LUÍS PAULO ALVES MAGALHÃES

ESTUDO SOBRE ENGENHARIA REVERSA E AVALIAÇÃO DA USABILIDADE DE FERRAMENTAS CASE PARA ENGENHARIA REVERSA DE SOFTWARE

Monografia de graduação apresentada ao Departamento de Ciência da Computação da Universidade Federal de Lavras como parte das exigências do curso de Ciência da Computação para obtenção do título de Bacharel em Ciência da Computação.

LAVRAS
MINAS GERAIS – BRASIL
2008

LUÍS PAULO ALVES MAGALHÃES

ESTUDO SOBRE ENGENHARIA REVERSA E AVALIAÇÃO DA USABILIDADE DE FERRAMENTAS CASE PARA ENGENHARIA REVERSA DE SOFTWARE

Monografia de graduação apresentada ao Departamento de Ciência da Computação da Universidade Federal de Lavras como parte das exigências do curso de Ciência da Computação para obtenção do título de Bacharel em Ciência da Computação.

Área de Concentração:

Engenharia de Software

Orientador:

Dr. Heitor Augustus Xavier Costa

LAVRAS
MINAS GERAIS – BRASIL
2008

Ficha Catalográfica preparada pela Divisão de Processo Técnico da Biblioteca Central da UFLA

Magalhães, Luís Paulo Alves

Estudo sobre Engenharia Reversa e Avaliação da Usabilidade de Ferramentas CASE para Engenharia Reversa de Software / Luís Paulo Alves Magalhães. Lavras – Minas Gerais, 2008. p. 64

Monografia de Graduação – Universidade Federal de Lavras. Departamento de Ciência da Computação.

1. Manutenção de Software. 2. Engenharia Reversa. 3. Ferramentas CASE de Engenharia Reversa. 4. Técnicas de Engenharia Reversa I. MAGALHÃES, L. P. A. II. Universidade Federal de Lavras. III. Estudo sobre Engenharia Reversa e Avaliação da Usabilidade de Ferramentas CASE para Engenharia Reversa de Software

LUÍS PAULO ALVES MAGALHÃES

ESTUDO SOBRE ENGENHARIA REVERSA E AVALIAÇÃO DA USABILIDADE DE FERRAMENTAS CASE PARA ENGENHARIA REVERSA DE SOFTWARE

Monografia de graduação apresentada ao Departamento de Ciência da Computação da Universidade Federal de Lavras como parte das exigências do curso de Ciência da Computação para obtenção do título de Bacharel em Ciência da Computação.

Aprovada em 17/11/2008

Prof. Dr. Antônio Maria Pereira de Resende

Prof. Dr. Valter Vieira de Camargo

Prof. Dr. Heitor Augustus Xavier Costa

(Orientador)

LAVRAS
MINAS GERAIS – BRASIL

"Mas aqueles que esperam no Senhor renovam as suas forças. Voam bem alto como águias; correm e não ficam exaustos, andam e não se cansam" Is.40:31

A Deus, à minha mãe Marta, ao meu pai Eloídes, ao meu irmão Gérson e ao meu tio Wellington dedico este trabalho, atribuindo-lhes significativos e insubstituíveis lugares em meu coração e expressando a minha imensa gratidão por todo amor, cuidado, companheirismo e contribuição para a minha formação. Vocês são muito especiais para mim!

AGRADECIMENTOS

Ao Deus Todo-Poderoso, em primeiro lugar, pelo seu imensurável amor demonstrado a cada dia através das infinitas bênçãos derramadas sobre a minha vida por graça e misericórdia.

Aos meus pais Eloídes e Marta, vidas que representam tudo para mim, dignos da minha eterna gratidão pelo carinho, pelo incentivo, pela confiança, pelo apoio em todos os momentos, enfim, pelo exemplo de vida dado, responsável pela formação do meu caráter.

Ao meu irmão Gérson, amigo verdadeiro, companheiro de todos os momentos, pronto para o que der e vier.

Ao meu tio Wellington, peça fundamental para o meu desenvolvimento como estudante. Sem esta mente brilhante no meu caminho, talvez eu não chegaria onde cheguei. Obrigado por todo apoio oferecido quando meus pais não estavam presentes. Existem coisas na vida que não há dinheiro que pague.

Aos meus amigos e irmãos de república: Adílio, Gabriel, Guilherme e Gustavo (O namorador). Foram mais que especiais os momentos que passamos juntos! Jamais me esquecerei. Muito obrigado pela mão estendida sempre que precisei. Foram vidas muito usadas por Deus para me ensinar muitas coisas. Cada um de vocês está guardado em um lugar bem especial no fundo do meu coração.

Ao grande companheiro Paulo Júnior (O Caboclo), sempre presente e disposto a ajudar. É com satisfação que digo: “O que seria de mim sem o Caboclo!?” Inúmeras foram as vezes que me estendeste a mão durante os apertos da faculdade. Mais que amigo, é uma pessoa muito especial cuja lembrança jamais sairá da memória. Obrigado por tudo, Caboclo! Também lhe digo que existem coisas nesta vida que não há dinheiro que pague.

SUMÁRIO

Lista de Figuras

Lista de Tabelas

1 INTRODUÇÃO 2
- **1.1 Motivação 2**
- **1.2 Objetivo 4**
- **1.3 Metodologia de Desenvolvimento 4**
  - **1.3.1 Tipo da Pesquisa 4**
  - **1.3.2 Procedimentos Metodológicos 4**
- **1.4 Estrutura da Monografia 5**

2 REENGENHARIA DE SOFTWARE 7
- **2.1 Considerações Iniciais 7**
- **2.2 Importância 8**
- **2.3 Atividades 9**
- **2.4 Considerações Finais 10**

3 ENGENHARIA REVERSA DE SOFTWARE 11
- **3.1 Considerações Iniciais 11**
- **3.2 Conceitos 11**
- **3.3 Níveis de Abstração 14**
- **3.4 Redocumentação 16**
- **3.5 Recuperação do Projeto 18**
- **3.6 Considerações Finais 19**

4 TÉCNICAS DE ENGENHARIA REVERSA 20
- **4.1 Considerações Iniciais 20**
- **4.2 Fusion/RE 20**
- **4.3 Análise de Estruturas do Código-Fonte 23**
- **4.4 Análise de Estruturas do Banco de Dados 24**
- **4.5 Considerações Finais 25**

5 FERRAMENTAS CASE DE ENGENHARIA REVERSA 26
- **5.1 Considerações Iniciais 26**
- **5.2 Modelagem do Código-Fonte 27**
- **5.3 Modelagem do Banco de Dados 30**
- **5.4 Considerações Finais 34**

6 USABILIDADE 35
- **6.1 Considerações Iniciais 35**
- **6.2 Conceitos 35**
- **6.3 Sub-Características de Usabilidade da Norma ISO/IEC 9126 ..... 37**

6.4 Testes de Usabilidade 37
6.5 Considerações Finais 38

7 USO DAS FERRAMENTAS CASE 39
7.1 Considerações Iniciais 39
7.2 O Software 39
7.3 Uso da Técnica Análise de Estruturas do Código-Fonte 39
7.4 Uso da Técnica Análise de Estruturas do Banco de Dados 40
7.5 Fusion-RE/I 43
7.5.1 Etapa 1 – Recuperação de Visões Funcionais 43
7.5.2 Etapa 2 – Recuperação de Visões Estruturais 47
7.6 Resultados 49
7.7 Considerações Finais 50

8 ANÁLISE COMPARATIVA DA CARACTERÍSTICA DE QUALIDADE DE USABILIDADE DAS FERRAMENTAS CASE . 51
8.1 Considerações Iniciais 51
8.2 Critérios de Avaliação das Ferramentas CASE 51
8.3 Avaliação das Ferramentas CASE 51
8.4 Considerações Finais 56

9 CONSIDERAÇÕES FINAIS 57
9.1 Conclusões 57
9.2 Contribuições 57
9.3 Trabalhos Futuros 58

REFERÊNCIAS BIBLIOGRÁFICAS 60

LISTA DE FIGURAS

Figura 3-1 – Relação entre Nível de Abstração, Interatividade e Completude (Fonte: Harandi; Ning (1990)) 16

Figura 3-2 – Relação entre Nível de Abstração, Interatividade e Completude. 16

Figura 4-1 – Visão Geral do *Fusion-RE/I* (Fonte: Feltrim (1999)) 21

Figura 4-2 – Ordem das Tarefas Prescritas pelo *Fusion-RE/I* (Fonte: Feltrim (1999)) 23

Figura 5-1 – Exibição de Diagrama de Classes na ArgoUML 28

Figura 5-2 – Exibição de Diagrama de Classes na Umbrello UML Modeller 28

Figura 5-3 – Exibição de Diagrama de Classes na IDE NetBeans 29

Figura 5-4 – Exibição de Diagrama de Classes na WithClass 30

Figura 5-5 – Modelagem do Banco de Dados Usando a DBConstructor 31

Figura 5-6 – Modelagem do Banco de Dados Usando a DBWrench 32

Figura 5-7 – Modelagem do Banco de Dados Usando a DBVisualizer 33

Figura 5-8 – Modelagem do Banco de Dados Usando a SQuirreL 33

Figura 6-1 – Subcaracterísticas de Usabilidade 37

Figura 7-1 – Diagrama de Classes do SisGAGRO com Atributos e Métodos e sem Relacionamentos 40

Figura 7-2 – Diagrama de Classes do SisGAGRO sem Atributos e Métodos 41

Figura 7-3 – Diagrama de Estados do SisGAGRO 42

Figura 7-4 – Diagrama de Atividade Cadastro de Fazenda do SisGAGRO 42

Figura 7-5 – Modelo do Banco de Dados do SisGAGRO 43

Figura 7-6 – Modelo de Ciclo de Vida do SisGAGRO 44

Figura 7-7 – Diagrama de Estados Representando as Sentenças Iniciais do Ciclo de Vida 45

Figura 7-8 – Descrição de Operação “Criar Hiperbase” (Fonte: Feltrim (1999)) 46

Figura 7-9 – Descrição de Operação “Criar Fazenda” do SisGAGRO..... 46

Figura 7-10 – Diagrama de Classes do Modelo de Objetos 47

Figura 7-11 – Modelo de Análise *Fusion-RE/I* e UML 47

Figura 7-12 – Diagrama de Componentes do Quadro de Chamadas 48

Figura 7-13 – Visões Estruturais *Fusion-RE/I* e UML..... 49

LISTA DE TABELAS

Tabela 7-1 – Lista de Procedimentos de Implementação de Insumos e Sêmen 48

Tabela 8-1 – Perguntas e Justificativas para a Sub-característica Inteligibilidade 52

Tabela 8-2 – Perguntas e Justificativas para a Sub-característica Atratividade..... 52

Tabela 8-3 – Perguntas e Justificativas para a Sub-característica Apreensibilidade . 53

Tabela 8-4 – Perguntas e Justificativas para a Sub-característica Operacionalidade 53

Tabela 8-5 – Resultado da Avaliação da Usabilidade das Ferramentas CASE 54

Tabela 8-6 – Reflexão dos Resultados Referentes às Ferramentas CASE para Engenharia Reversa do Código-Fonte 55

Tabela 8-7 – Reflexão dos Resultados Referentes às Ferramentas CASE para Engenharia Reversa do Banco de Dados 55

Estudo sobre Engenharia Reversa e Avaliação da Usabilidade de Ferramentas CASE para Engenharia Reversa de Software

RESUMO

O crescimento do mercado de software a cada dia acarreta o aumento do uso de técnicas de desenvolvimento, muitas vezes informais. A manutenção de tais software torna-se problemática, uma vez que a documentação associada ao software, na maioria das vezes, não está de acordo com o código implementado. Dessa forma, quando diante da manutenção do produto, o engenheiro de software encontra uma documentação informal e incompleta, que não reflete o software existente. Nesse contexto é que se encontra a Engenharia Reversa de Software, com o propósito de recuperar as informações de projeto perdidas ou não geradas durante a fase de desenvolvimento, e de documentar o real estado do software. As dificuldades passam a existir quando os sistemas apresentam problemas como: i) dificuldade de compreensão das regras de negócio; ii) problemas na estruturação do código; e iii) desconhecimento das razões que levaram a determinadas decisões. O objetivo deste trabalho é estudar e analisar a usabilidade de ferramentas CASE para realizar a engenharia reversa de software. Para isso, algumas técnicas de engenharia reversa foram pesquisadas e estudadas, bem como ferramentas CASE automatizadas que apóiem estas técnicas. Além disso, estas ferramentas CASE foram usadas em um software real para nortear a análise.

Palavras-chave: Engenharia de Software; Manutenção de Software; Engenharia Reversa.

Study on Reverse Engineering and Evaluation of the Usability of CASE Tools for Software Reverse Engineering

ABSTRACT

The growth of the software market has leading to an increasing use of development techniques, which are, sometimes, informal ones. The maintenance of such software is problematic, since its documentation rarely reflects the implemented code. In this context Reverse Engineering of Software can help by means of recovering the project information lost or not generated during the development phase and documenting the current software state. The difficulties begin to exist when the systems have problems such as: i) difficulty on understanding the rules of business, ii) problems in the code structuring; and iii) ignorance of the reasons that led to certain decisions. The purpose of this work is to study and analyze the usability of CASE tools to perform reverse engineering software. For this reason, some reverse engineering techniques were researched and studied, as well as automated tools that support these techniques. Moreover, these tools were used in an actual software to guide the analysis.

Keywords: Software Engineering, Software Maintenance, Reverse Engineering.

1 INTRODUÇÃO

Muitas empresas, instituições e organizações que fazem uso de sistemas computacionais para realizarem serviços essenciais enfrentam sérias dificuldades quando estes sistemas estão defasados. A necessidade de manutenção de um software ao longo do tempo é irrefutável. É utopia a confecção de um software perfeito que atenda a todas as necessidades e não precise em momento algum ser modificado [Lehman, 1985]. Seja por um *bug* ou por uma melhoria, a manutenção está presente no ciclo de vida do software. Além disso, o crescente avanço da tecnologia faz com que qualquer software necessite de manutenção, mesmo que as melhores e as mais atuais técnicas de análise, projeto e codificação tenham sido aplicadas.

A manutenção é uma das atividades mais importantes do ciclo de vida de um software. Não se justifica ter um software que não reflita a realidade do processo de negócio da organização. A manutenção é uma das atividades mais custosas da engenharia de software. De acordo com Pressman (2001), a manutenção de software é responsável por mais de 60 % do tempo de desenvolvimento de um software.

No contexto de software, que precisa ser alterado para que seu tempo de vida útil na organização em que está implantado seja estendido, há várias tecnologias que podem ser usadas. Para isso, há a necessidade dos mantenedores[1] terem pleno conhecimento da funcionalidade do software. Isso pode não ser trivial, uma vez que a documentação existente do software raramente é consistente com a sua funcionalidade atual; isso quando esta documentação existe [Costa, 2005].

Assim sendo, uma das técnicas empregadas é a engenharia reversa que consiste em realizar um processo de exame e de compreensão do software existente, para recapturar ou recriar o projeto (*design*) e decifrar os requisitos atualmente implementados, apresentando-os em nível ou grau mais alto de abstração e não envolvendo mudanças no software ou criação de software [Costa; Sanches, 1996].

1.1 Motivação

À medida que o mercado de software cresce, aumenta-se o uso de técnicas de desenvolvimento, muitas vezes informais. A manutenção de software torna-se

[1] Pessoas responsáveis pela tarefa de manutenção.

problemática, uma vez que a documentação associada ao software, na maioria das vezes, não está de acordo com o seu código-fonte. Nessas condições, diante da necessidade de manutenção, o desenvolvedor encontra uma documentação informal e incompleta, muitas vezes não refletindo o software existente.

Rugaber (1992) afirma que a maior parte do desenvolvimento de software é gasto na manutenção e não no desenvolvimento e grande parte do processo de manutenção é dirigida ao entendimento do software. Sendo assim, se é desejável melhorar o processo de manutenção de software, é necessário facilitar o processo de sua compreensão. A engenharia reversa aborda diretamente o problema de compreensão do software.

A atividade de manutenção consiste de três etapas [Schneidewind, 1987]: i) entendimento; ii) modificação; e iii) revalidação. Notadamente, as etapas de entendimento e de modificação estão relacionadas com a disponibilização de informações do software, ou seja, se apóiam na existência, na consistência, na completude e na atualização correta dos documentos que o compõem.

Nesse contexto, a engenharia reversa tem o propósito de recuperar as informações de projeto perdidas durante a fase de desenvolvimento e documentar o real estado do software. Ela é uma técnica usada para recuperar informações a partir do código-fonte do software, visando a obtenção de sua representação em um nível mais alto de abstração. Dessa forma, ela visa facilitar o entendimento do software, primordial para a sua manutenção. De acordo com Pressman (2001), manutenibilidade e inteligibilidade estão intimamente ligadas: é impossível realizar a manutenção em um software sem antes entendê-lo.

Realizar engenharia reversa com o auxílio de ferramentas CASE é necessário. Quesitos como ganho de tempo e isenção de erros de execução – características intrínsecas à automação de processo – são contemplados quando ferramentas CASE são utilizadas. Porém, a produtividade muitas vezes não é explorada suficientemente pela dificuldade de interação entre o usuário e a ferramenta CASE [Silveira, 1999].

Segundo ACM SIGHI (1992), uma interface bem definida entre o usuário e a ferramenta CASE é fundamental para que as suas funções sejam exploradas com facilidade. A interatividade, conceito abordado com mais detalhes no capítulo de

engenharia reversa, é extremamente importante quando se insere uma ferramenta no contexto da engenharia reversa. À medida que o nível de abstração aumenta, caso a interatividade seja expressiva, a completude e o entendimento do software estarão comprometidos. Desta forma, a manutenção do software será árdua.

Assim, o estudo da usabilidade de ferramentas CASE que apóiam o processo de engenharia reversa se faz necessário. Existem muitas ferramentas CASE no mercado, mas muitas delas deixam a desejar em quesitos de qualidade, por exemplo, a usabilidade.

1.2 Objetivo

O objetivo deste trabalho é fundamentar conceitos de Engenharia Reversa e estudar e analisar a usabilidade de ferramentas CASE (*Computer Aided Software Engineering*) para realizar a engenharia reversa de software. Para isso, algumas técnicas de engenharia reversa foram pesquisadas e estudadas, bem como ferramentas CASE que apóiem estas técnicas. Além disso, estas ferramentas CASE foram usadas em um software real para nortear a análise.

A análise da usabilidade foi realizada sob a luz da norma ISO/IEC 9126 [ISO/IEC 9126-1, 2001], que apresenta a definição da característica de usabilidade. O foco da análise foi em quatro de suas sub-catacterísticas: i) inteligibilidade; ii) apreensibilidade; iii) operacionalidade; e iv) atratividade.

1.3 Metodologia de Desenvolvimento

1.3.1 Tipo da Pesquisa

Conforme Jung (2004) e Marconi; Lakatos (2003) e observando o método científico, tem-se que a presente pesquisa é classificada quanto à natureza como tecnológica, quanto aos objetivos como explicativa, quanto aos procedimentos como operacional, quanto ao local como de laboratório, quanto à base teórica como documental e quanto ao tempo transversal e longitudinal.

1.3.2 Procedimentos Metodológicos

O presente trabalho foi realizado no período de março de 2008 a outubro de 2008, iniciando-se por um levantamento bibliográfico, na internet e em bibliotecas digitais e impressas, de artigos científicos relacionados ao tema.

Os primeiros esforços deste trabalho foram estudar o conceito, a história e a importância da engenharia reversa. A busca da origem do termo, a história e a aplicabilidade da engenharia reversa nas áreas industriais e comerciais a contextualizou na manutenção de software. A importância da engenharia reversa para os engenheiros de software foi apresentada, diante da necessidade de fazer manutenção do software.

Em seguida, foram estudadas técnicas de engenharia reversa. Estas técnicas foram pesquisadas e estudadas com o objetivo de melhorar o processo de redocumentação do software, que tem como finalidade produzir meios que facilitem o seu entendimento.

Foi feita uma análise de ferramentas CASE de engenharia reversa. Baseado nas informações obtidas na análise de estrutura e no funcionamento do software, algumas ferramentas CASE que realizam a engenharia reversa foram apresentadas e brevemente demonstradas.

Realizou-se um estudo de caso sobre um software real. Este estudo de caso foi realizado, aplicando técnicas de engenharia reversa sobre o software, buscando identificar e justificar qual a melhor técnica a ser utilizada.

Analisou a usabilidade de algumas ferramentas CASE de engenharia reversa. Tendo como característica de qualidade a usabilidade, foram elaboradas questões de acordo com as suas sub-características definidas na norma ISO/IEC 9126 [ISO/IEC 9126-1, 2001]. Por fim, foi feita a valoração das ferramentas CASE de engenharia reversa. Uma avaliação das ferramentas CASE foi proposta, conforme a resposta das questões elaboradas.

1.4 Estrutura da Monografia

Este trabalho está organizado da seguinte forma.

O capítulo 2 apresenta breve revisão de literatura sobre reengenharia de software.

O capítulo 3 apresenta breve revisão de literatura sobre engenharia reversa de software.

O capítulo 4 apresenta sucintamente três técnicas de engenharia reversa.

O capítulo 5 apresenta oito ferramentas CASE de engenharia reversa de código-fonte e de banco de dados.

O capítulo 6 apresenta breve revisão de literatura sobre usabilidade.

O capítulo 7 apresenta o uso das ferramentas CASE abordadas no capítulo anterior em um software real.

O capítulo 8 apresenta uma análise da usabilidade das ferramentas CASE de engenharia reversa, cujo foco foi em quatro sub-características de usabilidade: inteligibilidade, apreensibilidade, operacionalidade e atratividade.

O capítulo 9 apresenta algumas conclusões e contribuições e sugere alguns trabalhos futuros.

2 REENGENHARIA DE SOFTWARE

2.1 Considerações Iniciais

Confeccionar um software perfeito que atenda a todas as necessidades e não precise em momento algum ser modificado é algo impossível. Seja por um bug ou por uma melhoria, o ciclo de vida do software tem a manutenção como atividade sempre presente. Construir um software aplicando as melhores técnicas de projeto e codificação não é suficiente para deixar o software isento da necessidade de manutenção, em vista do crescente avanço da tecnologia.

De acordo com Sommerville (2001), a reengenharia de software se ocupa de reimplementar software legado, para que sua manutenção seja mais fácil. A reengenharia pode envolver a redocumentação, a reorganização e a reestruturação do software, a tradução do software para uma linguagem de programação mais moderna e a modificação e a atualização da estrutura e dos valores dos dados do software. A funcionalidade do software não é modificada e, normalmente, a sua arquitetura permanece a mesma.

O entendimento do software é primordial para a sua manutenção. Pressman (2001) aponta que a manutenibilidade e a inteligibilidade estão intimamente ligadas: é impossível realizar manutenção em um software sem antes entendê-lo.

A engenharia reversa é uma das tarefas a serem realizadas durante a reengenharia. De acordo com Benedusi et al. (1992), pode-se definir engenharia reversa como uma coleção de teorias, metodologias e técnicas capazes de suportar a extração e a abstração de informações de um software existente, produzindo documentos consistentes, quer seja a partir somente do código-fonte ou por meio da adição de conhecimento e da experiência que não podem ser automaticamente reconstruídos a partir do código.

A seção 2.2 trata a importância da reengenharia na engenharia de software, justificando e mostrando sua finalidade, o que ela deve contemplar e como ela o faz. A seção 2.3 aborda alguns conceitos de reengenharia, bem como as atividades que a compreendem.

2.2 Importância

Não se justifica ter um software que não reflita a realidade do processo de negócio da organização. Assim sendo, a manutenção é uma das atividades mais importantes do ciclo de vida de um software e a mais custosa da engenharia de software. De acordo com Pressman (2001), a manutenção de software é responsável por mais de 60% do tempo de desenvolvimento de um software.

A manutenção de software é dificultada muitas vezes pela falta e/ou desatualização da documentação do software. Geralmente, isso ocorre pelo fato do software ter sido construído há tempos (de maneira ad hoc e sem a preocupação de elaborar o mínimo de documentação) ou pelo fato das atualizações na documentação não terem sido realizadas, à medida que ocorreram modificações no software. A manutenibilidade, caracterizada principalmente pelo entendimento do software, está fortemente relacionada à disponibilidade de documentação do software.

A realidade é o software tem sido desenvolvido sem a preocupação desejável com as fases subseqüentes, dificultando as atividades posteriores. Segundo Pfleeger (2001), os documentos gerados durante o processo de desenvolvimento de software são importantes por permitir a redução de diversos problemas na manutenção.

A produção de software subsiste com uma constante necessidade de mudanças. A dificuldade de realizar manutenção em software não é um problema novo. A facilidade de manutenção, colocada em segundo plano por décadas, é a ênfase da engenharia reversa [Schneider, 2001].

Neste contexto, a reengenharia vem como peça fundamental para contornar esse problema, mais especificamente uma de suas atividades, a engenharia reversa. Segundo Schneider (2001), a reengenharia é o exame de um software para reconstituí-lo em uma forma nova e a implementação subseqüente dessa nova forma.

O principal propósito da reengenharia é a busca por melhorias que permitam produzir algo de melhor qualidade ou, pelo menos, de qualidade comparável ao produto inicial. Ela consiste na reorganização e na modificação do software com objetivo de torná-lo mais fácil de manter.

2.3 Atividades

De acordo com os autores Novais; Prado (2001), a reengenharia de software permite obter o entendimento do domínio da aplicação, recuperando as informações das etapas de análise e de projeto e organizando-as de forma coerente e reutilizável. A reengenharia de software abrange um conjunto de atividades [Pressman, 2001]:

- Análise de Inventário. Esta atividade diz respeito à avaliação sistemática de cada aplicação, com o objetivo de identificar quais são as aplicações candidatas à reengenharia. A análise de inventário consiste em realizar levantamento das informações sobre as aplicações, não importando o meio onde estejam. Além disso, ela deve fornecer uma descrição detalhada sobre cada aplicação ativa, classificando as informações colhidas de acordo com a sua importância para o negócio, longevidade, estado de manutenção e algum outro critério local importante que possa servir como base para avaliar a necessidade de uma reengenharia;
- Reestruturação de Documentos. Documentação desatualizada ou inexistente é uma característica que software em funcionamento possui há muito tempo. Esta atividade cria um conjunto de documentos necessários para o suporte de longo prazo de uma aplicação. A reconstrução completa da documentação é de vital importância ao entendimento do software;
- Engenharia Reversa. A finalidade desta atividade é reverter um software em suas definições mais abstratas de desenvolvimento com o objetivo de compreender, como funciona e como não funciona para poder ser modificado de acordo com as necessidades apontadas pela reengenharia. A engenharia reversa realiza o processo de análise de um software, em um esforço de extrair informação de projeto de dados, arquitetural e procedimental;
- Reestruturação de Código. Apesar de ter uma arquitetura relativamente sólida, software legado pode ter módulos codificados de tal maneira que o torna difícil de ser entendido, testado e mantido. Transgressões da programação devem ser anotadas, analisadas e reestruturadas. Para evitar que haja alguma irregularidade causada pela mudança de estrutura, testes devem ser realizados e a documentação deve ser atualizada;
- Reestruturação de Dados. Diferentemente da reestruturação de código, que ocorre em um nível relativamente baixo de abstração, a reestruturação de dados é uma atividade de reengenharia de escala plena. Na maioria dos casos, a reestruturação de dados começa com uma atividade de engenharia reversa. A arquitetura de dados atual é

dissecada e os modelos de dados necessários são definidos. A reestruturação de dados realiza a atualização dos dados em uma nova arquitetura ou estrutura seguindo os princípios mais atuais, porém conservando a mesma funcionalidade;

- Engenharia Avante (*Forward Engineering*). Esta atividade parte de uma abstração de alto nível de implementação lógica independente para a implementação física do software. Uma seqüência de requisitos de projeto para implementação é seguida. A engenharia avante não apenas recupera informação de projeto de software existente, mas usa essa informação para alterá-lo ou reconstituí-lo em um esforço para aperfeiçoar sua qualidade global.

2.4 Considerações Finais

A reengenharia é a área da engenharia de software responsável por possibilitar a reconstrução do software sem alterar a sua funcionalidade. Isto é, a partir da realização de algumas atividades, é possível obter um software reengenheirado, por exemplo, a documentação atualizada que serve de auxílio para a sua manutenção. Cabe ressaltar a integridade da funcionalidade durante o processo de reengenharia, o que significa a não alteração de quaisquer regras de negócio ou mesmo das funções do software.

A engenharia reversa, uma das atividades da reengenharia, é o foco deste trabalho. Assim, técnicas de engenharia reversa, ferramentas CASE (que implementam estas técnicas) e o uso destas ferramentas CASE em um software real são apresentados nos próximos capítulos.

3 ENGENHARIA REVERSA DE SOFTWARE

3.1 Considerações Iniciais

Concomitantemente ao crescimento do mercado de software, percebe-se o crescente uso de técnicas de desenvolvimento software, muitas vezes informais, acarretando problemas na sua manutenção e no seu uso. Além disso, modificações e adições de novas características produzem efeitos colaterais inesperados, não previstos na documentação, que acaba se tornando informal e incompleta, não refletindo o software existente e dificultando o seu gerenciamento [Jabur, 2007].

A engenharia reversa é um ramo da engenharia de software responsável por possibilitar a recuperação de informações perdidas ao longo do desenvolvimento do software. De acordo com Chikofsky; Cross II (1990), a engenharia reversa pode ser definida como o processo de análise para identificar seus componentes e seus inter-relacionamentos e criar suas representações em outra forma ou em um nível mais alto de abstração.

A engenharia reversa é o inverso da engenharia progressiva. A engenharia progressiva parte de uma abstração para chegar a uma implementação, enquanto a engenharia reversa parte de um código-fonte existente e recria modelos perdidos, não criados ou não modificados de acordo com alterações feitas no código.

A seção 3.2 aborda alguns conceitos relacionados à engenharia reversa considerados importantes para melhor compreensão. A seção 3.3 apresenta os níveis de abstração, isto é, formas de representação do software. A seção 3.4 trata de redocumentação, parte da engenharia reversa que visa criar novas visões do software por meio da análise do código-fonte, com o objetivo de melhorar a sua compreensão. A seção 3.5 versa sobre recuperação de projeto, parte da engenharia reversa que visa recuperar informações necessárias para melhor compreensão do que o software faz, como ele faz e por que ele o faz.

3.2 Conceitos

Várias são as definições de engenharia reversa presentes na literatura. A seguir, são apresentadas algumas:

- Processo de exame e de compreensão do software existente, para recapturar ou recriar o projeto e decifrar os requisitos atualmente implementados, apresentando-os em grau ou nível mais alto de abstração. Não envolvem mudanças no software ou criação de software [Chikofsky; Cross II, 1990];
- Processo de análise do esforço em criar uma representação do software em nível de abstração mais alto que o código-fonte [Pressman, 2001];
- Coleção de teorias, de métodos e de técnicas capazes de apoiar: i) o projeto e a implementação de um processo para extrair e abstrair informações do software existente e produzir documentação consistente com o código; e ii) a adição de conhecimentos e de experiências à documentação, que não podem ser automaticamente reconstruídas a partir do código [Benedusi *et al*., 1992];
- Processo de análise de um software para identificar os seus componentes e os inter-relacionamentos destes componentes para criar uma representação do software em outra forma, em um nível mais alto de abstração que o código-fonte [Waters; Chikofsky, 1994];
- Processo de engenharia para entender, analisar e abstrair o software para uma nova forma em alto nível de abstração [Stephen; Lynn, 1995];
- Processo por meio do qual um software é examinado para identificar ou especificar a definição em nível de sistema, em nível de requisitos ou em nível de projeto [Sage, 1995];
- A engenharia reversa é o processo de analisar o software com o objetivo de recuperar o seu projeto e a sua especificação. Se o código-fonte estiver disponível, ele é a entrada para o processo de engenharia reversa. Caso contrário, a engenharia reversa precisa começar com o código executável [Sommerville, 2001];
- A engenharia reversa é uma técnica usada para recuperar informações a partir do código-fonte, visando a obtenção de sua representação em nível mais alto de abstração. Ela se destina a criar visões do software em diferentes níveis de abstração, facilitando o entendimento com o principal objetivo de ajudar na manutenção [Feltrim, 1999].

Enfim, o objetivo da engenharia reversa é derivar o projeto ou a especificação de um software a partir de seu código-fonte. A engenharia reversa é usada durante o processo de reengenharia a fim de recuperar o projeto (design) que os engenheiros usam para a

compreensão antes de reorganizar sua estrutura. Contudo, a engenharia reversa não precisa ser sempre seguida da reengenharia [Sommerville, 2001]:

- O projeto e a especificação de um software existente podem passar por engenharia reversa, de modo que sirvam como entrada à especificação de requisitos, para a substituição desse software;
- Como alternativa, o projeto e a especificação podem passar por engenharia reversa, de modo que estejam disponíveis para ajudar na manutenção do software. Com essas informações adicionais, pode não ser necessário fazer a reengenharia do código-fonte.

Conforme Oman (1990) e Chikofsky; Cross II (1990), mediante o nível de entendimento do software e o escopo das informações usadas, duas categorias de engenharia reversa são definidas:

- Visualização de código. Esta categoria consiste na criação ou na mudança de representações semanticamente equivalentes dentro de um mesmo nível de abstração, dando ênfase à criação de visões gráficas. O objetivo é melhorar o entendimento do software criando representações a partir de dados coletados do código-fonte. O processo de visualização de código, também denominado redocumentação, cria as representações a partir de informações obtidas apenas da análise do código fonte, embora a representação dessas informações possa se diversificar. As formas das representações são consideradas visões alternativas, cujo objetivo é melhorar a compreensão do sistema global. A forma mais simples e mais antiga de engenharia reversa é a visualização de código. A intenção é recuperar a documentação que existiu ou que deveria ter existido sobre o software. A visualização de código não transcende a visão em nível estrutural e não atribui significados ao software analisado. Recuperações mais ambiciosas como a função, os propósitos ou a essência do software exigem nível de entendimento maior e são definidas como entendimento do código-fonte;
- Entendimento do programa. Esta categoria visa a recuperações mais complexas, como função, propósitos ou essência do software definidos como o entendimento do código-fonte. A partir de uma combinação de código-fonte, documentos existentes, experiências pessoais e conhecimentos gerais do problema, recriam-se abstrações de projeto. Essa categoria se difere da visualização de código, pois ela tem o objetivo de compreender o software, ao invés de fornecer visões para auxiliar o usuário a

entendimento o software [Lucas *et al.*, 2005]. No entendimento do código-fonte, também denominado recuperação de projeto, o conhecimento do domínio das informações externas e deduções é adicionado às observações feitas sobre o software, examinando-o, de modo a obter informações com nível mais alto de abstração. Sintetizando, devem ser produzidas as informações necessárias para entender o que, como e por que o software faz [Biggerstaff, 1989].

Há alguns pontos a serem considerados ao realizar engenharia reversa em um software. A caracterização de um bom processo é verificada com um bom produto mediante os objetivos traçados no início do processo. Em outras palavras, alguns tópicos, como níveis de abstração, completude e interatividade, devem ser avaliados. O entendimento, envolvido na categoria entendimento do progama, vai além do conhecimento em nível implementacional e estrutural, pois se busca obter o conhecimento em nível funcional e em nível de domínio. Um completo entendimento de códgio-fonte busca reconstruir a função do software e o processo pelo qual o software foi desenvolvido. Rugaber et al. (1990) enfatizam a importância da recuperação de decisões de projeto tomadas durante o desenvolvimento original para uma completa estrutura de entendimento. A categoria de entendimento do programa é a forma mais crítica de engenharia reversa, porque ela tenta aproximar do raciocínio humano na busca de entendimento.

3.3 Níveis de Abstração

Existem várias formas de representar o software que podem ser extraídas do código-fonte. Algumas são de mais alto nível, como modelos de fluxo de dados e de controle. Conforme Pressman (2001), o nível de abstração de um processo de engenharia reversa e as ferramentas usadas para executá-lo referem-se à sofisticação da informação de projeto. Os níveis de abstração, na visão deste autor, são:

- Representações do projeto procedimental (abstração de baixo nível);
- Informação do software e da estrutura de dados (nível de abstração mais alto);
- Modelos de fluxo de dados e de controle (nível de abstração relativamente alto);
- Modelos entidade-relacionamento (alto nível de abstração).

À medida que o nível de abstração aumenta, informações são fornecidas ao engenheiro de software, permitindo entendimento mais fácil do software. A partir da

engenharia reversa e com base nos diferentes níveis e graus de abstração, o software pode ser visualizado de diferentes maneiras [Harandi; Ning, 1990]:

- Nível implementacional: compreende a abstração das características da linguagem de programação e as características específicas da implementação;
- Nível estrutural: detalhes da linguagem de programação são abstraídos para revelar sua estrutura a partir de diferentes perspectivas. O resultado é uma representação explícita das dependências entre os componentes do software;
- Nível funcional: abrange a abstração da função de um componente. Essa visão relaciona partes do software às suas funções, revelando as relações lógicas entre elas;
- Nível de domínio: abstrai o contexto no qual o software está operando.

É relevante ressaltar que uma forma de representação extraída do código pode diferir de uma representação similar que foi desenvolvida no processo de engenharia progressiva. A forma extraída refletirá a idiossincrasia da representação do código muito mais do que a representação original, pois esta reflete a compreensão do problema pelo analista/projetista [Costa; Sanches, 1996].

Existe uma diferença básica entre nível de abstração e grau de abstração. O nível de abstração é inerente a cada estágio do software. Nos estágios iniciais do ciclo de vida, as informações possuem alto nível de abstração; nos estágios finais, as informações possuem baixo nível de abstração. Cada fase no processo de desenvolvimento de software é um refinamento do nível de abstração do software. Quanto mais alto for o nível de abstração, mais fácil será entender o software como um todo, pois o nível de abstração diz respeito a estágios iniciais do ciclo de vida e fornece uma visão global do software. A Figura 3-1 ilustra a relação entre nível de abstração e o processo de desenvolvimento de software. O grau de abstração, por sua vez, diz respeito à quão fácil é entender/abstrair o software e está relacionado a uma mesma atividade no seu ciclo de vida. Informações em uma forma mais global possuem alto grau de abstração e, em uma forma mais detalhada, informações possuem baixo grau de abstração. Em outras palavras, o grau de abstração está relacionado ao nível de detalhe das informações [Penteado, 1999].

Em um processo de desenvolvimento de software, os estágios iniciais envolvem conceitos mais gerais, mentos detalhados, independentes da implementação, enquanto os estágios finais enfatizam os detalhes de implementação. O aumento de detalhes durante o

processo de desenvolvimento conceitua os níveis de abstração. Nos estágios iniciais do software, requisitos de alto nível são planejados e definidos, quando comparados à própria implementação.

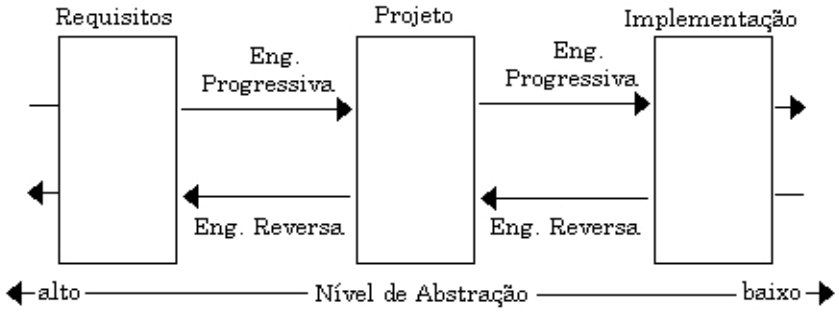

Figura 3-1 – Relação entre Nível de Abstração, Interatividade e Completude (Fonte: Harandi; Ning (1990))

Há dois conceitos correlacionados ao nível de abstração [Pressman, 2001]:

- Completude. Quanto mais alto for o nível de abstração da representação, menor é a completude. A completude está relacionada o quão detalhada é a representação. Completude de um processo de engenharia reversa refere-se ao nível de detalhe fornecido em um nível de abstração;
- Interatividade. A interação entre as pessoas e as ferramentas CASE para efetuar a engenharia reversa é importante. A interatividade se refere ao grau em que uma pessoa interage com ferramentas CASE automáticas para criar um processo efetivo de engenharia reversa. Na maioria dos casos, à medida que o nível de abstração aumenta, a interatividade precisa aumentar ou a completude será prejudicada.

A Figura 3-2 mostra que, enquanto o nível de abstração aumenta, se a interatividade fica um pouco a desejar (seta vermelho claro) a completude tende a cair significativamente (seta vermelho escuro). Por outro lado, para que a completude tenha um pequeno aumento (seta verde claro), a interatividade deve ter um aumento expressivo (seta verde escuro).

Figura 3-2 – Relação entre Nível de Abstração, Interatividade e Completude.

3.4 Redocumentação

A redocumentação visa criar novas visões do software por meio da análise do código-fonte, com o objetivo de melhorar a sua compreensão. A criação dessas visões

adicionais do código-fonte, geralmente gráficas, tem o objetivo de recriar a documentação que existiu ou que deveria ter existido do software [Feltrim, 1999]. A redocumentação, conhecida como visualização de código, é a criação ou a revisão de uma representação da abstração semântica do software.

A redocumentação pode ser vista como uma parte da engenharia reversa, mas de forma mais simplificada. A redocumentação envolve a análise estática do código-fonte para a produção de uma documentação, sendo examinados o uso de variáveis, as chamadas de componentes, os caminhos de controle, o tamanho do componente, os parâmetros de chamada, os caminhos de teste e alternativas que ajudem no entendimento do que/como o código-fonte realiza [Pfleeger, 2001]. A redocumentação não busca recuperar modelos de projeto de forma automática ou semi-automática, diferentemente da engenharia reversa.

A redocumentação deve ser feita de forma sistemática. Segundo Anquetil; Oliveira (2002), processos de redocumentação precisam ser estabelecidos de forma a identificar por onde começar a redocumentação, quais documentos devem ser gerados e úteis ao desenvolvimento e o que, como e por quem deve ser realizados.

Tendo como verdade que a redocumentação é essencial para a manutenção de software, ela é uma necessidade. Sabendo que a documentação é considerada nos processos atuais como parte do desenvolvimento ou como manutenção de documentos existentes, Anquetil; Oliveira (2002) definem um processo redocumentação que garante gerar documentação suficiente para apoiar a atividade de manutenção. Para a definição do processo, são consideradas três características básicas:

- Ser baseado na engenharia reversa. O processo de redocumentação deve ser baseado em uma abordagem *bottom-up* (a partir do código-fonte), não devendo ser realizadas novamente as fases de desenvolvimento;
- Gerar a documentação mínima necessária. Para diminuir os custos de redocumentação e maximizar as chances da documentação gerada ser sempre mantida atualizada, Anquetil; Oliveira (2002) seguiram a redocumentação definida em Pressman (2001): limitar a redocumentação ao mínimo necessário;
- Buscar automação quando possível. Tentar definir artefatos no processo que possam ser gerados automaticamente e semi-automaticamente desde que mantenham informação de valor para manutenção.

Anquetil; Oliveira (2002) dividem o processo de redocumentação em três grandes fases e estas divididas em atividades:

- Fase de Preparação: consiste no levantamento das informações sobre o software a ser documentado. A partir dos dados coletados, é feito o planejamento do que será documentado. Essa fase é composta de duas atividades: i) Inventariar o Software; e ii) Auditar o Software;
- Fase de Planejamento: consiste no planejamento do que será redocumentado, indicação de quem fará a documentação e definição do cronograma para as atividades a serem realizadas. Esta fase é composta por uma atividade: i) Planejar a Redocumentação;
- Fase de Redocumentação: consiste na redocumentação propriamente dita. Esta fase é composta de quatro atividades: i) Definir Visão de Alto Nível; ii) Gerar Referências Cruzadas; iii) Definir Subsistemas; e iv) Gerar Documentação de Baixo Nível.

3.5 Recuperação do Projeto

Em um processo de reengenharia, a fase mais árdua é a engenharia reversa, pois a obtenção de documentação e a recuperação de um projeto de software, partindo de um código-fonte, é mais árdua que aplicar a engenharia avante (forward engineering) com o projeto recuperado e documentado [Peres et al., 2003]. Peres et al. (2003) propõem um conjunto de padrões de processo para a engenharia reversa baseada em transformações, cuja contribuição é a obtenção de um projeto recuperado do software legado em alto nível de abstração, representado em UML (Unified Modeling Language) [UML, 2008]. Desta forma, garante-se a evolução e a manutenibilidade do software, tornando-o mais expressivo e de fácil entendimento. Esses padrões usam transformações que agilizam o processo de engenharia reversa e facilitam a obtenção da documentação do projeto.

A modelagem é a referência das atividades de um projeto e o seu uso levará à construção de software de qualidade [Booch et al., 2000]. A partir dela, consegue-se visualizar a arquitetura e compreender, simplificar, reaproveitar e gerenciar riscos. Sem uma modelagem, o processo de desenvolvimento pode ter um custo maior se o escopo do projeto crescer, pois não haverá documentação para auxiliar futuras modificações.

A recuperação do projeto visa resgatar informações necessárias para compreender melhor o que o software faz, como ele faz e por que ele faz. A recuperação do projeto, conhecida como entendimento do programa, é a adição do domínio de conhecimento e

informações externas para identificar abstrações de alto nível no software, além daquelas obtidas diretamente pelo exame do software.

De acordo com Veronese et al. (2002), a engenharia reversa atua no auxílio à recuperação da documentação e ao entendimento do software, possibilitando que sua manutenção seja realizada de forma menos árdua. Conforme Feltrim (1998), a recuperação de projeto possibilita melhor entendimento do software por parte do engenheiro de software.

3.6 Considerações Finais

Uma das fases mais importantes da reengenharia é a engenharia reversa. Ela é responsável por gerar a documentação do software legado, partindo do software pronto, em estado final, suscetível apenas a atividades de manutenção. A engenharia reversa é uma técnica usada para recuperar informações a partir dos documentos do software relativos ao seu código-fonte, visando a sua representação em nível mais alto de abstração.

Algumas técnicas e métodos são usados para realizar a engenharia reversa. No próximo capítulo, são relatadas algumas técnicas de engenharia reversa para a geração da documentação.

4 TÉCNICAS DE ENGENHARIA REVERSA

4.1 Considerações Iniciais

Este capítulo apresenta sucintamente as técnicas *Fusion/RE*, análise de estruturas do código-fonte e análise de estruturas do banco de dados mais comumente usadas para realizar a engenharia reversa.

A seção 4.2 apresenta breve descrição de *Fusion/RE*. A seção 4.3 apresenta breve descrição da análise de estruturas do código-fonte. A seção 4.4 apresenta breve descrição da análise de estruturas do banco de dados.

4.2 Fusion/RE

Uma técnica que se destaca em trabalhos de reengenharia, como os de Martins et al. (2002), Novais (2002), Bossonaro (2008), Jesus (2000), Fukuda (2000) e Penteado (1996), é a técnica de engenharia reversa *Fusion/RE*. Esta técnica é usada para obter o entendimento e revitalizar a estrutura do código legado, segundo o paradigma orientado a objetos, visando reutilizar a funcionalidade deste código na reconstrução do software.

Na visão de Penteado; Lemos (1999), o *Fusion/RE* foi criado com o objetivo de recuperar o projeto de software construído com o paradigma procedimental, reconstruindo-o usando o paradigma de orientação a objetos. Esta reconstrução envolve seis passos concernentes ao processo de reengenharia, contudo apenas os quatro primeiros passos referem-se à engenharia reversa. Resumidamente, estes quatro passos são:

1. Revitalizar a arquitetura do software legado. Recuperar a documentação básica do software, baseada na documentação disponível, com a elaboração da lista “X chama/chamado por X”;
2. Recuperar o Modelo de Análise do Sistema Atual (MASA). A partir das bases de dados e do código-fonte, é criado um pseudo-modelo orientado a objetos do software legado. Muitos procedimentos podem conter anomalias, ou seja, um mesmo procedimento pode usar várias estruturas de dados;
3. Criar o Modelo de Análise do Sistema (MAS). Os diagramas do MASA são abstraídos, as pseudo-classes do MASA são generalizadas e as anomalias dos procedimentos são eliminados;

4. Mapear o MAS para o MASA. Classes, atributos e procedimentos do MAS são mapeados para elementos correspondentes do MASA.

De acordo com Feltrim (1999), o *Fusion-RE/I* (*Fusion* – Reverse Engineering/Interface) é uma técnica de engenharia reversa que, visando facilitar o processo, parte da interface do software para a recuperação de informações úteis à sua manutenção. Desta forma, é possível recuperar visões funcionais e estruturais do software [Costa, 1997]. A Figura 4-1 mostra uma visão geral do *Fusion-RE/I*, representando os elementos requeridos e os obtidos na realização do processo.

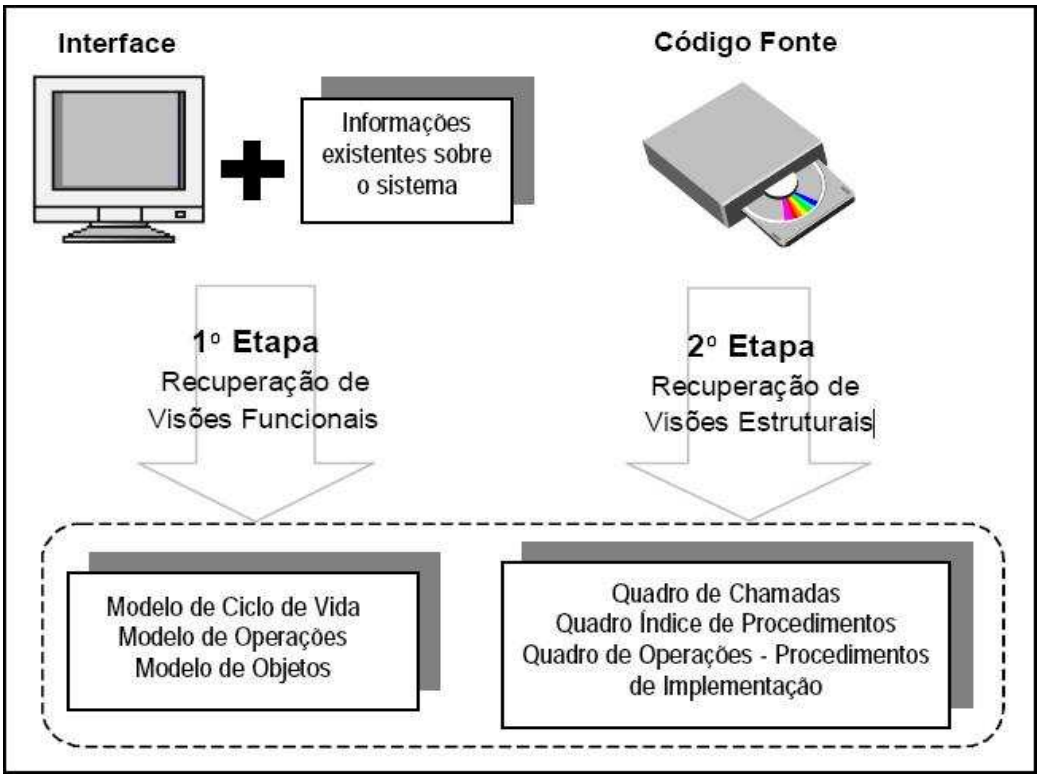

Figura 4-1 – Visão Geral do *Fusion-RE/I* (Fonte: Feltrim (1999))

O *Fusion-RE/I* foi criado na tentativa de reunir as melhores técnicas existentes e compreende duas fases distintas [Feltrim, 1999] (Figura 4-2):

- Recuperação de Visões Funcionais (Etapa 1). Visando obter a abstração da funcionalidade do software, em uma primeira etapa do *Fusion-RE/I*, são realizadas duas tarefas [Feltrim, 1999]:
    - Obter informações existentes sobre o software: busca a informação disponível sobre o software em análise. Isso envolve reunir a documentação existente (manuais, listagem de código, etc.) e obter informações relevantes, como o domínio do software e a linguagem de implementação. Entrevistas com os

usuários podem ser úteis, pois, muitas vezes, informações importantes podem não estar documentada. Tendo reunido esta documentação, ela deve ser analisada para identificar informações relacionadas aos requisitos do software, ao projeto arquitetural, de dados e procedimental, ao ambiente onde o software é executado, à organização dos arquivos em disco, etc;

- Recuperar modelo de análise do software: após obter as informações sobre o software, a tarefa de recuperar informações da fase de análise é iniciada. As informações recuperadas são obtidas, investigando a interface do software. Segundo Costa (1997), a tarefa de recuperação do modelo de análise compreende a elaboração dos três modelos da fase de análise do *Fusion-RE/I*: i) de ciclo de vida; ii) de operações; e iii) de objetos. Pressupõe-se que essa tarefa exija grande esforço, em função da complexidade do software;

- Recuperação de Visões Estruturais (Etapa 2). Nesta etapa, o código-fonte do software legado é usado para identificar procedimentos que implementam as operações do software discriminadas na etapa anterior. As atividades a serem realizadas são:
    - Elaborar quadro de procedimentos de implementação: o objetivo é identificar cada procedimento, sua funcionalidade e a seqüência de chamadas desse procedimento. Para tal, dois quadros são utilizados: i) quadro de chamadas para cada arquivo de código-fonte do software; e ii) quadro geral de procedimentos;
    - Elaborar quadro das operações – procedimentos de implementação: os procedimentos que implementam as operações da interface são identificados e, de acordo com a sua funcionalidade, eles são alocados à interface ou a um dos temas definidos anteriormente. Em seguida, são identificados os *links* entre os documentos da primeira etapa do método com os respectivos códigos-fonte que os implementam. Nas primeiras colunas do quadro, são colocadas as opções do *menu* e as operações de cada opção (descrição da interface). Na próxima coluna, são colocados os procedimentos que implementam cada operação, de acordo com a hierarquia de chamadas descrita no quadro de chamadas. As próximas colunas do quadro são usadas para alocar os procedimentos à interface ou a um dos temas definidos. Assim, tem-se uma coluna para interface e uma para cada tema definido. O nível de profundidade

com que os procedimentos são detalhados nesse quadro depende do interesse em questão.

Etapa 1. Recuperação de Visões Funcionais
a. Obtenção de informações existentes sobre o sistema
b. Recuperação do modelo de análise
b1. Elaboração do modelo de ciclo de vida
b2. Elaboração do modelo de operações
b3. Elaboração do modelo de objetos
Etapa 2. Recuperação de Visões Estruturais
a. Elaboração do Quadro de procedimentos de implementação
a1. Elaboração do quadro de chamadas
a2. Elaboração do quadro índice de procedimentos
b. Elaboração do quadro de operações - procedimentos de implementação

Figura 4-2 – Ordem das Tarefas Prescritas pelo *Fusion-RE/I* (Fonte: Feltrim (1999))

4.3 Análise de Estruturas do Código-Fonte

O objetivo desta técnica é a definição de classes (paradigma de orientação a objetos) a partir da análise das estruturas de dados de um software, isto é, análise do seu código-fonte. Bons indicadores de classes são obtidos examinando o código-fonte do software com a intenção de agrupar algumas de suas variáveis e verificar as estruturas compostas de dados, como registros, arquivos, listas, vetores e árvores. Além disso, é necessário avaliar a interação das variáveis internas com as estruturas de dados globais para que sejam definidas possíveis classes que implementem essa interação.

A visualização de código (criação ou mudança de representações semanticamente equivalentes dentro de um nível de abstração, a partir de dados coletados do código-fonte) é o foco desta técnica. O código-fonte legado possui lógica de programação, decisões de projeto, requisitos do usuário e regras de negócio que podem ser recuperadas e reconstruídas em um modelo sem a perda da semântica [Lucas et al., 2005].

A engenharia reversa identifica componentes para futuro reuso na construção de um software, outra característica importante e fundamental do conceito de orientação a objetos. Usando uma ferramenta CASE com um analisador léxico, os artefatos disponíveis no código-fonte podem ser recuperados. Após a análise do código-fonte, a ferramenta CASE possui conhecimento da estrutura de classes descrita. Esta estrutura é apresentada ao

usuário para navegação pela estrutura e para análise das classes que compõe o projeto [Dantas Filho et. al., 2000].

Duas vertentes da engenharia reversa são observadas em software orientado a objeto: i) parte estática (extração das informações do código-fonte); e ii) parte dinâmica (extração a partir do código-fonte e extração a partir de monitoramento do software em tempo de execução).

4.4 Análise de Estruturas do Banco de Dados

Há vários motivos para fazer a engenharia reversa de banco dados, entre elas: i) recuperar a descrição do conteúdo do banco dados perdida ao longo do tempo, em decorrência das modificações para implementar mudanças necessárias; ii) passar de um sistema gerenciador de banco dados (SGBD) de um fornecedor para outro; iii) mudar a arquitetura do banco dados centralizada para cliente/servidor; iv) implementar interface de banco dados mais inteligente para o usuário; e v) integrar os bancos dados isolados.

Talvez essa seja uma das técnicas mais difundidas e usadas de engenharia reversa. A maior necessidade de alteração de software legado diz respeito à forma como os dados são armazenados. Logo, é necessário extrair os seus modelos conceituais dos dados para que eles possam ser acomodados aos novos paradigmas de gerenciamento de banco de dados [Schneider, 2001].

De acordo com Schneider (2001), para representar um esquema de banco de dados em um outro esquema, é necessário definir os dados e a forma como eles se relacionam, ou seja, obter o modelo conceitual de dados. Com este modelo, é possível alterar o modelo físico de um banco de dados de acordo com as necessidades apontadas pela reengenharia.

O objetivo é criar um modelo de banco de dados a partir de um banco de dados existente. Os modelos de banco de dados mostram sua estrutura para saber como as tabelas e as visões se relacionam com outros elementos, sem mostrar os dados reais. Isso pode simplificar a criação de um novo banco de dados ou a compreensão da estrutura de um banco de dados existente. As tarefas fundamentais deste método são:

- Conexão ao sistema de gerenciamento de banco de dados de destino no qual se deseja realizar a engenharia reversa;

- Determinação de objetos de banco de dados específicos da plataforma, como tabelas, chaves primárias, índices e código de banco de dados, nos quais se deseja realizar a engenharia reversa;
- Extração, análise e validação do esquema em um diagrama de modelo de banco de dados que pode ser revisado e modificado.

4.5 Considerações Finais

O entendimento dos dados ocorre em diferentes níveis de abstração: no nível das estruturas do código-fonte e no nível das estruturas globais de dados. O uso de técnicas facilita o processo de engenharia reversa, de forma que a documentação fique mais robusta quanto possível. A especificidade de cada técnica, em outras palavras, o fato de cada técnica tratar de uma parte específica do software, e a complementação de uma técnica à outra proporcionam um conjunto de métodos que podem construir uma documentação relativamente completa do software.

Neste trabalho, é abordada a extração da estrutura estática do software. A estrutura estática de um software escrtio na linguagem de programação Java pode ser extraída a partir do código-fonte, sem a necessidade de executar e monitorar o software. Diagramas UML são usados para representar esta estrutura.

5 FERRAMENTAS CASE DE ENGENHARIA REVERSA

5.1 Considerações Iniciais

Um dos benefícios oferecidos pelas ferramentas CASE é orientar e disciplinar o processo de desenvolvimento de software. Elas contribuem para a qualidade do software obrigando o desenvolvedor a criar um modelo do software antes de construí-lo [Voxxel, 1998]. Ferramentas CASE é uma classificação que abrange aplicações baseadas em computadores que auxiliam atividades de engenharia de software, desde análise de requisitos e modelagem até programação e testes [Issa, 2006].

A importância de engenharia reversa combinada com o uso de ferramentas CASE para aumentar a compreensão, a manutenção e a recuperação de documentos na reconstrução de um software é relatada em Martins (2003).

A existência de software legado orientado a objetos estimulou a criação de ferramentas CASE para auxiliar o processo de manutenção, fornecendo suporte à recuperação de modelos em uma notação padrão a partir do código-fonte (por exemplo, UML). A engenharia reversa atua no auxílio à recuperação da documentação e ao entendimento de software, possibilitando realizar sua manutenção de forma menos árdua.

Este capítulo relaciona algumas ferramentas CASE que realizam a engenharia reversa. Elas são apresentadas em dois grupos: i) para modelar código-fonte; e ii) modelar banco de dados. Para o primeiro grupo, foram selecionadas ferramentas CASE que usam UML. Para o segundo grupo, foram escolhidas ferramentas CASE que usam a linguagem SQL (Structured Query Language). A escolha foi em decorrência de UML e SQL serem padrões estabelecidos e reconhecidos na academia e na indústria [OMG, 2008; ODMG, 2008].

A seção 5.2 apresenta breve descrição das ferramentas CASE selecionadas para realizar a engenharia reversa do software construindo a sua modelagem usando UML. A seção 5.3 apresenta breve descrição das ferramentas CASE selecionadas para realizar a engenharia reversa do banco de dados.

5.2 Modelagem do Código-Fonte

As ferramentas CASE apresentadas nesta seção realizam a engenharia reversa do software construindo a sua modelagem usando UML. A seguir, elas são apresentadas acompanhadas de breve descrição e de um screenshot após a geração do Diagrama de Classe:

- ArgoUML

  A ArgoUML [ArgoUML, 2001] (Figura 5-1) é uma ferramenta CASE que possui um editor de diagramas UML. Ela é capaz de gerar código em Java, construindo um projeto orientado a objetos. Uma de suas principais características é o apoio à engenharia reversa, gerando Diagramas de Classes importando códigos-fonte em Java. A ArgoUML é licenciada como software livre e é escrita em Java; assim, ela é portável para diferentes sistemas operacionais. Esta ferramenta CASE apresenta algumas dificuldades: i) Diagramas de Classes são gerados com as classes sobrepostas, atrapalhando a sua observação; ii) diagramas gerados são difíceis de serem editados; iii) opção de desfazer um comando (*undo*) não existe; iv) opções claras de saída para o usuário não são fornecidas; v) componentes dos diagramas são difíceis de serem manuseados; vi) sua interação é difícil; e vii) importação de classe é feita uma a uma. Pontos positivos: i) dimensão da área de trabalho é satisfatória; e ii) linguagem do usuário é usada;

- Umbrello

  A Umbrello UML Modeller [Umbrello, 2008] (Figura 5-2) é uma ferramenta CASE que constrói diagramas UML e pode auxiliar no desenvolvimento de software. Esta ferramenta CASE é um software livre, está disponível gratuitamente, está integrada ao projeto KDE (*K Desktop Environment*) e está disponível nas plataformas Linux, FreeBSD e Solaris. Ela gera e imprime diagramas, cria classes em Java, SQL, PHP e Python, permite exportar em formato .png, suporta arquivos XMI (*XML Metadata Interchange*), possui visualizador de código-fonte e permite executar engenharia reversa. Além disso, ela fornece uma estrutura modular para a engenharia reversa de classes Java. Um problema identificado foi: i) Diagrama de Classes não é gerado todo de uma única vez, necessitando colocar classe por classe. Como pontos positivos: i) linguagem do usuário é usada; ii) opções claras de saída para o usuário são usadas; e iii) interação com usuário é facilitada;

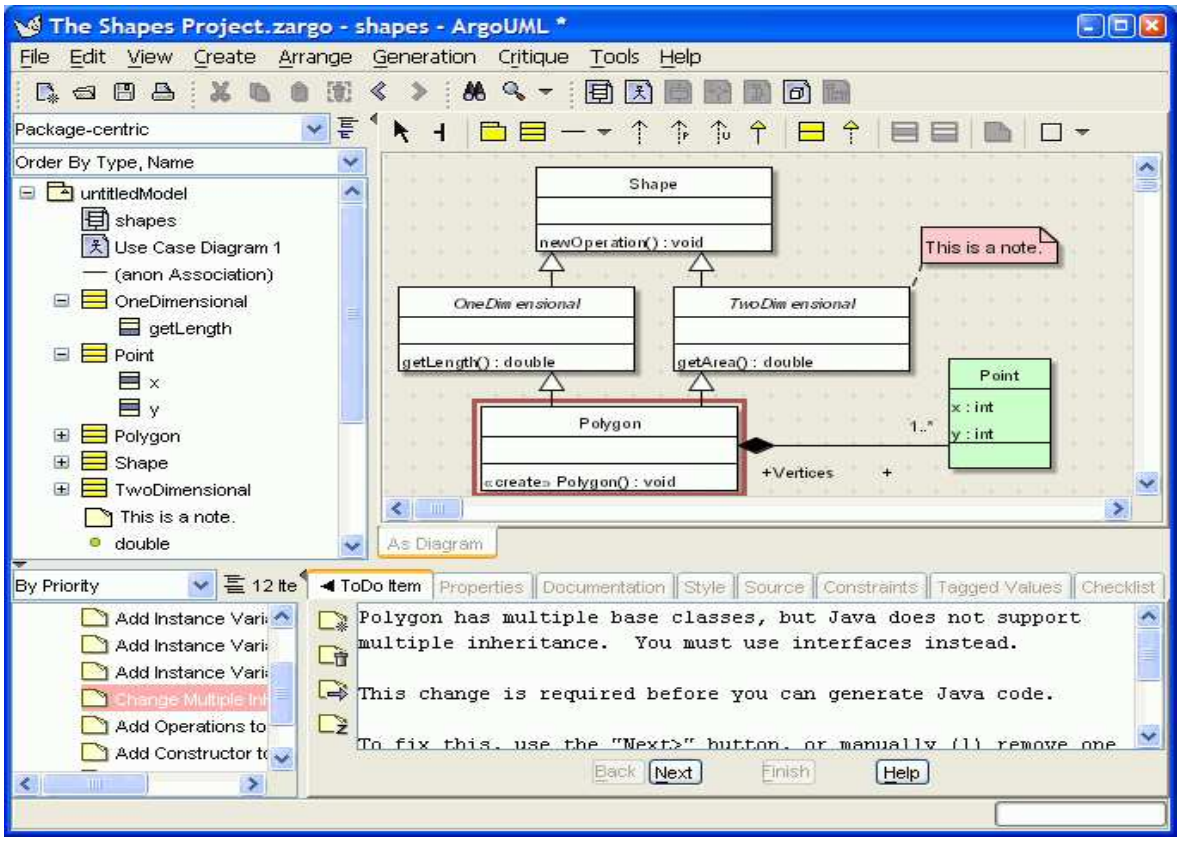

Figura 5-1 – Exibição de Diagrama de Classes na ArgoUML

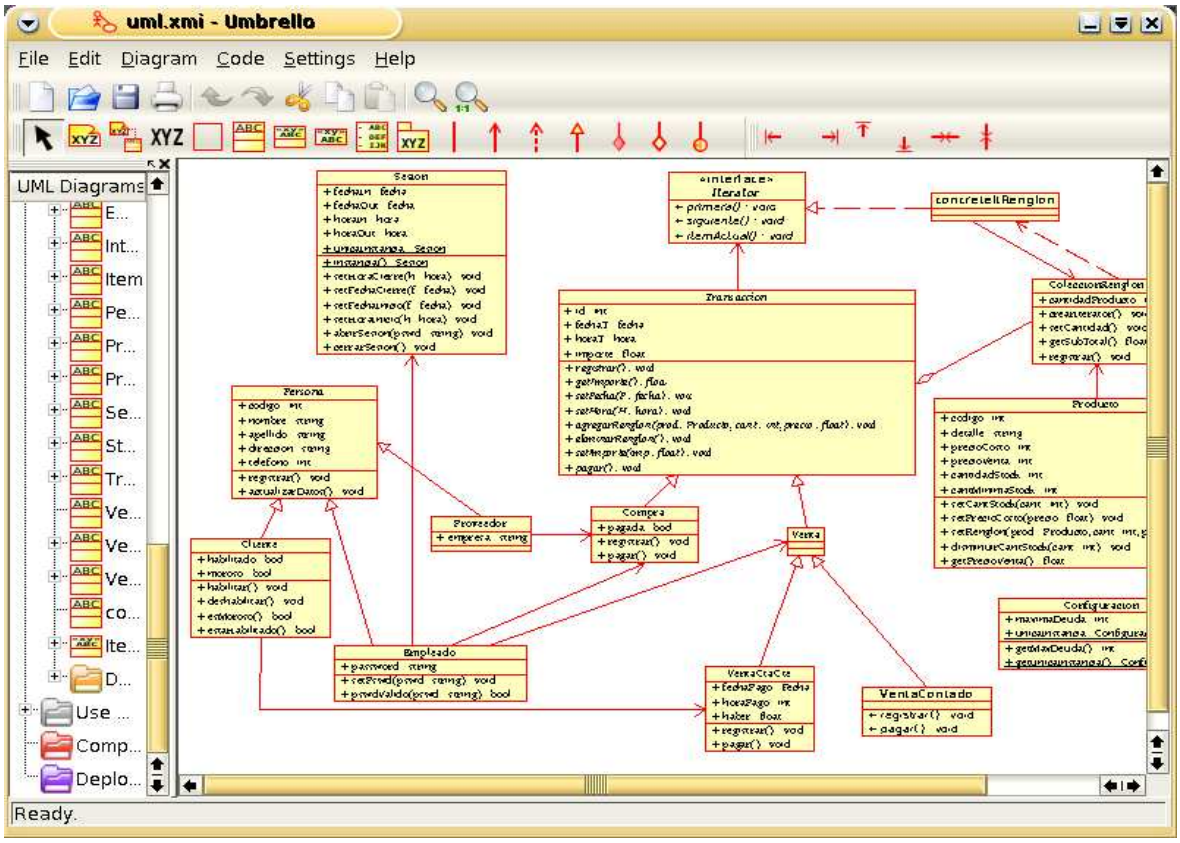

Figura 5-2 – Exibição de Diagrama de Classes na Umbrello UML Modeller

- NetBeans

  A IDE NetBeans [Netbeans, 2008] (Figura 5-3) é um ambiente de desenvolvimento multiplataforma que auxilia programadores a escrever, compilar, *debugging* e instalar aplicações, sendo arquitetada em forma de uma estrutura reusável visando simplificar o desenvolvimento e aumentar a produtividade. Além disso, ela possibilita reverter o código-fonte de um software Java em um projeto UML [UMLNetbeans, 2008]. Ela é considerada apropriada para geração de documentos a partir da engenharia reversa. Não foram detectados problemas relevantes. Pontos positivos se destacaram: i) saídas claras e ícones intuitivos para o usuário são amplamente usados; ii) linguagem do usuário é usada; iii) usuários experientes têm apoio; iii) situações comuns são consistentes, ou seja, usuário não demanda tempo para adivinhar se diferentes palavras ou ações possuem a mesma semântica; e iv) área de trabalho dos diagramas é flexível;

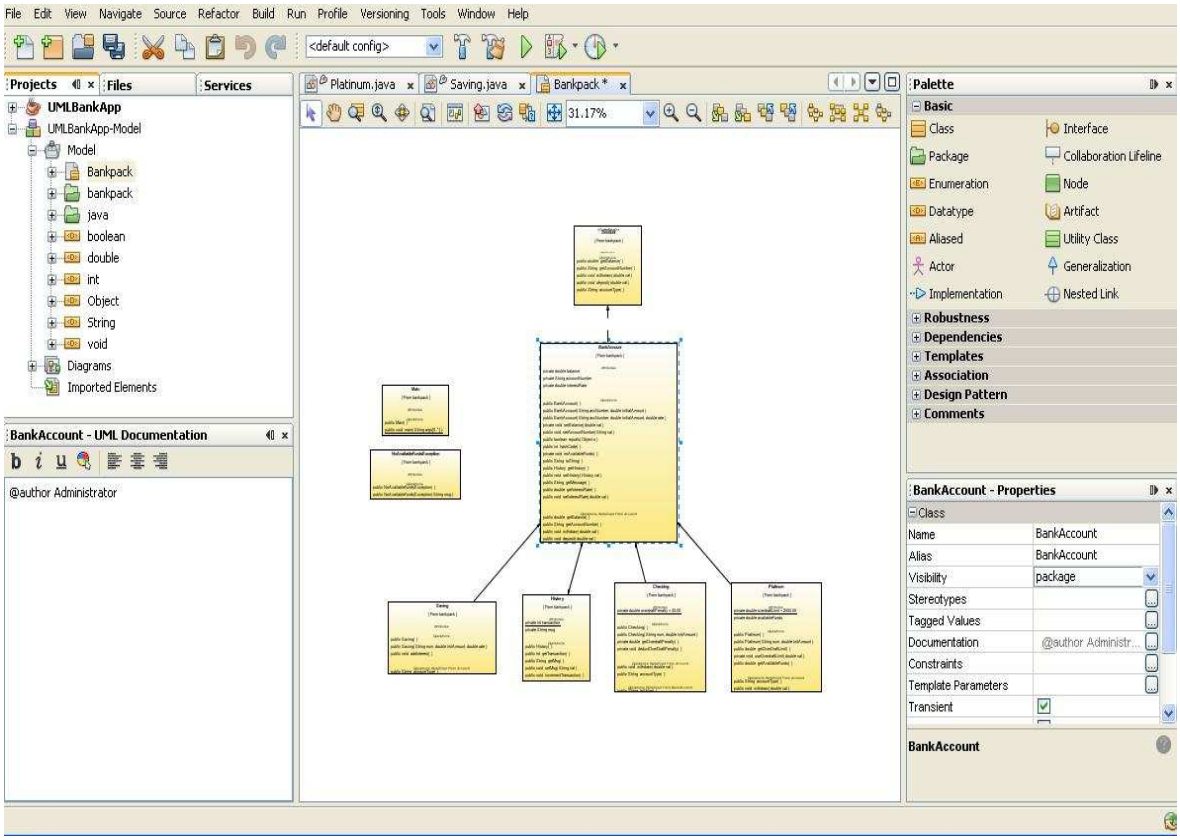

Figura 5-3 – Exibição de Diagrama de Classes na IDE NetBeans

- WithClass

  A WithClass [WithClass, 1998] (Figura 5-4) possui uma linguagem de customização simples para geração automática de código e documentação de modelos. Ela realiza a engenharia reversa de projetos existentes, permitindo documentá-los em diagramas,

retrabalhá-los e aproveitar as boas soluções de projeto existentes. Além disso, ela tem simplicidade de uso e interface simples e coerente com o usuário. As linguagens de programação disponíveis são Visual Basic, Delphi, C++ e Java.

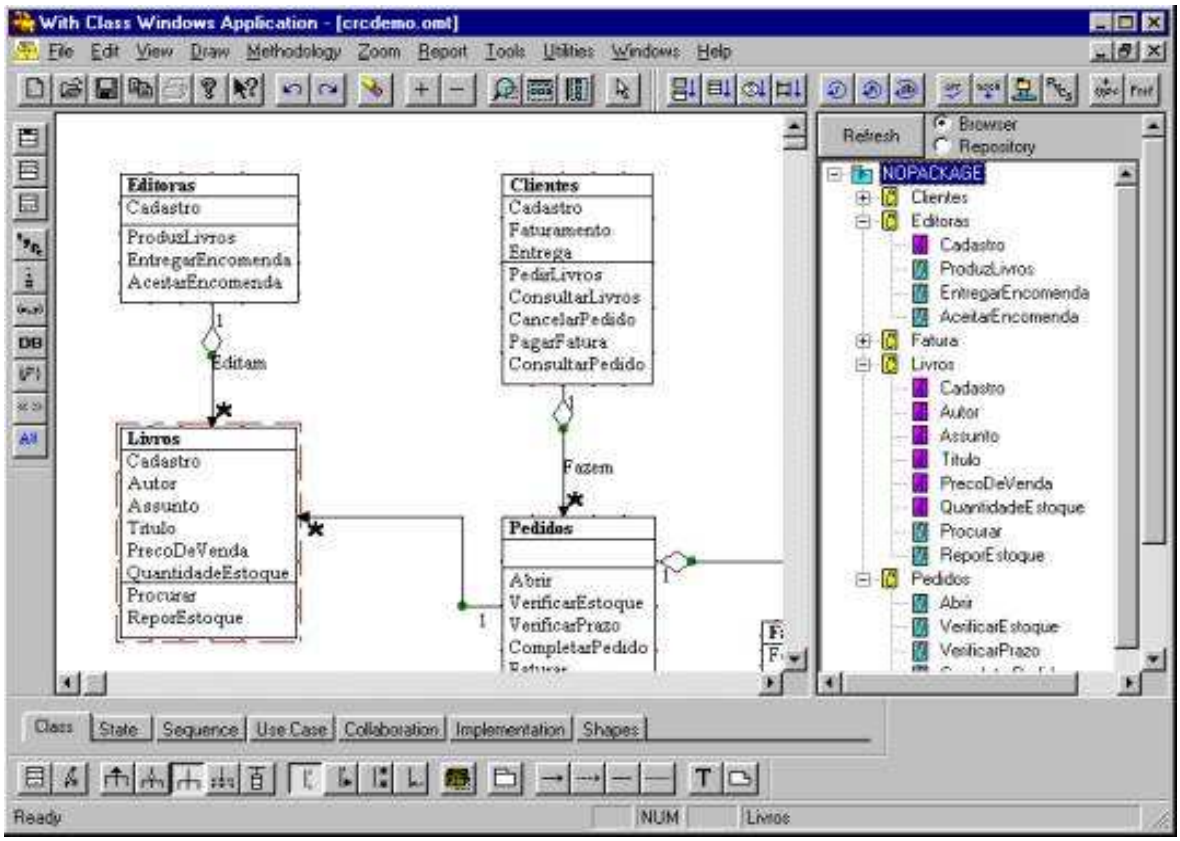

Figura 5-4 – Exibição de Diagrama de Classes na WithClass

5.3 Modelagem do Banco de Dados

As ferramentas CASE apresentadas nesta seção realizam a engenharia reversa do banco de dados. A partir do script do banco de dados, as ferramentas CASE conseguem realizar a engenharia reversa e descobrir os componentes do banco de dados e seus inter-relacionamentos, fornecendo visão mais abstrata e inteligível do banco de dados. A seguir, elas são apresentadas acompanhadas de breve descrição e de um screenshot após a geração da modelagem do banco de dados:

- DBConstructor

  A DBConstructor [DBConstructor, 2003] (Figura 5-5) foi desenvolvida e criada para facilitar e agilizar o processo de desenvolvimento, criação e manutenção de *Database Schemas*. Esta ferramenta CASE mantém documentação completa da estrutura do banco de dados e permite criar *scripts* de DDL (*Data Definition Language*) para múltiplas plataformas de banco de dados, sendo o processo de geração dos *scripts*

simples e rápido. Os *scripts* podem ser gerados para vários SGDBs: IBM DB2, Interbase, Oracle, MySQL e SQL Server. Isso é possível mediante o uso de *database platform definitions*, que podem ser criadas e redefinidas pelo usuário. Algumas de suas principais características são: i) criação de *scripts* de DDL para várias plataformas; ii) importação da estrutura de um banco de dados existente; iii) facilidade para gerar *scripts*; iv) geração de *scripts* de atualização para os objetos de um projeto; e v) simplicidade da visualização da estrutura de um banco de dados;

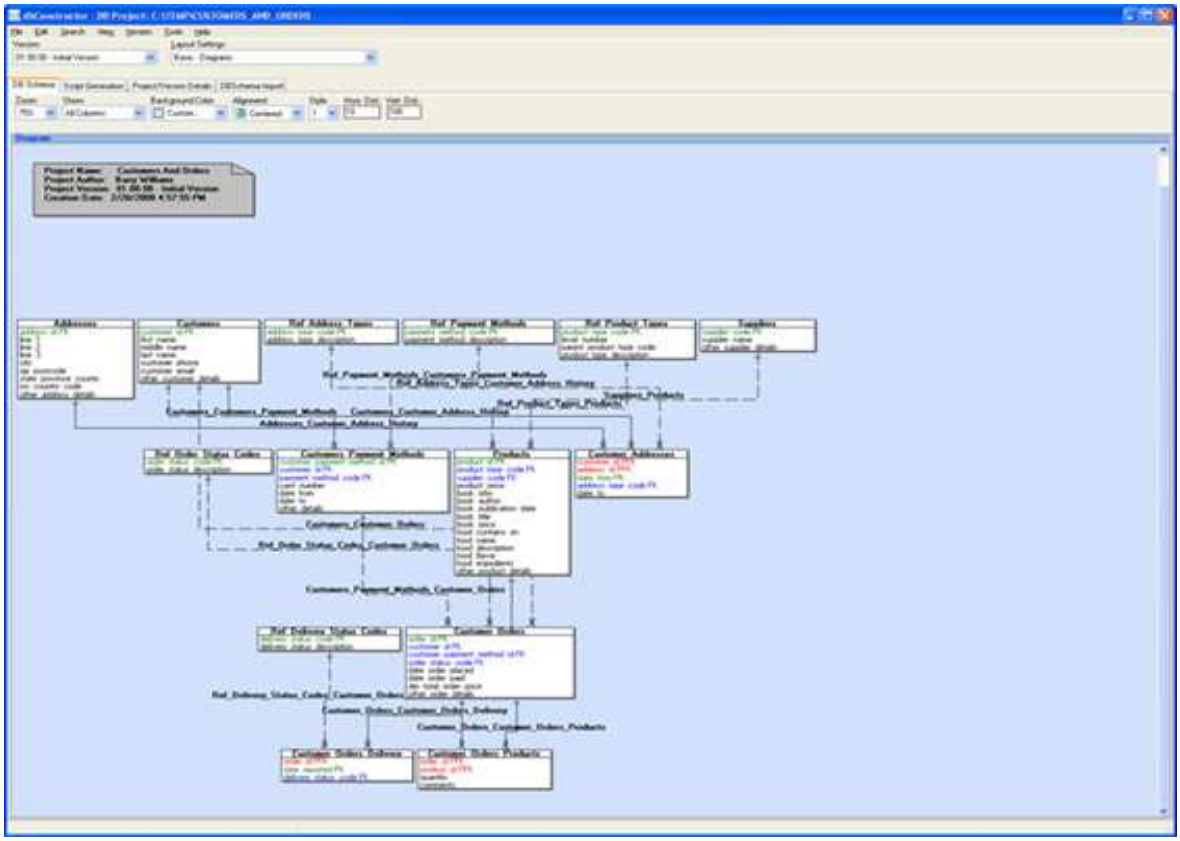

Figura 5-5 – Modelagem do Banco de Dados Usando a DBConstructor

- DBWrench

  A DBWrench [DBWrench, 2007] (Figura 5-6) é uma ferramenta CASE capaz de fazer modelos, construir o modelo diretamente no banco de dados e fazer engenharia reversa de um banco de dados existente. Ela é leve, é escrita em Java, é baseada em JDBC (*Java Database Connectivity*), roda em qualquer plataforma, é proprietária e é capaz de criar diagramas de entidade-relacionamento. A DBWrench possibilita acesso aos SGBDs Microsoft SQL Server, MySQL e PostegreSQL, sendo simples de aprender e fácil de usar. Algumas das suas principais características são: i) simplicidade na adição

de chave estrangeira; ii) presença do *Query Editor* mostrando resultados de *query* em tabela ou em texto; e iii) conversão de um SGBD para outro;

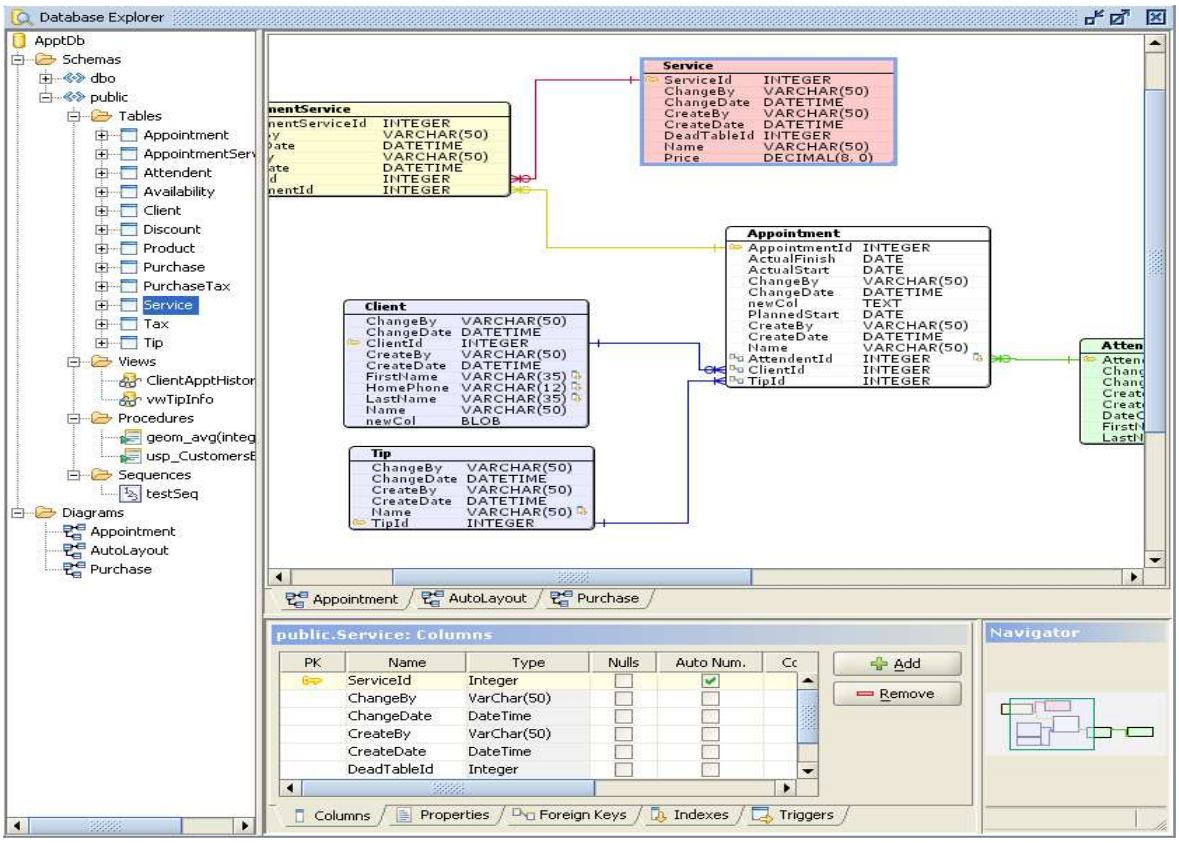

Figura 5-6 – Modelagem do Banco de Dados Usando a DBWrench

- DBVisualizer

  A DBVisualizer [DBVisualizer, 2008] (Figura 5-7) é uma ferramenta CASE multiplataforma 100% Java e é usada para gerenciar e navegar simultaneamente por banco de dados de qualquer tipo usando controladores JDBC. Além disso, ela permite múltiplas conexões aos SGBDs: Oracle, Sybase, DB2, MySQL, Informix, SQL Server, PostgreSQL, Cloudscape, McKoi, SAP DB, Mimer e InstantDB. A DBVisualizer possui clara representação gráfica das relações entre tabelas, tipos de dados, índices, privilégios e procedimentos armazenados, capaz de executar sentenças SQL;

- SquirreL

  A SQuirreL [Squirrel, 2003] (Figura 5-8) é uma ferramenta CASE gratuita e escrita em Java. Entre outras funções, ela gera diagramas a partir de banco de dados, permitindo exportá-los para imagem ou imprimi-los, aceita diversos outros SGBDs, usando JDBC, e possui interface simples. O *plugin graph* permite ao usuário criar gráficos de tabela. O teclado é bem explorado com atalho para várias operações.

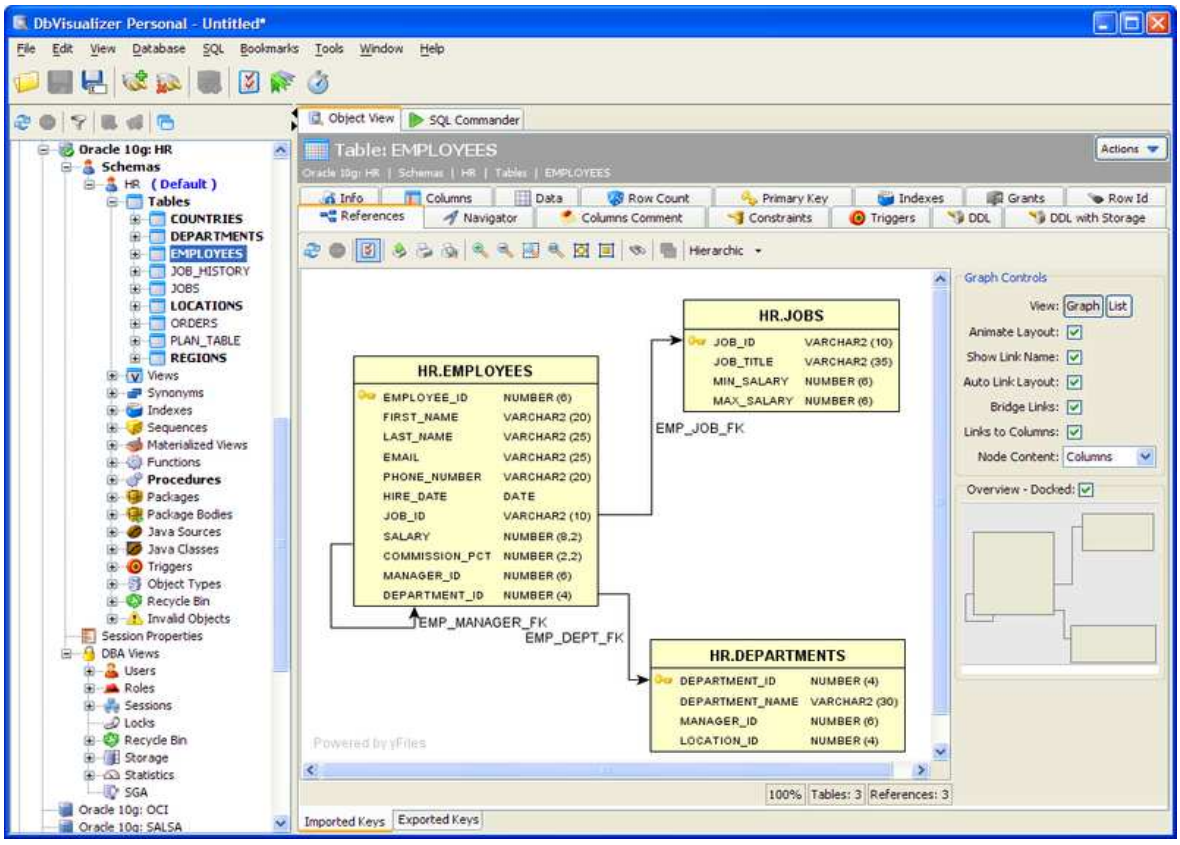

Figura 5-7 – Modelagem do Banco de Dados Usando a DBVisualizer

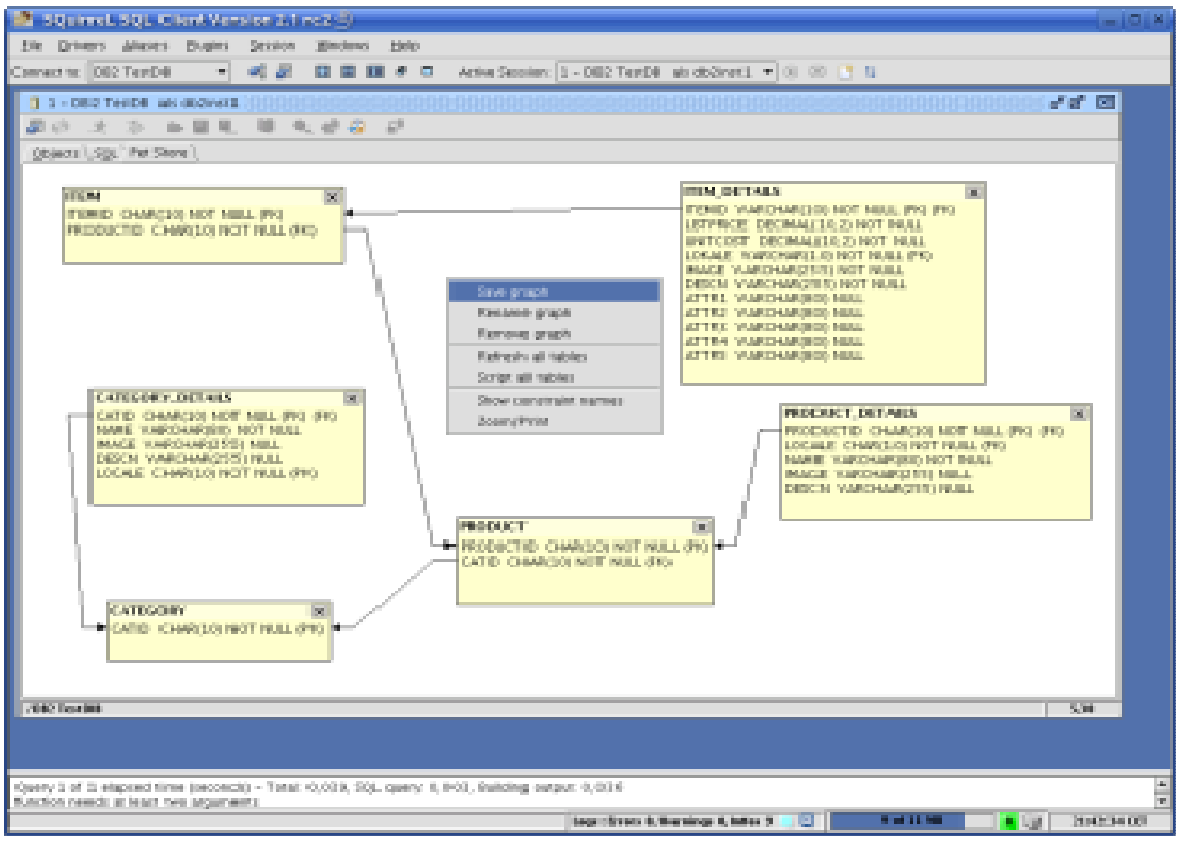

Figura 5-8 – Modelagem do Banco de Dados Usando a SQuirreL

5.4 Considerações Finais

As ferramentas CASE são fundamentais para o processo de desenvolvimento de software. Um dos maiores benefícios oferecidos por elas é orientar e disciplinar a engenharia reversa software. As ferramentas CASE contribuem para a qualidade do software obrigando o desenvolvedor a criar um modelo do software antes de construí-lo. De modo análogo ao desenvolvimento de software, o uso de ferramentas CASE durante o processo de engenharia reversa é indispensável.

6 USABILIDADE

6.1 Considerações Iniciais

Este capítulo apreesnta conceitos concernentes à usabilidade. Considerando que a interação com a ferramenta CASE é uma questão de extrema importância no contexto deste trabalho, a análise da usabilidade dessas ferramentas se faz necessária.

A seção 6.2 apresenta algumas definições de usabilidade. A seção 6.3 aborda detalhadamente as subcaracterísticas de usabilidade, conforme a norma ISO/IEC 9126. A seção 6.4 faz algumas considerações sobre testes de usabilidade.

6.2 Conceitos

Na interação humano-computador e na Ciência da Computação, usabilidade normalmente se refere à simplicidade e à facilidade com que um interface, um programa de computador ou um website podem ser utilizados. Usabilidade é o termo empregado para descrever a qualidade da interação dos usuários com uma determinada interface [Bevan, 1998]. Segundo Nielsen (1993), esta qualidade está associada aos seguintes princípios:

- Facilidade de aprendizado. O usuário aprende rápido?;
- Facilidade de lembrar como realizar uma tarefa após algum tempo. O usuário memoriza o que aprendeu?;
- Produtividade. O usuário fez suas tarefas com rapidez?;
- Baixa taxa de erros. O usuário comete poucos erros durante a interação?;
- Satisfação subjetiva do usuário. O usuário gosta de utilizar o sistema?

Considera-se que a interface tem problema de usabilidade se determinado usuário ou grupo de usuários encontra dificuldades para realizar uma tarefa com a interface. Tais dificuldades podem ter origens variadas e ocasionar perda de dados, diminuição da produtividade e total rejeição do software por parte dos usuários.

A usabilidade está diretamente ligada ao diálogo na interface e à capacidade do software em permitir que o usuário alcance suas metas de interação com o sistema. Ser de fácil aprendizagem, permitir utilização eficiente e apresentar baixa taxa de erros são aspectos fundamentais para a percepção da boa usabilidade por parte do usuário.

De acordo com Shneiderman (1998), usabilidade é a propriedade de uma interface que permite classificá-la quanto à:

- facilidade de aprendizado;
- eficiência de uso/desempenho na execução de tarefas;
- retenção com o tempo;
- minimização da quantidade de erros;
- satisfação subjetiva.

O termo usabilidade foi definido na norma ISO/IEC 9126 [ISO/IEC 9126-1, 2001], sobre qualidade de software, como um conjunto de atributos de software relacionado ao esforço necessário para seu uso e para o julgamento individual de tal uso por determinado conjunto de usuários.

Segundo a norma ISO/IEC 9241 [ISO 9241-11, 1997], três medidas são fundamentais para a evidenciar a boa usabilidade de uma interface:

- efetividade: permite ao usuário alcançar objetivos iniciais de interação, sendo a efetividade avaliada em termos de finalização de uma tarefa e em termos de qualidade do resultado obtido;
- eficiência: refere-se à quantidade de esforço e de recursos necessários para chegar a um determinado objetivo. Os desvios que o usuário faz durante a interação e a quantidade de erros cometidos podem servir para avaliar o nível de eficiência da ferramenta;
- satisfação: refere-se ao nível de conforto do usuário ao utilizar a interface e é a maneira de alcançar seus objetivos ao usar a ferramenta CASE. Assim, a satisfação é difícil de medir e de quantificar, pois está relacionada com fatores subjetivos.

O intuito da usabilidade não é subestimar a capacidade dos usuários de utilizar um determinado sistema, masidentificar possíveis erros e tratá-los de maneira que facilite o seu pelos usuários.

Segundo Valdestilhas; Almeida (2005), a usabilidade de uma interface é um dos fatores mais importantes para o sucesso ou insucesso do software. Porém, a usabilidade freqüentemente é ignorada por diversas razões, por exemplo, financeiras e técnicas . O conceito de usabilidade está fundamentado em pesquisas interdisciplinares rigorosas e resulta em soluções de fácil implementação.

6.3 Sub-Características de Usabilidade da Norma ISO/IEC 9126

Usabilidade é a capacidade do software de ser entendido, lido, usado e atrativo ao usuário quando utilizado em condições estabelecidas, oferecendo a ele, em um determinado contexto de operação, a realização de tarefas, de maneira eficaz, eficiente e agradável. Usabilidade está dividida em cinco sub-características: Inteligibilidade, Apreensibilidade, Operacionalidade, Atratividade e Conformidade [ISO/IEC 9126-1, 2001] (Figura 6-1). Contudo, este trabalho trata as quatro primeiras sub-catacterísticas:

- Inteligibilidade: capacidade do software de fazer o usuário saber se ele é adequado e como ele pode ser usado por tarefas particulares;
- Apreensibilidade: capacidade do software de fazer o usuário entendê-lo;
- Operacionalidade: capacidade do software de fazer o usuário aprendê-lo e controlá-lo;
- Atratividade: capacidade do software de fazer o usuário vê-lo atraente.

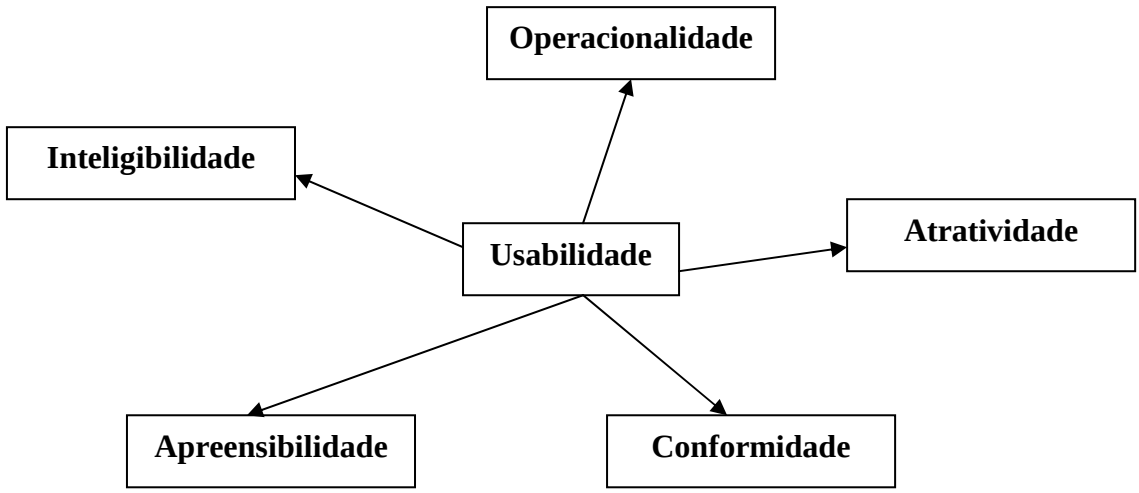

Figura 6-1 – Subcaracterísticas de Usabilidade

6.4 Testes de Usabilidade

Segundo Mayheu (1999), o teste de usabilidade é um técnica formal que pode envolver usuários representando a população alvo para aquele determinado sistema. Estes usuários são designados para desenvolver tarefas típicas e críticas havendo coleta de dados para serem posteriormente analisados. Contudo, o teste de usabilidade caracteriza-se por utilizar diferentes técnicas voltadas, em sua maioria, para a avaliação da ergonomia[2] dos sistemas interativos:

[2] Segundo Iida (1997, p. 1) *apud* Royas; Marziale (2001), ergonomia é o estudo do relacionamento entre o homem e o seu trabalho, equipamento e ambiente, e particularmente a aplicação dos conhecimentos de anatomia, fisiologia e psicologia na solução dos problemas surgidos desse relacionamento.

- Avaliação Heurística;
- Critérios Ergonômicos;
- Inspeção Baseada em Padrões, Guias de Estilo ou Guias de Recomendações;
- Inspeção por *Checklists*;
- Percurso (ou Inspeção) Cognitivo;
- Teste Empírico com Usuários;
- Entrevistas e Questionários.

De acordo com Cybis; Bertiol (2007), algumas técnicas de avaliação para testes de usabilidade podem incluir uma lista de métodos que direciona os esforços dos usuários em realizar uma variedade de tarefas em um protótipo ou sistema. Enquanto o usuário realiza estas tarefas, ele é observado por inspetores que coletam dados referentes aos processos de interação. Os dados coletados consistem em: i) erros cometidos pelo usuário; ii) quando e onde eles confundem-se ou frustram-se; iii) rapidez com que o usuário realiza a tarefa; iv) se os usuários obtêm sucesso na realização da tarefa; e v) satisfação do usuário com a experiência.

Entretanto, testes de usabilidade que envolvem usuários reais no procedimento de interação transformam-se em procedimento mais oneroso e complexo. A utilização de heurísticas, por exemplo, permite identificar erros mais difíceis de serem identificados. Mas, estudos apontam que a utilização conjunta de ambos os processos, aplicação de heurísticas e testes de usabilidade, é melhor abordagem de investigações de usabilidade.

6.5 Considerações Finais

Este capítulo abordou alguns conceitos de usabilidade, além de suas sub-características definidas na norma ISO/IEC 9126. Estes conceitos são importantes para a análise comparativa da usabilidade das ferramentas CASE.

7 USO DAS FERRAMENTAS CASE

7.1 Considerações Iniciais

Este capítulo apresenta um estudo de caso de cada técnica abordada no capítulo 4 em um software real. Em cada uma das técnicas, uma ferramenta CASE é usada para auxiliar a técnica. As ferramentas CASE abordadas neste capítulo foram apresentadas no capítulo 5.

A seção 7.2 apresenta o software escolhido para ser usado no estudo de caso, descrevendo algumas de suas características e relatando seu estado atual. A seção 7.3 apresenta o uso da análise de estruturas do código-fonte. A seção 7.4 apresenta o uso da análise de estruturas de banco de dados. A seção 7.5 apresenta o uso de *Fusion/RE*.

7.2 O Software

O software escolhido para realizar o estudo de caso é o Sistema Gerenciador de uma Agropecuária (SisGAGRO) a ser implantado em uma empresa de atividade rural, Agro-Sanjutá – Fazenda dos Varões, situada na cidade de Campo Belo/MG. O processo de desenvolvimento contou com a elaboração da documentação de parte do software apenas na fase inicial. A documentação foi construída usando os diagramas UML.

A documentação existente não está atualizada. Além de comentários entre as linhas de código, alguns diagramas criados no início do desenvolvimento não refletem o software atual. Algumas funções do software foram modificadas, caracterizando a incompatibilidade do software com a documentação. Novas funções foram inseridas no software, caracterizando a não existência de documentação.

7.3 Uso da Técnica Análise de Estruturas do Código-Fonte

O surgimento de software legado orientado a objetos na última década estimulou a criação de ferramentas CASE que auxiliassem o processo de manutenção, fornecendo suporte à recuperação de modelos em uma notação padrão a partir do código. A engenharia reversa atua no auxílio à recuperação da documentação e ao entendimento desse software, possibilitando que a sua manutenção seja realizada de forma menos árdua.

A ferramenta CASE de engenharia reversa escolhida para automatizar o processo de geração da documentação do código-fonte foi IDE Netbeans. Esta ferramenta CASE foi apresentada na seção 5.2 e gera diagramas UML. Por motivos do volume da documentação e de ser basicamente manutenção de cadastros, decidiu-se escolher apenas uma função (Manter Cadastro de Animal) e explorá-la.

A seguir, são apresentados o Diagrama de Classes (Figura 7-1 e Figura 7-2), o Diagrama de Estado (Figura 7-3) e o Diagrama de Seqüência (Figura 7-4), representando a realidade atual do software e caracterizando a engenharia reversa de código-fonte.

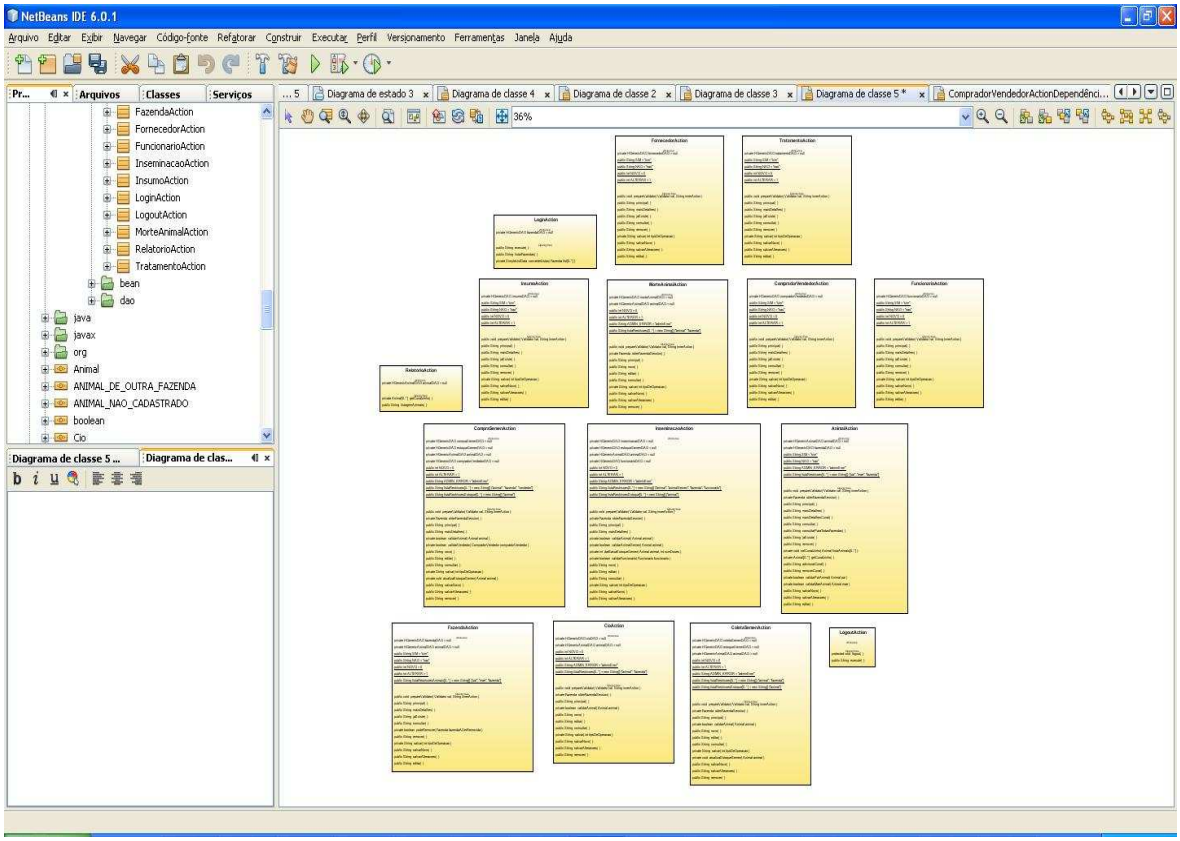

Figura 7-1 – Diagrama de Classes do SisGAGRO com Atributos e Métodos e sem Relacionamentos

7.4 Uso da Técnica Análise de Estruturas do Banco de Dados

A engenharia reversa do banco de dados consiste em criar um modelo de banco de dados a partir de um banco de dados existente. Conforme apresentado na seção 4.4, os modelos de banco de dados mostram graficamente a estrutura do banco de dados para que seja possível ver como os elementos do banco de dados, como tabelas e visões, se relacionam com outros elementos, sem mostrar os dados reais. A ferramenta CASE escolhida para realizar a engenharia reversa do banco de dados foi a DBWrench. Esta

ferramenta CASE é capaz de criar diagramas de entidade-relacionamento e realizar engenharia reversa de banco de dados existente.

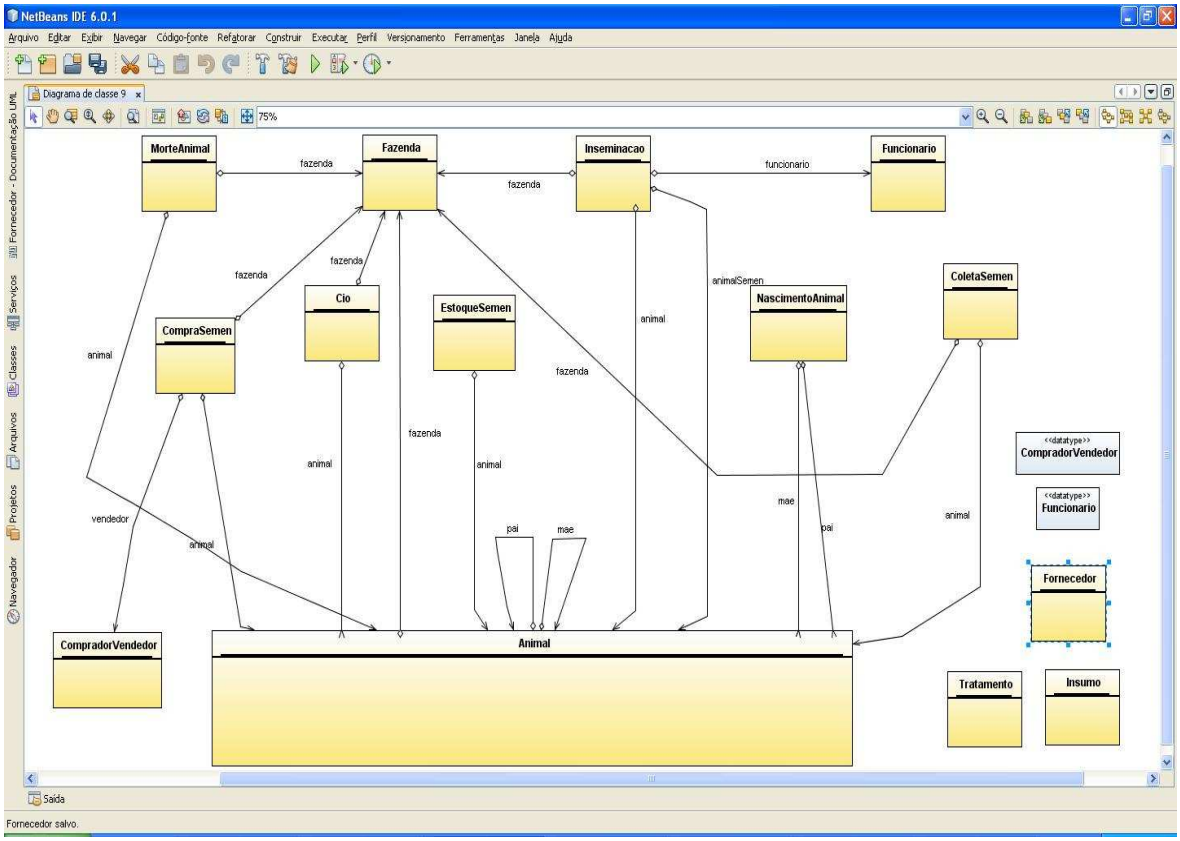

Figura 7-2 – Diagrama de Classes do SisGAGRO sem Atributos e Métodos

A Figura 7-5 mostra o modelo do banco de dados do SisGAGRO. Os relacionamentos entre as tabelas não aparecem, pois o desenvolvedor optou por fazer todo tipo de tratamento de dados no próprio código-fonte. O relacionamento entre as tabelas é definido nas linhas de código-fonte do software ao invés de ser no banco de dados.

Em um software em que o relacionamento entre as tabelas existe, é fundamental o seu entendimento para a compreensão do software. As regras de negócio muitas vezes são definidas diretamente no próprio banco de dados através do relacionamento entre as classes.

Neste caso, esta técnica não contribui para a redocumentação desse software no sentido de facilitar o seu entendimento, pois a sua lógica foi feita no próprio código-fonte. Desta forma, a modelagem do banco de dados não facilita o entendimento do software, uma vez que tabelas isoladas têm pouco significado quando o foco é a relação entre elas. Por este motivo, essa técnica não é a mais adequada a ser utilizada para o tipo de implementação em estudo.

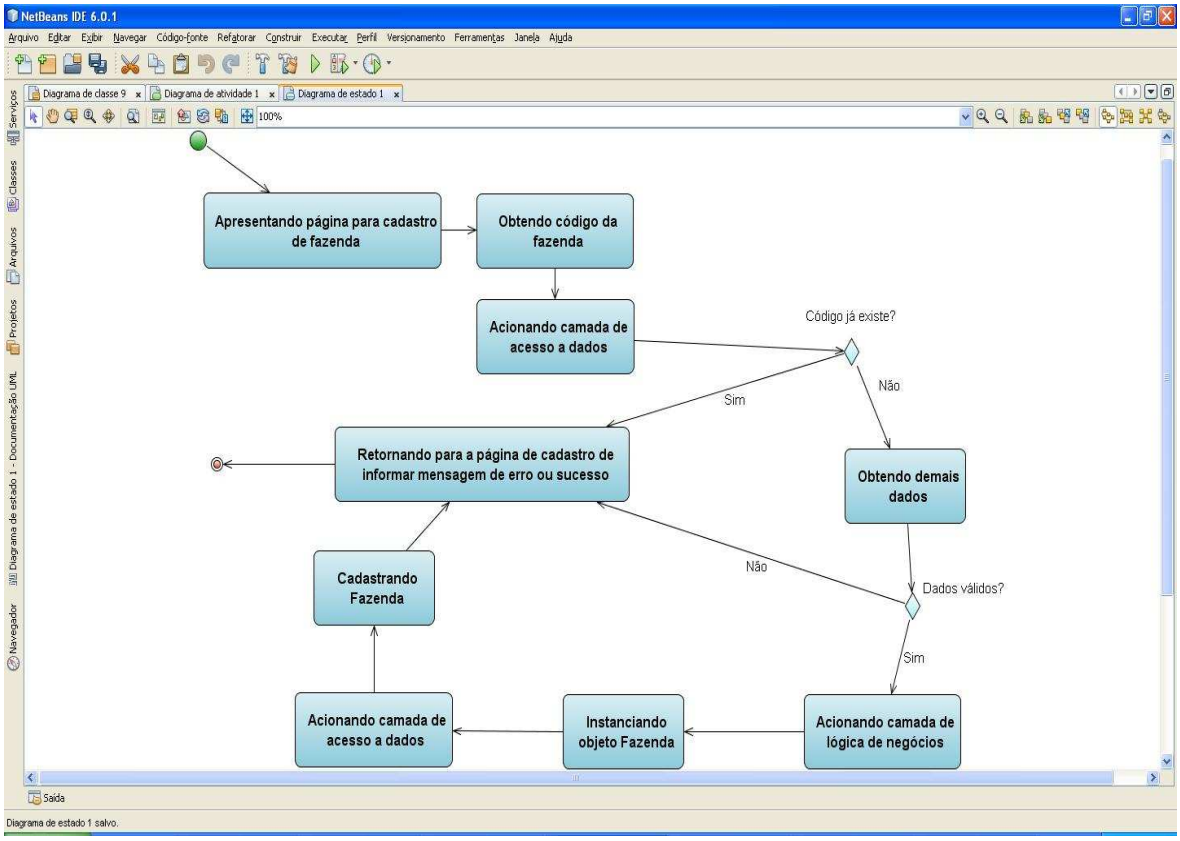

Figura 7-3 – Diagrama de Estados do SisGAGRO

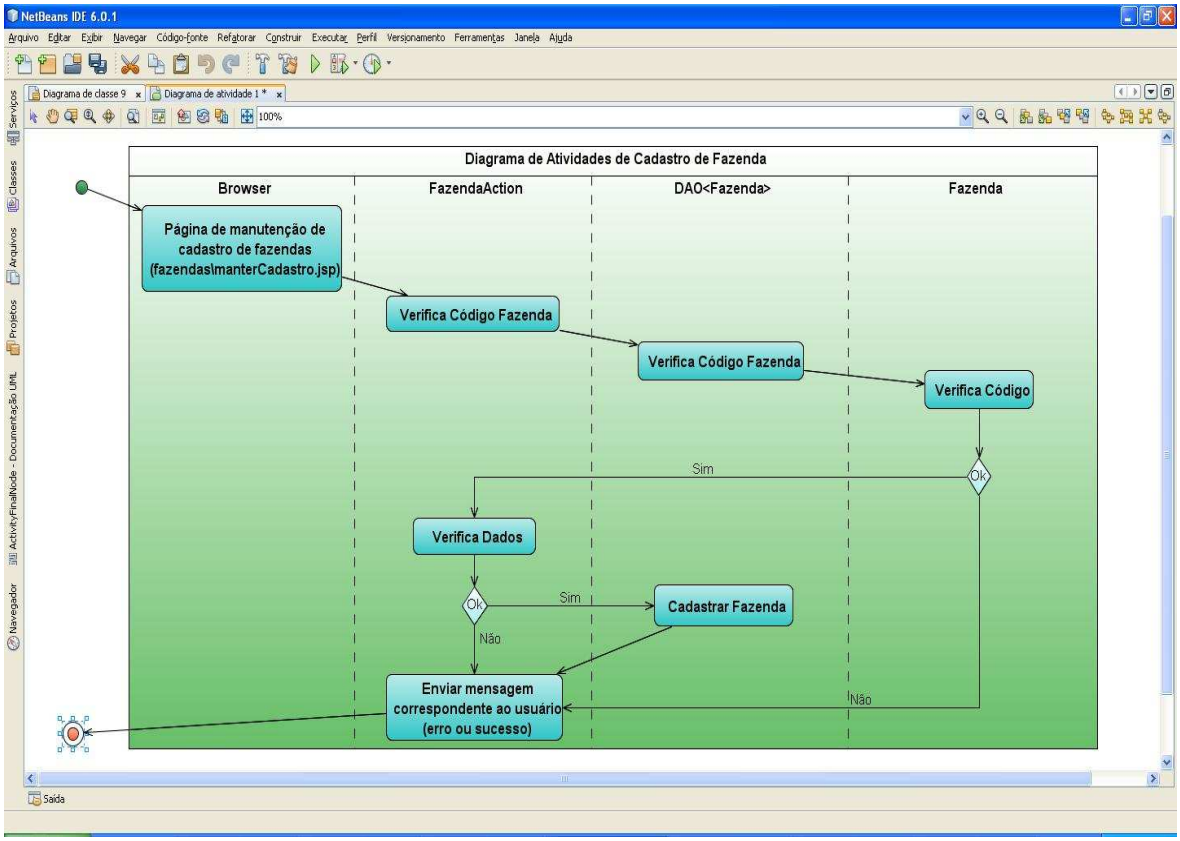

Figura 7-4 – Diagrama de Atividade Cadastro de Fazenda do SisGAGRO

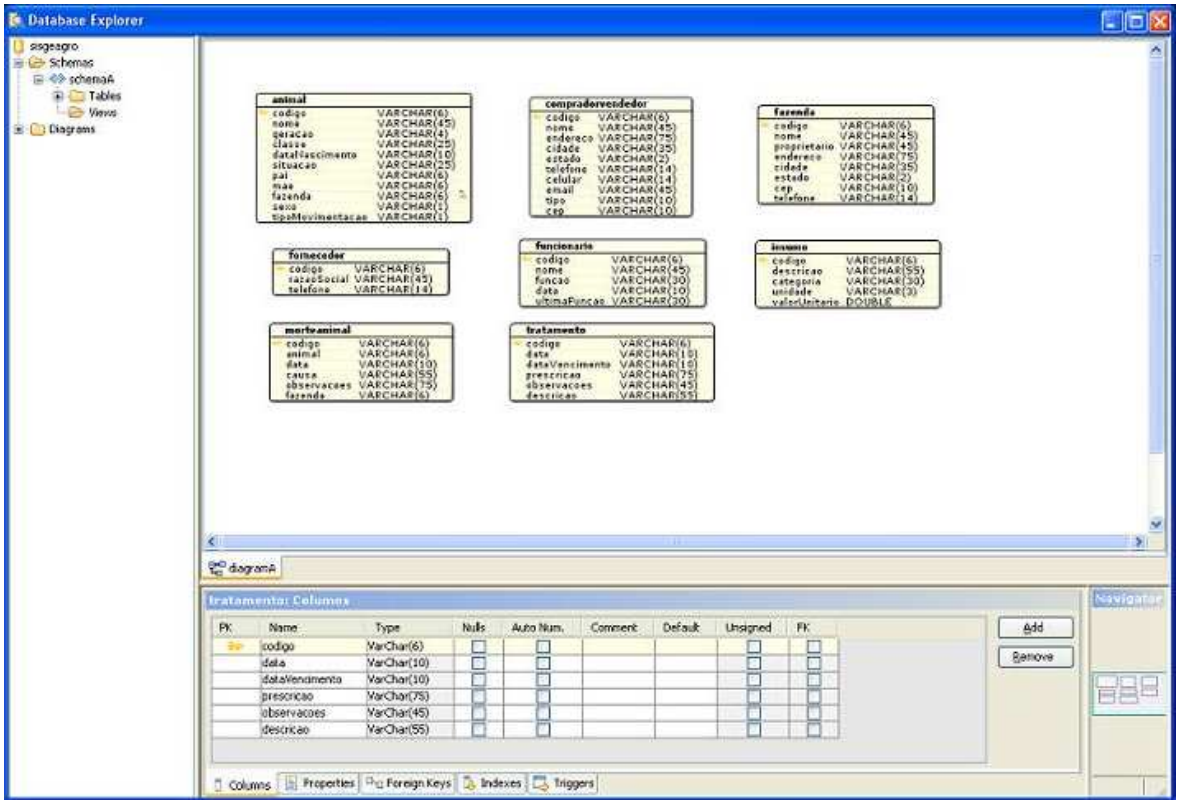

Figura 7-5 – Modelo do Banco de Dados do SisGAGRO

7.5 Fusion-RE/I

Entre as técnicas em estudo, a *Fusion-RE/I* é a mais robusta. Como relatado na seção 4.2, *Fusion-RE/I* foi criado na tentativa de reunir as melhores técnicas, o que, em partes, justifica sua robustez. A princípio, criado com o objetivo de recuperar o projeto de software procedimental em uma forma de objetos, *Fusion-RE/I* não se limita a isso. Esta seção usa *Fusion-RE/I* para realizar a engenharia reversa do SisGAGRO. *Fusion-RE/I* possui uma seqüência de passos pré-definidos que auxiliam o processo de engenharia reversa. As duas próximas seções apresentam cada um desses passos detalhado e sua execução.

7.5.1 Etapa 1 – Recuperação de Visões Funcionais

Antes de iniciar o desenvolvimento do software, foi feita uma entrevista com a empresa interessada no serviço para saber qual era o produto que ele desejava obter. Foram feitas algumas anotações, uma filmagem de parte da entrevista e uma gravação de voz de outra parte da entrevista.

Depois de iniciado o desenvolvimento do software, algumas dúvidas foram sanadas com o dono da empresa. Cabe salientar que o desenvolvedor foi funcionário da empresa por algum tempo (aproximadamente 4 anos); assim sendo, ele tem bom conhecimento das regras de negócio da empresa. Por este motivo, não houve necessidade de muitas reuniões.

Existe uma pré-documentação do software, que possui cerca de 30 páginas contendo a especificação do trabalho, mas está desatualizada. Apesar disso, ela foi usada para auxiliar no início do desenvolvimento do software.

Após a parte de especificação de requisitos, foram elaborados diagramas UML relativos ao software: i) três Diagramas de Casos de Uso; ii) Diagrama de Classes; iii) três Diagramas de Estados; iv) três Diagramas de Seqüência; e v) três Diagramas de Atividades. Além destes diagramas, foi elaborado o modelo de projeto do software.

De acordo com Feltrim (1999), sendo o Diagrama de Estados um elemento de modelagem poderoso sob o aspecto da modelagem de seqüências de eventos e de estados de um objeto e semanticamente bem definido, era esperado que a representação do aspecto comportamental do software por meio destes diagramas revelasse informações sobre o software em análise. De fato, os Diagramas de Estados possibilitaram melhor visualização da seqüência de operações representada no modelo de ciclo de vida, permitindo que fossem feitas correções apropriadas no modelo, para que ele ficasse consistente com a interface do SisGAGRO (Figura 7-6 e Figura 7-7).

```
life cycle SISGEAGRO - (FAZENDA | Saída)

FAZENDA = (ANIMAIS | COMPRADORES/VENDEDORES | INSUMOS | FORNECEDORES |
TRATAMENTOS VETERINÁRIOS | FUNCIONÁRIOS | VENDAS | MORTES | COMPRA DE
SÊMEN | COLETA DE SÊMEN | INSEMINAÇÃO ARTIFICIAL | CIO ) + ...
```

Figura 7-6 – Modelo de Ciclo de Vida do SisGAGRO

O modelo de ciclo de vida é composto por expressões regulares que definem a seqüência de eventos a que o software pode interagir durante a sua vida. Ela descreve o comportamento de como o software se comunica com o ambiente, desde sua criação até o seu término [Masiero, 1995; Penteado, 1996].

O modelo de operações define o comportamento das operações do software, declarativamente, de forma textual. Isto é feito usando quadros que especificam as operações: i) nome da operação; ii) descrição da operação; iii) valores a que operação pode

ter acesso (apenas leitura); iv) valores que a operação pode modificar; v) lista de eventos que pode ser enviada a agentes (entidades ativas que interagem com o software) pela operação; vi) pré-condições; e vii) pós-condições, relacionando o estado do software antes da operação e o estado do software posterior à efetivação da operação.

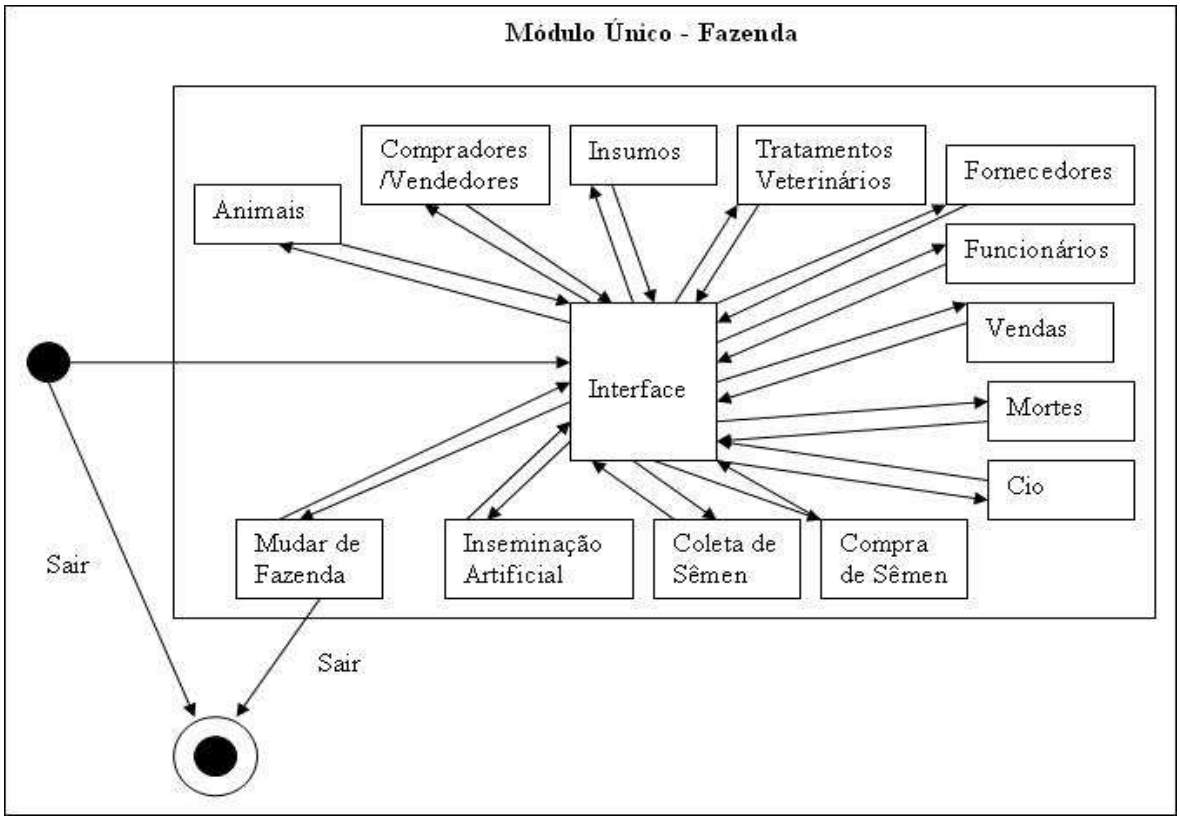

Figura 7-7 – Diagrama de Estados Representando as Sentenças Iniciais do Ciclo de Vida

O modelo de operações é decorrente do modelo de ciclo de vida. Este modelo especifica o comportamento de uma operação de forma declarativa em termos das mudanças de estado do software e eventual geração de eventos de saída [Masiero, 1995]. Para ilustrar e entender o modelo de operações do SisGAGRO, foi escolhido apenas um método (Criar Fazenda) para representar o que seria feito no software. A Figura 7-8 apresenta o *template* usado para apresentar o modelo de operações do método (Figura 7-9).

O modelo de objetos descreve a estrutura do software e o modelo da interface descreve o comportamento do software. Por sua vez, o modelo de interface é composto de dois modelos que capturam diferentes aspectos do comportamento [Coleman *et al.*, 1996]: i) modelo do ciclo de vida; e ii) modelo de operações. O objetivo do modelo de objetos é representar conceitos existentes no domínio do software e o relacionamento entre eles [Masiero, 1995]. A notação é derivada do modelo entidade-relacionamento estendido,

sendo composta por classes, relacionamentos entre classes, atributos de classes e de relacionamentos, agregações, especializações e generalizações [Coleman *et al*., 1996]. A Figura 7-10 mostra um Diagrama de Classes representando o modelo de objetos.

Operação:	b_Criar_hiperbase
Descrição:	Cria uma nova hiperbase vazia
Lê:	
Modifica:	
Envia:	
Assume:	
Resultado:	Se existe uma hiperbase aberta, então é criada uma nova área para edição de hiperbase e a hiperbase que estava aberta é fechada Alterações não salvas na hiperbase que estava aberta estarão perdidas

Figura 7-8 – Descrição de Operação "Criar Hiperbase" (Fonte: Feltrim (1999))

Operação:	Cadastrar Fazenda
Descrição:	Cadastra uma nova fazenda vazia
Lê:	
Modifica:	
Envia:	
Assume	
Resultado:	Se existe uma fazenda aberta, então é criada uma nova área para edição de fazenda e a fazenda que estava aberta é fechada. Alterações não salvas na fazenda que estava aberta estarão perdidas.

Figura 7-9 – Descrição de Operação "Criar Fazenda" do SisGAGRO

Conforme sugerido por Feltrim (1999) e baseados nos estudos realizados sobre *Fusion-RE/I* e UML, foi elaborada uma proposta para a representação das informações da fase de análise recuperadas pelo uso de *Fusion-RE/I* usando UML. O *Fusion-RE/I* se difere do *Fusion/RE* porque leva à produção das visões do software sob o paradigma de orientação a objetos, ao contrário do segundo, que é uma estratégia de engenharia reversa para mudança de paradigma do software legado procedural para o padrão orientado a objetos. O objetivo dessa proposta foi encontrar as correspondências entre as duas notações, de forma que a informação contida nos modelos de análise de *Fusion-RE/I* pudesse ser representada adequadamente em UML. A Figura 7-11 mostra resumidamente quais elementos de UML foram usados para representar os modelos de análise de *Fusion-RE/I*.

7.5.2 Etapa 2 – Recuperação de Visões Estruturais

Existe um quadro de chamadas para cada arquivo do software. Nesse quadro, são descritos os procedimentos incluídos no arquivo, bem como quais são chamados e quais o chamam. Feltrim (1999) propõe o uso dos *packages* e do Diagrama de Componentes para construir o quadro de chamadas. Desta forma, foi feita a elaboração de um Diagrama de Componentes de um método do SisGAGRO (Figura 7-12).

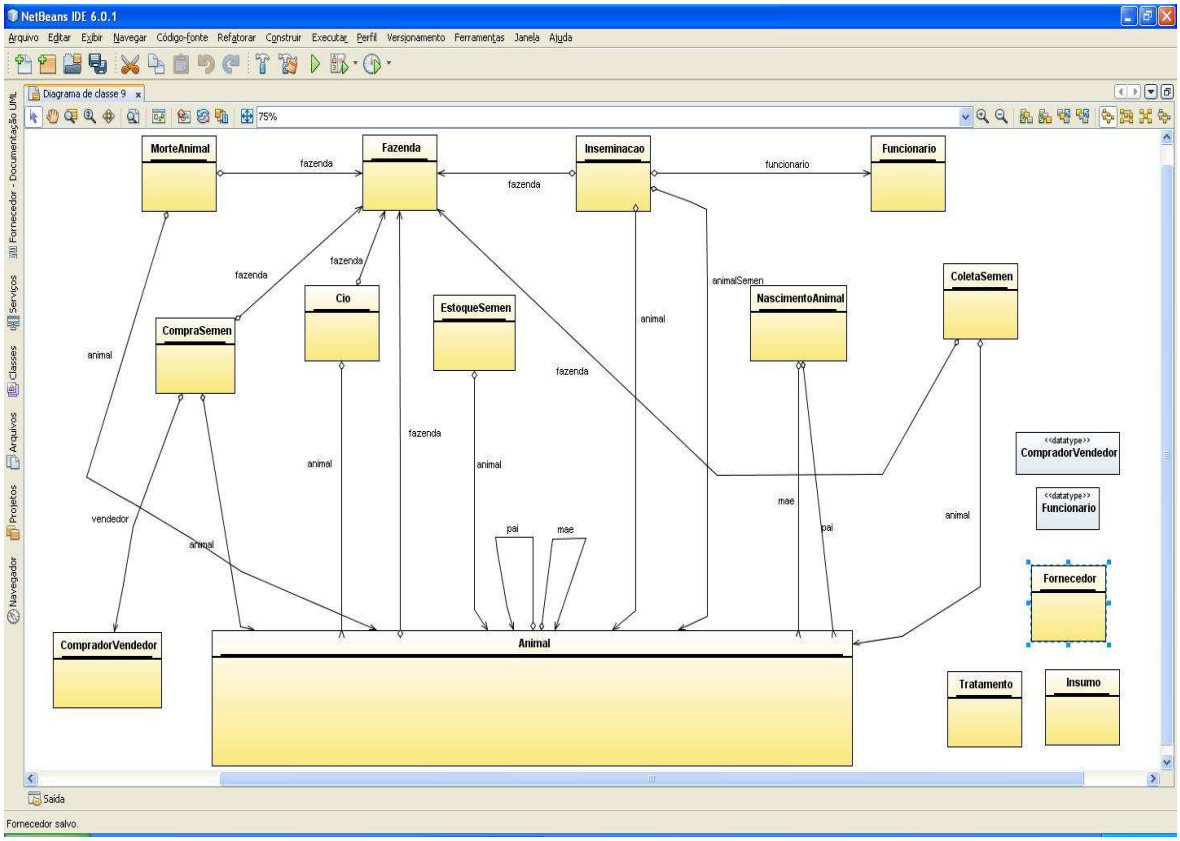

Figura 7-10 – Diagrama de Classes do Modelo de Objetos

Fusion-RE/I	UML
Modelo de Objetos Fusion	Modelo de Objetos UML
Modelo de Ciclo de Vida	Diagrama de Estados
Modelo de Operações	

Figura 7-11 – Modelo de Análise *Fusion-RE/I* e UML

O quadro de índices é proveniente do quadro de chamadas e trata-se de uma lista, em ordem alfabética, dos procedimentos de implementação do software, com suas respectivas localizações (arquivo e diretório). A finalidade desse quadro é agilizar o trabalho de manipulação dos procedimentos. Por tratar-se apenas de uma lista textual, esse

quadro não necessita de uma representação específica em UML. A Tabela 7-1 apresenta a lista de procedimentos de Insumos e de Sêmen do SisGAGRO.

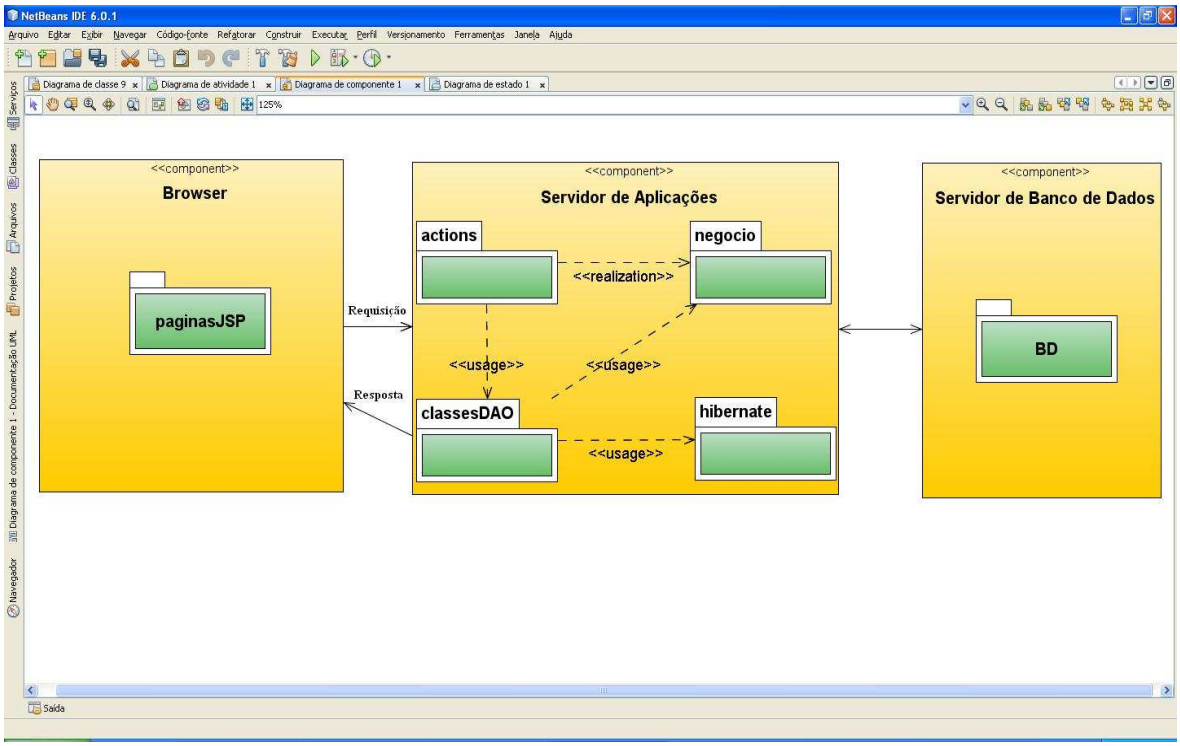

Figura 7-12 – Diagrama de Componentes do Quadro de Chamadas

Tabela 7-1 – Lista de Procedimentos de Implementação de Insumos e Sêmen

Sistema: SisGAGRO – Sistema de Gerenciamento Agropecuário		
Procedimentos		**Diretório/Arquivo**
Insumos	Consultar	sisgagro\build\web\insumos\consultar.jsp
	Editar	sisgagro\build\web\insumos\editar.jsp
	Novo	sisgagro\build\web\insumos\novo.jsp
	Principal	sisgagro\build\web\insumos\principal.jsp
Sêmen	Consultar	sisgagro\build\web\semen\compra\consultar.jsp
	Detalhes	sisgagro\build\web\semen\compra\detalhes.jsp
	Editar	sisgagro\build\web\semen\compra\editar.jsp
	Novo	sisgagro\build\web\semen\compra\novo.jsp
	Principal	sisgagro\build\web\semen\compra\principal.jsp

O objetivo desse quadro é identificar os procedimentos que implementam as operações de interface e classificá-los, de acordo com sua funcionalidade, à interface ou a um dos temas definidos. Por ser uma tabela, a UML não fornece uma representação direta. Não foi encontrado em UML um diagrama para relacionar, diretamente, os procedimentos de implementação com as operações da interface com o usuário, como acontece no Quadro de Operações – Procedimentos de Implementação. Assim, optou-se por deixar o mapeamento desse quadro em aberto. A Figura 7-13 mostra resumidamente quais

elementos de UML podem ser usados para a representação das visões estruturais de *Fusion-RE/I*.

Fusion-RE/I	UML
Quadro de Chamadas	Diagrama de Componentes
Quadro de Índice	—
Quadro de Operações – Procedimentos de Implementação	—

Figura 7-13 – Visões Estruturais *Fusion-RE/I* e UML

7.6 Resultados

As três técnicas de engenharia reversa usadas têm focos diferentes. A primeira técnica, análise de estruturas do código-fonte, tem o foco no código-fonte, isto é, nas linhas de código-fonte do software. Ela analisa as estruturas e propõe a geração de diagramas para a representação do software em nível mais alto de abstração.

A segunda técnica, análise de estruturas de banco de dados, tem o foco no banco de dados. Compreender o relacionamento entre as classes do banco de dados é importante para entender as regras de negócio usadas no software. Pôde-se observar que os resultados não apresentam relacionamento entre as classes. Isso ocorreu, pois o desenvolvedor optou por realizar os tratamentos e estabelecer os relacionamentos em nível de aplicação. Em outras palavras, as tabelas do banco de dados são relacionadas no código-fonte.

A terceira técnica, *Fusion-RE/I*, é mais completa e complexa. Ela busca recuperar artefatos desde a concepção do software. As informações necessárias vão desde entrevistas com o interessado na criação do software a criação de um modelo de objetos. Sua robustez é verificada com a quantidade de passos e a relativa dificuldade de executar cada um deles. *Fusion-RE/I* foi criada com a propósito de reunir as melhores técnicas existentes.

Inicialmente, dá-se a recuperação das informações do software antes de iniciar o seu desenvolvimento, por exemplo entrevistas, documentação e manuais. Em seguida, a recuperação do modelo de análise, composto pelo modelo de ciclo de vida, modelo de operações e modelo de objetos. Esta fase inicial compreende a parte de recuperação de visões funcionais. A recuperação de visões estruturais se encarrega da elaboração do quadro de procedimentos de implementação, do quadro de índices de procedimento e do quadro de operações.

Para o software em questão, o uso de *Fusion-RE/I* é menos apropriado, em vista de sua robustez e da relativa simplicidade do software avaliado. A segunda técnica se torna inútil quando o relacionamento das tabelas no banco de dados é substituído pela implementação deste relacionamento no código-fonte do software. Desta forma, a técnica mais apropriada para explorar no software é a primeira técnica. Ela é relativamente simples em aplicar e atende às necessidades atuais. A presença da implementação do relacionamento das tabelas em nível de aplicação, ou seja, no código-fonte, valoriza o seu uso, visto que o seu foco é o código-fonte.

7.7 Considerações Finais

Neste capítulo, foi realizado o estudo de caso de cada técnica de engenharia reversa do sistema SisGAGRO. O uso das técnicas foi apoiado pelas ferramentas CASE para a execução da tarefa, uma vez que sem o seu uso, a engenharia reversa seria árdua.

Além disso, foi discutido o porquê de escolher UML para a representação dos modelos recuperados pelo uso de *Fusion-RE/I*. Com o estudo de caso, foi possível chegar à conclusão de que as técnicas análise de estruturas do código-fonte e análise de estruturas do banco de dados são simples, ao passo que *Fusion-RE/I* é mais robusta, mais completa e mais complexa.

8 ANÁLISE COMPARATIVA DA CARACTERÍSTICA DE QUALIDADE DE USABILIDADE DAS FERRAMENTAS CASE

8.1 Considerações Iniciais

Este capítulo apresenta uma análise comparativa da característica de qualidade de usabilidade das ferramentas CASE para engenharia reversa de software à luz da norma ISO/IEC 9126 [ISO/IEC 9126-1, 2001].

A seção 8.2 define alguns critérios de avaliação das ferramentas CASE. A seção 8.3 apresenta o resultado da avaliação das ferramentas CASE segundo os critérios definidos.

8.2 Critérios de Avaliação das Ferramentas CASE

Foi realizada uma avaliação das oito ferramentas CASE de engenharia reversa. Para cada ferramenta CASE, as sub-características inteligibilidade, apreensibilidade, operacionalidade e atratividade foram consideradas. Foram elaboradas algumas perguntas (critérios) baseadas nos trabalhos de Vilella (2003), Filgueiras (2003), Winckler; Pimenta (2008), Pagliuso (2004), Nielsen (1993) e Bevan (1998). A Tabela 8-1, a Tabela 8-2, a Tabela 8-3 e a Tabela 8-4 apresentam 20 perguntas, divididas em 4 grupos de 5 perguntas, representando as quatro sub-características de usabilidade abordadas. Associada a cada pergunta, existe uma justificativa, caracterizando a relevância do seu uso.

8.3 Avaliação das Ferramentas CASE

A Tabela 8-5 apresenta o resultado da avaliação preliminar realizada pelo autor deste trabalho. A subjetividade pode ser amenizada pelas considerações e reflexões dos resultados. A avaliação foi realizada seguindo o padrão sugerido no formulário de avaliação de Companhia de Saneamento do Distrito Federal [CAESB, 2008]:

- Atende Totalmente (2 pontos);
- Atende Parcialmente (1 ponto);
- Não Atende (0 ponto).

Os resultados obtidos permitem identificar as ferramentas CASE NetBeans (em relação ao código-fonte), 37 pontos, e DBWrench (em relação ao banco de dados), 33 pontos, detentores dos resultados mais significativos.

Tabela 8-1 – Perguntas e Justificativas para a Sub-característica Inteligibilidade[3]

		Pergunta	Justificativa
Inteligibilidade	1	**É fácil identificar pela interface da ferramenta CASE onde e como realizar a tarefa sem ajuda de *help* e afins?**	Consultar manuais e ajuda é razoavelmente entediante. É preferível prezar pela visibilidade para possibilitar a execução de uma tarefa sem consultas cansativas.
	2	**A ferramenta CASE segue uma forma padrão? (Não é diferente das ferramentas CASE existentes e desempenham o mesmo papel)**	A padronização é fundamental para que a ferramenta CASE seja entendida. A ferramenta CASE ter um padrão firmado é, de certa forma, seguir uma estrutura com a qual os usuários estão acostumados.
	3	**Componentes da interface com o usuário, por exemplo, *menus*, caixas de texto ou listas de seleção, são utilizados respeitando as suas características funcionais?**	A organização adequada de elementos da interface é fundamental para o usuário adquirir o domínio da ferramenta CASE sem complicações e ter condições de determinar qual é melhor para uma tarefa.
	4	**Os ícones de navegação são usados de forma a efetivamente ajudar o usuário a reconhecer imediatamente uma classe de itens?**	A facilidade de reconhecer uma classe de itens pelos ícones de navegação (botões, *menus*, etc.) possibilita o usuário visualizar os recursos que a ferramenta CASE oferece de forma mais ágil.
	5	**É fácil entender os conceitos usados?**	O entendimento dos conceitos usados pela ferramenta CASE é importante para que o esforço do usuário em compreender o conceito lógico e a aplicabilidade da ferramenta CASE seja mínimo.

Tabela 8-2 – Perguntas e Justificativas para a Sub-característica Atratividade[4]

		Pergunta	Justificativa
Atratividade	1	**As funções da ferramenta CASE são facilmente percebidas pelo usuário?**	A atração e a motivação do usuário são estimuladas quando as funções da ferramenta CASE são vistas sem muito esforço.
	2	**A ferramenta CASE possui ícones que auxiliam o entendimento de funções?**	"Uma imagem vale mais do que mil palavras"
	3	**A ferramenta CASE possui sistema de cores para representação e visualização das tarefas?**	As cores são um exemplo de satisfação subjetiva.
	4	**A ferramenta CASE usa linguagem do usuário?**	Usar a linguagem do usuário é essencial para atraí-lo. Trabalhar com uma ferramenta CASE a qual não se entende a nomenclatura é desestimulante, além de dificultar a intuição ao tentar realizar uma tarefa.
	5	**A ferramenta CASE mantém consistência a situações comuns, evitando o usuário adivinhar se diferentes palavras/ações têm a mesma semântica?**	A existência de sinônimos na ferramenta CASE pode deixar o usuário confuso. Uma ferramenta CASE coesa e consistente possui valor subjetivo.

[3] Baseada em Marçal; Beren (2005), Vilella (2003), Filgueiras (2003) Pimenta (2008), Pagliuso (2004).
[4] Retirada de Nielsen (1993)

Tabela 8-3 – Perguntas e Justificativas para a Sub-característica Apreensibilidade[5]

		Pergunta	Justificativa
Apreensibilidade	1	**A ferramenta CASE sugere novas funções, isto é, à medida que é usada, é possível perceber a quantidade de recursos que ela oferece?**	É importante a sugestão de funções relacionadas à função usada. É possível que uma função sugerida seja a próxima a ser usada. A sua sugestão facilita uso da ferramenta CASE, diminui o tempo e proporciona satisfação ao usuário.
	2	**Recursos para facilitar a apreensão do funcionamento da aplicação são facilmente identificáveis?**	Seções de ajuda e FAQ's[6] disponíveis e facilmente identificáveis colaboram para apreensão da ferramenta CASE.
	3	**A ferramenta CASE possui clareza do *help*? O mecanismo de ajuda da ferramenta CASE é fácil de entender?**	Possuir um *help* que não seja fácil de entender anula a vantagem que ele traz, pois o propósito para o qual ele foi feito não é alcançado com razoável facilidade.
	4	**A ferramenta CASE é auto-explicativa e clara na execução de suas funções?**	Quando uma função é executada de forma auto-explicativa e clara, o esforço do usuário em aprender a usar a ferramenta CASE é menor.
	5	**A ferramenta CASE possui mecanismos de memória que registram passos da última tarefa realizada?**	A memorização da última tarefa por parte da ferramenta CASE é um mecanismo que facilita o aprendizado do usuário, pois a memória do ser humano é auxiliada pela ferramenta CASE.

Tabela 8-4 – Perguntas e Justificativas para a Sub-característica Operacionalidade[7]

		Pergunta	Justificativa
Operacionalidade	1	**A ferramenta CASE é prática, isto é, não requer muito esforço para realizar determinada tarefa?**	A praticidade da ferramenta CASE motiva o usuário a usá-la, pois realizar uma tarefa em vez de complicada é simples e prática.
	2	**A ferramenta CASE realiza a mesma tarefa por diferentes caminhos?**	A mesma tarefa realizada de uma forma diferente oferece flexibilidade para a sua execução, proporcionando ao usuário optar pela que melhor se adapte.
	3	**A ferramenta CASE realiza as tarefas de modo satisfatório?**	O início da tarefa e a obtenção e a exibição dos resultados de forma satisfatória cooperam para a aceitação da ferramenta CASE.
	4	**O grau de dificuldade para a localização de uma função na ferramenta CASE é relativamente baixo?**	Quanto mais fácil for localizar uma função na ferramenta CASE, mais fácil será o aprendizado. O esforço do usuário para operar e controlar a operação é menor.
	5	**O acesso direto às tarefas de alta prioridade é oferecido na barra de ferramentas?**	Os recursos presentes na barra de ferramentas possibilitam a praticidade e a agilidade. Em se tratando de tarefas de alta prioridade, o resultado é mais visível.

[5] Baseada em Marçal; Beren (2005), Vilella (2003), Filgueiras (2003) Pimenta (2008), Pagliuso (2004)
[6] *Frequently Asked Questions*
[7] Baseada em Marçal; Beren (2005), Vilella (2003), Filgueiras (2003) Pimenta (2008), Pagliuso (2004)

Tabela 8-5 – Resultado da Avaliação da Usabilidade das Ferramentas CASE

		Engenharia Reversa do Código-Fonte				Engenharia Reversa do Banco de Dados			
Critérios		**ArgoUML**	**WithClass**	**Umbrello**	**NetBeans**	**DBWrench**	**DBVisualizer**	**DBConstructor**	**SQuirreL**
Inteligibilidade	1	1	1	1	2	2	1	1	1
	2	2	2	2	2	2	2	1	2
	3	2	2	2	2	2	2	2	2
	4	1	2	2	2	1	2	1	1
	5	1	1	1	2	2	1	1	2
Sub-total		**7**	**8**	**8**	**10**	**9**	**8**	**6**	**8**
Apreensibilidade	1	1	1	2	2	2	1	1	1
	2	1	1	1	2	1	1	1	1
	3	0	1	1	2	2	1	2	1
	4	1	1	2	2	2	2	1	2
	5	0	2	1	2	1	1	1	1
Sub-total		**3**	**6**	**7**	**10**	**8**	**6**	**6**	**6**
Operacionalidade	1	1	2	2	2	2	2	2	1
	2	1	1	1	1	1	1	1	1
	3	2	2	1	2	2	2	2	2
	4	1	1	1	1	1	1	1	1
	5	2	2	2	2	2	2	2	2
Sub-total		**7**	**8**	**7**	**8**	**8**	**8**	**8**	**7**
Atratividade	1	1	1	1	2	1	1	1	1
	2	1	2	2	2	2	1	2	2
	3	2	1	2	1	2	2	2	1
	4	1	2	1	2	2	2	2	2
	5	2	1	2	2	1	1	1	1
Sub-total		7	7	8	9	8	7	8	7
Total		**24**	**29**	**30**	**37**	**33**	**29**	**28**	**28**

Para uma análie pouco mais criteriosa, foram escolhidos cinco resultados de cada categoria das Ferramentas CASE (Engenharia Reversa do Código-Fonte e Engenharia Reversa do Banco de Dados) é realizada uma reflexão. Priorizou-se a escolha de resultados com maior discrepância para esta reflexão ser mais significativa, isto é, que abordasse pontos mais críticos. A Tabela 8-6 e a Tabela 8-7 apresentam o comentáriodos resutados obtidos na avaliação das Ferramentas CASE.

Tabela 8-6 – Reflexão dos Resultados Referentes às Ferramentas CASE para Engenharia Reversa do Código-Fonte

Resultados	Reflexão
Critério Inteligibilidade (5ª pergunta)	Os conceitos utilizados pela ferramentas NetBeans são mais intuitivos. A ferramenta Netbeans utiliza a linguagem do usuário.
Critério Apreensibilidade (3ª pergunta)	O *help* da ferramenta ArgoUML não oferece ajuda. Por incrível que pareça, ele oferece apenas informações técnicas do software. Já o *help* da ferramenta NetBeans, além de tudo, possui suporte online oferecendo ajuda/tutoriais inclusive em vídeo.
Critério Apreensibilidade (5ª pergunta)	Além de a ferramenta ArgoUML não possuir recurso de memorização da última tarefa, a visualização adequada do que se está fazendo às vezes é comprometida. Quando se realiza a engenharia reversa, apesar de ter os relacionamentos corretos, as classes ficam sobrepostas, dando impressão de que existe apenas uma única classe.
Critério Operacionalidade (2ª pergunta)	A possibilidade de realizar uma tarefa utilizando a barra de menu ou os botões exibidos na barra de ferramentas, por exemplo, são evidenciadas de forma equivalente em cada uma das ferramentas. Porém, nem tudo o que se pode realizar através dos menus é possível realizar por outro caminho.
Critério Atratividade (1ª pergunta)	A forma de exibição dos recursos da ferramenta NetBeans permite identificar com facilidade as funções da ferramenta. O grande diferencial é a exibição de várias “mini-janelas” como paleta, propriedades, projects, entre outros.

Tabela 8-7 – Reflexão dos Resultados Referentes às Ferramentas CASE para Engenharia Reversa do Banco de Dados

Resultados	Reflexão
Critério Inteligibilidade (1ª pergunta)	A realização da engenharia reversa do banco de dados foi feita sem complicações na ferramenta DBWrench, ao contrário das outras ferramentas.
Critério Apreensibilidade (4ª pergunta)	Para a realização da engenharia reversa do banco de dados, o esforço utilizando a ferramenta DBConstructor foi bem maior, já que a dificuldade para compreender o que a ferramenta estava executando era evidente.
Critério Operacionalidade (1ª pergunta)	O esforço dispendido para realizar engenharia reversa utilizando a ferramenta SquirreL foi bem maior que as outras ferramentas. É uma ferramenta simples, mas por outro lado, um tanto mais complicada.
Critério Atratividade (2ª pergunta)	Alguns ícones utilizados pela ferramenta DBVisualizar não são tão intuitivos.
Critério Atratividade (3ª pergunta)	A ferramenta SquirreL é a que possui a interface mais simples das ferramentas analisadas. As cores não são tão exploradas.

8.4 Considerações Finais

Na linha de raciocínio, a preocupação com a interface deve ser constante durante o desenvolvimento de software – para os desenvolvedores – e durante a escolha de uma ferramenta CASE que possa auxiliar as atividades de desenvolvimento. A avaliação permite identificar as ferramentas que atendem a critérios de usabilidade, caracterizando o atendimento de um dos pontos da qualidade de software.

9 CONSIDERAÇÕES FINAIS

A seção 9.1 apresenta uma breve conclusão da relevância do trabalho. A seção 9.2 apresenta as contribuições do trabalho. A seção 9.3 faz uma síntese dos trabalhos que podem ser desenvolvidos com base nesta pesquisa, objetivando a sua continuidade.

9.1 Conclusões

Este trabalho permitiu verificar a importância da engenharia reversa no processo de desenvolvimento de software, bem como a documentação ser um apoio relevante para o software. A manutenção muitas vezes é árdua por falta do entendimento do software, das regras de negócio, entre outros. As técnicas de engenharia reversa estudadas, juntamente com as ferramentas CASE possibilitando a automação do processo, permitem recriar ou atualizar a documentação existente, de forma que reflita o sistema atual e facilite a compreensão do software.

Existem muitas ferramentas CASE no mercado que realizam engenharia reversa do código-fonte e do banco de dados. Porém, algumas delas deixam a desejar em quesitos de qualidade (por exemplo, usabilidade). O presente trabalho possibilitou verificar pontos fracos e pontos fortes de algumas ferramentas CASE no que diz respeito à usabilidade, classificando algumas como maior usabilidade que outras. Essa avaliação é útil para quem deseja utilizar estas ferramentas CASE para realizar engenharia reversa e precisa de um parâmetro para escolher uma ferramenta CASE.

Como visto, a usabilidade está associada aos seguintes princípios: i) facilidade de aprendizado; ii) facilidade de lembrar como realizar uma tarefa após algum tempo; iii) rapidez no desenvolvimento de tarefas; iv) baixa taxa de erros; e v) satisfação subjetiva do usuário. Com baixa usabilidade, uma ferramenta CASE pode afetar a interatividade, comprometendo o processo de engenharia reversa.

9.2 Contribuições

Foi realizada uma avaliação de oito ferramentas CASE de engenharia reversa quanto a usabilidade. Estas ferramentas CASE foram organizadas em duas categorias: i) quatro ferramentas CASE que realizam engenharia reversa do código-fonte; e ii) quatro ferramentas CASE que realizam engenharia reversa do banco de dados. Esta avaliação

permite identificar pontos fracos e pontos fortes de cada ferramenta CASE, fundamentais no momento da escolha de uma ferramenta CASE para realizar engenharia reversa.

O estudo de caso, usando as três técnicas – *Fusion-RE/I*, análise de estruturas do código-fonte e análise de estruturas do banco de dados – e apresentando detalhadamente os resultados, possibilita a identificação do foco de cada técnica de engenharia reversa, proporcionando a escolha adequada de uma técnica para um software, de acordo com o tipo de implementação e com a robustez do software.

A análise das ferramentas CASE permite verificar qual tem maior grau de usabilidad. O conjunto de perguntas elaborado com base em alguns artigos, livros e na norma ISO/IEC 9126-1, que tratam da usabilidade de uma software, facilita a avaliação das ferramentas CASE.

9.3 Trabalhos Futuros

Como propostas de trabalhos futuros, pretende-se aplicar técnicas de reengenharia em um software no qual se fez engenharia reversa. Existem muitos softwares legados atualmente e a migração para linguagens mais atuais, isto é, em constante uso no mercado não é uma tarefa simples, muito menos a sua manutenção. Por isso, estudar técnicas de reengenharia focadas na migração de um software se faz uma proposta de trabalho interessante.

Como sugestão de trabalhos futuros, pretende-se realizar estudos de reengenharia para adaptar modelos que foram classificados como incompletos após a engenharia reversa. Outra sugestão é o estudo de outras técnicas de engenharia reversa, bem como de outras ferramentas CASE, pois novas ferramentas CASE surgem para suprir funções que as outras deixam de atender e para acompanhar o avanço da tecnologia.

Outra sugestão de trabalhos futuros é o refinamento do conjunto de perguntas sobre a usabilidade das ferramentas CASE. Para obter resultados mais precisos da avaliação, pode-se formar um grupo de pessoas devidamente capacitadas para realizar a avaliação individual e, em seguida, fazer uma média para cada ferramenta CASE. Além disso, pode-se utilizar dos variados testes de usabilidade, bem como das heurísticas de usabilidade propostos no capítulo 6.

Analisar as ferramentas CASE abordadas neste trabalho segundo a qualidade e confiabilidade do resultado gerado é outra sugestão de trabalhos futuros. Não se justifica utilizar uma ferramenta que seja fácil de usar e não gere resultados válidos. A verificação destes resultados é tão quanto ou mais importante que a usabilidade da ferramenta.

Por fim, pode-se avaliar estas ferramentas CASE segundo as demais características de qualidade apresentadas na norma ISO/IEC 9126.

REFERÊNCIAS BIBLIOGRÁFICAS

ACM SIGCHI. **"Curricula for Human-Computer Interaction".** Tehnical Report. NY: ACM. 1992 Disponível em: <http://www.acm.org/sigchi.>

Anquetil, N., Oliveira, K. M. **Processo de Redocumentação: Uma Necessidade. Simpósio Brasileiro de Qualidade de Software.** Universidade Católica de Brasília, 2002.

ArgoUML. Disponível em: <http://argouml.tigris.org/> Acesso em: Fevereiro 2008

Benedusi, P.; Cimitile, A.; Carlini, U. **Reverse Engineering Processes, Design Document Production, and Structure Charts.** Journal Systems and Software. v. 19. p. 225-245. 1992.

Bevan, N. **Usability Issues in Web Site Design**. Proceedings of UPA'98. Washington DC. p. 22-26. Disponível em: <http://www.usability.serco.com/papers/usweb98.pdf>. Acesso em: Fevereiro 2008.

Biggerstaff, T. **Design recovery for maintenance and reuse.** IEEE Software, 22(7):36.49, July 1989.

Booch, G.; Rumbaugh, J; Jacobson, I. **UML – Guia do Usuário.** Editora Campus, 2000, Rio de Janeiro.

Bossonaro, A. A. **Estratégia de Reengenharia de Software usando Transformações**. T. J. Disponível em: <http://www.recope.dc.ufscar.br>. Acesso em: Junho 2008.

CAESB. Companhia de Saneamento do Distrito Federal. **Edital de Chamada de Projetos para o Programa de Responsabilidade Social da CAESB.** 2008. Disponível em: <http://www.caesb.df.gov.br/SCRIPTS/Downloads/EDITAL%20RESP.SOCIAL%202007-08.pdf>. Acesso em: Setembro 2008.

Chikofsky, E. J.; Cross II, J. H. **Reverse Engineering and Design Recovery: A Taxonomy.** IEEE Software, v.7, n.1, p.13-17, 1990.

Coleman, D.; Arnold, P.; Bodoff, S. **Desenvolvimento Orientado a Objetos: O Método Fusion.** Ed. Campos, Rio de Janeiro, 1996.

Costa, H. A. X. **Critérios e Diretrizes de Manutenibilidade para a Construção do Modelo de Projeto Orientado a Objetos.** Tese de Doutorado. Escola Politécnica da Universidade de São Paulo. 2005. 199p.

Costa, R. M. **Um Método de Engenharia Reversa para Auxiliar a Manutenção de Software.** Dissertação de Mestrado. Universidade Federal de São Carlos, 1997. 100p.

Costa, R. M. da; Sanches, R. **Ferramentas de Engenharia Reversa no Apoio à Qualidade de Software.** Instituto de Ciências Matemáticas e de Computação. Relatórios Técnicos do ICMC/USP. São Carlos. 1996. Disponível em:

<ftp://ftp.icmc.usp.br/pub/BIBLIOTECA/rel_tec/RT_045.pdf>. Acesso em: Outubro 2008.

Cybis, W.A; Betiol, A.H.; Faust, R. **Ergonomia e Usabilidade – Conhecimentos, Métodos e Aplicações.** Novatec Editora. 2007.

Dantas Filho, J. L. R.; Esperanço, C. P.; Silveira, D. S.; Schmitz, E. A. **Engenharia Reversa em Sistemas de Informação Utilizando XMI.** Resumo, PUC-Rio, 2000.

DBConstructor. Disponível em: <http://www.dbconstructor.com/home/home.aspx>. Acesso em: Fevereiro 2008.

DBVisualizer. Disponível em: <http://www.dbvis.com/products/dbvis/>. Acesso em: Setembro 2008.

DBWrench. Disponível em: <http://www.dbwrench.com/>. Acesso em: Fevereiro 2008.

Feltrim. V. D. **Apoio à Documentação de Engenharia Reversa de Software por Meio de Hipertextos.** Dissertação de Mestrado. Universidade Federal de São Carlos. 1999.

Feltrim, V. D. **Apoio à Documentação de Engenharia Reversa de Software por meio de Hipertextos.** In: III Workshop de Teses em Engenharia de Software, 1998, Maringá. III Workshop de Teses em Engenharia de Software, 1998. p. 1-4.

Filgueiras, L. V. L. **Engenharia da Usabilidade. Laboratório de Tecnologia de Software.** Seminário de Pesquisa, 2003. Disponível em: <http://www.poli.usp.br/pro/procsoft/tpcsepusp04.pdf>. Acesso em: Setembro 2008.

Fukuda, A. P. **Refinamento Automático de Sistemas Orientados a Objetos Distribuídos.** Dissertação de Mestrado. Universidade Federal de São Carlos. 2000.

Harandi, M. T.; Ning, J. Q. **Knowledge-Based Program Analysis.** IEEE Software, v. 7, n. 1, p. 74-81, jan.1990.

Iida, I. Ergonomia. Projeto e produção. São Paulo: Edgard Blücher, 1997.

ISO 9241-11: **Ergonomic requirements for office work with visual display terminals (VDTs). Part 11 — Guidelines for specifying and measuring usability.** Gènève: International Organisation for Standardisation. 1997.

ISO/IEC 9126-1. **Software Engineering – Product Quality – Part 1: Quality Model.** International Organization for Standardization 2001.

Issa, L. V. N. **Desenvolvimento de Interface com o Usuário Dirigido por Modelos e Geração Automática de Código.** Dissertação de Mestrado. Universidade Federal de Minas Gerais. 2006.

Jabur, W. P. **Engenharia Reversa de Software.** Florianópolis. I Congresso sobre Direito de Autor e Interesse Público. 2007.

Jesus, E. S. **Engenharia Reversa de Sistemas Legados Usando Transformações.** Dissertação de Mestrado. Universidade Federal de São Carlos. 2000.

Jung, C. F. **Metodologia para Pesquisa e Desenvolvimento: aplicada a novas tecnologias, produtos e processos.** Axcel Books do Brasil Editora, Rio de Janeiro, RJ, 2004

Lehman, M. M.; Belady, L. **Program Evolution: Processes of Software Change.** Academic Press, 1985. 538p.

Lucas, F. R.; Muñoz, H. J.; Santana, M.; Neto, M. **Lírio – Uma Ferramenta para Criação de Diagramas de Seqüência Utilizando Engenharia Reversa.** Salvador/BA. Faculdade Ruy Barbosa de Ciência da Computação, 2005.

Marçal, E. K.; Beren, I. M. **Auditoria da Qualidade de um *Software* de Contabilidade.** I Seminário de Ciências Contábeis. Blumenau/SC, 2005.

Marconi, M. A.; Lakatos, E. M. **Fundamentos de Metodologia Científica.** Editora Atlas, São Paulo, SP, 2003.

Martins, C. M., Pimenta, M. S., Price, A. M. A. **Migração de aplicações cliente/servidor para a plataforma Web usando Metamodelos.** The Second-Ibero American Symposium on Software Engineering and Knowledge Engineering October 2002.

Martins, C. R. L. **Estratégia de Migração de Aplicações Legadas Virtuais (tipo WIMP) para o Ambiente Web.** Dissertação de Mestre em Ciência da Computação. Universidade Federal do Rio Grande do Sul. 2003.

Masiero, P. C. **Análise Orientada a Objetos: Uma Introdução ao Método Fusion. IX Simpósio Brasileiro de Engenharia de Software.** Documento preparado como apoio ao tutorial homônimo. Recife. 1995.

Mayhew, D.J. **The usability engineering lifecycle: a practitioner's handbbok for user interface design.** San Francisco: Morgan Kaufmann. 1999.

UMLNetbeans. Disponível em: <http://www.netbeans.org/kb/55/uml-activity-diagram_pt_BR.html#intro>. Acesso em: Outubro 2008.

Netbeans, IDE. Disponível em: <http://www.netbeans.org/> Acesso em: Fevereiro 2008.

Nielsen, J. **Usability Engineering.** Academic Press, Cambridge, MA, 1993.

Novais, E. R. A, Prado, A. F. **Reengenharias de Software Orientadas a Componentes Distribuídos.** XV Simpósio Brasileiro de Engenharia de Software, Universidade Federal de São Carlos. 2001.

Novais, E. R. A.; **Reengenharias de Software Orientadas a Componentes Distribuídos.** São Carlos/SP, 2002, Dissertação de Mestrado, Universidade Federal de São Carlos.

ODMG. Object Data Management Group. Localização: <http://www.odbms.org/odmg.html>. Acesso em: Outubro 2008.

Oman, Paul W.; Cook, Curtis R. The book paradigm for improved maintenance. IEEE Software, v. 7, n. 1, p. 39-45, jan. 1990

OMG. The Object Management Group. Disponível em: <http://www.omg.org/>. Acesso em: Outubro 2008.

Pagliuso, P. B. B. **Método para Avaliação de Interface Web Baseado nos Princípios de Usabilidade – AvalUWeb.** Universidade Estadual de Campinas. Pós-Graduação em Engenharia Mecânica. 2004.

Penteado, R. A. D. **Um Método para Engenharia Reversa Orientada a Objetos.** São Carlos-SP, 1996. Tese de Doutorado. Universidade de São Paulo. 251p.

Penteado, R. A. D.; Lemos, G. S. **Garantia da Qualidade nos Processo de Engenharia Reversa e Reengenharia.** São Carlos/SP, 1999. Programa de Pós-Graduação/Ciência da Computação. Universidade Federal de São Carlos.

Peres, D. R.; Alvaro, A; Fontanette, V.; Garcia, V. C.; Prado, A. C.; Braga, R. T. V. **TB-REPP - Padrões de Processo para a Engenharia Reversa baseado em Transformações.** Universidade Federal de São Carlos, 2003.

Pfleeger, S. L. **Software Engineering: Theory and Practice.** 2ed. Prentice Hall, 2001.

Pressman, R. S. **Engenharia de Software.** São Paulo: Makron Books, 2001.

Royas, A. del V.; Marziale, M. H. P. A Situação de Trabalho do Pessoal de Enfermagem no Contexto de um Hospital Argentino: Um Estudo Sob a Ótica da Ergonomia. Revista Latino-Americana de Enfermagem - Ribeirão Preto - v. 9 - n. 1 - p. 102-108. janeiro 2001.

Rugaber, S. **Program Comprehension for Reverse Engineering.** AAAI Workshop on AI and Automated Program Understand, San Jose, California, p.106-110. July 1992. Disponível em: <http://www.cc.gatech.edu/reverse/papers.html>. Acesso em: Abril 2008

Rugaber, S.; Leblanc, R. J.; Ornburn, S. B. **Recognizing Design Decisions in Programs.** IEEE Software, v.7, n.1, p.46-54, 1990.

Sage, A. P. **Systems Engineering and Systems Management for Reengineering.** Journal Systems and Software, v.30, n.1, p.03-25, 1995

Schneider, R. L. **Engenharia Reversa na Engenharia de Software.** UFRJ, 2001.

Shneiderman, B. (1998) **"Designing the User Interface: Strategies for Effective Human-Computer Interaction"**, 3ª ed., Addison-Weslley.

Schneidewind, N. F. **The State of Software Maintanance**, IEEE Tranc. On Software Engineering, v.13 n.3, p. 303-310, 1987.

Silveira, D. S. **Fast Case uma Ferramenta Case para o Desenvolvimento Visual de Sistemas Orientados a Objetos.** 1999. Disponível em:

<http://www.inf.ufsc.br/~sbes99/anais/Sessao-Ferramenta-Completo/12-fastcase.pdf> Acesso em: Maio 2008

Sommerville, I. **Software Engineering.** Addison Wesley. 2001, São Paulo.

Squirrel. Disponível em: <http://www.squirrelsql.org/> Acesso em: Setembro 2008.

Stephen, R. M.; Lynn, M. M. **Software Migration and Reengineering: A Pilot Project in Reengineering.** Journal Systems and Software, v.30, n.1, p.137-50, 1995.

Umbrello. Disponível em: <http://www.umbrello.org/> Acesso em: Fevereiro 2008.

UML. **Unified Modeler Language.** Disponível em: <http://www.uml.org/> Acesso em: Junho 2008.

Valdestilhas, A.; Almeida, F. A. **A Usabilidade no desenvolvimento de aplicações para TV interativa.** Laboratório de Interação, Comunicação e Mídia. São José dos Campos, 2005

Veronese, G.; Correa, A.; Werner C.; Netto, F. J. **ARES: Uma Ferramenta de Engenharia Reversa Java-UML.** COOPE/UFRJ, Programa de Engenharia de Sistema e Computação. IM-DCC/UFRJ. 2002. XVI Simpósio Brasileiro de Engenharia de Software.

Vilella, R. M. **Conteúdo, Usabilidade e Funcionalidade: Três Dimensões para a Avaliação de Portais Estaduais de Governo Eletrônico na Web.** UFMG. Escola de Ciência da Informação. Dissertação de Mestrado. 2003.

Voxxel. **With Class uma ferramenta CASE para uso pessoal.** Developers Magazine, Ano 2, 1998.

Waters, R. C.; Chikofsky, E. J. **Reverse Engineering: Progress Along Many Dimensions.** Communications of the ACM. v.37, n.5, p.23-4, 1994.

Winckler, M.; Pimenta, M. S. **Avaliação de Usabilidade de Sites Web.** Disponível em: <http://www.funtec.org.ar/usabilidadsitiosweb.pdf>. Acesso em: Outubro 2008.

WithClass. Disponível em: <http://www.microgold.com>. Acesso em: Maio 2008.