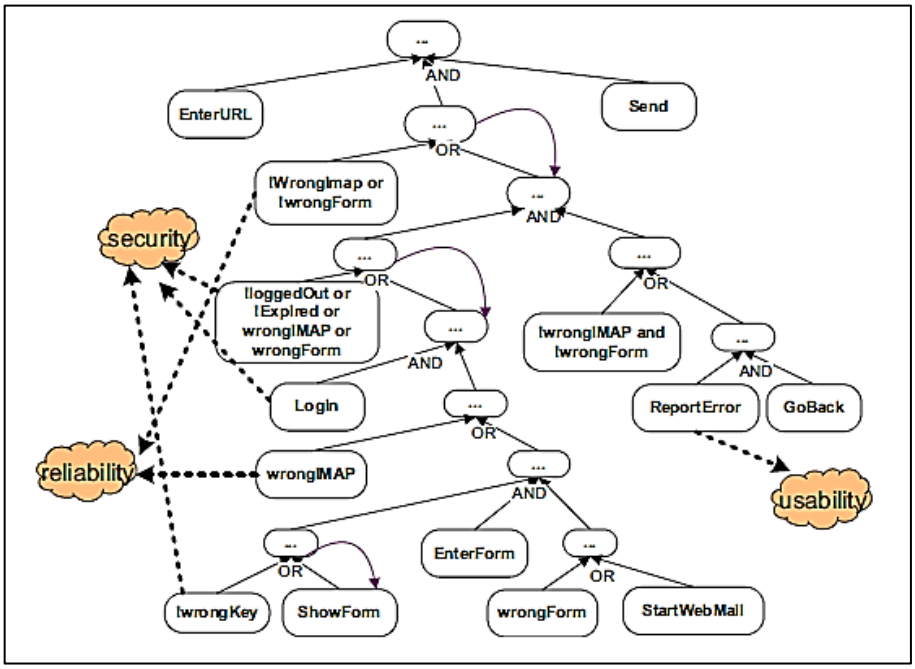
Após a construção do Modelo de Metas, obtém-se as estruturas em forma de árvore que representam os requisitos funcionais do software.

3.4.3.4 Identificação de Requisitos Não Funcionais

A identificação dos requisitos não funcionais é feita através da análise das árvores do Modelo de Metas, geradas através da tarefa Extração da Árvore do Modelo de Metas. Para isso, inicialmente devem ser definidos os requisitos não funcionais relevantes para o sistema analisado. No caso do exemplo, foram consideradas a segurança (*security*), confiabilidade (*reliability*) e usabilidade (*usability*). Seleciona-se, então um requisito não funcional e analisa-se o seu impacto sobre os nós das árvores do Modelo de Metas, associando-se o requisito ao nó que vai sofrer o impacto. Esse processo deve ser repetido para cada requisito não funcional definido para o sistema. A associação dos requisitos não funcionais à árvore da Figura 13 está apresentada na Figura 14.

Na Figura 14 estão representados os requisitos não funcionais identificados sobre o Modelo de Metas da Figura 13. Pode-se observar que eles estão atrelados às funções especificas. A segurança está associada às funções de entrada e saída do software, correspondentes às funções Login e LoggedOut e também à função wrongKey, responsável por validar a chave de acesso à função de acesso ao formulário. A confiabilidade está associada às funções de wrongIMAP e wrongForm que representam a validação do usuário. A usabilidade está associada à função ReportError, responsável por apresentação de erros ao usuário.

Figura 14 - Modelo de Metas com requisitos não funcionais



Fonte: Yu et al. (2005)

3.4.4 Vantagens e Desvantagens

O método de Engenharia Reversa através do Modelo de Metas pode ser aplicado em vários tipos de software legado, independentemente da linguagem de programação utilizada, e mesmo para programas desenvolvidos sem o uso da programação estruturada. O gráfico de estados é a ferramenta que auxilia a estruturação de códigos não estruturados.

A representação da árvore, utilizada no Modelo de Metas, ajuda a organizar o conteúdo do código através da identificação dos objetivos do usuário. Também auxilia no entendimento do código-fonte e na interdependência entre os objetivos identificados, além de apoiar a implementação dos requisitos não funcionais.

Uma desvantagem desse método é o fato de ser manual, tornando-o muito trabalhoso. A existência de ferramentas automatizadas poderia agilizar essa tarefa. Outra desvantagem é a refatoração ser realizada com base nos comentários existentes no código. Se não existirem comentários ou se os comentários estiverem desatualizados o esforço seria maior, pois seria necessário entender e interpretar

totalmente o código, além de eventualmente, ser necessário um especialista na linguagem em que o software está desenvolvido.

Apesar das desvantagens, esse método apresenta vários pontos positivos. Somente é necessário estimar com cuidado a complexidade para a aplicação do método.

3.5 COMPARAÇÃO DOS MÉTODOS

Cada método descrito neste capítulo apresenta um foco diferente para a realização da engenharia reversa; porém existem pontos em comum na aplicação de cada método. Objetivando analisar as semelhanças e diferenças entre os métodos, foram elencadas questões e aplicadas em cada método conforme a Tabela 4.

Tabela 4 - Comparativo entre Métodos de Engenharia Reversa

Questão	Métodos		
	Modelo de Meta	**Redocumentação**	**Interface**
Análise baseada em?	Código Fonte	Código Fonte	Telas
Modifica Software Legado?	Sim	Não	Não
Elabora documento?	Não	Sim	Sim
Elabora Modelo?	Sim	Não	Sim
Identifica Requisitos Funcionais?	Sim	Sim	Sim
Identifica Requisitos Não Funcionais?	Sim	Não	Não
Utiliza Ferramenta para Automatização?	Não	Sim	Sim
Quantidade de tarefas do método?	4	3	2
Atividade Principal	Refatoração do Código-Fonte	Extração dos Fatos	Mapeamento de Interface
Documento Final	Modelo de Metas	Modelos da UML	Gráfico de Interface
Foco da Análise	Sequência de invocações de Métodos / Agrupamento de Funções	Constantes / Domínios de Valor / Modelagem de Dados / Regras de Negócio	Interação entre Usuário e Software

Ao analisar a tabela comparativa, pode-se concluir que existem atividades semelhantes e diferentes nos métodos. Como exemplo, o método de Engenharia Reserva através de Modelos de Meta e o método de Redocumentação utilizam como base a análise do código-fonte, ao passo que o método de Engenharia Reversa

através de Interfaces de Sistemas Legados utiliza como base a análise da interface de usuário.

Após a comparação dos métodos, concluiu-se que seria possível unificá-los em um processo, de forma a utilizar as atividades semelhantes e explorar o potencial das diferenças, evidenciando o foco que cada método possui. O resultado dessa unificação é apresentado no próximo capitulo.

3.6 CONSIDERAÇÕES DO CAPÍTULO

Nesse capítulo foram descritos três métodos para a Engenharia Reversa em Software Legado, selecionados a partir de um conjunto de trabalhos analisados. Esses métodos descrevem focos diferentes para realizar a elicitação dos requisitos e, além disso, foram definidos para contextos diferentes.

O método de Engenharia Reversa através de Interface de Sistemas Legados é eficiente para a elicitação de requisitos visíveis através da interface de usuário. Devido a esse enfoque ele é mais apropriado em casos de migração da parte de interface de usuário, pois ignora os detalhes técnicos do software e pode não identificar os detalhes importantes que estão implícitos no software.

O método de Redocumentação de Sistema Legado possui foco em entender os detalhes técnicos através do código-fonte e, posteriormente transcrever os requisitos elicitados em diagramas da UML, complementando o método que analisa a interface do usuário. Os documentos gerados reconstituem os requisitos do software legado e, assim, permite realizar posteriormente sua Reengenharia.

O objetivo do método de Engenharia Reversa através de Modelo de Metas é semelhante ao método de Redocumentação de Sistema Legado, porém pelas suas considerações permitem obter maior detalhamento dos requisitos elicitados, pois é uma representação de forma estruturada das funções ou métodos que o software deve realizar. Seu produto final é o Modelo de Metas, que descreve a estrutura das funções do software para atender os objetivos do usuário. A identificação dos objetivos do usuário fornece uma visão mais estruturada de todo o sistema de software.

Os métodos analisados apresentam maior eficiência em pontos diferentes, dependendo do contexto e do objetivo final da aplicação da engenharia reversa.

Considerando como objetivo final a migração do software legado, observa-se que o método de Engenharia Reversa através de Modelo de Metas pode apresentar maior cobertura na elicitação de requisitos, pois fornece maior detalhamento das informações coletadas.

Com base nos estudos de cada método e com foco na migração do software legado e seus processos, conclui-se que os três métodos podem oferecer melhores resultados em conjunto, constituindo um processo de engenharia reversa que atenda o cenário de migração.

4 PROCESSO DE ENGENHARIA REVERSA PARA SOFTWARE LEGADO (ERSL)

Este capítulo tem como objetivo descrever o processo ERSL, definido a partir do estudo de três métodos de engenharia reversa, selecionados na literatura: Engenharia Reversa através de Interfaces de Sistemas Legados (STROULIA et al., 1999), Redocumentação de Sistemas Legados (GEET; EBRAERT; DEMEYER, 2010) e Engenharia Reversa através de Modelos de Meta (YU et al., 2005). Esses métodos foram selecionados porque cada um apresenta a aplicação da engenharia reversa em sistemas legados a partir de focos diferentes e, em conjunto, contribuiu para a definição do processo ERSL.

Para isso, é apresentada, inicialmente, a visão geral da migração de sistema legado, para mostrar o contexto em que o processo ERSL pode ser aplicado. A seguir, são apresentadas a visão geral do processo ERSL e a descrição das suas fases, através de suas atividades, seus participantes e os principais dados de entrada e de saída de cada atividade. Para essa descrição, foram utilizados os diagramas BPMN pois apresentam uma notação de fácil compreensão, apropriados para as áreas de tecnologia e áreas de negócio (CHINOSI; TROMBETTA, 2012).

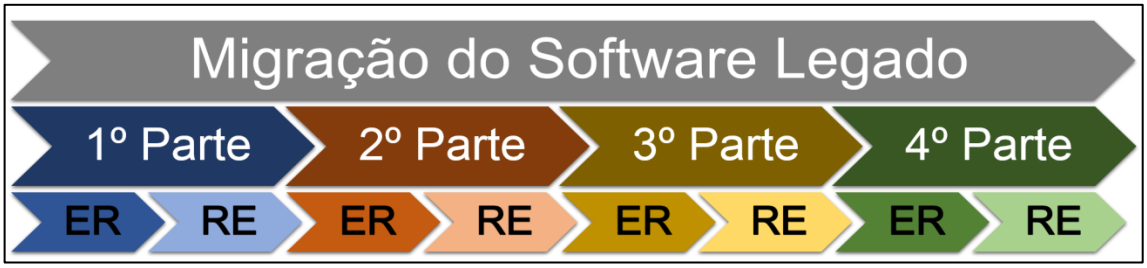
4.1 CONTEXTO DE APLICAÇÃO DO PROCESSO

O processo ERSL foi elaborado para ser utilizado na migração gradativa de sistema legado e, para isso, é necessário que se deixe claro o entendimento desse contexto, para que os resultados atinjam a meta do projeto.

Para que a migração seja gradativa, ou seja, por partes, deve-se inicialmente particionar o sistema a ser migrado. A engenharia reversa e a reengenharia são realizadas sobre cada parte, fazendo com que o processo de migração seja iterativo e incremental.

A Figura 15 apresenta um exemplo em que a migração de um software legado é feita através de quatro partes. Pode-se observar que a migração completa do sistema é finalizada quando todas as partes forem migradas, sendo que, para cada parte, é necessário realizar a engenharia reversa (ER) e a reengenharia (RE).

Figura 15 - Divisão de Partes para Migração de Software Legado

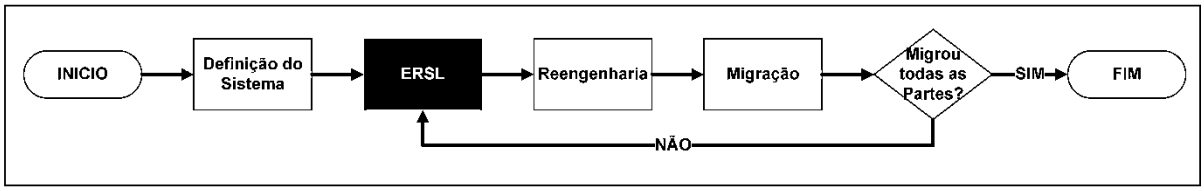


Fonte: do Autor

Nesse contexto, o processo ERSL pode ser aplicado na fase de engenharia reversa, cujo objetivo é realizar a elicitação dos requisitos das partes do software legado, para permitir posteriormente a realização da reengenharia.

A Figura 16 apresenta uma visão em alto nível de um processo de migração gradativa.

Figura 16 - Processo de Migração Gradativa



Fonte: do Autor

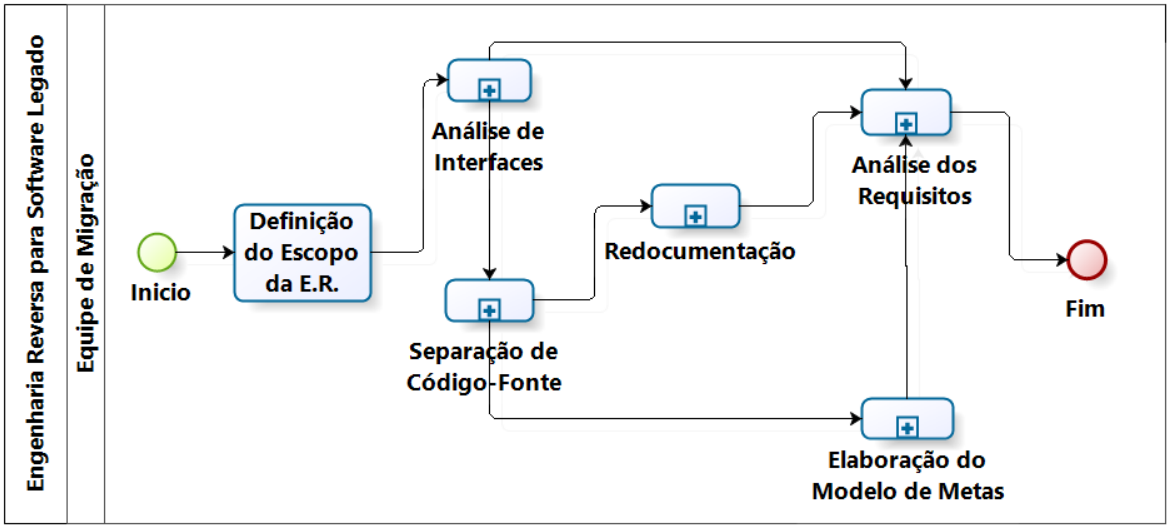
O processo de Definição do Sistema é responsável pela descrição do sistema a ser migrado, baseada no entendimento coletado através de usuários, responsáveis pelo sistema legado e eventuais documentos existentes. Com base nesses dados, o processo ERSL seleciona a parte a ser migrada, gera os seus requisitos e, a partir desses requisitos, o processo de Reengenharia reconstrói a parte selecionada do sistema. Ao concluir a reconstrução da parte selecionada, é executado o processo de Migração, que consiste em integrar a parte selecionada em outro software e/ou outra plataforma tecnológica. Em seguida, se ainda existirem partes a serem migradas,

volta-se para o processo ERSL para selecionar a próxima parte a ser migrada e assim por diante.

4.2 VISÃO GERAL DO PROCESSO ERSL

A visão geral do processo ERSL está apresentada na Figura 17.

Figura 17 - Visão Geral do Processo ERSL



O processo ERSL tem, como entrada, o Documento de Definição do Sistema, com as informações do sistema legado a ser migrado. Esse documento é gerado no processo de migração chamado de Definição do Sistema, que é predecessor do processo ERSL (Figura 16).

Inicialmente é executada a fase de Definição do Escopo da Engenharia Reversa, em que é definida a parte do sistema legado que vai ser o foco da aplicação do ERSL, considerando o planejamento da migração do sistema. Para isso, são considerados o conhecimento sobre o sistema legado, a prioridade da migração e os dados esperados pelo processo de reengenharia. Nesta fase é gerado o Documento de Escopo da Engenharia Reversa, que contém a lista de software ou módulos de software, que fazem parte do escopo em que o ERSL será aplicado para a elicitação de seus requisitos.

A fase seguinte é a de Análise de Interfaces do software legado, para a identificação de requisitos através da observação de cada interface de usuário, relativo ao escopo definido anteriormente. Como artefatos, são gerados o Documento de Características de Software e o Gráfico de Interfaces Detalhado. O Documento de Características de Software contém a descrição em alto nível dos requisitos, identificados através dessa fase. O Gráfico de Interfaces Detalhado contém o diagrama com o fluxo de navegação de telas, eventos que estimulam as transições entre cada interface de usuário e as informações de entrada e saída para cada interface. Esses artefatos fornecem o relacionamento entre as funcionalidades e as informações levantadas através da análise das interfaces do software. Esses artefatos são utilizados nas fases de Separação de Código-Fonte e de Análise dos Requisitos.

A fase de Separação de Código-Fonte é responsável por identificar o código-fonte que realiza as funções observadas através das interfaces analisadas e cada arquivo de código-fonte que não possuem uma relação direta com as interfaces, mas são necessários para que essa interface possa ser utilizada. Os artefatos resultantes são o Documento com Informações de Comentários Relevantes e os Arquivos de Código-Fonte que alimentam as fases de Redocumentação e de Elaboração do Modelo de Metas.

A fase de Redocumentação é responsável por coletar os requisitos contidos no código-fonte, gerando o artefato Documento de Características de Software Complementado, que complementa o documento correspondente gerado na fase de Análise de Interfaces, com as informações obtidas através dos arquivos de código-fonte.

A fase de Elaboração do Modelo de Metas é responsável pela construção do Modelo de Metas que representa os requisitos encontrados no código-fonte em forma de metas de usuário, conforme apresentado no método Engenharia Reversa através de Modelos de Meta (YU et al., 2005) descrito na seção 3.4 deste trabalho.

A fase de Análise dos Requisitos utiliza os artefatos gerados nas fases anteriores para analisar e representar os requisitos obtidos, gerando os Modelos UML, que serão utilizados na reengenharia da parte do sistema legado considerado.

4.3 PARTICIPANTES DO PROCESSO ERSL

Para a execução do processo ERSL são necessários os seguintes tipos de participantes: analista de migração, analista de sistemas, analista de requisitos e usuário do sistema.

4.3.1 Analista de Migração

O papel de Analista de Migração é exercido por um profissional que tenha vasta experiência em migração de sistema. Esse participante tem a responsabilidade de garantir que o processo ERSL seja aderente ao plano de migração do software legado. É o Analista de Migração que avalia a importância e estratégia da divisão do software legado em partes e a priorização dessas partes e/ou dos módulos que devem ser considerados no do processo ERSL. Desta forma, possui um papel indispensável na fase de Definição do Escopo da Engenharia Reversa e, além disso, é o responsável pela validação final do software migrado.

4.3.2 Analista de Sistemas

O papel de Analista de Sistemas é exercido por um profissional que tenha conhecimento dos processos e dos sistemas envolvidos na migração. Ele é responsável por liderar a execução das fases de Análise de Interfaces, Separação do Código-Fonte, Redocumentação e Elaboração do Modelo de Metas, além de participar na fase de Definição do Escopo da Engenharia Reversa e de Análise dos Requisitos.

4.3.3 Analista de Requisitos

O papel de Analista de Requisitos é exercido por um profissional com conhecimento e experiência nos processos de elicitação de requisitos. Ele é responsável por analisar os requisitos elicitados no processo ERSL, para a construção dos modelos UML, que são gerados para o processo de Reengenharia do Software Legado.

4.3.4 Usuário do Sistema

O papel de Usuário do Sistema é exercido por um profissional que tenha um grande conhecimento na utilização do software legado sobre o qual é aplicado o ERSL. Esse participante possui grande importância na estratégia que o Analista de Migração elabora, contribuindo com informações importantes que existem no processo de negócio em que o software legado está inserido.

4.4 DESCRIÇÃO DAS FASES DE ERSL

Tem-se, a seguir, a descrição das fases do processo ERSL, apresentadas na Figura 17. As tabelas contidas nesta seção, são exemplos de artefatos a serem gerados durante a execução das fases e atividades do processo ERSL.

4.4.1 Fase de Definição do Escopo da Engenharia Reversa

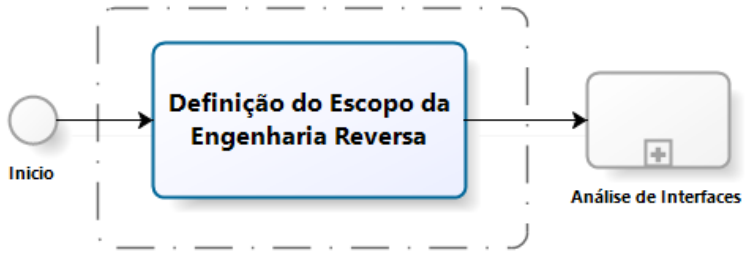
O objetivo principal dessa fase é definir o foco da aplicação do processo ERSL. Para isso, é necessário definir de forma clara as partes relevantes do software legado a serem consideradas na iteração da engenharia reversa. Com isso, evita-se o retrabalho de elicitar, analisar e documentar partes fora do contexto do trabalho e ter uma visão das expectativas dos resultados alcançados.

Para a execução desta fase é necessário o envolvimento dos seguintes participantes: Analista de Migração; Analista de Sistemas e Usuário do Sistema.

A fase de Definição do Escopo da Engenharia Reversa é liderada pelo Analista de Migração que, em conjunto com o Analista de Sistemas e o Usuário do Sistema, define a estratégia de migração do software legado considerado.

Conforme representado na Figura 18, esta fase possui somente uma atividade, sendo a primeira do ERSL, e é predecessora da fase de Análise de Interfaces. Como resultado, é produzido o artefato chamado de Documento de Escopo da Engenharia Reversa que contém a lista de nomes de software e/ou módulos de software do sistema legado que fazem parte do escopo da aplicação do ERSL.

Figura 18 - Atividades da fase de Definição do Escopo da Engenharia Reversa



O documento de Definição do Sistema, usado como entrada dessa fase, foi elaborado no processo de Definição do Sistema, anteriormente ao processo ERSL (Figura 16), e contém o escopo da migração, fornecendo uma visão geral do software legado. Esse documento é utilizado para selecionar as partes relevantes para a aplicação do processo ERSL.

O Usuário do Sistema fornece as informações sobre o processo em que o software legado está inserido e também sobre as suas funcionalidades. Com base nessas informações, o Analista de Sistemas define as partes do software legado que estão relacionadas a essas informações e, em conjunto com o Analista de Migração, elabora o Documento de Escopo da Parte Selecionada, que descreve as partes (software e/ou módulos de software) do software legado a serem consideradas para a aplicação do ERSL.

Os artefatos relacionados com essa fase são:

- ENTRADA:
  - Documento de Definição do Sistema
- SAÍDA:
  - Documento de Escopo da Engenharia Reversa

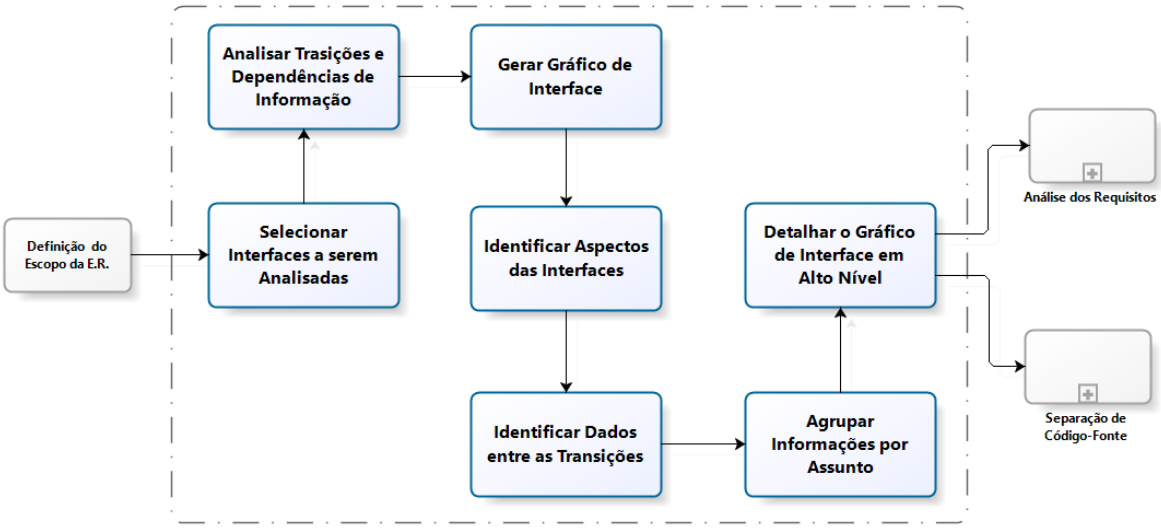
4.4.2 Fase de Análise de Interfaces

O objetivo principal da fase de Análise de Interface é ter o primeiro contato com o software legado, coletando dados úteis para o entendimento e a delimitação do contexto da análise. Foi baseada no processo de Engenharia Reversa através de

Interfaces de Sistemas Legados (STROULIA et al., 1999), descrito no item 3.2 do capítulo 3.

Essa fase está dividia em sete atividades (Figura 19) e tem início após a fase de Definição de Escopo da Engenharia Reversa, sendo, portanto, o ponto inicial da engenharia reversa propriamente dita. Para a sua execução é necessária a participação do Analista de Sistemas e do Usuário do Sistema.

Figura 19 - Atividades da fase de Análise de Interfaces



Os artefatos relacionados com essa fase são:

- ENTRADA:
  - Documento de Escopo da Engenharia Reversa: contém a lista de nomes de software e/ou módulos do sistema legado que fazem parte do escopo da aplicação do ERSL.
- SAÍDA:
  - Documento de Características de Software: contém a descrição em alto nível dos requisitos identificados com a análise da interface de usuário do software legado.
  - Gráfico de Interfaces Detalhado: contém o diagrama com a navegação entre interfaces, eventos que estimulam as transições e as informações de entrada e saída para cada interface de usuário.

Tem-se, a seguir, o detalhamento de cada atividade pertencente à fase Análise de Interfaces.

4.4.2.1 Selecionar Interfaces a Serem Analisadas

Esta atividade utiliza, como artefato de entrada, o Documento de Escopo da Engenharia Reversa para identificar as interfaces de usuário envolvidas no contexto a ser analisado. Para a sua execução, o Analista de Sistemas, em conjunto com o Usuário do Sistema, identifica as interfaces relacionadas com o software e/ou módulos selecionados para a aplicação do processo ERSL. Como artefato, é gerada a Lista de Interfaces para Análise, que contém o nome das interfaces identificadas.

Os artefatos relacionados com esta atividade são:

- ENTRADA:
  - Documento de Escopo da Engenharia Reversa
- SAÍDA:
  - Lista de Interfaces para Análise

4.4.2.2 Analisar Transições e Dependências de Informação

Esta atividade consiste na identificação do relacionamento entre as interfaces identificadas, ou seja, o Analista de Sistemas identifica a navegação entre as interfaces de usuário e as dependências de informação entre elas, tais como informações de entrada necessárias para uma interface ser apresentada. Com essa análise, pode-se construir um mapa com os relacionamentos entre as interfaces, gerando o artefato chamado Interfaces com Relacionamentos, que contém a descrição textual das interfaces identificadas, sua navegação e dependências de informações. A Tabela 5 apresenta um exemplo do artefato Interfaces com Relacionamentos.

Tabela 5 - Exemplo do Artefato Interfaces com Relacionamentos

ID	Nome da Interface	ID da Predecessora	Informação de Entrada
1	Menu	-	Login
2	Pedido	1	-
3	Escolha de Produtos	2	Pedido
4	Confirmação do Pedido	2	Pedido
5	Dados de Entrega	4	Pedido
6	Dados de Pagamento	5	Pedido; Entrega
7	Confirmação de Compra	6	Pedido; Entrega; Pagamento

As colunas ID e Nome da Interface identificam as interfaces mapeadas; a coluna ID da Predecessora contém o ID da interface que aciona a transição para interface identificada; a coluna Informação de Entrada é composta por informações necessárias para que a interface identificada seja acionada.

Os artefatos relacionados com esta atividade são:

- ENTRADA:
  - Lista de Interfaces para Análise
- SAÍDA:
  - Interfaces com Relacionamentos

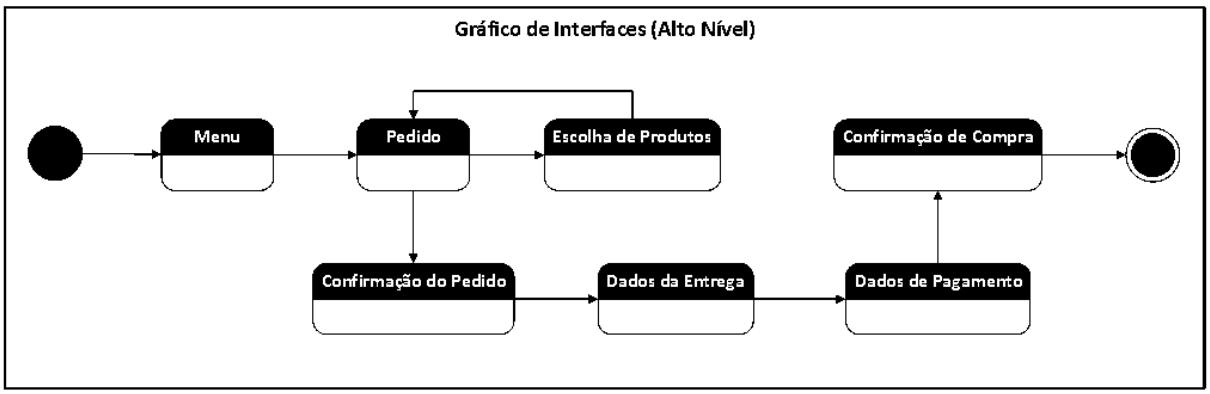
4.4.2.3 Gerar Gráfico de Interface

Esta atividade consiste em representar as interfaces selecionadas e os seus relacionamentos, através do modelo denominado de Gráfico de Interfaces. Esse modelo é baseado no diagrama de máquina de estados e o seu principal objetivo é representar, de forma gráfica, as interfaces selecionadas e os relacionamentos entre elas.

O Analista de Sistemas realiza a transformação do artefato Interfaces com Relacionamentos (representado na Tabela 5) no Gráfico de Interfaces em Alto Nível, que contém o diagrama com a navegação entre as interfaces selecionadas do software legado, objetivando facilitar o entendimento do relacionamento entre as

interfaces de usuário. A Figura 20 representa um exemplo de Gráfico de Interfaces em Alto Nível.

Figura 20 - Exemplo de Gráfico de Interfaces em Alto Nível



O Gráfico de Interfaces é de Alto Nível, pois representa uma visão abstrata do software legado, mostrando a lógica de operação, através do relacionamento entre as interfaces. Esse gráfico é detalhado posteriormente em outra atividade desta fase.

Na Figura 20, cada estado representa uma interface, com seu respectivo nome, e as setas representam a direção da navegação e do fluxo de informações de uma interface para outra.

Os artefatos relacionados com esta atividade são:

- ENTRADA:
  - Interfaces com Relacionamentos
- SAÍDA:
  - Gráfico de Interfaces em Alto Nível

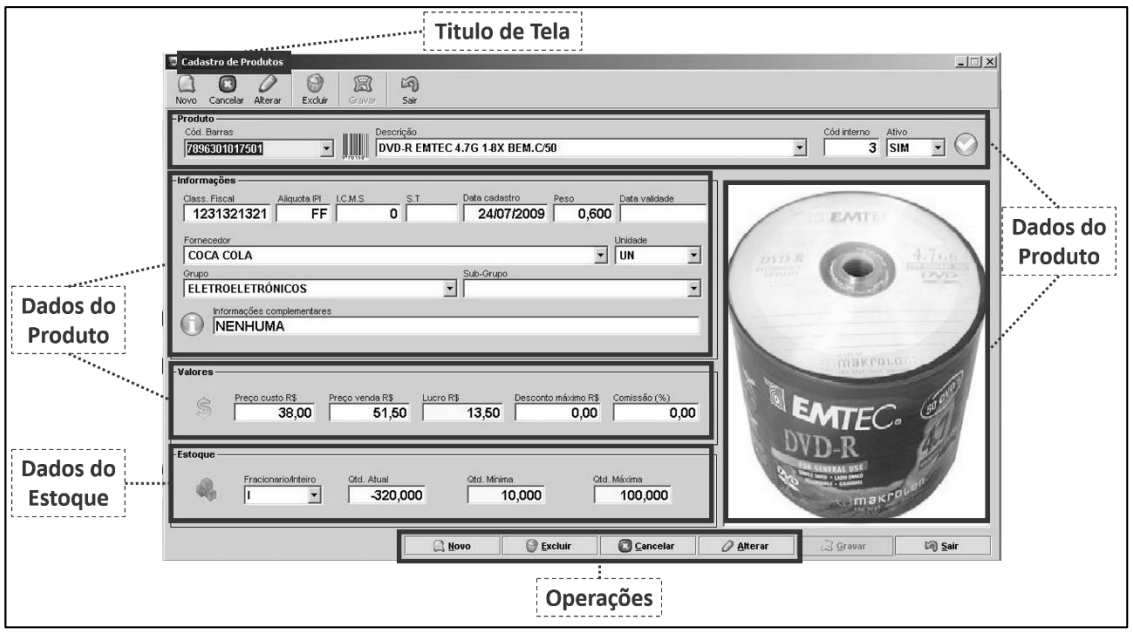
4.4.2.4 Identificar Aspectos das Interfaces

Esta atividade tem o objetivo de analisar os aspectos que cada interface possui, tais como: existência da identificação de tela; existência de títulos de tela; existência de data/hora e localização; caracteres especiais e localização dos campos.

Essa análise é realizada pelo Analista de Sistemas, pois existem dados técnicos que devem ser considerados para o entendimento do sistema.

A Figura 21 exemplifica uma interface e seus tipos de informação.

Figura 21 - Exemplo de interface e Classificação de Aspectos



O título da interface evidencia que ela é para uma operação do tipo CRUD (Incluir, Consultar, Atualizar e Excluir) de um produto. Essa informação é confirmada quando se observa o grupo de operações. Também é possível identificar as informações importantes para esta interface; no exemplo, observa-se que o aspecto classificado como Dados do Produto é extremamente importante para a interface exemplificada.

Após identificar as informações de cada interface, é gerado o artefato chamado de Documento de Características de Interface que registra os aspectos identificados em cada interface do Gráfico de Interfaces em Alto Nível.

A Tabela 6 apresenta uma parte de um Documento de Características de Interface, relativa à interface da Figura 21.

Tabela 6 - Exemplo de Documento de Características de Interface

A	B
ID	1
Nome	Manter Produtos
Título	Cadastro de Produtos
Cód. Identificação	-
Tipo	CRUD
Data / Hora / Localização	-
Grupo de Dados	PRODUTO / INFORMAÇÕES / VALORES / ESTOQUE
Qtd. Campos de Entrada	22
Campos de Entrada	Cód. Barras / Descrição / Cód. Interno / Ativo / Class. Fiscal / Alíquota IPI / ICMS / ST/ Data Cadastro / Peso / Data Validade / Fornecedor / Unidade / Grupo / Subgrupo / Inf. Complementares / Preço Custo / Preço Venda / Desconto Max. / Comissão / Fracionário ou Inteiro / Qtd. Atual
Qtd. Campos Saída	5
Campos Saída	Cód. Barras / Descrição / Lucro / Qtd. Mínima / Qtd. Máxima
Qtd. Botões	6
Botões	Novo / Cancelar / Alterar / Excluir / Gravar / Sair

Na coluna A estão listados os tipos de dados da informação e, na coluna B, informações observadas na interface.

Os artefatos envolvidos nesta atividade são:

- ENTRADA:
  - Gráfico de Interfaces em Alto Nível
- SAÍDA:
  - Documento de Características de Interface

4.4.2.5 Identificar Dados entre as Transições

Esta atividade consiste na identificação dos dados que trafegam quando ocorrem as transições das interfaces. Com base nos artefatos de Gráfico de Interface em Alto Nível e Interfaces com Relacionamentos, o Analista de Sistemas detalha as informações de entrada, geradas na atividade Analisar Transições e Dependências de Informação até o nível de dados. Desta forma, obtém o mapeamento dos dados de saída de uma interface A, que se tornam dados de entrada da interface B.

Após a identificação desses dados, o Documento de Características de Interface é complementado com os detalhes dos dados de entrada e saída para cada transição entre as interfaces, gerando um novo artefato chamado Documento Detalhado de Características de Interface.

A Tabela 7 representa as informações que devem ser acrescentadas no Documento de Características de Interface.

Tabela 7 - Exemplo de informações adicionados no Documento de Características de Interface

A	B
Qtd. Dados de Entrada	0
Dados de Entrada	-
Qtd. Dados de Saída	2
Dados de Saída	Cód. Barras / Descrição

Na coluna A estão listados os tipos de dados que classificam os dados da coluna B. Essas informações representam o resultado da análise de transição entre duas interfaces.

Os artefatos relacionados com esta atividade são:

- ENTRADA:
  - Interfaces com Relacionamentos
  - Gráfico de Interfaces em Alto Nível
  - Documento de Características de Interface

- SAÍDA:
    - Documento Detalhado de Características de Interface

4.4.2.6 Agrupar Informações por Assunto

Esta atividade objetiva a identificação dos relacionamentos entre as informações, com base na análise do Documento Detalhado de Características de Interface, separando e agrupando as informações coletas por assuntos relevantes, resultando no Documento de Características de Software. A Tabela 8 apresenta um exemplo do artefato Documento de Características de Software.

Tabela 8 - Exemplo de Documento de Características de Software

A	B
ID	1
Módulo	Produtos
Requisito	Gestão de Produtos
Funcionalidade	Cadastrar Produtos
Funções / Métodos	• Verificar Existência de Cadastro • Gerar Código de Identificação • Listar Fornecedores • Cadastrar Código de Barras • Cadastrar Estoque • Cadastrar Foto
Dependências	• Cadastro de Fornecedor
Regras de Negócio	
1	Necessário atrelar o produto a um fornecedor cadastrado
2	O produto deve ter data de início e termino de vigência

É comum que diversas informações do Documento Detalhado de Características de Interface possuam aspectos em comum, para uniformizar e

consolidar essas informações, o Analista de Sistemas organiza as informações em um novo artefato chamado Documento de Características de Software, que contém a descrição em alto nível dos requisitos identificados, através da fase de Análise de Interfaces. Esse documento posteriormente será incrementado em outras fases do ERSL.

Os artefatos envolvidos nesta atividade são:

- ENTRADA:
  - Documento Detalhado de Características de Interface
- SAÍDA:
  - Documento de Características de Software

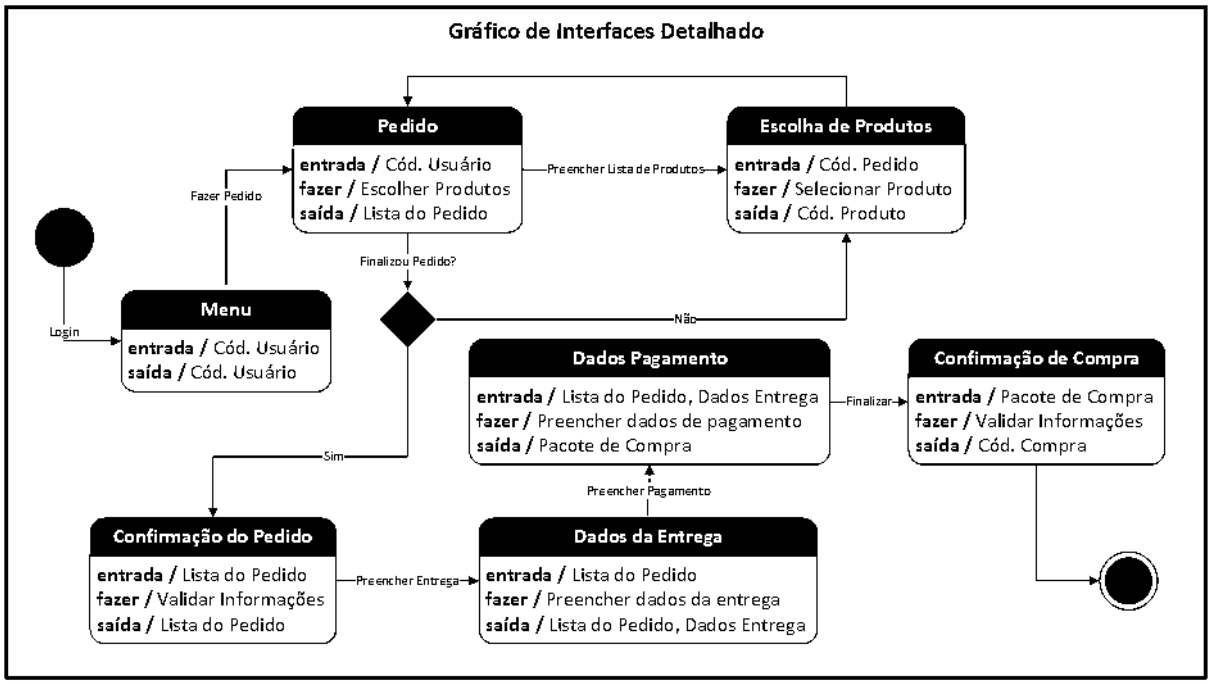
4.4.2.7 Detalhar o Gráfico de Interfaces em Alto Nível

Esta é a última atividade da fase de Análise de Interfaces e seu objetivo é incrementar e detalhar o Gráfico de Interfaces em Alto Nível. Nesta atividade são definidas as pré-condições e as pós-condições de cada interface.

Com base no artefato Documento Detalhado de Características de Interface, o Analista de Sistema realiza o detalhamento do Gráfico de Interfaces em Alto Nível. A sua finalidade é obter um documento com menor nível de abstração das informações sobre as interfaces do software legado, gerando o artefato chamado Gráfico de Interfaces Detalhado que contém o diagrama com a navegação entre interfaces, eventos que estimulam as transições e as informações de entrada e saída para cada interface de usuário.

Na Figura 22 tem-se um exemplo do Gráfico de Interfaces Detalhado, gerado a partir do Gráfico de Interfaces em Alto Nível. Nota-se que, nessa figura, os nós representam as interfaces mapeadas, contendo o detalhamento do objetivo e dos dados de entrada e saída, ligadas através dos conectores identificados através de eventos, que são estímulos para a transição entre interfaces. Também é possível utilizar uma estrutura de decisão para representar possíveis regras de negócio identificadas, ao analisar as transições entre interfaces.

Figura 22 - Exemplo do Gráfico de Interfaces Detalhado



O Gráfico de Interfaces Detalhado é um artefato de entrada para a fase de Análise dos Requisitos, pois possui informações relevantes sobre o funcionamento do software legado, tais como as dependências entre as interfaces de usuário, os fluxos de informações e também as iterações entre usuário e software.

Os artefatos envolvidos nesta atividade são:

- ENTRADA:
  - Documento Detalhado de Características de Interface
  - Gráfico de Interfaces em Alto Nível
- SAÍDA:
  - Gráfico de Interfaces Detalhado

4.4.3 Fase de Separação de Código-Fonte

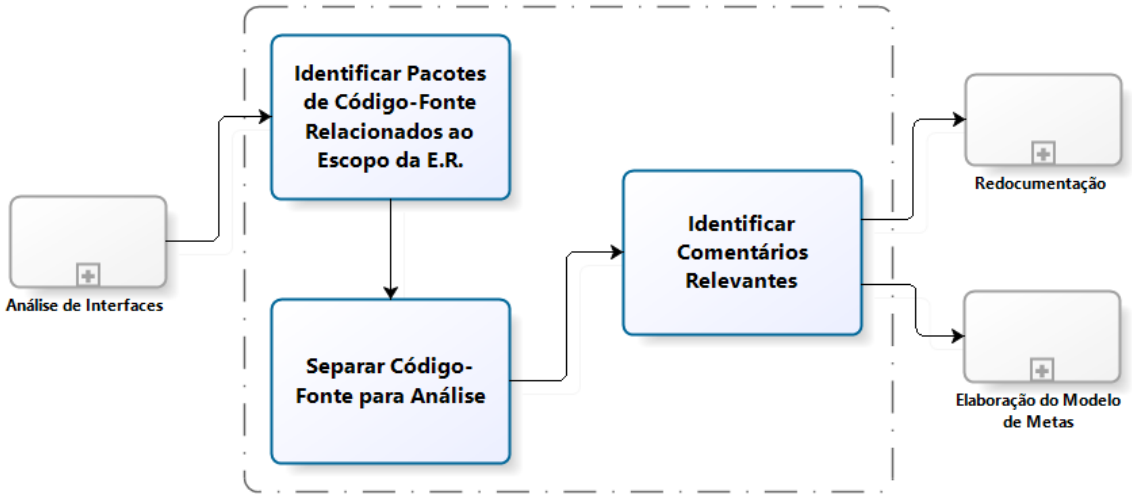
O objetivo principal da fase Separação de Código-Fonte é realizar a correta seleção do código-fonte para evitar análise de informações não relevantes ao estudo. Esta é uma fase intermediária que utiliza as informações coletadas na fase de Análise de Interfaces, para selecionar os arquivos de código-fonte que estão relacionados com

o escopo da engenharia reversa, gerando informações para as fases de Redocumentação e Elaboração do Modelo de Metas.

Todas as atividades desta fase foram elaboradas com base nos métodos de Redocumentação de Sistemas Legados (GEET; EBRAERT; DEMEYER, 2010) e de Engenharia Reversa através de Modelos de Meta (YU et al., 2005), pois ambos os métodos possuem como pré-requisito a seleção correta do código-fonte a ser analisado.

Esta fase tem início após a realização da fase de Análise de Interfaces e tem a participação do Analista de Sistemas, que exerce um papel técnico de análise. A Figura 23 representa o relacionamento das atividades pertencentes à fase de Separação de Código-Fonte.

Figura 23 - Atividades da fase de Separação do Código-Fonte e seus artefatos



Os artefatos relacionados com essa fase são:

- ENTRADA:
  - Documento de Características de Software: contém a descrição em alto nível dos requisitos identificados com a análise da interface de usuário do software legado.

- SAÍDA:
    - Arquivos de Código-Fonte: são arquivos contendo o código-fonte legado que estão relacionados com o escopo do ERSL.
    - Documento com Informações de Comentários Relevantes: contém a relação dos comentários relevantes, presentes no código-fonte do software legado.

Tem-se, a seguir, o detalhamento das atividades pertencentes à fase de Separação de Código-Fonte:

4.4.3.1 Identificar Pacotes de Código-Fonte Relacionados ao Escopo da Engenharia Reversa

Esta é a primeira atividade da fase de Separação de Código-Fonte, sendo executada logo após a fase de Análise de Interfaces. Seu objetivo é a identificação e a separação dos pacotes de código-fonte relacionados com o escopo da engenharia reversa, através da análise do Documento de Características de Software.

A identificação dos pacotes de código-fonte é realizada com base em cada interface de usuário analisada. Toda interface possui um pacote de código-fonte, necessário para seu funcionamento, o qual é considerado que tem relacionamento direto com a interface considerada. Porém, podem existir arquivos de código-fonte, que não estão nestes pacotes, e que possuem dependências com a interface considerada. Para uma total identificação, é necessário pesquisar as dependências indiretas através dos pacotes de relacionamento direto com a interface de usuário.

O Analista de Sistemas pesquisa nos repositórios e identifica os pacotes de código-fonte que estão relacionados diretamente com as interfaces descritas na fase Análise de Interfaces. Além disso, são também identificados os pacotes de código-fonte que possuem uma relação indireta com as interfaces identificadas. Como exemplo, podem-se citar os programas do tipo *batch* que realizam funcionalidades para que as interfaces tenham informações para utilização.

Os artefatos envolvidos nesta atividade são:

- ENTRADA:
    - Documento de Características de Software

- SAÍDA:
    - Pacotes de Código-Fonte

4.4.3.2 Separar Código-Fonte para Análise

Essa atividade objetiva a identificação dos arquivos de código-fonte presentes nos pacotes de código-fonte obtidos na atividade anterior. Seu foco é identificar os arquivos que tem relação com o escopo que foi definido para a engenharia reversa. Para isso, é necessário analisar os pacotes de código-fonte utilizando uma visão semântica.

A análise semântica deve ser realizada com base na utilização de cada interface analisada na fase de Análise de Interfaces; desta forma cada arquivo de código-fonte deve ser analisado considerando as invocações de funções.

Como exemplo, considera-se um arquivo A que invoca uma função x; porém as instruções contidas na função x encontram-se em um arquivo B. Desta forma os arquivos A e B devem ser separados.

Também deve-se considerar as estruturas de tabelas e os campos do banco de dados. É comum existirem funções e/ou métodos que realizam operações no banco de dados, principalmente operações do tipo consulta (*Select*). Para os casos de operações de consulta, devem-se localizar os arquivos de código-fonte que contém as funções e/ou métodos das operações de inserção (*insert*), atualização (*update*) e exclusão (*delete*), pois esses arquivos representam parte da funcionalidade que não está diretamente relacionada com cada interface de usuário.

Os artefatos envolvidos nesta atividade são:

- ENTRADA:
    - Pacotes de Código-Fonte
- SAÍDA:
    - Arquivos de Código-Fonte

4.4.3.3 Identificar Comentários Relevantes

Essa atividade objetiva localizar e registrar os comentários selecionados, que existem nos arquivos de código-fonte e que contribuam para o entendimento das possíveis regras de negócio do software legado. Para isso, deve-se realizar uma análise dos comentários existentes no código-fonte e relacioná-los com as invocações de métodos que eles explicam.

O resultado dessa atividade permite entender o funcionamento e a implementação do software legado, fornecendo subsídios para a execução das fases posteriores de Redocumentação e Elaboração do Modelo de Metas.

Através do resultado da análise dos comentários existentes nos arquivos de código-fonte, o Analista de Sistemas elabora o artefato chamado Documento com Informações de Comentários Relevantes. Esse documento contém o registro dos comentários relevantes extraídos do código-fonte, tais como: nome do arquivo de código-fonte legado; número da linha do comentário e texto do comentário.

Os artefatos envolvidos nesta atividade são:

- ENTRADA:
  - Arquivos de Código-Fonte
- SAÍDA:
  - Documento com Informações de Comentários Relevantes

4.4.4 Fase de Redocumentação

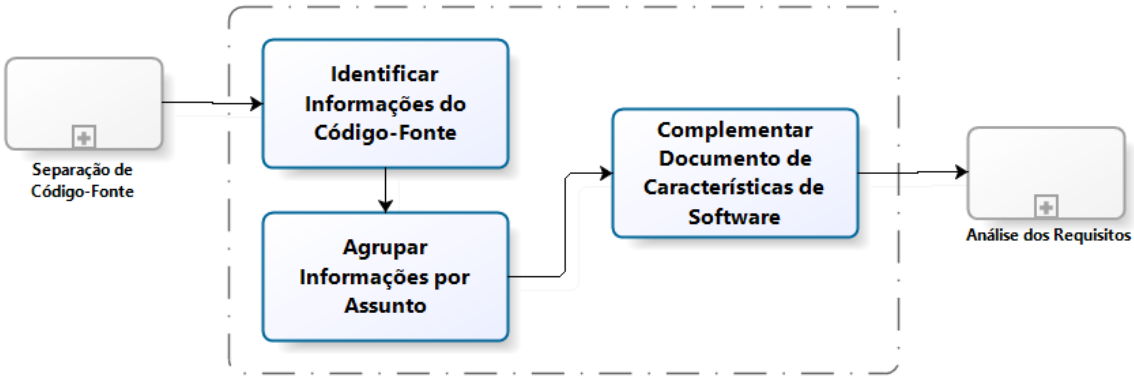
O objetivo principal da fase Redocumentação é analisar o código-fonte e elicitar os requisitos, com base nas informações relevantes para redocumentar o software legado, tais como modelo de dados, regras de negócio, sequência de eventos. Esta fase foi baseada no estudo do método de Redocumentação de Sistemas Legados (GEET; EBRAERT; DEMEYER, 2010).

Esta fase utiliza os arquivos de código-fonte separados na fase Separação de Código-Fonte, pois tem como pré-requisito estudar somente o que está dentro do escopo definido para o ERSL. É realizada pelo Analista de Sistemas pois, como na

fase Separação de Código-Fonte, as atividades necessitam de um perfil técnico de análise.

A Figura 24, apresenta as atividades dessa fase.

Figura 24- Atividades da fase de Redocumentação



Os artefatos relacionados com essa fase são:

- ENTRADA:
  - Documento de Características de Software: contém a descrição em alto nível dos requisitos identificados com a análise da interface de usuário do software legado.
  - Arquivos de Código-Fonte: são arquivos contendo o código-fonte legado que estão relacionados com o escopo do ERSL.
  - Documento com Informações de Comentários Relevantes: contém a relação dos comentários relevantes, presentes no código-fonte do software legado.
- SAÍDA:
  - Documento de Características de Software Complementado: contém a descrição em alto nível dos requisitos identificados com a análise das interfaces, complementado com a descrição dos requisitos identificados ao analisar o código-fonte legado.

Tem-se, a seguir, o detalhamento das atividades pertencentes à fase Redocumentação:

4.4.4.1 Identificar Informações do Código-Fonte

Essa atividade objetiva identificar as informações contidas nos arquivos de código-fonte. As informações podem ser sobre modelo de dados, regra de negócio, sequência de eventos, funções e quaisquer aspectos que estão implementados e que não são perceptíveis ao analisar as interfaces do software legado.

O Analista de Sistema coleta os aspectos contidos no código-fonte e os analisa para identificar os requisitos e como eles estão implementados, gerando um documento chamado de Documento de Características do Código-Fonte. Os aspectos são identificados através da análise de cada arquivo de código-fonte, baseada no entendimento semântico de como o software funciona.

A Tabela 9 ilustra o exemplo de um aspecto descrito no Documento de Características do Código-fonte. Na coluna A estão listados os tipos de dados que classificam a informação e na coluna B estão as informações encontradas no código-fonte. Nota-se que as informações identificadas nessa tabela são somente possíveis de serem coletadas através da análise dos Arquivos de Código-Fonte. Por exemplo, na linha Tipo, pode-se observar que o arquivo de código-fonte analisado é referente a um programa *batch*; além disso, a maioria das informações identificadas não são facilmente elicitadas somente pela análise das interfaces do software legado. Desta forma, constata-se que o papel da Redocumentação é fundamental para elicitar os detalhes dos requisitos no ERSL.

Os artefatos envolvidos nesta atividade são:

- ENTRADA:
  - Arquivos de Código-Fonte
  - Documento com Informações de Comentários Relevantes
- SAÍDA:
  - Documento de Características do Código-Fonte

Tabela 9 - Exemplo de Documento de Características do Código-Fonte

A	B
ID	1
Modulo	Produtos
Arquivo de Código	ListaProduto.vb
Tipo	BATCH
Qtd. Funções	3
Lista de Funções	• BuscarDataAtual() • BuscaCodigosProdutos(dataAtual) • BuscarDescricaoProduto(listCodProd)
Qtd. Variáveis	5
Lista de Variáveis	• dataAutal • dataAnterior • dataPosterior • listCodProd • listProd
Qtd. Invocações	3
Sequência de Invocações	1.0. BuscarDataAtual() 1.1. BuscaCodigosProdutos(dataAtual) 1.1.2. BuscarDescricaoProduto(listCodProd)
Qtd. Regras	1
Lista de Regras	Retornar Produtos Vigentes baseado na Data Atual
Qtd. Tabelas	3
Qtd. Operações	• TabDataMovimento • TabVigenciaProduto • TabDescricaoProduto
Qtd. Operações	1
Operações Banco de Dados	• Seleção

4.4.4.2 Agrupar Informações por Assunto

Esta atividade objetiva separar e agrupar as informações coletas, através da análise do Documento de Características do Código-Fonte, por tipo de assunto. É gerado, então, o Documento de Características Agrupadas do Código-Fonte.

O agrupamento das informações é feito, levando em consideração o relacionamento da informação com o requisito identificado a partir das Interfaces. Essa atividade permite, portanto, integrar o resultado obtido através da Análise das Interfaces, no contexto da Redocumentação.

Essa atividade é realizada pelo Analista de Sistema e, como resultado, tem-se o artefato Documento de Características Agrupadas do Código-Fonte similar ao documento exemplificado na Tabela 9, acrescido de uma linha chamada Requisito (s), conforme Tabela 10.

Tabela 10 - Exemplo de informação incrementada no Documento de Características do Código-Fonte

A	B
ID	1
Modulo	Produtos
Funcionalidade	Consulta de Produtos
Requisito (s)	Gestão de Produtos
Tipo	BATCH
Lista de Funções / Métodos	• BuscarDataAtual() • BuscaCodigosProdutos(dataAtual) • BuscarDescricaoProduto(listCodProd)
Lista de Regras	Retornar Produtos Vigentes baseado na Data Atual
Qtd. Operações	• TabDataMovimento • TabVigenciaProduto • TabDescricaoProduto
Operações Banco de Dados	• Seleção

Os artefatos envolvidos nesta atividade são:

- ENTRADA:
  - Documento de Características do Código-Fonte
- SAÍDA:
  - Documento de Características Agrupadas do Código-Fonte

4.4.4.3 Complementar Documento de Características de Software

Esta atividade objetiva complementar o artefato Documento de Características de Software, gerado na fase de Análise de Interfaces, com os requisitos identificados na fase Redocumentação. A atividade é realizada pelo Analista de Sistemas, que gera o artefato Documento de Características de Software Complementado.

Os artefatos envolvidos nesta atividade são:

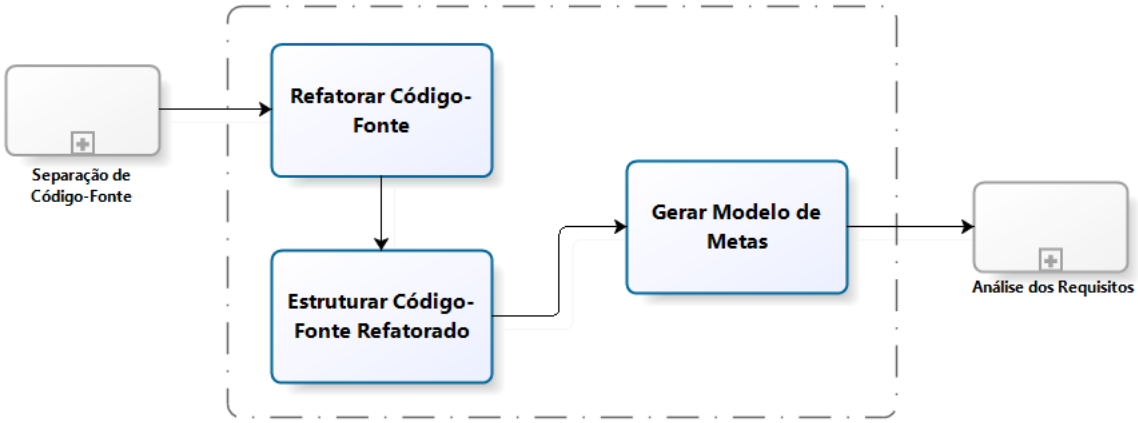
- ENTRADA:
  - Documento de Características Agrupadas do Código-Fonte
  - Documento de Características de Software
- SAÍDA:
  - Documento de Características de Software Complementado

4.4.5 Fase de Elaboração do Modelo de Metas

O objetivo principal da fase de Elaboração do Modelo de Metas é identificar as metas de usuário que o software legado atende, através da análise dos arquivos de código-fonte. É realizada pelo Analista de Sistemas e o resultado é representado através de diagramas que constituem o Modelo de Metas. Esta fase foi baseada no estudo do método Engenharia Reversa através de Modelos de Metas (YU et al., 2005).

Esta fase utiliza o resultado da fase de Separação de Código-Fonte, pois detalha o entendimento dos arquivos de código-fonte relacionados somente com o que está no escopo definido para ERSL. Na Figura 25, é apresentado o detalhamento dessa fase, através das suas atividades.

Figura 25 - Atividades da fase Elaboração do Modelo de Metas



Os artefatos relacionados com essa fase são:

- ENTRADA:
  - Arquivos de Código-Fonte: são arquivos contendo o código-fonte legado que estão relacionados com o escopo do ERSL.
  - Documento com Informações de Comentários Relevantes: contém a relação dos comentários relevantes, presentes no código-fonte do software legado.
- SAÍDA:
  - Modelo de Metas: contém os diagramas que representam as metas de usuário e suas respectivas funções necessárias para atendimento desta meta.

Tem-se, a seguir, o detalhamento das atividades pertencentes à fase Elaboração do Modelo de Metas:

4.4.5.1 Refatorar Código-Fonte

Essa atividade consiste em refatorar o código-fonte legado, em função dos comentários existentes.

O ponto inicial dessa atividade é analisar os comentários contidos no código-fonte e, a partir dessa análise, transformar o bloco de código em uma função. Porém, é provável que os comentários existentes estejam desatualizados ou nem existam.

Neste caso, é necessário ler, entender e separar os blocos de código, transformando, assim, cada bloco em uma respectiva função.

O Analista de Sistema realiza a refatoração proposta, que consiste em localizar esses blocos, transformá-los em funções e substitui-los em respectivas invocações dessas funções. O artefato resultante é o Código-Fonte Refatorado, que consiste de arquivos de código-fonte com somente as invocações das funções, abstraindo-se a lógica implementada em cada método ou função.

Os artefatos envolvidos nesta atividade são:

- ENTRADA:
  - Arquivos de Código-Fonte
  - Documento com Informações de Comentários Relevantes
- SAÍDA:
  - Código-Fonte Refatorado

4.4.5.2 Estruturar Código-Fonte Refatorado

Essa atividade consiste em transformar o código-fonte refatorado, obtido na atividade Refatorar Código-Fonte, em um código-fonte estruturado. A execução desta atividade necessita de grande atenção, para não alterar nenhuma regra de negócio.

No início da atividade, o Analista de Sistemas deve verificar se o código-fonte refatorado já está estruturado; em caso positivo, não é necessário executar essa atividade. Se o código-fonte não estiver estruturado, deve-se transformar o código-fonte refatorado em um gráfico de estados e consecutivamente em código-fonte estruturado, conforme método contido na seção 3.4.

Os artefatos envolvidos nesta atividade são:

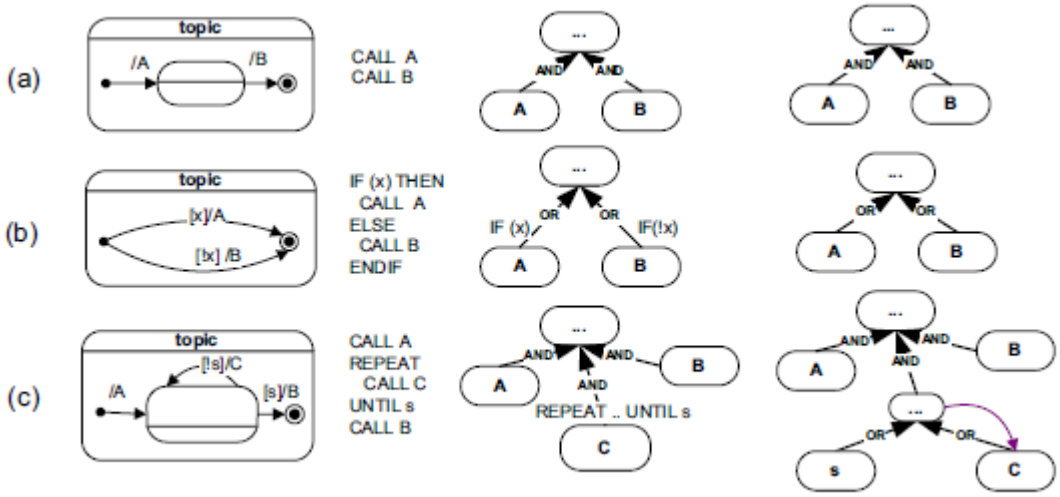
- ENTRADA:
  - Código-Fonte Refatorado
- SAÍDA:
  - Código-Fonte Estruturado

4.4.5.3 Gerar Modelo de Metas

Essa atividade tem como objetivo construir os diagramas do Modelo de Metas. Esse modelo representa as metas de usuário que o software legado atende e as funções necessárias para que essas metas sejam atingidas.

Para transformar o código-fonte estruturado em um diagrama de Modelo de Metas, o Analista de Sistemas deve identificar, no código-fonte estruturado, os padrões apresentados na Figura 26 e transformá-lo em diagramas. Maiores detalhes dessa transformação podem ser vistos na seção 3.4.2 do capítulo 3.

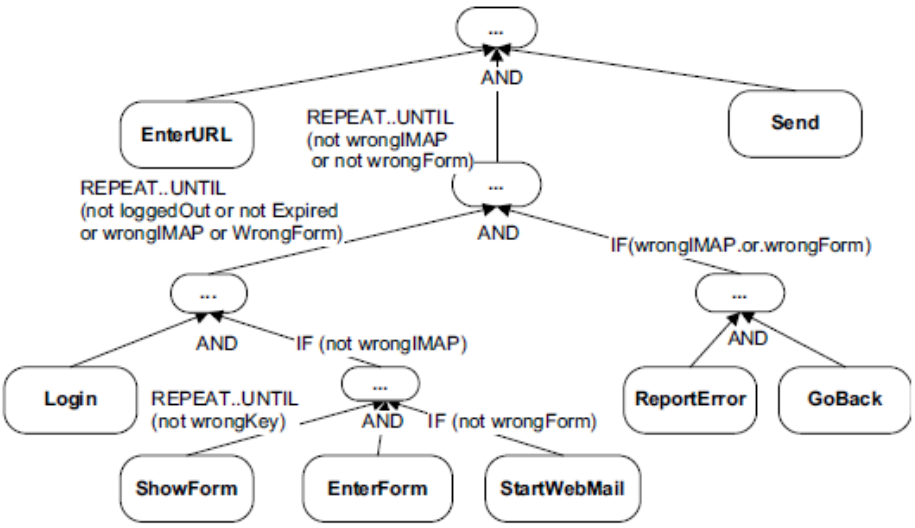
Figura 26 - Padrões para extração do Modelo de Metas



Fonte: Yu et al. (2005)

Após a aplicação dos padrões apresentados na Figura 26, chega-se no diagrama de Modelo de Metas que representa as metas de usuário identificadas no Código-Fonte Estruturado e as respectivas funções necessárias para atendimento de cada meta. Como resultado, obtém-se o diagrama representado na Figura 27.

Figura 27 – Exemplo de Modelo de Metas extraído do código estruturado



Fonte: Yu et al. (2005)

Os artefatos envolvidos nesta atividade são:

- ENTRADA:
  - Código-Fonte Estruturado
- SAÍDA:
  - Modelo de Metas

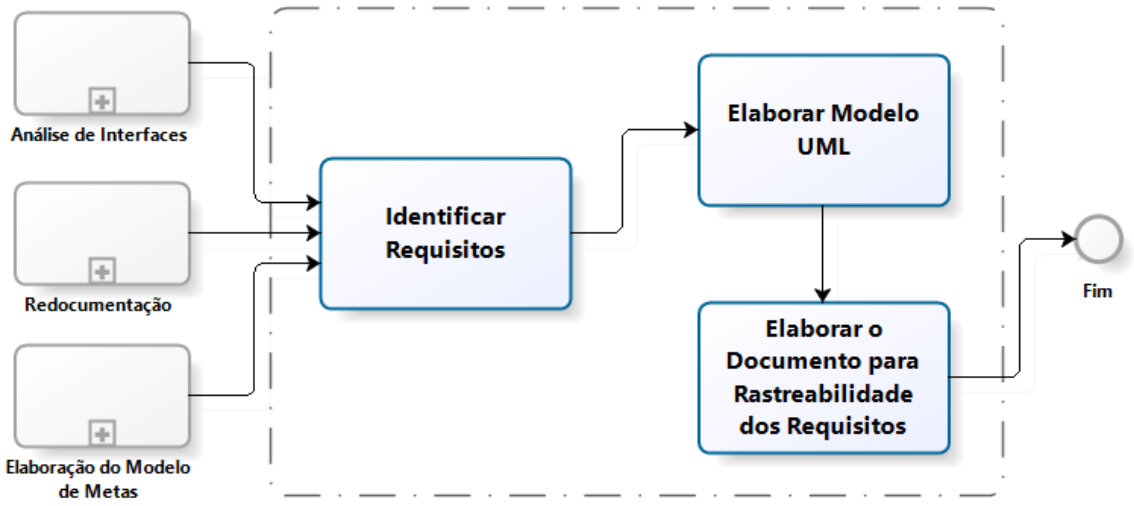
4.4.6 Fase de Análise dos Requisitos

O objetivo principal da fase Análise dos Requisitos é analisar os requisitos elicitados nas fases anteriores e criar modelos e/ou documentação que serão utilizadas para a reengenharia do software legado. Esta fase é a última do processo ERSL.

A representação dos requisitos utilizada nesta fase, deve ser selecionada entre as técnicas definidas pela Engenharia de Software. No caso do processo ERSL, a UML foi considerada como representação a ser utilizada. Essa fase tem a participação de Analista de Sistemas, Analista de Requisitos e Analista de Migração que, com base nos requisitos identificados nas fases anteriores, geram a documentação adequada.

A Figura 28 apresenta as atividades que compõem essa fase.

Figura 28 - Atividades da fase Análise dos Requisitos



Os artefatos relacionados com essa fase são:

- ENTRADA:
  - Documento de Características de Software Complementado: contém a descrição em alto nível dos requisitos identificados com a análise das interfaces, complementado com a descrição dos requisitos identificados ao analisar o código-fonte legado.
  - Gráfico de Interfaces Detalhado: contém o diagrama com a navegação entre interfaces, eventos que estimulam as transições e as informações de entrada e saída para cada interface de usuário.
  - Modelo de Metas: contém os diagramas que representam as metas de usuário e suas respectivas funções necessárias para atendimento desta meta.
- SAÍDA:
  - Modelo UML: contém diagrama(s) da UML, representando os requisitos do software legado elicitados através do processo ERSL.
  - Mapa de Rastreabilidade de Requisitos: contém a rastreabilidade dos requisitos, mostrando o relacionamento entre o requisito elicitado e a fase do processo em que ele foi identificado.

Tem-se, a seguir, o detalhamento de cada atividade pertencente à fase Análise dos Requisitos:

4.4.6.1 Identificar Requisitos

Essa atividade tem como objetivo analisar os artefatos gerados nas fases anteriores, identificando os requisitos elicitados. Cada uma das fases anteriores teve o propósito de elicitar os requisitos com focos diferentes e, nessa atividade, eles serão unificados em documentos do software legado. Observa-se que é comum encontrar os mesmos requisitos em artefatos que foram gerados em fases diferentes; por isso, os diversos documentos devem ser analisados e o seu conteúdo unificado.

Para unificar os requisitos em um artefato, o Analista de Sistemas e o Analista de Requisitos elaboram o Documento de Requisitos do Software Legado que contém a descrição dos requisitos elicitados, do software legado, referentes ao escopo definido para a aplicação do ERSL. Um exemplo da descrição de requisitos pode ser mostrado através da Tabela 11.

Nesse exemplo, o requisito foi descrito através do nome do requisito, das funções que compõem este requisito, da sua dependência com outro requisito (para realizar o cadastro de um produto é necessário que o fornecedor deste já esteja cadastrado no software) e das regras de negócio que estão envolvidas com o requisito.

Os artefatos envolvidos nesta atividade são:

- ENTRADA:
  - Documento de Características de Software Complementado
  - Gráfico de Interfaces Detalhado
  - Modelo de Metas
- SAÍDA:
  - Documento de Requisitos do Software Legado

Tabela 11 - Exemplo de Documento de Requisitos do Software Legado

A	B
ID	1
Requisito	Cadastro de Produto
Funcionalidades	• Incluir Produto • Alterar Produto • Excluir Produto • Consultar Produto
Dependências	2 – Cadastro de Fornecedor
Regras de Negócio	
1	Necessário atrelar o produto à um fornecedor cadastrado
2	O produto deve ter data de início e termino de vigência
3	Ao cadastrar um novo produto deve utilizar a data atual como data de início de vigência
4	Ao alterar um produto existente, a vigência do produto é encerrada e é cadastrado uma nova vigência.
5	O produto nunca será excluído fisicamente, é somente encerrado a vigência.
6	A consulta dos produtos pode ser realizada por: produtos vigentes, produtos não vigentes ou todos (vigentes e não vigentes).

4.4.6.2 Elaborar Modelo UML

Essa atividade tem como objetivo elaborar os modelos UML que representem, de forma padronizada, os requisitos identificados com a aplicação do ERSL. Os modelos UML mais apropriados para a documentação dependem do tipo do software e do objetivo da reengenharia. Esta atividade possui um nome abrangente, pois a decisão sobre qual modelo da UML deve ser construído, depende do Analista de Migração, que define a estratégia a ser empregada na reengenharia do software legado. Podem-se utilizar, por exemplo, os modelos de Casos de Uso, de Classes, de Diagrama de Sequência e de Diagrama de Estados.

A construção destes modelos é realizada pelo Analista de Requisitos, que transforma o Documento de Requisitos do Software Legado em diagramas dos modelos. Os Modelos UML são os resultados do processo ERSL.

Os artefatos envolvidos nesta atividade são:

- ENTRADA:
  - Documento de Requisitos do Software Legado
- SAÍDA:
  - Modelo UML

4.4.6.3 Elaborar Documento para Rastreabilidade dos Requisitos

Essa atividade tem como objetivo relacionar os requisitos com os artefatos em que foram identificados.

Para manter a informação dos requisitos identificados com os documentos gerados nas fases do processo ERSL, é feito o mapeamento dos requisitos do Modelo UML com os documentos em que cada requisito foi elicitado.

O Analista de Requisitos e o Analista de Sistemas devem criar o artefato chamado de Mapa de Rastreabilidade dos Requisitos, que contém esse mapeamento, através da análise do Modelo UML e dos artefatos gerados pela aplicação do processo ERSL. A Tabela 12 é um exemplo do Mapa de Rastreabilidade dos Requisitos.

Tabela 12 - Exemplo de Mapa de Rastreabilidade dos Requisitos

Requisito do Modelo UML	Identificado na Fase		
	Analise de Interfaces	**Redocumentação**	**Modelo de Metas**
Cadastro de Produto	Identificado	Identificado	Identificado
Cadastro de Fornecedor		Identificado	Identificado
Controle do Estoque	Identificado	Identificado	
Notificação de Falta de Produto			Identificado

Nesse exemplo, pode–se notar que a coluna da esquerda (Requisito do Modelo UML) contém os requisitos identificados na atividade Elaborar Modelo UML e as demais colunas contêm a informação da fase do processo em que o requisito foi identificado.

A conclusão do Mapa de Rastreabilidade dos Requisitos corresponde ao final do processo ERSL, permitindo, portanto, o início do processo de reengenharia do software legado.

Os artefatos envolvidos nesta atividade são:

- ENTRADA:
  - Documento de Características de Software Complementado
  - Gráfico de Interfaces Detalhado
  - Modelo de Metas
  - Modelo UML
- SAÍDA:
  - Mapa de Rastreabilidade de Requisitos

4.5 CONSIDERAÇÕES DO CAPÍTULO

Nesse capítulo foi apresentado o processo ERSL, definido dentro do contexto de migração gradativa de um software legado, através de fases, atividades, artefatos e participantes. A execução correta das atividades, assim como sua sequência e os artefatos gerados, são essenciais para se obter os requisitos elicitados e, com isso, aplicar a reengenharia do software legado.

É importante entender que os artefatos gerados no processo ERSL constituem os tipos de documentos do software legado. Mesmo que não se realize a reengenharia deste software legado, após o término do ERSL, obtém-se os documentos necessários para prosseguir, posteriormente, as fases da migração.

O processo ERSL foi aplicado sobre uma parte de um software legado real, que está sofrendo o processo de migração, para avaliar a sua eficácia na identificação de requisitos.

5 APLICAÇÃO DO PROCESSO ERSL

Este capítulo tem como objetivo descrever a aplicação do processo ERSL em um ambiente real de migração, desde a definição do cenário, comentários sobre as fases executadas, resultados obtidos, até conclusões sobre a aplicação do processo.

5.1 CENÁRIO DA APLICAÇÃO DO PROCESSO ERSL

O sistema legado escolhido para aplicação do processo está inserido no contexto de uma companhia de seguros. Nesse segmento de negócio, os produtos da companhia são vendidos por parceiros que não fazem parte do quadro de funcionários, chamados de Corretores de Seguros. Com base neste princípio de vendas, todo o seguro vendido possui o corretor de seguros como intermediador da venda.

Essa companhia possui um grande sistema legado proprietário, construído entre os anos de 1990 a 2000, e é composto de vários módulos de software, sendo que cada módulo possui uma responsabilidade específica no sistema.

5.1.1 Software Legado Analisado

O módulo do software legado escolhido foi o Módulo de Consulta de Produção, responsável por calcular e apresentar a produção de vendas para cada corretor de seguro, ou seja, todas as vendas são apresentadas, para que o corretor acompanhe o seu desempenho.

Esse módulo possui interfaces de usuário, desenvolvidas na linguagem VB6 e vb.Net, sobre arquitetura cliente-servidor; também possui módulos de programa do tipo *batch*, desenvolvidos em VB6 e *scripts*, os quais são executados diretamente no banco de dados. Todos os dados são armazenados no banco de dados Sybase ASE.

5.1.2 Motivação para Escolha do Software Legado

Esse módulo de software foi escolhido pois se enquadra como um software legado devido à sua estabilidade em relação à manutenção, à grande importância

para o negócio da companhia e por estar em execução no ambiente produtivo há mais de 15 anos. Sua última manutenção classificada como corretiva foi realizada há 2 anos atrás, ou seja, poucos erros têm sido identificados, e sua última manutenção evolutiva foi realizada há 4 anos atrás.

Além das motivações apresentadas anteriormente, existe também a necessidade de migração de todo o sistema utilizado na companhia seguradora para um novo software, pois o seu software legado utiliza tecnologias obsoletas e, além disso, a realização da manutenção, motivada por mudanças de estratégia dos negócios, é muito difícil.

5.2 INFORMAÇÕES COLETADAS COM O PROCESSO ERSL

Com a aplicação do processo ERSL foi possível coletar detalhes de implementação que são imperceptíveis para os usuários do software legado. Cada fase do processo identificou informações do software legado em níveis diferentes de abstração. Para fins comparativos, as informações coletadas em cada fase do processo ERSL foram consolidadas de forma quantitativa para essa seção.

5.2.1 Número de Elementos Identificados

Com a finalidade de avaliar quantitativamente o resultado das fases do processo ERSL, foram considerados os elementos identificados nos artefatos gerados, cuja quantidade teve variações em relação aos métodos utilizados em cada uma das fases. A Tabela 13, mostra que as fases de Análise de Interfaces, Redocumentação e Elaboração de Modelo de Metas objetivam níveis diferentes de detalhamentos de requisitos e também focos diferentes da engenharia reversa.

Os elementos considerados para comparar o resultado das fases foram: requisitos, funções, regras de negócio, dados e dependências entre funções e/ou requisitos.

Tabela 13 - Quantidade de elementos identificados por fase do ERSL

Quantidade	Análise de Interfaces	Redocumentação	Modelo de Metas
Requisitos	3	4	3
Funções	41	76	20
Regras	2	7	5
Dados	36	98	30
Dependências	2	5	13

A fase de Análise de Interfaces resultou em menores quantidades de regras e dependências entre funções e/ou requisitos, pois esse tipo de elemento está implícito no código-fonte implementado e não é possível ser percebido apenas pelas interfaces. A fase de Redocumentação foi a que resultou em maiores quantidades de elementos identificados, pois é aplicado em um nível de abstração muito baixo, através da interpretação do código. A fase de Elaboração de Modelo de Metas resultou em menor quantidade de dados e função, pois ela objetiva elevar o nível de abstração baseado no código-fonte; apesar disso, permitiu o entendimento da estrutura do programa, através do Modelo de Metas.

5.2.2 Elementos Identificados

Com base na Tabela 13, pode-se concluir que os elementos identificados foram diferentes em cada uma das fases do processo ERSL.

A fase de Análise de Interfaces resultou na identificação de elementos do software legado, em um nível mais alto de abstração. Porém, isso foi esperado, pois sua principal função foi auxiliar o entendimento sobre a utilização do software legado e sobre as informações geradas que são importantes para os usuários. Os principais elementos identificados durante a fase de Análise de Interfaces são:

- Dados fornecidos por usuários
- Dados apresentados para usuários
- Eventos estimulados por ações do usuário
- Dependências de dados entre interfaces

A fase de Redocumentação forneceu maior detalhamento na identificação de elementos; isso ocorreu, pois, o foco da sua aplicação é sobre o código-fonte. Os principais elementos identificados durante a fase de Redocumentação são:

- Regras de Negócio
- Detalhamento de Funções
- Constantes e Domínios de Valores
- Modelagem de Dados

A fase de Elaboração de Modelo de Metas resultou informações em um nível mais abstrato do que as outras fases; porém, permitiu a identificação de informação de relevante importância, para o mapeamento das dependências entre funções necessárias na implementação dos requisitos. Os principais elementos identificados com a fase de Elaboração de Modelo de Metas são:

- Dependências entre funções
- Dependências entre requisitos
- Regras entre dependências

Ao comparar as informações identificadas em cada uma das fases, nota-se que foram obtidas informações com detalhamento de diferente nível de abstração que, em conjunto, forneceram requisitos com maior completeza, elevando a qualidade dos requisitos elicitados com o processo.

5.3 ARTEFATOS GERADOS COM O PROCESSO ERSL

A aplicação do processo ERSL resultou em 20 artefatos gerados; dentre eles são descritos, nessa seção, três artefatos que se destacam por fornecerem visões diferentes dos requisitos elicitados.

Os artefatos descritos são: Documento de Características de Software, Gráfico de Interfaces Detalhado e o Modelo de Metas.

5.3.1 Documento de Características de Software

O Documento de Características de Software é um artefato que foi gerado na fase de Análise de Interfaces e posteriormente complementado na fase de Redocumentação. Para avaliar a evolução do artefato, foram apresentados, na Tabela 14, os números de elementos contidos quando gerado na fase de Análise de Interfaces e após ser complementado na fase de Redocumentação.

Tabela 14 - Quantidade de elementos do Documento de Características de Software

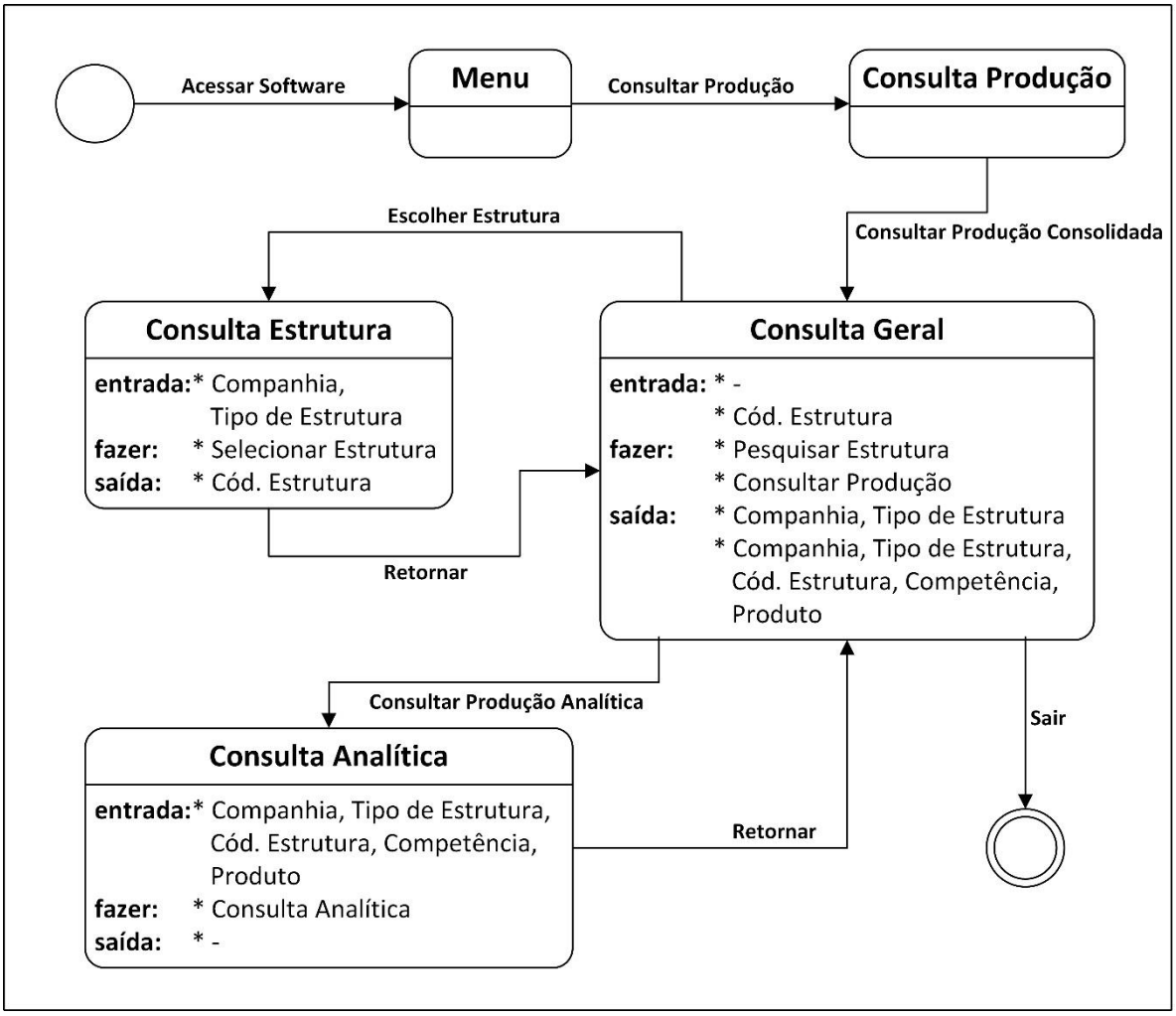
Quantidade	Análise de Interfaces	Redocumentação
Requisitos	3	3
Funções / Métodos	17	23
Dependências	2	2
Regras de Negócio	7	13

Pode-se notar que os elementos incrementados com a Redocumentação foram as funções e/ou métodos e de regras de negócio. Isso ocorreu, pois, a análise que a Redocumentação aplica no nível de código-fonte é mais efetiva na identificação de regras de negócio e detalhamento das funções do software, que não são perceptíveis aos usuários, diferentemente dos requisitos e dependências que são perceptíveis aos usuários do software legado.

5.3.2 Gráfico de Interfaces Detalhado

O Gráfico de Interfaces Detalhado, que é o resultado da aplicação da fase de Análise de Interfaces, foi fundamental para a execução das fases seguintes do processo ERSL, pois auxiliou no mapeamento de cada arquivo de código-fonte relacionado com o escopo do processo ERSL. A Figura 29 é o Gráfico de Interfaces Detalhado do software legado analisado.

Figura 29 - Gráfico de Interfaces Detalhado do Software Legado Estudado



Ao analisar a Figura 29, percebeu-se que a interface Consulta Geral pode receber entradas diferentes, fazer ações diferentes e também fornecer saídas diferentes, pois é uma interface que pode ter estados diferentes dependendo da ação do usuário, além de ser uma interface central para o software analisado. As interfaces Consulta Estrutura e Consulta Analítica fornecem e dependem respectivamente dos dados da interface Consulta Geral.

5.3.3 Modelo de Metas

O Modelo de Metas, que é o resultado da fase de Elaboração do Modelo de Metas, apresenta de forma estruturada as funções necessárias para atender os

requisitos identificados nas fases de Análise de Interfaces e de Redocumentação. As Figura 30 e Figura 31 apresentam o Modelo de Metas para o software estudado.

Figura 30 - Diagrama de Modelo de Metas do Software Legado Estudado (Parte 1)

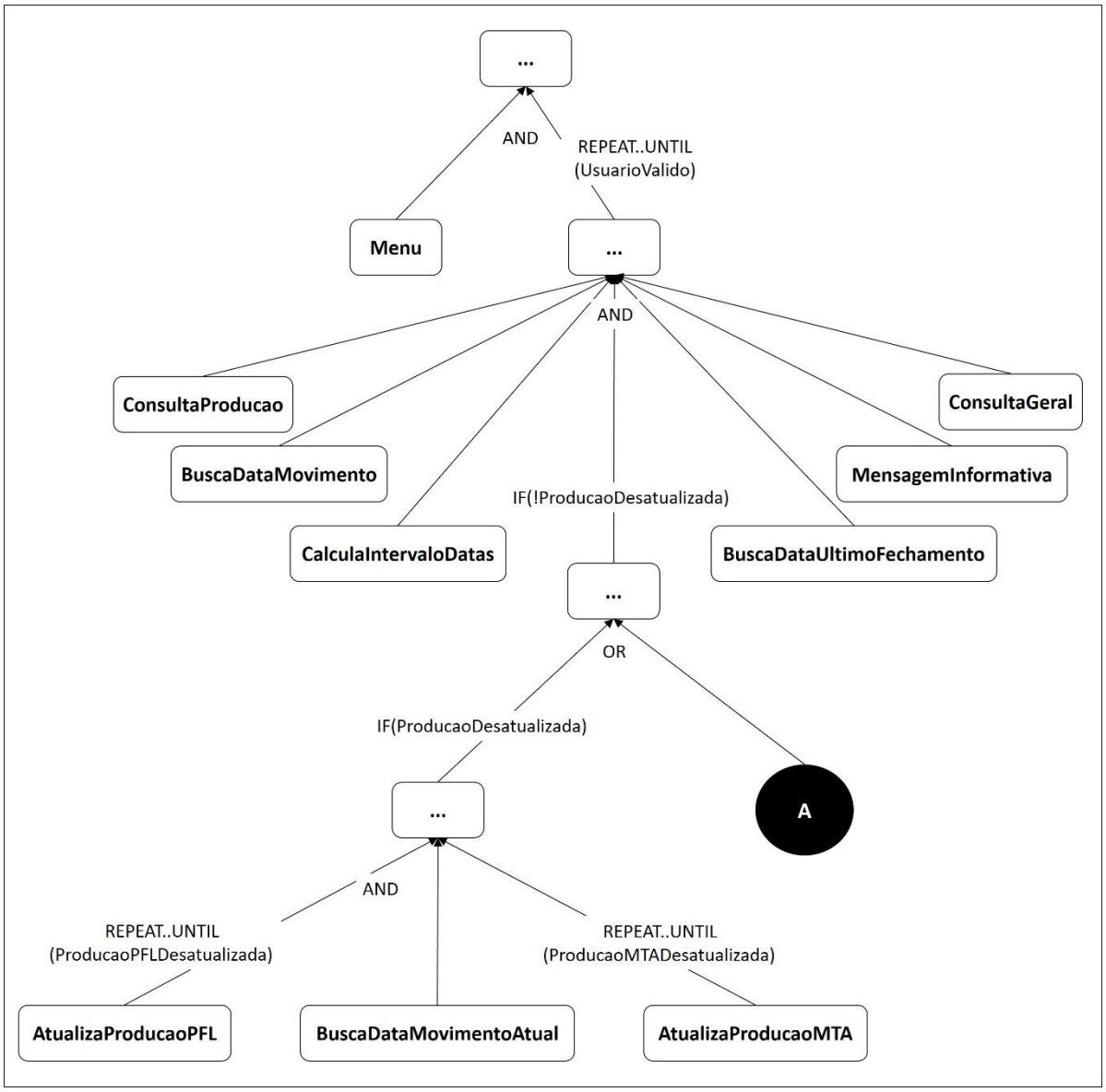
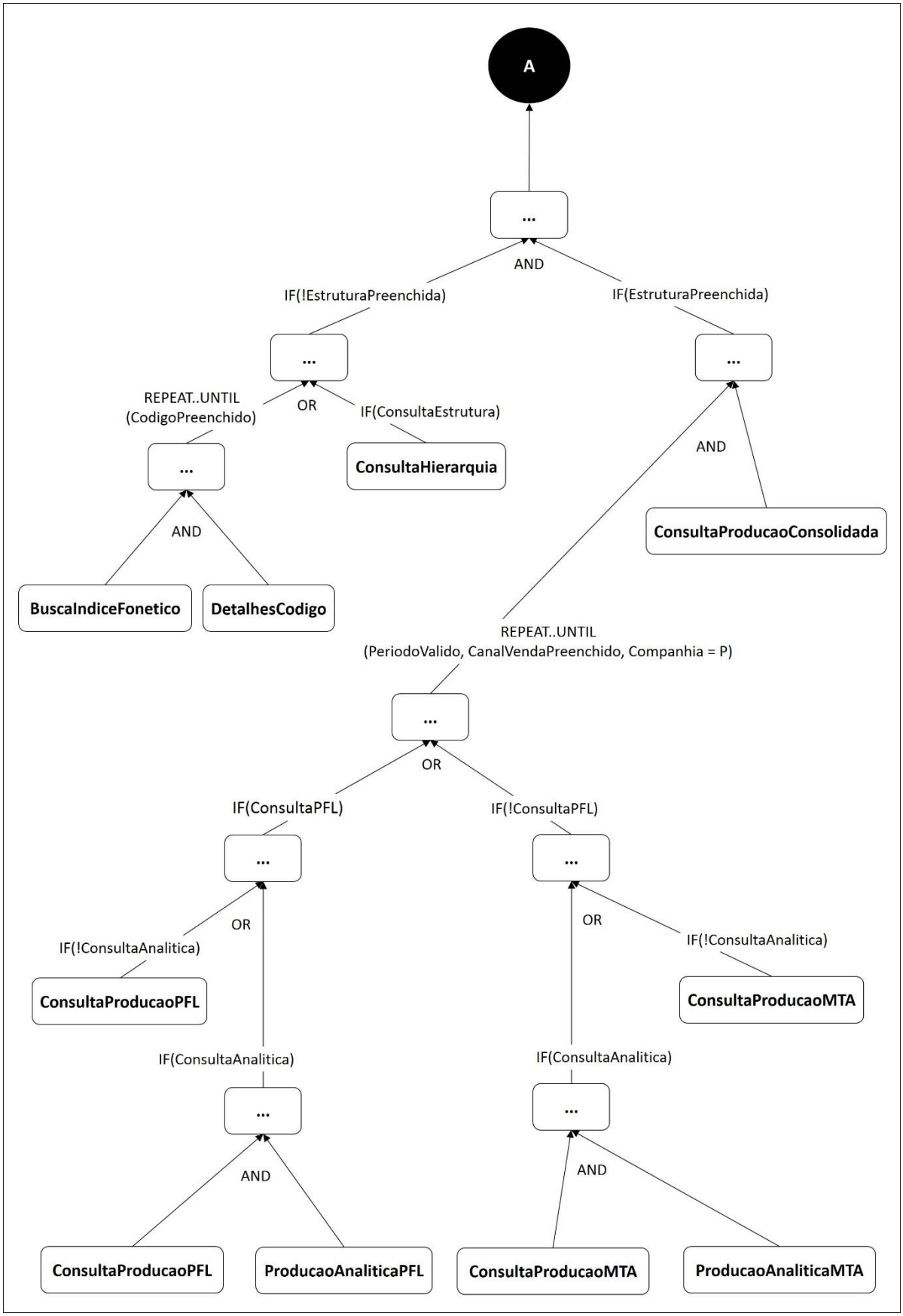


Figura 31 - Diagrama de Modelo de Metas do Software Legado Estudado (Parte 2)



O diagrama da Figura 30 representa a primeira parte do Modelo de Metas do software legado estudado. O nó localizado na parte superior representa a meta de usuário identificada como Consulta de Produção, o restante dos nós representa de forma estruturada as funções que o software possui para atender a meta de usuário. O diagrama da Figura 31 representa a segunda parte do Modelo de Metas, contendo o restante das funções que o software legado possui para atender a meta de usuário explicitada na Figura 30.

O Modelo de Metas e o Gráfico de Interfaces Detalhado apresentam informações semelhantes, apesar de serem gerados em fases diferentes. Essa semelhança ocorre, pois, o conteúdo dos dois artefatos tem o foco no usuário. O Gráfico de Interfaces Detalhado resulta da execução do software do ponto de vista do usuário e o Modelo de Metas representa as metas do usuário em relação ao software legado.

5.4 REQUISITOS IDENTIFICADOS COM O PROCESSO ERSL

Com a execução da fase de Análise dos Requisitos, obtiveram-se conclusões sobre as fases de Análise de Interfaces, Redocumentação e Elaboração de Modelo de Metas, pois cada uma destas fases identificou requisitos, que no final se tornaram dependentes ou foram incorporados em outros requisitos.

Ao término da fase de Análise dos Requisitos, todos os requisitos identificados nas fases anteriores foram consolidados, resultando em três requisitos:

1) Informações de Produção Atualizadas: todas as vendas realizadas devem ser enviadas para o software legado realizar o cálculo da produção.
2) Estrutura Hierárquica: os corretores de seguros devem estar em uma estrutura hierárquica por região de atuação, para que as consultas possam ser realizadas em qualquer nível da hierarquia.
3) Consulta de Produção: apresentar a produção em valores e quantidade de vendas para qualquer nível na hierarquia dos corretores.

A Tabela 15 apresenta os requisitos identificados com suas respectivas quantidades de funcionalidades e regras de negócio.

Tabela 15 - Requisitos Identificados do Software Legado Estudado

Requisito	Funcionalidades	Regras de Negócio
Informações de Produção Atualizadas	2	1
Estrutura Hierárquica	3	3
Consulta de Produção	7	10

Na Tabela 15, pode-se constatar que a quantidade de funcionalidades e regras de negócio que estão relacionadas com o requisito Consulta de Produção é muito maior que os outros requisitos, ficando evidente que o requisito principal do software legado estudado é a Consulta de Produção.

A construção do Mapa de Rastreabilidade dos Requisitos possibilitou comparar as fases de elicitação dos requisitos. Esse artefato é importante para a validação dos requisitos elicitados. A Tabela 16 apresenta o Mapa de Rastreabilidade dos Requisitos.

Tabela 16 - Mapa de Rastreabilidade dos Requisitos do Software Legado Estudado

Requisito do Modelo UML	Identificado na Fase		
	Analise de Interfaces	**Redocumentação**	**Modelo de Metas**
Informações de Produção Atualizadas		Identificado	Identificado
Estrutura Hierárquica	Identificado	Identificado	Identificado
Consulta de Produção	Identificado	Identificado	Identificado

Com a Tabela 16, conclui-se que apenas o requisito Informações de Produção Atualizadas que não foi identificado em todas as fases de elicitação. Ao comparar os requisitos identificados nas fases de elicitação, tem-se resultados semelhantes; isso

ocorre, pois, cada uma das fases identifica o requisito em um nível de detalhamento diferente.

5.5 PONTOS POSITIVOS E DIFICULDADES

Com a aplicação do processo ERSL, foram encontrados pontos positivos, que confirmam a coerência do processo, e dificuldades que apontam a necessidade de possíveis melhorias no processo.

5.5.1 Pontos Positivos com a Aplicação do Processo

Os pontos positivos da aplicação do processo ERSL estão relacionados à distribuição e sequência das atividades.

Em primeiro lugar, pode-se citar que a fase de Análise de Interfaces, executada como primeira fase de engenharia reversa, auxilia no entendimento do software legado e facilita a execução das demais fases do processo ERSL.

Outro ponto positivo está relacionado com a distribuição das fases do processo, pois durante a sua aplicação foi claramente perceptível que cada uma das fases complementa os requisitos elicitados na fase anterior, aumentando a qualidade dos requisitos identificados.

Por fim, também deve ser citado como positivo a fase de Análise dos Requisitos, pois contribuiu para finalizar o processo, realizando a validação e consolidação dos requisitos elicitados.

5.5.2 Dificuldades Encontradas com a Aplicação do Processo

A dificuldade na aplicação do processo ERSL foi sentida durante a fase de Separação do Código-Fonte. Isso ocorreu, pois, o repositório, em que os códigos-fonte são armazenados, é compartilhado por diversos sistemas de software, dificultando a localização da parte que pertencia ao software legado analisado. Para resolver esse problema, foi necessário utilizar uma ferramenta proprietária da companhia de seguros, a qual auxilia na localização das referências e dependências que um determinado arquivo de código-fonte possui. Uma ferramenta mais específica

de análise de código pode melhorar as atividades manuais que foram muito trabalhosas.

Outra dificuldade encontrada foi a ausência de comentários nos arquivos de código-fonte, dificultando a identificação de regras de negócio e também dificultando a aplicação da fase de Elaboração de Modelo de Metas.

5.6 CONSIDERAÇÕES DO CAPÍTULO

Nesse capítulo foram relatados os resultados obtidos com a aplicação do processo ERSL definido no capítulo 4.

Em geral, o processo ERSL pode ser considerado eficiente para elevar a qualidade dos requisitos elicitados através da sua aplicação, no contexto da engenharia reversa. Pode-se chegar a essa conclusão, pois a cada fase do processo o detalhamento dos requisitos é incrementado.

É importante, também, entender que o processo ERSL é um experimento de unificação de métodos da engenharia reversa com foco em software legado, podendo ser aprimorado principalmente com ferramentas que automatizam atividades que não dependem de análise ou interpretação humana.

6 CONSIDERAÇÕES FINAIS

Este capítulo descreve as conclusões obtidas com este trabalho, suas principais contribuições, as recomendações e os trabalhos futuros.

6.1 CONCLUSÕES

Neste trabalho foram apresentadas as dificuldades que as corporações encontram para realizar a manutenção em software legado e uma das estratégias para resolver esse problema é migrar o software legado para um novo sistema. Uma das atividades importantes na migração do software legado é a de elicitação de requisitos, a qual não é trivial nesse contexto.

A engenharia reversa é uma das técnicas mais aplicadas quando existe a necessidade de elicitar requisitos de um software legado e, baseado neste cenário, este trabalho apresentou o processo ERSL. Esse processo foi desenvolvido com foco em migração de software legado, porém pode ser aplicado para outros contextos, cujo o objetivo é elicitar requisitos por meio da engenharia reversa.

Com a aplicação do processo na obtenção de requisitos de um software legado real, pode-se concluir que o processo mostrou ser eficiente, pois sistematiza a análise de código fonte e eleva o nível de qualidade dos requisitos elicitados. Também pode ser considerado como um processo iterativo e incremental, pois o software legado é dividido em partes e o processo pode ser aplicado por partes, até que todos os requisitos do software legado tenha sido identificados.

O processo ERSL foi definida com todas as atividades realizadas de forma manual. Constatou-se que as atividades de extração de informações do código-fonte são trabalhosas e tediosas e as ferramentas para a sua automação seria de grande valia, deixando o analista para atividades que dependem de interpretação humana.

6.2 CONTRIBUIÇÕES

Na literatura é comum encontrar métodos específicos para a engenharia reversa para software legado; no entanto, eles possuem um objetivo em comum de

elicitar os requisitos do software legado, mas as abordagens são diferentes, pois depende para que os requisitos estão sendo elicitados.

A primeira contribuição deste trabalho consiste da seleção e estudo de três métodos distintos de engenharia reversa para software legado. A identificação das suas diferenças e semelhanças foi importante para sua melhor compreensão e esta comparação foi fundamental para a construção do processo ERSL, pois permitiu experimentar as diferentes abordagens de cada um deles.

A segunda contribuição está voltada à unificação de métodos, que possuem focos diferentes para realizar a engenharia reversa em software legado, em um processo que procura cobrir as eventuais deficiências de cada método com os pontos fortes de outros. Os requisitos obtidos, a partir da combinação de fontes e técnicas diversas, melhoraram o nível de qualidade dos requisitos elicitados.

Outra contribuição foi definir o processo ERSL no contexto da migração do software legado para um novo sistema de software, pois os estudos encontrados na literatura possuem uma abordagem voltada à migração do software legado, indicando somente a importância das atividades de elicitação de requisitos, não explorando como realizar essa atividade.

6.3 RECOMENDAÇÕES E TRABALHOS FUTUROS

O objetivo de escolher a engenharia reversa está sempre relacionado à manutenção de software que não possui documentação e, portanto, o trabalho é centrado no entendimento do código-fonte, para dele extrair os requisitos. Nesse contexto, as maiores dificuldades estão ligadas a como e em que sequência o código-fonte deve ser selecionado para agilizar a análise, que em geral não estão estruturados e nem possuem comentários.

Seguindo este raciocínio, os trabalhos futuros podem ser voltados aos estudos e métodos para selecionar e analisar corretamente o código-fonte dos programas de software. Outro trabalho interessante consiste em estudar e pesquisar sobre ferramentas que podem auxiliar e automatizar o processo proposto nesse trabalho.

REFERÊNCIAS

CHIKOFSKY, ELLIOT J.; CROSS, JAMES H. . Reverse Engineering and Design Recovery: A Taxonomy. **IEEE Software**, v. 7, n. 1, p. 13-17, 1990.

CHINOSI, MICHELE; TROMBETTA, ALBERTO. BPMN: An introduction to the standard. **Computer Standards & Interfaces**, v. 34, n. 1, p. 124-134, 2012.

GEET, JORIS V.; EBRAERT, PETER; DEMEYER, SERGE. Redocumentation of Legacy Banking System. In: ERCIM WORKSHOP ON SOFTWARE EVOLUTION (EVOL) AND INTERNATIONAL WORKSHOP ON PRINCIPLES OF SOFTWARE EVOLUTION (IWPSE), 10. , 2010, Bélgica. **Anais**... Bélgica: ACM, 2010. p. 33-41.

PRESSMAN, ROGER S. . **Engenharia de Software**: Uma Abordagem Profissional. Tradução de Ariovaldo Griesi; Mario Moro Fecchio, Revisão técnica de Reginaldo Arakaki; Julio Arakaki; Renato Manzan de Andrade. 7º ed. . Porto Alegre: AMGH Editora LTDA., 2011. 780 p.

SALVATIERRA, GONZALO et al. . Legacy System Migration Approaches. **IEEE Latin America Transactions**, v. 11, n. 2, p. 840-851, 2013.

SOMMERVILLE, IAN. **Engenharia de Software**. Tradução de Ivan Bosnic, Kalinka G. de O. Gonçalves, Revisão técnica de Kechi Hirama. 9º ed. . São Paulo: Pearson Prentice Hall, 2011. 530 p.

STROULIA, ELENI et al. . Reverse Engineering Legacy Interfaces: An Interaction-Divren Approach. In: WORKING CONFERENCE ON REVERSE ENGINEERING, 6. , 1999, Atlanta. **Anais**... Atlanta: IEEE, 1999. p. 292-302.

WU, BING et al. . Legacy System Migration: A Legacy Data Migration Engine. In: INTERNATIONAL DATABASE CONFERENCE, 17. , 1997, República Checa. **Anais**... República Checa: Czechoslovak Computer Experts, 1997. p. 129-138.

YU, YIJUN et al. . Reverse Engineering Goal Models from Legacy Code. In: IEEE INTERNATIONAL CONFERENCE ON REQUIREMENTS ENGINEERING (RE'05), 13. , 2005, Paris. **Anais**... Paris: IEEE Computer Society, 2005. p. 363-372.