UNIVERSIDADE ESTADUAL DO NORTE DO PARANÁ

***CAMPUS* LUIZ MENEGHEL**

WANDERCY ALVES JUNIOR

REFATORAÇÃO DE PROGRAMAS EM DELPHI

Bandeirantes

2009

WANDERCY ALVES JUNIOR

REFATORAÇÃO DE PROGRAMAS EM DELPHI

Trabalho de conclusão de curso submetido à Universidade Estadual do Norte do Paraná - Campus Luiz Meneghel, como requisito parcial para obtenção do grau de Bacharel em Sistemas de Informação.
Orientador: Prof. José Reinaldo Merlin

Bandeirantes
2009

WANDERCY ALVES JUNIOR

REFATORAÇÃO DE PROGRAMAS EM DELPHI

Trabalho de conclusão de curso submetido à Universidade Estadual do Norte do Paraná - Campus Luiz Meneghel, como requisito parcial para obtenção do grau de Bacharel em Sistemas de Informação.
Orientador: Prof. José Reinaldo Merlin

COMISSÃO EXAMINADORA

_____
Prof. José Reinaldo Merlin
UENP – *Campus* Luiz Meneghel

_____
Prof. Carlos Eduardo Ribeiro
UENP – *Campus* Luiz Meneghel

_____
Prof. Cristiane Yanase Hirabara Castro
UENP – *Campus* Luiz Meneghel

Bandeirantes, __ de_____ de 2009

DEDICATÓRIA

À minha família, Camila e Miguel, que são as pessoas mais importantes na minha vida.

AGRADECIMENTOS

Agradeço a Deus pela minha vida e pelas coisas boas que já me aconteceram.

Aos meus pais que sempre me aconselharam, e me apoiaram em minhas escolhas.

Agradeço ao meu orientador que me proporcionou este desafio e me ajudou a concluir este trabalho.

"Se eu soubesse antes o que sei agora

erraria tudo exatamente igual..."

Humberto Gessinger

RESUMO

A manutenção em *software* é uma tarefa difícil de ser realizada e de custo elevado, pois para fazer qualquer alteração no código é necessário antes compreendê-lo, e isso leva tempo. Os problemas na manutenção são geralmente resultado de falhas no desenvolvimento do *software*. Uma atividade que pode reduzir os problemas de manutenção é a refatoração, que visa tornar o código mais simples e mais fácil de compreender, reduzindo o tempo gasto nas alterações. Neste trabalho foram estudadas algumas técnicas de refatoração, que foram aplicadas em um sistema escrito em *Delphi*.

Palavras-chave: Manutenção de *Software,* Refatoração, Reengenharia de *Software, Delphi*.

ABSTRACT

The maintenance of software is a difficult task to be performed and it has a high cost, because to make any changes in the code is necessary to understand it before, and that takes time. The problems in maintenance are usually the result of failures in software development. An activity can reduce problems of maintenance is *refactoring*, to make the code simpler and easier to understand, reducing the time spent on changes. In this work we study some techniques of *refactoring*, which they were applied on a system written in Delphi.

Keywords: Software Maintenance, Refactoring, Reengineering, Software, Delphi.

LISTA DE FIGURAS

SUMÁRIO

1. INTRODUÇÃO

Devido aos prazos geralmente apertados, os desenvolvedores de *softwares*, para entregar os sistemas, acabam não se preocupando em deixar o código mais claro e fácil de se entender ou em deixá-lo bem estruturado, causando muitos problemas para a equipe que vai fazer a manutenção futuramente.

A reengenharia é uma maneira de aumentar a vida útil de um *software* já existente traduzindo-o para uma linguagem mais atual, geralmente sem alterar as funções básicas do *software*. É geralmente usada em sistemas legados convertendo-os para novos paradigmas.

Na refatoração, que é um tipo especial de reengenharia, é alterada a estrutura do *software* sem modificar as suas funcionalidades. A linguagem de programação original é mantida e alterações são feitas no código-fonte explorando as boas práticas de programação orientada a objetos.

Neste trabalho algumas técnicas de refatoração foram empregadas para melhorar o código-fonte de parte de um sistema de *software* que foi desenvolvido em *Delphi*, que é um ambiente de desenvolvimento bastante utilizado por desenvolvedores de *softwares* em diversas áreas.

1.1. Motivação e Justificativa

Este projeto foi proposto devido à grande quantidade de sistemas de *software* que são desenvolvidos na linguagem de programação *Delphi* e principalmente devido àqueles que já estão em uso e precisam ser mantidos através de manutenção.

A manutenção não é uma tarefa fácil e exige considerável esforço, pois para fazer qualquer alteração no código é necessário antes compreendê-lo. Muitas vezes o desenvolvedor olha para o código e não consegue fazer isso de imediato e, assim, acaba levando muito tempo para realizar a manutenção do *software*.

Se o sistema fosse projetado de maneira adequada ele não precisaria de tantas linhas de código que acabam gerando redundância e ficando confuso.

A refatoração ajuda a tornar o código mais legível, modificando-o depois que ele é escrito, mas sem causar alterações nas funcionalidades do sistema.

Utilizando essa técnica o desenvolvedor ganha mais tempo na correção de erros e na inclusão de novas funcionalidades, reduzindo os custos e otimizando o programa.

1.2. Objetivos

1.2.1. Objetivo Geral

O objetivo desse trabalho é mostrar várias técnicas e padrões existentes de refatoração (*refactoring*) e selecionar algumas técnicas mais adequadas para aplicar em programa desenvolvido em *Delphi*.

1.2.2. Objetivos Específicos

- Pesquisar as diversas de técnicas de refatoração existentes, e em quais situações elas se aplicam;
- Analisar parte do código-fonte de um sistema desenvolvido em *Delphi*; e
- Refatorar código-fonte analisado, a fim de melhorar sua manutenibilidade.

1.3. Metodologia

Realizou-se um estudo bibliográfico sobre os temas manutenção de *software*, refatoração e desenvolvimento de programas em *Delphi*.

Várias técnicas de refatoração foram escolhidas para serem utilizadas neste trabalho. Depois de feito este estudo algumas refatorações foram empregadas para otimizar o *software* analisado. Realizou-se testes no *software* antes e depois de cada refatoração para verificar se nenhuma funcionalidade foi alterada.

1.4. Organização do Trabalho

Este trabalho está organizado da seguinte forma:

O Capitulo 2 apresenta o estudo bibliográfico que fundamenta os conceitos utilizados no desenvolvimento deste trabalho. Na seção 2.1 está a definição do que é manutenção de *software*, um estudo sobre os custos de manutenção e quais são as maiores dificuldades encontradas na manutenção de um *software*. Na seção 2.2 é definido o conceito de Reengenharia de *Software*, para que serve essa prática, em quais situações ela deve ser empregada. O item 2.3 aborda o tema refatoração, definido o que é refatoração, quais são os tipos de refatoração existentes, e como aplicá-las. A sessão 2.4 fala um pouco sobre a linguagem *Delphi* e os sistemas desenvolvidos nesse ambiente.

No Capitulo 3 estão descritas as etapas do desenvolvimento do trabalho. O Item 3.1 descreve o *software* escolhido e as funções que ele desempenha. No Item 3.2 está a análise do código-fonte antes das refatorações que mostram as primeiras falhas encontradas no código. A seção 3.3 descreve as refatorações feitas no código divididas em dois itens, o primeiro mostra as alterações mais simples feitas logo após a análise do código e o segundo mostra as alterações mais complexas executadas por último.

O Capitulo 4 mostra as conclusões obtidas com a aplicação das refatorações no código-fonte analisado.

2. FUNDAMENTAÇÃO TEÓRICA

2.1. Manutenção de *Software*

A vida útil de um sistema grande e complexo varia de 5 a 10 anos. Todavia não é difícil encontrar sistemas com 15 ou 20 anos, por essa razão os peritos em informática vêem a entrega de um sistema ao usuário como o início de sua vida útil. Desse ponto em diante o sistema irá evoluir até que seja eliminado ou substituído (FOURNIER, 1994).

O *software* é, por natureza, flexível e pode ser mudado. Como os requisitos mudam ao se alterar as circunstâncias de negócios, o *software* que suporta o negócio também deve evoluir e mudar (SOMMERVILLE, 2003).

Sommerville (2003) diz que na manutenção de *software* as mudanças são feitas em respostas aos requisitos modificados, mas a estrutura fundamental do *software* permanece estável.

A partir dessa idéia diz-se que a manutenção de *software* é o processo geral de modificação de um sistema depois que ele foi colocado em uso, mas não envolve mudanças maiores na arquitetura do sistema (SOMMERVILLE, 2003). Então qualquer trabalho feito no *software* depois que ele se torna operacional ou passa para a produção é chamado de manutenção (PARIKH, 1990).

Para Sousa (2002 *apud* BARROS, 1998), o conceito de manutenção de *software* é visto como o processo de modificação de um ou mais programas depois que o *software* foi entregue ao cliente e está sendo usado.

Scussiato (1998) afirma que a manutenção de sistemas de computador deve ser encarada como uma atividade de extrema importância e necessária para adequar os produtos de *software* às necessidades da organização. A manutenção de sistemas não pode ser considerada só como uma fase ou etapa do ciclo de vida de *software* na organização.

Segundo Lima (2001), a atividade de manutenção do sistema de *software* desenvolvida deve ser entendida como um processo de modificação do sistema de *software* ou de seus componentes, a fim de corrigir defeitos, melhorar desempenhos ou outros atributos, ou ainda, adaptá-lo a um ambiente em constante transformação.

Pressman (2002) diz que manutenção de *software* não é só consertar erros. Pode-se definir esse conceito descrevendo quatro atividades que são desenvolvidas depois de um programa ser liberado para o uso. Ele divide as tarefas de manutenção de *software* em quatro tipos: corretiva, adaptativa, perfectiva ou preventiva. E diz que apenas 20% dos trabalhos de manutenção são usados para consertar os erros, e que os outros 80% são gastos em adaptações de sistemas existentes através de modificações no ambiente externo, melhorando o que os usuários solicitaram, e submetendo a aplicação à reengenharia (PRESSMAN, 2002).

Sommerville (2003) define três tipos de manutenção de *software*:

- Manutenção para repara defeitos no *software.*
- Manutenção para adaptar o *software* a um ambiente operacional diferente.
- Manutenção para fazer acréscimo à funcionalidade do sistema ou modificá-la.

2.1.1. Dificuldade de Manutenção

Sousa (2002) diz que a principal dificuldade na execução das tarefas de manutenção origina-se do fato de que a complexidade de grandes sistemas aumenta dramaticamente com o tempo, em grande parte à mudança da estrutura do programa.

Esta dificuldade é composta pela informação imprecisa, desatualizada e violações flagrantes dos padrões de programação da organização.

O alto grau de erro associado com execução das tarefas de manutenção é geralmente resultado direto do aumento da complexidade dos programas

Segundo Pressman (2002), a maioria dos problemas de manutenção de *software* está ligada às deficiências na maneira segundo a qual o *software* foi planejado e desenvolvido. A falta de controle e disciplina no desenvolvimento da engenharia de *software* quase sempre se traduz em problemas durante sua manutenção.

2.1.2. Custos de Manutenção

A manutenção é considerada uma das fases mais longas e de alto custo, pois envolve a correção de erros não descobertos anteriormente (desenvolvimento) e o aumento de funções do sistema à medida que novos requisitos são identificados (SOMMERVILLE, 2003).

De acordo Sommerville (2003) é difícil medir o esforço relativo dedicado aos diferentes tipos de manutenção, mas alguns levantamentos mostraram os resultados apresentados no Gráfico 1.

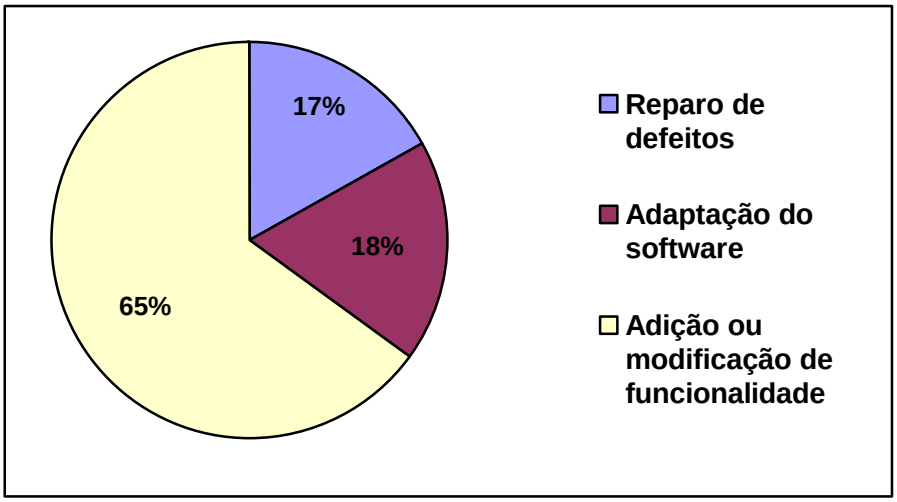

Gráfico 1: Distribuição de Esforço de Manutenção (SOMMERVILLE, 2003).

A partir destes números, pode-se observar que reparar defeitos em um sistema não é a atividade de manutenção mais dispendiosa, em vez disso, a evolução do sistema para atender novos ambientes e a requisitos novos ou modificados consome a maior parte do esforço de manutenção (SOMMERVILLE, 2003).

Pressman (2002) afirma que a manutenção de *software* existente pode ser responsável por mais de 60% de todo o esforço despendido por uma organização de desenvolvimento e a porcentagem continua a crescer à medida que mais *software* é produzido.

Na década de 70 estimava-se que o custo de manutenção em um *software* seria de aproximadamente 40% a 60%. Na década de 80 aumentou para 50% a 70% e na de 90 acreditava-se que seria de 70% a 80% (HUGO, 2002).

Hugo (2002) também mostra o motivo dessa estimativa ser tão alta:

- É difícil ou impossível rastrear o *software*;
- Difícil entender o "programa de outra pessoa";
- A "outra pessoa" não está próxima;
- Documentação não existe ou é ruim;
- Maioria dos *softwares* não é projetada para mudança; e
- Manutenção não é vista como um trabalho glamouroso.

A manutenção geralmente é cara e inevitável devido às mudanças de requisitos, necessidade de ajustes e aumento de funcionalidades antes não previstas (MACORATTI, 2008).

Por que o custo é tão alto?

- Não há controles sobre prazos ou planejamento sobre equipes e recursos;
- O levantamento de requisitos não é feito de forma integrada com o cliente e dentro de padrões definidos; e
- O controle de qualidade é deficiente.

O Gráfico 2 mostra como os custos podem diminuir à medida que mais esforço é dedicado durante o desenvolvimento do sistema a fim de produzir um sistema com maior facilidade de manutenção (SOMMERVILLE, 2003).

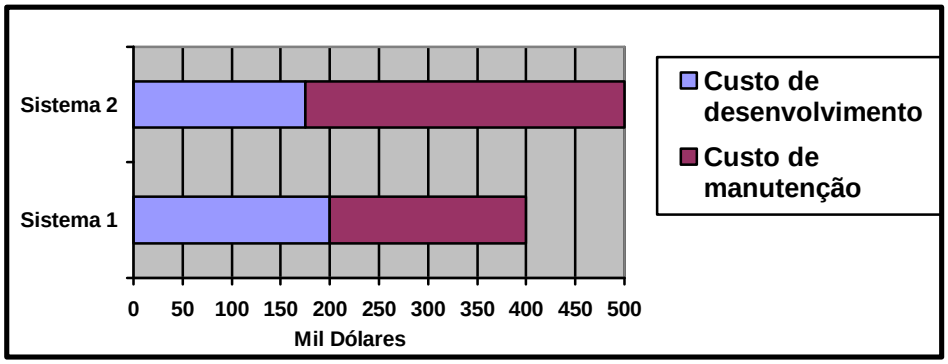

Gráfico 2: Custos de Desenvolvimento e Manutenção

Por reduzir os custos de compreensão, análise e teste há um efeito multiplicador que facilita a manutenção. É importante projetar e implementar um sistema para reduzir os custos de manutenção. É dispendioso acrescentar funcionalidade depois da entrega porque é necessário compreender o sistema e analisar o impacto dessas mudanças.

No sistema 1 (apresentado no Gráfico 2) o custo de 25 mil dólares foi investido em uma manutenção mais fácil. Isso possibilitou economizar 100 mil dólares em manutenção.

Sommerville (2003) concluiu que existem alguns fatores importantes que elevam os custos da manutenção: estabilidade da equipe, responsabilidade contratual, habilidade da equipe e idade e estrutura do programa.

E explica que os três primeiros surgem do fato de que muitas organizações fazem distinção entre desenvolvimento e manutenção do sistema. Esta última é vista com prioridade quase nula porque não há incentivo em investir no desenvolvimento pensando em reduzir custo de manutenção (SOMMERVILLE, 2003).

Boas técnicas de engenharia de *software* durante o desenvolvimento e o código bem estruturado, contribuem para a redução de custos de manutenção (SOMMERVILLE, 2003).

2.2. Reengenharia

Sommerville (2003) afirma que a reengenharia é diferente de outras modificações em *software*, no sentido de que nenhuma funcionalidade nova é adicionada ao sistema. Em vez disso, o sistema é modificado, a fim de tornar mais fácil a sua compreensão e alteração. A reengenharia pode envolver algumas alterações estruturais, mas geralmente não envolve grandes mudanças de arquitetura.

Segundo Sommerville (2003) a reengenharia de *software* se ocupa de reimplementar sistemas legados, para que sua manutenção seja mais fácil, aumentando assim a sua vida útil.

O processo de reengenharia pode envolver redocumentar, organizar, traduzir o sistema para uma linguagem de programação mais moderna e modificar e

atualizar a estrutura e os valores dos resultados do sistema. A funcionalidade do *software* não é modificada e normalmente a arquitetura do sistema também permanece a mesma (SOMMERVILLE, 2003).

2.3. Refatoração

De acordo com Fowler (2004), a refatoração é definida como:

> "... uma técnica disciplinada para reestruturar um código fonte existente, alterando sua estrutura interna sem mudar o seu comportamento externo. Sua essência está em uma série de pequenas transformações que preservam comportamento. Cada transformação (chamada de refatoração) faz pouca coisa, mas uma seqüência de pequenas transformações produz uma reestruturação significativa. Uma vez que cada refatoração é pequena, é mais improvável que algo dê errado. O sistema se mantém funcionando integralmente após cada refatoração, reduzindo as chances de o sistema sofrer um dano grave durante a reestruturação..."

Então refatoração é uma maneira disciplinada de aperfeiçoar o código que minimiza a chance de introdução de falhas. Em essência, refatorar é melhorar o projeto do código após este ter sido escrito (FOWLER, 2004).

2.3.1. Vantagens

Muitas vezes o desenvolvedor olha para o código e não consegue compreendê-lo logo. Isso faz com que uma simples alteração leve horas ou até dias para ser executada (SCHAAB, 2006).

Um sistema projetado de maneira incorreta precisa de muita linha de código, isso gera duplicações de funções no programa. Se for difícil entender o projeto a partir do código também será difícil dar-lhe manutenção, afirma Fowler (2004).

Fowler (2004) cita as seguintes vantagens para a utilização da refatoração:

- Melhora o projeto de *software*;
- Torna o *software* mais fácil de entender;
- Ajuda a encontrar falhas; e

- Ajuda a programar mais rapidamente.

A refatoração ajuda a tornar o código mais legível. Ao refatorar, o desenvolvedor tem código que funciona, mas que não está estruturado da forma ideal. Um pouco mais de tempo refatorando faz com que o código comunique melhor o seu propósito (FOWLER, 2004).

A refatoração melhora a qualidade do projeto, melhora a legibilidade, reduz as falhas e conseqüentemente aumenta a velocidade das alterações (Fowler, 2004).

A refatoração é processo fundamental no desenvolvimento de um *software*, pois otimiza a vida do programa e reduz o tempo gasto para desenvolvê-lo.

2.3.2. Quando refatorar

Fowler (2004) afirma que a refatoração deve ser utilizada quando o “código cheira mal” (do inglês *bad smells in code*).

Martin Fowler e Kent Beck criaram o termo *bad smell* (mau cheiro) que se refere às características do código fonte que indicam má qualidade e que podem sofrer refatoração. Eles elaboraram uma lista de 22 *bad smells* que são apresentados a seguir (FOWLER, 2004).

- **Código Duplicado**. Um código duplicado quase sempre representa uma falha. Ele leva a fazer modificações paralelas, pois quando é feito alteração em um lugar, é necessário fazer em outro também. (MCCONNELL, 2005).
- **Método Longo**. Uma maneira de melhorar um sistema é aumentar sua modularidade – aumentar o número de rotinas bem definidas e com bons nomes, que executam apenas uma atividade (MCCONNELL, 2005). Métodos longos são problemáticos porque freqüentemente contêm muita informação que fica escondida pela complexidade da lógica associada (FOWLER, 2004).
- **Classes Grandes**. Quando uma classe tentar fazer muita coisa, ela freqüentemente tem variáveis de instância demais. Quando uma classe tem variáveis de instância em excesso, o código duplicado não deve

estar longe (FOWLER, 2004).

- **Lista de Parâmetros Longa**. Listas de parâmetros longas são difíceis de entender, porque se tornam inconsistentes e difíceis de usar (FOWLER, 2004). E uma lista de parâmetros muito longa, é sinal de alerta de que a abstração da interface da rotina não foi bem elaborada (MCCONNELL, 2005).
- **Alteração Divergente**. O *software* deve ser estruturado para tornar as alterações mais fáceis. Quando existe a necessidade de uma alteração, é desejável ir a um único ponto do sistema, claramente definido, e fazer essa alteração (FOWLER, 2004).
- **Cirurgia com Rifle**. Ele acontece quando é necessário fazer muitas alterações pequenas em muitas classes, cada vez que é preciso executar uma mudança no *software*. Quando as alterações estão por toda parte, elas são difíceis de encontrar, sendo fácil deixar de fazer alguma alteração importante e podendo ocasionar algum erro (FOWLER, 2004).
- **Inveja dos Dados**. Quando um método parece mais interessado em uma classe diferente daquela na qual ele se encontra é um indício do *bad smell* de “inveja dos dados” (FOWLER, 2004).
- **Grupos de Dados**. Agrupamentos de dados que perambulam juntos deveriam ser criados no seu próprio objeto (FOWLER, 2004).
- **Obsessão Primitiva**. Os tipos de dados primitivos podem ser usados para representar um número infinito de entidades do mundo real. É interessante criar uma classe para representar uma entidade comum dentro do sistema (MCCONNELL, 2005).
- **Comandos *Switch***. Um dos fenômenos mais óbvios do código orientado a objetos é sua comparativa falta de comandos *switch* (FOWLER, 2004).
- **Hierarquias Paralelas de Herança**. Esse é um caso especial do *bad smell* cirurgia com rifle. Nessa situação, cada vez que é criada uma subclasse de uma determinada classe, é preciso também criar uma subclasse de outra (FOWLER, 2004).
- **Classe Ociosa**. Cada classe criada custa dinheiro para manter e compreender. Uma classe que não esteja fazendo o suficiente para se pagar deve ser eliminada (FOWLER, 2004). Se uma classe parece não

estar valendo à pena, verifique a possibilidade de atribuir todas as responsabilidades dela para outras classes e, então, eliminá-la (MCCONNELL, 2005).

- **Generalidade Especulativa**. Esse tipo de *bad smell* ocorre quando as pessoas dizem: “Bom, acho que precisaremos disso algum dia” e então são criados todos os tipos de ganchos e casos especiais para lidar com coisas que não são exigidas. O resultado disso é algo difícil de entender e manter (FOWLER, 2004).
- **Campo Temporário**. Muitas vezes é encontrado um objeto no qual uma variável de instância recebe um valor apenas em determinadas circunstâncias. Tal código é difícil de entender, porque se espera que um objeto precise de todas as suas variáveis (FOWLER, 2004).
- **Cadeias de Mensagens**. Acontece quando um cliente pede um objeto para outro objeto, ao qual o cliente pede então outro objeto, ao qual o cliente pede então outro objeto e assim por diante. Navegar dessa maneira significa que o cliente está acoplado à estrutura de navegação. Qualquer mudança no relacionamento intermediário obriga a uma mudança no cliente (FOWLER, 2004).
- **Intermediário**. Ao verificar que a maior parte do código em uma classe está apenas passando chamadas para rotinas em outras classes, deve ser considerada a possibilidade de eliminar o intermediário e chamar essas outras classes diretamente (MCCONNELL, 2005).
- **Intimidade Inadequada**. O encapsulamento talvez seja a ferramenta mais poderosa para tornar os programas intelectualmente controláveis e para minimizar os efeitos da propagação das alterações de código. Sempre que existir uma classe que sabe mais do que deve sobre outra classe, deve-se privilegiar o encapsulamento mais forte, não o mais fraco (MCCONNELL, 2005).
- **Classes Alternativas com Interfaces Diferentes**. Ocorre quando as interfaces de duas classes são diferentes e, no entanto as classes são completamente similares (KERIEVSKY, 2005).
- **Biblioteca de Classes Incompleta**. Muitas vezes é ruim, e normalmente impossível, modificar uma biblioteca de classes para fazer algo que fosse

necessário que ela fizesse (FOWLER, 2004).

- **Classes de Dados**. Essas são classes que possuem atributos, métodos de acesso para os atributos e mais nada. Tais classes são depósitos burros de dados e quase que certamente manipuladas nos mínimos detalhes por outras classes (FOWLER, 2004).
- **Herança Recusada**. Subclasses conseguem herdar métodos e dados de seus pais, mas e se elas não precisarem do que lhes é dado, então elas recebem tudo e escolhem apenas alguns para usar (FOWLER, 2004).
- **Comentários**. Os comentários têm um papel importante a desempenhar, mas não devem ser usados como suporte para explicar um código mal elaborado (MCCONNELL, 2005).
- **Uma classe tem coesão fraca**. Uma classe que possui uma miscelânea de responsabilidades não relacionadas entre si deve ser subdividida em várias classes, cada uma das quais tendo a seu encargo um conjunto coeso de responsabilidades (MCCONNELL, 2005).

A refatoração deve ser feita o tempo todo, em pequenas quantidades. Deve-se empregar a refatoração nas seguintes situações (FOWLER, 2004):

- Quando for preciso acrescentar rotinas;
- Quando for preciso adicionar uma classe;
- Quando for preciso consertar uma falha;
- Durante a revisão do código;
- Quando houver módulos propensos a erros; e
- Quando houver módulos de alta complexidade.

A refatoração pode ser considerada uma técnica ou ferramenta de auxílio no desenvolvimento e manutenção, contribuindo para o tempo de vida do s*oftware*, já que o acréscimo de novas funcionalidades passa a ser feita de maneira fácil e rápida (FOWLER, 2004).

Mas há situações em que o código está tão confuso que seria mais fácil reescrevê-lo novamente do zero do que refatorá-lo. Um sinal claro da necessidade de reescrever é quando o código simplesmente não funciona (FOWLER, 2004).

2.3.3. Refatorando

Segundo Mens e Tourwé (2004), o processo de refatoração consiste em um número distinto de atividades. Eles sugerem as seguintes atividades:

- Identificação de locais do *software* com oportunidades de refatoração;
- Determinar qual refatoração pode ser aplicado no local identificado;
- Garantir que as refatorações escolhidas preservem o comportamento;
- Aplicar as refatorações escolhidas aos seus respectivos locais;
- Avaliar o efeito da refatoração na característica de qualidade do *software* (complexidade, legibilidade, manutenibilidade) ou de processo (produtividade, custo, esforço); e
- Manter a consistência entre o código refatorado e outros artefatos (como documentação, modelos, testes e outros).

2.3.4. Catálogo de refatorações

Fowler (2004) elaborou um catálogo contendo 72 refatorações divididas em seis conjuntos.

Conjunto 1: Compondo métodos.

É o conjunto de refatorações que lidam com a composição de métodos (FOWLER, 2004).

- **Extrair Método**. Existe um fragmento de código que pode ser agrupado. Deve-se transformar o fragmento em um método que o nome explique o seu propósito.
- **Internalizar Método**. O corpo de um método é tão claro quanto seu nome. Deve-se colocar o corpo do método dentro do corpo do que o chama e remover o método.
- **Internalizar Variável Temporária**. Existe uma variável temporária que recebe uma única atribuição de uma expressão, e essa variável está atrapalhando outras refatorações. Devem-se substituir todas as

referências a essa variável temporária pela expressão.

- **Substituir Variável Temporária por Consulta**. Está sendo usada uma variável temporária para armazenar o resultado de uma expressão. Deve-se extrair a expressão para um método e substituir todas as referências à variável temporária pelo novo método criado.
- **Introduzir Variável Explicativa**. Quando existir uma expressão complicada, coloca-se o resultado da expressão em uma variável temporária cujo nome explique o seu propósito.
- **Dividir Variável Temporária**. Existe uma variável temporária que mais de uma vez recebe uma atribuição, mas não é uma variável de laço e nem um acumulador temporário. Para esse caso, deve-se criar uma variável temporária separada para cada atribuição.
- **Remover Atribuições a Parâmetros**. O código faz uma atribuição a um parâmetro. Deve-se usar uma variável temporária no lugar da atribuição.
- **Substituir Método por Objeto Método**. Existe um método longo que usa variáveis locais de um modo que não se pode usar a refatoração Extrair Método. Deve-se transformar o método em seu próprio objeto de modo que todas as variáveis locais se tornem campos desse objeto. Pode-se então decompor o método em outros métodos no mesmo objeto.
- **Substituir o Algoritmo**. Deseja-se substituir um algoritmo por um mais claro. Substitui-se o corpo do método pelo novo algoritmo.

Conjunto 2: Movendo recursos entre objetos.

Esse conjunto de refatorações auxilia no processo de atribuição de responsabilidade nos componentes orientados a objetos (FOWLER, 2004).

- **Mover Método**. Um método está usando ou sendo usado por mais recursos de outra classe do que a classe na qual ele está definido. Cria-se um novo método com um corpo similar na classe que ele mais usa. Transforma-se o método antigo em uma simples delegação ou remove-se completamente.
- **Mover Campo**. Um campo é usado por outra classe mais do que a classe na qual ele está definido. Nessa situação, deve-se criar um novo campo na classe alvo e alterar todos os usuários desse campo.

- **Extrair Classe**. Existe uma classe fazendo um trabalho que deveria ser feito por duas. Deve-se criar uma nova classe e mover os campos e métodos pertinentes da classe antiga para a nova.
- **Internalizar Classe**. Uma classe não está fazendo muita coisa. Mover todos os seus recursos para outra classe e apagar a classe ociosa.
- **Ocultar Delegação**. Um cliente referencia uma classe delegada de um objeto. Criam-se métodos no servidor para ocultar a delegação.
- **Remover Intermediário**. Uma classe está executando uma quantidade excessiva de delegações simples. Deve-se fazer com que o cliente chame o delegado diretamente.
- **Introduzir Método Externo**. Uma classe servidora precisa de um método adicional, mas não é possível modificá-la. Deve-se criar um método na classe cliente com uma instância da classe servidora como seu primeiro parâmetro.
- **Introduzir Extensão Local**. Uma classe servidora precisa de diversos métodos adicionais, mas não é possível modificá-la. Cria-se uma nova classe que contenha esses métodos adicionais. Essa extensão da classe deve ser tornada uma subclasse ou um envoltório da classe original.

Conjunto 3: Organizar dados.

Esse conjunto de refatorações discute maneiras de tornar mais fácil o trabalho com dados dentro de sistemas orientados a objetos (FOWLER, 2004).

- **Auto-Encapsular Campo**. Um campo está sendo acessado diretamente, mas o acoplamento a esse campo está se tornando inadequado. Devem-se criar métodos de acesso ao campo (gravação e leitura) e usar apenas esses métodos para acessar o campo.
- **Substituir Atributo por Objeto**. Se existe um dado que precisa de comportamento ou dados adicionais, então se transforma o dado em um objeto.
- **Mudar de Valor para Referência**. Se existe uma classe com muitas instâncias iguais e se deseja substituí-las por um único objeto, se transforma o objeto em um objeto por referência.
- **Mudar de Referência para Valor**. Se existe um objeto que é pequeno,

imutável e difícil de gerenciar, se transforma o objeto em um objeto por valor.

- **Substituir Vetor por Objeto**. Existe um vetor nos quais certos elementos significam coisas diferentes. Então é preferível substituir o vetor por um objeto que tenha um campo para cada elemento.
- **Duplicar Dados Observados**. Existem dados do domínio disponíveis somente em um controle GUI[1] e métodos do domínio precisam acessá-los. Então se copia os dados para um objeto do domínio. Configura-se um *Observer*[2] para sincronizar os dois fragmentos de dados.
- **Transformar Associação Unidirecional em Bidirecional**. Existem duas classes que precisam usar os recursos uma da outra, mas há uma conexão unidirecional apenas. Devem-se acrescentar ponteiros de retorno e alterar os modificadores para atualizar ambos os conjuntos.
- **Transformar Associação Bidirecional em Unidirecional**. Se existe uma associação bidirecional, mas uma classe não precisa mais dos recursos da outra; então se elimina o lado desnecessário da associação.
- **Substituir Números Mágicos por Constantes Simbólicas**. No caso de existir um número literal com um significado especial, pode-se criar uma constante com um nome de acordo com seu propósito e substituir por ela.
- **Encapsular Campo**. Se existir um campo público, deve-se torná-lo privado e fornecer métodos de acesso.
- **Encapsular Coleção**. Quando um método retornar uma coleção, fazê-lo retornar uma visão apenas de leitura e fornecer métodos de adição e remoção.
- **Substituir Registro por Classe de Dados**. É necessário interagir com variáveis do tipo registro em um ambiente de programação tradicional. Deve-se criar um objeto de dados para o registro.
- **Substituir Enumeração por Classe**. Se uma classe tem uma

[1] Graphic User Interface (Interface Gráfica do Usuário).
[2] Observer: padrão de projeto que define uma dependência um-para-muitos entre objetos, de modo que, quando um objeto muda de estado, todos os seus dependentes são automaticamente notificados e atualizados.

enumeração que não afeta seu comportamento, então se substitui a enumeração por uma nova classe.

- **Substituir Enumeração por Subclasses**. Quando existe uma enumeração imutável que afeta o comportamento da classe deve-se substituí-la por subclasses.
- **Substituir Enumeração pelo Padrão *State/Strategy***. Se uma enumeração afeta o comportamento da classe, mas não se podem usar subclasses, então a enumeração deve ser substituída por um objeto que represente o estado do objeto original.
- **Substituir Subclasse por Campos**. Quando as subclasses se diferem somente apenas em métodos que retornam dados, devem-se transformar os métodos em campos na superclasse e eliminar as subclasses.

Conjunto 4: Simplificando expressões condicionais.

Esse conjunto de refatorações visa simplificar a lógica condicional em projetos de *software* (FOWLER, 2004).

- **Decompor Condicional**. Existe uma estrutura condicional complicada (*if-then-else*). Extraem-se os métodos da condição, da parte após o *then* e da parte após o *else*.
- **Consolidar Expressão Condicional**. Se uma seqüência de testes condicionais tem o mesmo resultado, é interessante criar uma única expressão condicional e extraí-la.
- **Consolidar Fragmentos Condicionais Duplicados**. Se o mesmo fragmento de código aparece em todos os ramos de uma expressão condicional, então se move ele para fora da expressão.
- **Remover *Flag* de Controle**. Existe uma variável que está atuando como uma *flag* de controle para uma série expressões booleanas. Usar um *break* ou um *return* no lugar dela é mais indicado.
- **Substituir Condição Aninhada por Cláusulas Guarda**. Um método tem lógica condicional que não deixa claro o fluxo normal da execução. Usar cláusulas-guarda para todos os casos especiais.
- **Substituir Comando Condicional por Polimorfismo**. Existe um comando condicional que seleciona diferentes comportamentos de

acordo com o tipo de um objeto. Mover cada ramificação do comando condicional para um método de sobrescrita em uma subclasse e tornar abstrato o método original.

- **Introduzir Objeto Nulo**. Quando existem verificações repetidas de um valor nulo, então se substitui o valor nulo por um objeto nulo.
- **Introduzir Asserção**. Uma seção de código faz alguma suposição sobre o estado do programa. Tornar a suposição explícita com uma classe de asserção.

Conjunto 5: Tornando as chamadas de métodos mais simples.

Esse conjunto de refatorações explora maneiras de tornar as interfaces mais diretas (FOWLER, 2004).

- **Renomear Método**. Quando o nome de um método não revela seu propósito, é indicado alterar o nome do método.
- **Acrescentar Parâmetro**. Se um método precisar de mais informações de quem o chama, então se deve acrescentar um parâmetro para um objeto que possa passar essa informação.
- **Remover Parâmetro**. Deve-se remover um parâmetro quando esse não é mais usado pelo corpo do método.
- **Separar Pesquisa do Modificador**. Existe um método que retorna um valor, mas também altera o estado de um objeto. Nesses casos, devem-se criar dois métodos, um para consulta e um para modificação.
- **Parametrizar Método**. Quando diversos métodos fazem coisas semelhantes, mas com diferentes valores contidos no corpo do método, é interessante criar um método que use um parâmetro para os diferentes valores.
- **Substituir Parâmetro por Métodos Explícitos**. Se um método executa diferentes códigos de acordo com os valores de um parâmetro, então se cria um método separado para cada valor do parâmetro.
- **Preservar o Objeto Inteiro**. Existem casos em que estão sendo lidos diversos valores de um objeto e passando esses valores como parâmetro em uma chamada de método. Quando isso acontecer, se envia todo o objeto.
- **Substituir Parâmetro por Método**. Quando um objeto chama um

método e depois passa o resultado como parâmetro para um método, é aconselhável remover o parâmetro e deixar o destinatário chamar o método.

- **Introduzir Objeto Parâmetro**. Quando existe um grupo de parâmetros que naturalmente andam juntos, se substituem eles por um objeto.
- **Remover Método de Gravação**. Um campo deve ser gravado na sua criação e nunca alterado. Para esse caso, devem-se remover quaisquer métodos de gravação desse campo.
- **Ocultar Método**. Quando um método não é usado por nenhuma outra classe, deve-se torná-lo privado.
- **Substituir Construtor por um *Factory Method***. Quando se deseja fazer mais do que uma simples construção quando se cria um objeto, deve-se substituí-lo por um *factory method*.
- **Encapsular *Downcast***. Se um método retorna um objeto que precisa sofrer *downcast* por seus solicitantes, deve-se mover o *downcast* para dentro do método.
- **Substituir Código de Erro por Exceção**. Existem condições em que um método retorna um código especial para indicar um erro, neste caso, é preferível gerar uma exceção.
- **Substituir Exceção por Teste**. Se estiver sendo gerada uma exceção em uma condição que o solicitante poder ser verificado primeiro, então se deve alterar o solicitante para fazer o teste primeiro.

Conjunto 6: Lidando com generalização.

Esse conjunto de refatorações lida com a movimentação de métodos e atributos em uma hierarquia de herança (FOWLER, 2004).

- **Subir Campo na Hierarquia**. Quando duas subclasses têm o mesmo campo, mover o campo para a superclasse.
- **Subir Método na Hierarquia**. Se existem dois métodos nas subclasses que produzem o mesmo resultado, mover os métodos para a superclasse.
- **Subir o Corpo do Construtor na Hierarquia**. Se existem construtores em subclasses com corpos quase idênticos, criar um construtor para a

superclasse e chamá-lo a partir de métodos das subclasses.

- **Descer Método na Hierarquia**. Se algum comportamento na superclasse é relevante apenas para algumas das subclasses, então mover esse comportamento para essas subclasses.
- **Descer Campo na Hierarquia**. Se um campo é usado apenas por algumas subclasses, então mover esse campo para essas subclasses.
- **Extrair Subclasse**. Quando uma classe tem características que são usadas apenas em algumas instâncias, então se deve criar uma subclasse para esse subconjunto de características.
- **Extrair Superclasse**. Quando existem duas classes com características semelhantes, criar uma superclasse e mover as características em comum para ela.
- **Extrair Interface**. Diversos clientes usam o mesmo subconjunto da interface de uma classe ou duas classes têm parte de suas interfaces em comum. Então se deve extrair o subconjunto para uma interface.
- **Condensar Hierarquia**. Deve-se juntar uma classe e uma subclasse quando não são muito diferentes.
- **Criar um Método Padrão**. Quando existem dois métodos em subclasses diferentes que executam passos semelhantes na mesma ordem, mas esses passos são diferentes. Então se devem colocar os passos em métodos com a mesma assinatura, de modo que os métodos originais se tornem o mesmo. Assim pode-se subi-los na hierarquia.
- **Substituir Herança por Delegação**. Quando uma subclasse usa apenas parte da interface de uma superclasse ou não quer herdar dados. Então se deve criar um campo para a superclasse, ajustar os métodos para delegarem para a superclasse e remover a herança.
- **Substituir Delegação por Herança**. Se existe delegação e está sendo escrito frequentemente muitas delegações simples para toda a interface, então se deve tornar a classe que está delegando uma subclasse da delegada.

2.3.5. Relacionando refatorações com *bad smells*

Fowler (2004) relata quais as refatorações mais adequadas para cada *bad smell*, que são mostradas no Quadro 1.

Bad Smell	**Refatorações comuns**
Alteração Divergente	Extrair Classe
Biblioteca de classes incompleta	Introduzir Método Externo, Introduzir Extensão Local
Cadeias de mensagens	Ocultar Delegação
Campo Temporário	Extrair Classe, Introduzir Objeto Nulo
Cirurgia com rifle	Mover Método, Mover Campo, Internalizar Classe
Classe Ociosa	Internalizar Classe, Condensar Hierarquia
Classes alternativas com interfaces diferentes	Renomear Método, Mover Método
Classes de Dados	Mover Método, Encapsular Campo, Encapsular Coleção
Classes Grandes	Extrair Classe, Extrair Subclasse, Extrair Interface, Substituir Atributo por Objeto
Código duplicado	Extrair Método, Extrair Classe, Subir Método na Hierarquia, Criar um Método Padrão
Comandos *Switch*	Substituir Comando Condicional por Polimorfismo, Substituir Enumeração por Subclasses, Substituir Enumeração pelo Padrão *State/Strategy*, Substituir Parâmetro por Métodos Explícitos, Introduzir Objeto Nulo
Comentários	Extrair Método, Introduzir Asserção
Generalidade Especulativa	Condensar Hierarquia, Internalizar Classe, Remover Parâmetro, Renomear Método
Grupos de Dados	Extrair Classe, Introduzir Objeto Parâmetro, Preservar o Objeto Inteiro
Herança Recusada	Substituir Herança por Delegação
Hierarquias Paralelas de herança	Mover Método, Mover Campo
Intermediário	Renomear Intermediário, Internalizar Método, Substituir Delegação por Herança
Intimidade Inadequada	Mover Método, Mover Campo, Transformar Associação Bidirecional em Unidirecional, Substituir Herança por Delegação, Ocultar Delegação
Inveja dos Dados	Mover Método, Mover Campo, Extrair Método

Quadro 1 Relação de qual Refatoração utilizar para resolver determinado *bad smell*.

2.4. A Linguagem de Programação *Delphi*

O *Delphi* é um ambiente de desenvolvimento orientado a objetos. É uma ferramenta de desenvolvimento muito produtiva, permitindo metodologias de desenvolvimento capazes de construir robustos e complexos aplicativos em pouquíssimo tempo, e não exigindo códigos complexos caso haja boa elaboração na metodologia (ROBSON, 2005).

O ambiente de desenvolvimento do *Delphi* utiliza uma linguagem que antes era conhecida como *Object Pascal,* mas hoje é chamada de "Linguagem *Delphi*". A linguagem de programação *Delphi* é baseada em conceitos da POO (Programação Orientada a Objetos) e no RAD – *Rapid Application Development* (desenvolvimento visual) (CANTÙ, 2006).

Frick (2005) afirma que a principal linguagem utilizada no desenvolvimento de *software* para o setor corporativo é *Delphi* com 36,8% e 102 indicações conforme a Tabela 1.

Tabela 1 Principais Linguagens Utilizadas no Desenvolvimento de Software Para o Setor Corporativo em 2005.

Linguagem	Nº de Indicações*	%**
Delphi	102	36,8
Visual Basic	60	21,7
C++ / Visual C++ / C++ Builder	50	18,1
SQL	49	7,7
HTML / DHTML	44	15,9
XML	39	14,1
Java	38	13,7
Java Script	33	11,9
Clipper	31	11,2
ASP	26	9,4
WebServices / .Net	20	7,2
FoxPro / Visual FoxPro	15	5,4
Progress	15	5,4
C / Visual C	14	5,1
Pascal	12	4,3
COBOL	1	4,0
PHP	10	3,6
JSP / Servlet	10	3,6

**Incluem-se respostas múltiplas. ** Sobre o total de produtos: 277*
Fonte: FRICK (2005).

3. DESENVOLVIMENTO

3.1. Análise do Software Escolhido

O s*oftware* escolhido para a execução deste trabalho foi um Sistema Gerenciador de Escola e Hospital Veterinário, o qual foi desenvolvido para administrar as aulas e os atendimentos no hospital veterinário do *Campus* Luiz Meneghel da Universidade Estadual do Norte do Paraná. Além de gerenciar os atendimentos, o *software* faz o controle de estoque da farmácia do hospital veterinário, mantendo registros das entradas e saídas dos produtos e cadastros de fornecedores, produtos, pacientes, disciplinas e requisitantes (professores).

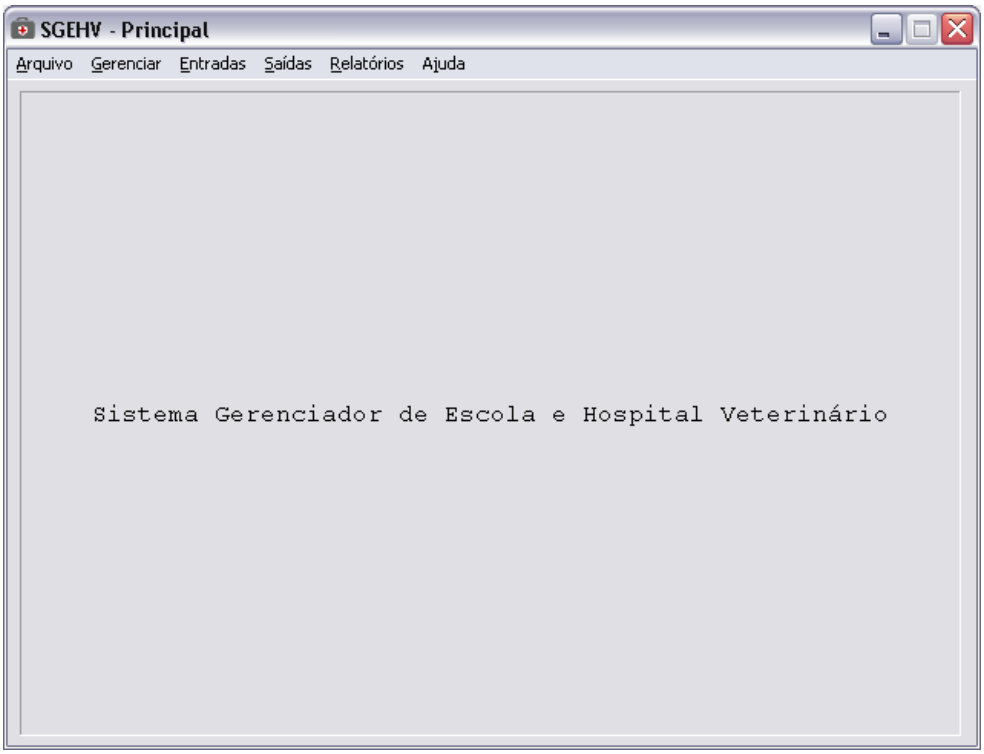

Figura 1 - Sistema Gerenciador de Escola e Hospital Veterinário

As refatorações serão feitas no módulo do Cadastro de Saída que é um dos módulos mais complexos desse sistema, pois apresenta uma grande quantidade de campos, atualizando duas tabelas no banco de dados (saidas e itens_saidas), além de fazer consultas em Pacientes, Requisitantes, Disciplinas, e em Produtos. Como o

software já está em uso, é importante que não haja alterações visíveis ao usuário final, alterando apenas a estrutura do código-fonte.

Antes de conhecer o código-fonte é importante saber como o programa funciona e como ele é utilizado. A Figura 2 mostra a interface do Cadastro de Saída que é o modulo no qual serão feitas as refatorações.

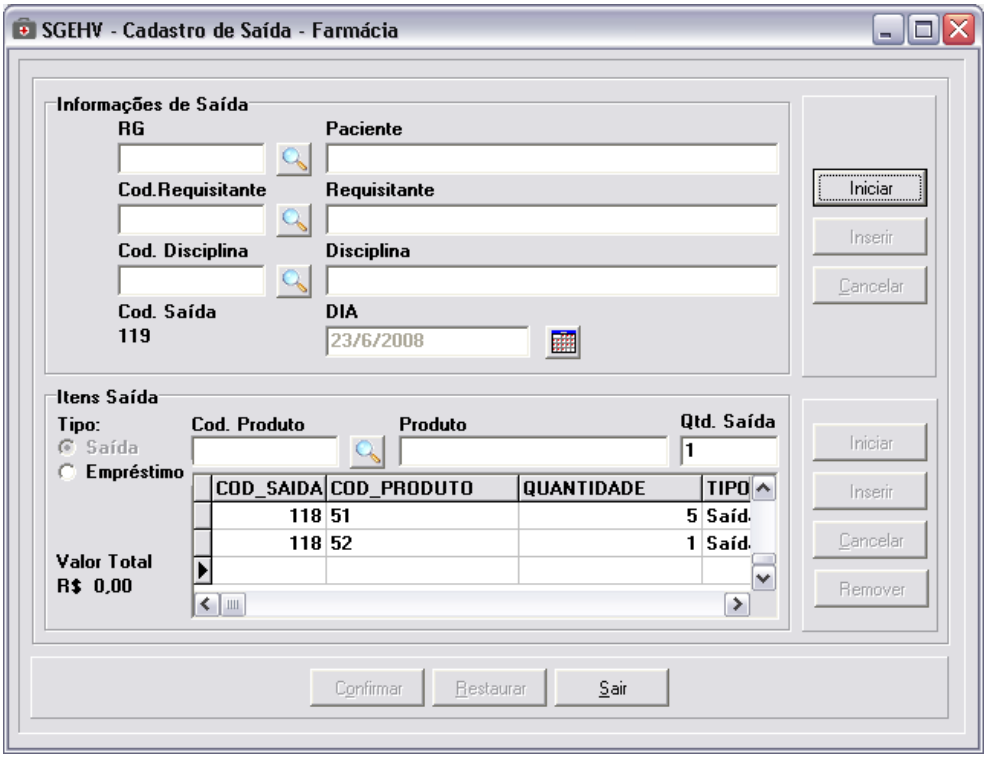

Figura 2 - Janela do Cadastro de Saída

A Figura 2 mostra o Cadastro de Saída, que possui campos referentes às informações da saída e campos referentes aos produtos.

Apresenta também a tela em seu estado inicial, antes de se aplicar a refatoração. Como se pode perceber, os botões “Iniciar” e “Sair” são os únicos que estão habilitados. Ao clicar no botão “Iniciar” os campos “RG”, “Cod. Requisitante”, “Cod. Disciplina” e “Dia”, assim como os respectivos botões de consulta são habilitados. O botão “Iniciar” é desabilitado e os botões “Inserir” e “Cancelar” são habilitados.

Depois de se fornecer as informações da saída o usuário pode clicar em “Inserir” ou em “Cancelar”; se clicar em “Cancelar” a tela volta ao seu estado inicial, se clicar em “Inserir” os campos com as informações fornecidas são desabilitados e habilitam-se os botões “Confirmar” e “Restaurar“. Se o usuário clicar em “Restaurar”

o registro que foi inserido não é efetivado e o cadastro volta ao seu estado inicial. Se o usuário clica em “Confirmar” o segundo botão “Iniciar” é habilitado e os botões “Confirmar” e “Restaurar” são desabilitados.

Ao clicar no botão “iniciar” os campos “Cod. Produto” e “Qtd. Saída” são habilitados, pode-se mudar o tipo da saída (saída ou empréstimo) e o botão de consulta ao lado do campo “Cod. Produto” também pode ser utilizado. O botão “Iniciar” é desabilitado e os botões “Inserir” e “Cancelar” são habilitados.

Depois de fornecer as informações do produto o usuário pode clicar em “Inserir“ ou em “Cancelar”, ambos habilitam o botão “Iniciar” e desabilitam os campos das informações dos produtos. Os botões “Inserir” e ”Cancelar” são desabilitados e os botões “Confirmar” e “Restaurar” são habilitados. O usuário pode continuar inserindo itens através do botão “Iniciar”.

Se o usuário clica em “Restaurar” a inserção dos itens não é efetivada e o usuário pode recomeçar a inserir os itens clicando novamente em “Iniciar”.

Ao clicar em “Confirmar” os itens inseridos serão salvos no banco de dados e a saída deles será efetivada. Os botões dos itens serão desabilitados e a janela volta ao seu estado inicial como mostrado na Figura 2.

3.1.1 Análise do Código-Fonte

Durante a análise do código-fonte foram encontrados alguns fatores que dificultavam sua leitura e compreensão, como trechos de código duplicados, métodos longos e variáveis temporárias cujas funções não eram claras. Nos casos em que as rotinas eram baseadas em expressões condicionais (*if*) também foram encontrados espaços para refatorações.

3.2. Refatorações Aplicadas

As refatorações estão apresentadas em duas seções. Primeiro estão as alterações mais simples, feitas no código logo após a análise. E depois as

refatorações mais complexas que foram executadas no código após as primeiras alterações. O código-fonte refatorado está disponível nos Apêndices.

É importante ressaltar que o *software* foi testado após cada alteração no código, para garantir que erros não foram gerados pelas alterações e que as funcionalidades também não foram alteradas.

3.2.1. Alterações Mais Simples

Algumas alterações mais simples, como renomear variáveis, renomear métodos e classes, devem ser feitas antes das refatorações mais complexas, pois minimizam as possibilidades de erro nessas refatorações, deixando o código mais fácil de entender e de encontrar falhas.

No código-fonte analisado encontraram-se alguns trechos de código comentados, os quais se tornaram desnecessários quando alguma alteração foi feita durante o desenvolvimento, mas o desenvolvedor optou por deixar o código na forma de comentário como um *backup* para o caso de precisar restaurar o código. Esses comentários deveriam ter sido eliminados quando as alterações estivessem validadas.

Verificou-se que os trechos comentados realmente eram desnecessários e estavam apenas fazendo volume, atrapalhando a leitura do código.

Alguns componentes da interface dificultavam a compreensão do código-fonte, pois eram muito difíceis de ser identificados sem vê-los no *layout* da interface (Figura 3). Eles foram renomeados de acordo com as suas funções (Tabela 3).

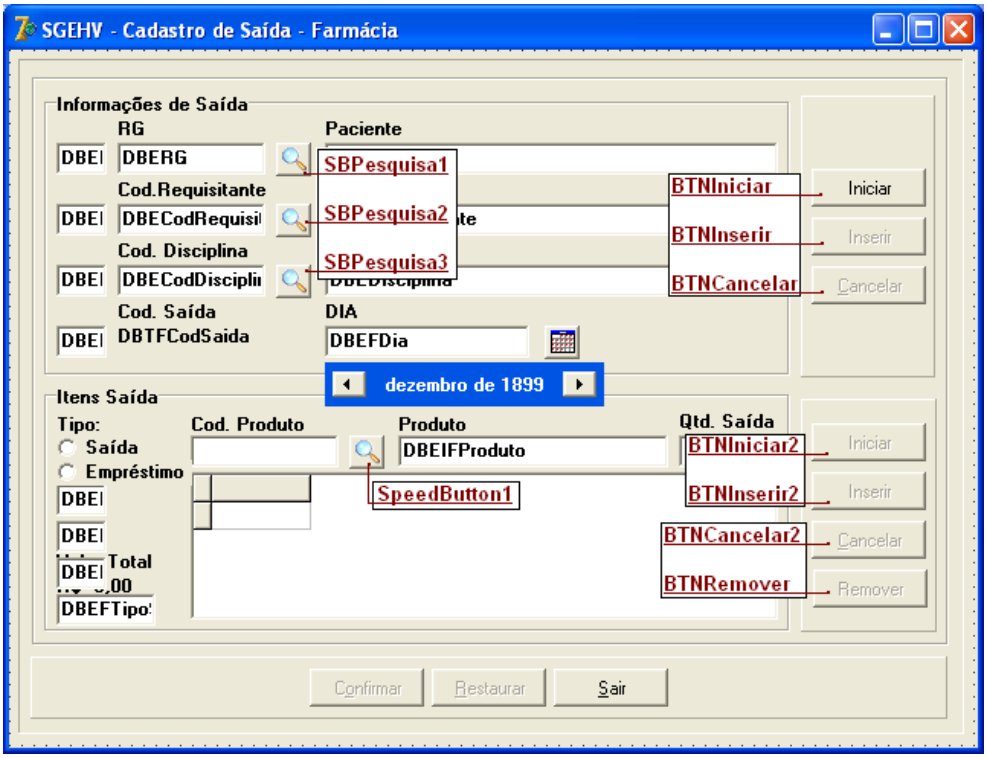

Figura 3 - Nomes dos Componentes da Interface

Antes	Depois
SBPesquisa1 (TSpeedButton)	SBPesquisaRG
SBPesquisa2 (TSpeedButton)	SBPesquisaCodRequisitante
SBPesquisa3 (TSpeedButton)	SBPesquisaCodDisciplina
SpeedButton1 (TSpeedButton)	SBPesquisaCodProduto
POpcoes (TPanel)	POpcoesNota
Panel2 (TPanel)	POpcoesItens
BTNIniciar (TBitBtn)	BTNIniciarNota
BTNIniciar2 (TBitBtn)	BTNIniciarItens
BTNInserir (TBitBtn)	BTNInserirNota
BTNInserir2 (TBitBtn)	BTNInserirItens
BTNCancelar (TBitBtn)	BTNCancelarNota
BTNCancelar2 (TBitBtn)	BTNCancelarItens

Quadro 2 Componentes Renomeados

Alguns componentes, que permaneciam invisíveis durante a execução e não eram utilizados pelo usuário, tinham a função de armazenar informações de outros componentes temporariamente ou gravar essas informações no banco de dados. Eles foram substituídos por variáveis, quando sua função era de armazenamento, ou

por comandos de acesso à tabela, quando servia para gravar informações no banco de dados.

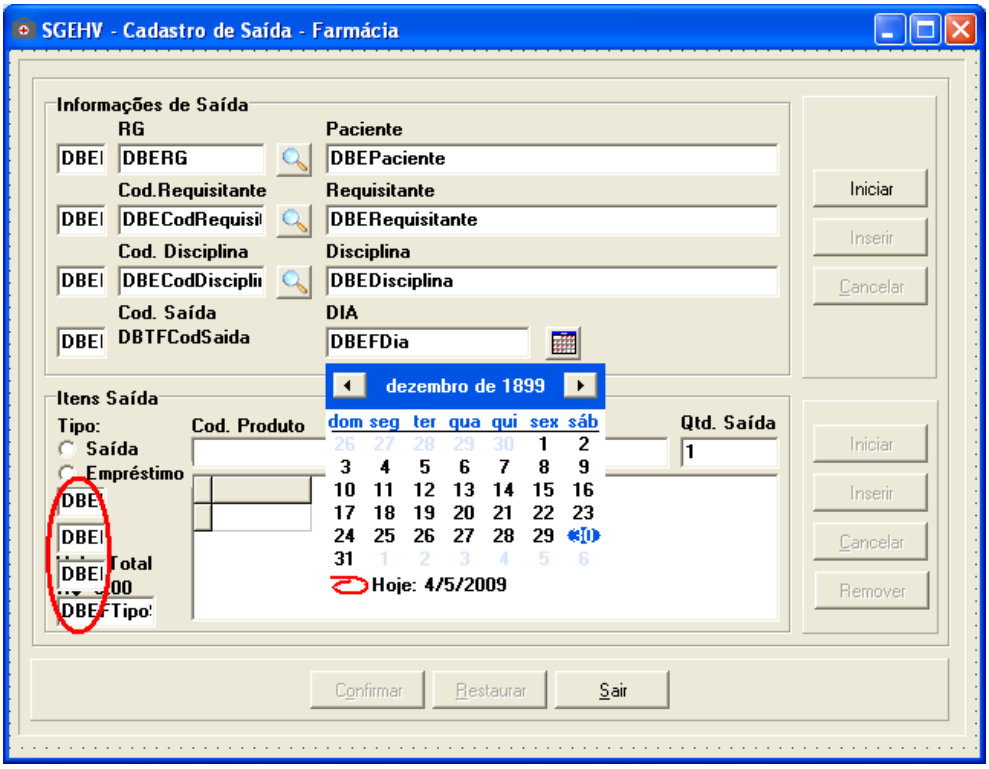

Figura 4 - Componentes Invisíveis Durante a Execução

O componente “DBEFTipoSaida”, foi substituído pela variável global TipoSaida (*string*) .

```
procedure TF_Cad_Saida_Farmacia.RBAulaClick(Sender: TObject);
begin
  DBEFTipoSaida.Text:=RBAula.Caption;
  RBAula.Enabled:=FALSE;
  RBCirurgia.Enabled:=TRUE;
end;

procedure TF_Cad_Saida_Farmacia.RBCirurgiaClick(Sender: TObject);
begin
  DBEFTipoSaida.Text:=RBCirurgia.Caption;
  RBCirurgia.Enabled:=FALSE;
  RBAula.Enabled:=TRUE;
end;
```

Código depois da refatoração:

```
procedure TF_Cad_Saida_Farmacia.RBAulaClick(Sender: TObject);
begin
  TipoSaida:=RBAula.Caption;
  RBAula.Enabled:=FALSE;
  RBCirurgia.Enabled:=TRUE;
end;

procedure TF_Cad_Saida_Farmacia.RBCirurgiaClick(Sender: TObject);
begin
  TipoSaida:=RBCirurgia.Caption;
  RBCirurgia.Enabled:=FALSE;
  RBAula.Enabled:=TRUE;
end;
```

Os componentes “DBEFTipoSaida”,” DBEIFCodSaida”,” DBEIFCodProduto” e ”DBEIFQtd” foram substituídos por comandos de acesso aos seus respectivos campos na tabela do banco de dados.

```
DBEFTipoSaida.Text:= TipoSaida;
DBEIFCodSaida.Text:=DBTFCodSaida.Caption;
DBEIFCodProduto.Text:=EIFCodProduto.Text;
DBEIFQtd.Text:=EIFQtdSaida.Text;
```

Código depois da refatoração:

```
DM.IBQItensFarmaciaTIPO.AsString:=' TipoSaida;
DM.IBQItensFarmaciaCOD_SAIDA.AsString:=DBTFCodSaida.Caption;
DM.IBQItensFarmaciaCOD_PRODUTO.AsString:=EIFCodProduto.Text;
DM.IBQItensFarmaciaQUANTIDADE.AsString:=EIFQtdSaida.Text;
```

Encontrou-se no código uma expressão que atribuía um valor a um componente da interface, essa expressão não era tão complexa, mas continha muitas funções de conversão de dados de *string* para *float* e vice-versa que tornavam a expressão difícil de ser lida. Nesse caso pode-se utilizar uma ou mais variáveis temporárias para diminuir complexidade da expressão.

```
LValorTotal.Caption := FloatToStr(StrToFloat(LValorTotal.Caption) +
(StrToFloat(DM.IBQProdutosPRECO_VENDA.AsString) * StrToFloat
(EIFQtdSaida.Text)));
LValorTotal.Caption := FloatToStr (RoundTo (StrToFloat (LValorTotal.Caption), -
2));
```

Para simplificar a expressão usou-se três variáveis *float* para guardar os valores literais convertidos e calcular a expressão antes de atribuir o resultado ao componente que irá exibí-lo, o arredondamento é feito somente na exibição do resultado final.

```
Total:= StrToFloat(LValorTotal.Caption);
Preco:= StrToFloat(DM.IBQProdutosPRECO_VENDA.AsString);
Qtd :=  StrToFloat (EIFQtdSaida.Text);
Total := Total + Preco * Qtd;
LValorTotal.Caption := FloatToStr (RoundTo Total, -2));
```

3.2.2. Alterações Complexas

Durante a análise do código-fonte percebeu-se que alguns objetos eram alterados sempre ao mesmo tempo, mas como eram objetos diferentes, o desenvolvedor usou expressões condicionais distintas para testar e manipular cada objeto, quando essas as alterações poderiam estar em uma única expressão que testa todos os objetos.

```
if BTNCommit.Enabled = FALSE then
  BTNCommit.Enabled:=TRUE;
if BTNRollback.Enabled = FALSE then
  BTNRollback.Enabled:=TRUE;
```

Nesse exemplo anterior a condição pode ser representada de maneira mais simples utilizando a clausula "*not*" ao invés de fazer comparações de valores "*Boolean"*.

```
if not BTNCommit.Enabled and not BTNRollback.Enabled then
begin
  BTNCommit.Enabled:=TRUE;
  BTNRollback.Enabled:=TRUE;
end;
```

Durante a análise foram encontradas duas variáveis – cont (*string*) e cont2 (*integer*) – que dificultavam a compreensão do código, pois os seus nomes não deixavam suas funções explicitas. Mas simplesmente alterar os nomes dessas variáveis não resolveria o problema, pois as informações contidas nelas não

representavam significados claros. Então foi realizada uma análise mais específica ao redor dessas duas variáveis.

- No evento “OnClick” do botão “btnInserirNota”, cont (*string*) recebe '1' e cont2 (*integer*) recebe 1.
- No evento “OnClick” do botão “btnInserirItens”, cont (*string*) recebe '2' e cont2 (*integer*) recebe 0.
- No evento “OnClick” do botão “btnCommit” se o valor de cont (*string*) for '1' os botões do cadastro da nota fiscal são desabilitados e o botão “btnIniciarItens” é habilitado.

  Se o valor de cont for diferente de '1' o botão “btnIniciarNota” é habilitado. Depois é feito outro teste verificando se o valor de cont2 (*integer*) é igual a 0. Se a condição for verdadeira cont (*string*) recebe '0'.
- No evento “OnClick” do botão “btnRollback” é feito um teste se cont2 (*integer*) é igual a 1. Se a condição for verdadeira o botão “btnInserirNota” habilitado.

De acordo com essa análise a variável cont (*string*) “diz” qual cadastro foi efetuado, se o seu valor é ‘1’ significa que a última inserção foi um cadastro de saída, e se ela vale ‘2’ significa que a última inserção foi um cadastro de item. Ela também é usada quando o usuário clica no botão “Confirmar” (btnCommit), onde é testado seu valor e de acordo com esse valor, o sistema segue uma determinada rotina.

Esse problema pode ser resolvido com a refatoração “Substituir Números Mágicos por Constantes Simbólicas”, que implica em substituir esses números por constantes com um nome de acordo com seu propósito.

Os valores de cont2 (*integer*) variam entre dois valores: 0 ou 1. Se o valor for 0 significa que a última inserção foi de um item. Se o seu valor é 1 significa que a última inserção foi o cadastro de uma saída. Essa variável é utilizada quando o usuário clica no botão “Restaurar” (btnRollback) determinando a rotina a ser seguida.

No evento OnClick do botão “btnCommit” é feito um teste em cont2 (*integer*), se o seu valor é 1, ou seja, se a última inserção foi o cadastro de uma saída, a variável cont (*string*) recebe ‘0’. Tal rotina pode ser excluída do código sem prejudicar nenhuma outra função no sistema, pois esse valor não representa nenhum significado.

As duas variáveis, apesar cada uma ser de um tipo diferente e de serem manipuladas de forma diferente, elas possuem a mesma função. Então elas podem ser substituídas por uma única variável. É importante que essa variável tenha um nome que identifique melhor sua função.

O próximo exemplo mostra uma parte do código que utilizava as duas variáveis:

```
if cont = '1' then
begin
  BTNIniciarNota.Enabled:=FALSE;
  BTNIniciarNota.Default:=FALSE;
  BTNIniciarItens.Default:=TRUE;
  BTNIniciarItens.Enabled:=TRUE;
  BTNCommit.Enabled:=FALSE;
  BTNRollback.Enabled:=FALSE;
  DM.IBTrans.Commit;
  DM.IBTrans.StartTransaction;
end
else
begin
  BTNIniciarNota.Enabled:=TRUE;
  BTNIniciarNota.Default:=TRUE;
  BTNIniciarItens.Default:=FALSE;
  BTNIniciarItens.Enabled:=FALSE;
  BTNRemoverItens.Enabled:=FALSE;
  BTNCommit.Enabled:=FALSE;
  BTNRollback.Enabled:=FALSE;
  DM.IBTrans.Commit;
  DM.IBTrans.StartTransaction;
end;
if cont2 = 0 then
  cont:='0';
```

As variáveis “cont” e “cont2” foram substituídas pela variável “UltimoRegistroInserido” que recebe uma *string* quando um registro é inserido identificando se o cadastro foi uma “saída” ou um “item” conforme o próximo exemplo.

```
if UltimoRegistroInserido = 'saida' then
begin
  BTNIniciarNota.Default:=TRUE;
  BTNIniciarNota.Enabled:=TRUE;
  BTNInserirNota.Default:=FALSE;
  BTNCommit.Enabled:=FALSE;
  BTNRollback.Enabled:=FALSE;
  DM.IBTrans.Commit;
  DM.IBTrans.StartTransaction;
end;
if UltimoRegistroInserido = 'item' then
begin
```

```
    BTNIniciarItens.Default:=TRUE;
    BTNInserirItens.Default:=FALSE;
    BTNRemoverItens.Enabled:=FALSE;
    BTNCommit.Enabled:=FALSE;
    BTNRollback.Enabled:=FALSE;
    DM.IBTrans.Commit;
    DM.IBTrans.StartTransaction;
  end;
```

Em várias partes do código a execução das rotinas é orientada por cláusulas condicionais (*if*), se o mesmo fragmento de código aparece em todos os ramos de uma expressão condicional, então deve-se utilizar a refatoração “Consolidar Fragmentos Condicionais Duplicados”, que consiste em mover estes fragmentos para fora da expressão.

```
Begin
  if UltimoRegistroInserido=‘saida’ then
  begin
    BTNIniciarNota.Default:=TRUE;
    BTNIniciarNota.Enabled:=TRUE;
    BTNInserirNota.Default:=FALSE;
    BTNCommit.Enabled:=FALSE;
    BTNRollback.Enabled:=FALSE;
    DM.IBQFarmacia.Close;
    DM.IBTrans.Rollback;
  end;
  if UltimoRegistroInserido = ‘item’ then
  begin
    BTNIniciarItens.Default:=TRUE;
    BTNInserirItens.Default:=FALSE;
    BTNRemoverItens.Enabled:=FALSE;
    BTNCommit.Enabled:=FALSE;
    BTNRollback.Enabled:=FALSE;
    DM.IBQFarmacia.Close;
    DM.IBTrans.Rollback;
  end;
  DM.IBQFarmacia.Open;
  DM.IBQFarmacia.Last;
  DM.IBQItensFarmacia.Open;
  DM.IBQItensFarmacia.Last;
  DM.IBQDisciplinas.Open;
  DM.IBQRequisitantes.Open;
  DM.IBQPacientes.Open;
  LValorTotal.Caption:='0,00';
end;
```

Depois de mover as linhas repetidas o código do exemplo anterior ficou da seguinte forma:

```
Begin
  if UltimoRegistroInserido= ‘item’ then
  begin
    BTNIniciarNota.Default:=TRUE;
    BTNIniciarNota.Enabled:=TRUE;
    BTNInserirNota.Default:=FALSE;
  end;
  if UltimoRegistroInserido= ‘saída’ then
  begin
    BTNIniciarItens.Default:=TRUE;
    BTNInserirItens.Default:=FALSE;
    BTNRemoverItens.Enabled:=FALSE;
  end;
  BTNCommit.Enabled:=FALSE;
  BTNRollback.Enabled:=FALSE;
  DM.IBQFarmacia.Close;
  DM.IBTrans.Rollback;
  DM.IBQFarmacia.Open;
  DM.IBQFarmacia.Last;
  DM.IBQItensFarmacia.Open;
  DM.IBQItensFarmacia.Last;
  DM.IBQDisciplinas.Open;
  DM.IBQRequisitantes.Open;
  DM.IBQPacientes.Open;
  LValorTotal.Caption:='0,00';
end;
```

4. CONCLUSÃO

4.1. Considerações Finais

A manutenção em software é uma atividade complexa, com alto custo e sujeita a erros. Uma das atividades que podem contribuir para minimizar esses problemas é a refatoração do código-fonte, cujo objetivo é modificar um programa para torná-lo mais legível e, por isso, com manutenção mais fácil.

Este trabalho apresentou um estudo de caso, aplicando refatoração em um software de gerenciamento de farmácia de hospital veterinário, escrito em *Delphi*.

O código-fonte analisado apresentava muitos fatores dificultavam sua leitura e compreensão, como trechos de código duplicados, métodos longos e variáveis temporárias cujas funções não eram claras e bem definidas. Muitas rotinas eram baseadas em expressões condicionais (*if*), que apresentavam problemas como o mesmo fragmento de código em todos os ramos de uma expressão condicional ou testes condicionais semelhantes que poderiam ser simplificados.

Através de algumas técnicas de refatoração aplicadas esses problemas puderam ser resolvidos, tornando o código-fonte mais fácil de ser lido e compreendido, e com isso, facilitando futuras alterações no código.

4.2. Contribuição do trabalho

Esse trabalho mostra a importância de se manter um *software* com o código bem estruturado através das técnicas de refatoração, que podem reduzir o custo total do *software* e aumentar a sua vida útil. Mostrando algumas alterações que são possíveis de se fazer em um sistema escrito em *Delphi* para tornar mais fácil a compreensão do seu código.

4.3. Trabalhos Futuros

Para a realização deste trabalho a refatoração foi aplicada de forma manual. Sabe-se que para uma melhor produtividade e qualidade é interessante utilizar ferramentas automatizadas para realização de tarefas de engenharia de software. Como trabalhos futuros sugere-se investigar a existência ou desenvolvimento de ferramentas para refatoração automática de código escrito em *Delphi*.

REFERÊNCIAS

CANTÙ, Marco. **Dominando o Delphi 2005: A Bíblia**. São Paulo: Pearson Prentice Hall, 2006.

FOURNIER, Roger. **Guia Prático para Desenvolvimento e Manutenção de Sistemas Estruturados**. São Paulo: Makron Books, 1994.

FOWLER, Martin. **Refactoring: Improving the Design of Existing Code**. Porto Alegre: Bookman. 2004.

FRICK, Silvia T. F. de. **O Mercado de Software Corporativo no Brasil em 2005**. Disponível em: < si.uniminas.br/~marcio/SIE/> Acesso em: 29 de janeiro de 2009.

HUGO, Marcel. **Engenharia de Software**. Artigo. Blumenau, 2002.

KERIEVSKY, Joshua. **Refactoring to Patterns**. Addison-Wesley, 2005.

LIMA, Mirian Fuá de**. Análise do Processo de Manutenção de Software em Empresas de Porto União da Vitória.** Monografia de Pós-Graduação em Nível de Especialização em Tecnologia de Desenvolvimento de Sistemas. Universidade Regional de Blumenau, Blumenau. 2001.

MACORATTI, José Carlos. **Desenvolvimento e Manutenção de Software.** Disponível em: <www.macoratti.net/vb_mds1.htm>. Acesso em: 17 de Outubro de 2008.

McCONNELL, Steve. **Code Complete Um guia prático para a construção de Software**. Porto Alegre: Bookman, 2005

MENS, T. e **TOURWÉ**, T. **A Survey of Software Refactoring**. IEEE Transactions on Software Engineering, Vol. 30, No. 2, 2004.

PARIKH, Girish. Reengenharia de Software: Técnicas de Manutenção de Programas e Sistemas. Rio de Janeiro: LTC. 1990.

PRESSMAN, Roger S. **Engenharia de Software.** 5ª Ed. Rio de Janeiro: Mcgraw-hill, 2002. 843 p.

ROBSON. Um Pouco sobre o Delphi. Artigo. 2005. Disponível em <http://www.scriptfacil.com/materia/685/artigo/um-pouco-sobre-o-delphi.html> Acesso em : 15 de setembro de 2008.

SCHAAB, Gustavo Luis. **Ferramenta de Suporte ao Processo de Refatoração em Projetos de Software**. 122 f. Monografia - Instituto de Ciências Exatas e Tecnológicas, Novo Hamburgo, 2006**.**

SCUSSIATO, Edésio. **Processo de Manutenção de Sistemas Baseado na Norma ISO-IEC 12207**. Monografia de Pós-Graduação em Nível de Especialização em

Tecnologia de Desenvolvimento de Sistemas - Centro de Ciências Exatas e Naturais, Universidade Regional de Blumenau, Blumenau. 1998.

SOMMERVILLE, Ian. **Engenharia de Software.** 6ª Edição São Paulo: Addison Wesley, 2003. 592 p.

SOUSA, Marcelo P. de. **Avaliação da Qualidade do Processo de Manutenção de Software Utilizando a Norma NBR ISO/IEC 12.207**. Monografia. Blumenau. Junho, 2002.

ANEXO A

DOCUMENTAÇÃO DO SISTEMA GERENCIADOR DE ESCOLA E HOSPITAL VETERINÁRIO DA UENP – FALM

Banco de Dados:

Banco feito em SQL utilizando Interbase do Delphi 7, a UI utilizada foi do IBExpert Free;
Usuário IBExpert: SYSDBA
Senha IBExpert: masterkey
Charset: ASCII

Tabelas na ordem de dependência:

```
 CREATE TABLE aulas(
cod_aula INTEGER NOT NULL,
aula VARCHAR (80) NOT NULL,

CONSTRAINT pk_aulas PRIMARY KEY (cod_aula),
CONSTRAINT unq_aula UNIQUE (aula)
);

CREATE TABLE doacoes(
cod_doacao INTEGER NOT NULL,
dia DATE NOT NULL,
doador VARCHAR (80) NOT NULL,
cnpj_cpf VARCHAR (20) NOT NULL,

CONSTRAINT pk_doacoes PRIMARY KEY (cod_doacao)
);

CREATE TABLE fornecedores(
cod_fornecedor INTEGER NOT NULL,
razao_social VARCHAR (80) NOT NULL,
endereco VARCHAR (60) NOT NULL,
bairro VARCHAR (60) NOT NULL,
cidade VARCHAR (60) NOT NULL,
cep VARCHAR (8) NOT NULL,
uf VARCHAR (2) NOT NULL,
telefone VARCHAR (10) NOT NULL,
insc_estadual VARCHAR (20) NOT NULL,
cnpj VARCHAR (20) NOT NULL,

CONSTRAINT pk_fornecedores PRIMARY KEY (cod_fornecedor),
CONSTRAINT unq_cnpj UNIQUE (cnpj),
CONSTRAINT unq_insc_estadual UNIQUE (insc_estadual)
);

CREATE TABLE pacientes(
```

```
rg VARCHAR(20) NOT NULL,
nome VARCHAR (60) NOT NULL,
especie VARCHAR (10) NOT NULL,
proprietario VARCHAR (80) NOT NULL,

CONSTRAINT pk_pacientes PRIMARY KEY (rg)
);

CREATE TABLE requisitantes(
cod_requisitante INTEGER NOT NULL,
nome VARCHAR (80) NOT NULL,

CONSTRAINT pk_requisitantes PRIMARY KEY (cod_requisitante),
CONSTRAINT unq_nome UNIQUE (nome)
);

CREATE TABLE unidades(
cod_unidade INTEGER NOT NULL,
tipo VARCHAR (10) NOT NULL,

CONSTRAINT pk_unidades PRIMARY KEY (cod_unidade),
CONSTRAINT unq_tipo UNIQUE (tipo)
);

CREATE TABLE usuarios(
login VARCHAR (20) NOT NULL,
senha VARCHAR (12) NOT NULL,

CONSTRAINT pk_usuarios PRIMARY KEY (login)
);

CREATE TABLE entradas(
nota_fiscal VARCHAR (20) NOT NULL,
cod_fornecedor INTEGER NOT NULL,
dia DATE NOT NULL,
valor_total NUMERIC (10,2),

CONSTRAINT pk_entradas PRIMARY KEY (cod_fornecedor,nota_fiscal),
CONSTRAINT fk_entradas_fornecedores FOREIGN KEY (cod_fornecedor) REFERENCES
fornecedores (cod_fornecedor)
);

CREATE TABLE produtos(
cod_produto VARCHAR(20) NOT NULL,
cod_unidade INTEGER NOT NULL,
nome VARCHAR (80) NOT NULL,
preco_venda NUMERIC (10,2) NOT NULL,
quantidade_total NUMERIC (10,4) NOT NULL,

CONSTRAINT pk_produtos PRIMARY KEY (cod_produto),
```

```
CONSTRAINT fk_produtos_unidades FOREIGN KEY (cod_unidade) REFERENCES
unidades (cod_unidade) ON DELETE CASCADE,
CONSTRAINT chk_produtos_preco_venda CHECK (preco_venda >= 0),
CONSTRAINT chk_produtos_quantidade_total CHECK (quantidade_total >= 0)
);

CREATE TABLE itens_doacoes(
cod_doacao INTEGER NOT NULL,
cod_produto VARCHAR(20) NOT NULL,
quantidade NUMERIC (10,4) NOT NULL,

CONSTRAINT pk_itens_doacoes PRIMARY KEY (cod_doacao,cod_produto),
CONSTRAINT fk_itens_doacoes_doacoes FOREIGN KEY (cod_doacao) REFERENCES
doacoes (cod_doacao)  ON DELETE CASCADE,
CONSTRAINT fk_itens_doacoes_produtos FOREIGN KEY (cod_produto) REFERENCES
produtos (cod_produto),
CONSTRAINT chk_itens_doacoes_quantidade CHECK (quantidade >0)
);

CREATE TABLE itens_entradas(
cod_produto VARCHAR(20) NOT NULL,
nota_fiscal VARCHAR (20) NOT NULL,
cod_fornecedor INTEGER NOT NULL,
quantidade NUMERIC (10,4) NOT NULL,
valor_item NUMERIC (10,2) NOT NULL,

CONSTRAINT pk_itens_entradas PRIMARY KEY
(cod_produto,nota_fiscal,cod_fornecedor),
CONSTRAINT fk_itens_entradas_produtos FOREIGN KEY (cod_produto) REFERENCES
produtos (cod_produto),
CONSTRAINT fk_itens_entradas_entradas FOREIGN KEY (cod_fornecedor,nota_fiscal)
REFERENCES entradas (cod_fornecedor,nota_fiscal),
CONSTRAINT chk_itens_entradas_quantidade CHECK (quantidade > 0),
CONSTRAINT chk_itens_entradas_valor_item CHECK (valor_item >= 0)
);

CREATE TABLE saidas(
cod_saida INTEGER NOT NULL,
rg VARCHAR(20) NOT NULL,
cod_aula INTEGER NOT NULL,
cod_requisitante INTEGER NOT NULL,
valor_total_saida NUMERIC (10,2) NOT NULL,
dia DATE NOT NULL,

CONSTRAINT pk_saidas PRIMARY KEY (cod_saida),
CONSTRAINT fk_saidas_pacientes FOREIGN KEY (rg) REFERENCES pacientes (rg),
CONSTRAINT fk_saidas_aulas FOREIGN KEY (cod_aula) REFERENCES aulas
(cod_aula),
CONSTRAINT fk_saidas_requisitantes FOREIGN KEY (cod_requisitante) REFERENCES
requisitantes (cod_requisitante),
```

```
CONSTRAINT chk_saidas_valor_total_saida CHECK (valor_total_saida >=0)
);

CREATE TABLE itens_saidas(
cod_saida INTEGER NOT NULL,
cod_produto VARCHAR(20) NOT NULL,
quantidade NUMERIC (10,4) NOT NULL,
tipo VARCHAR(10) NOT NULL,

CONSTRAINT pk_itens_saidas PRIMARY KEY (cod_saida,cod_produto),
CONSTRAINT fk_itens_saidas_saidas FOREIGN KEY (cod_saida) REFERENCES saidas
(cod_saida),
CONSTRAINT fk_itens_saidas_produtos FOREIGN KEY (cod_produto) REFERENCES
produtos (cod_produto),
CONSTRAINT chk_itens_saidas_quantidade CHECK (quantidade > 0)
);
```

Relatórios, triggers e generators encontram-se dentro do diretório SQL.

ANEXO B

Código-Fonte Original

```
unit U_Cad_Saida_Farmacia;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, StdCtrls, Buttons, ExtCtrls, ComCtrls, Mask, DBCtrls, Grids,
  DBGrids;

type
  TF_Cad_Saida_Farmacia = class(TForm)
    PCadSaidaFarmacia: TPanel;
    GBSaidaFarmacia: TGroupBox;
    POpcoes: TPanel;
    BTNIniciar: TBitBtn;
    BTNInserir: TBitBtn;
    BTNCancelar: TBitBtn;
    GBInfoSaida: TGroupBox;
    DBECodRequisitante: TDBEdit;
    DBECodDisciplina: TDBEdit;
    Label4: TLabel;
    Label3: TLabel;
    DBERG: TDBEdit;
    Label2: TLabel;
    Label1: TLabel;
    SBPesquisa1: TSpeedButton;
    Label10: TLabel;
    Label9: TLabel;
    Label8: TLabel;
    DBEFDia: TDBEdit;
```

Label6: TLabel;
SPData: TSpeedButton;
MCData: TMonthCalendar;
GBItensSaida: TGroupBox;
LValorTotal: TLabel;
Label11: TLabel;
Label5: TLabel;
DBEIFCodSaida: TDBEdit;
Label14: TLabel;
DBEIFCodProduto: TDBEdit;
Label15: TLabel;
DBEIFQtd: TDBEdit;
DBEIFProduto: TDBEdit;
Label16: TLabel;
DBGItensFarmacia: TDBGrid;
DBTFCodSaida: TDBText;
EIFCodProduto: TEdit;
EIFQtdSaida: TEdit;
SBPesquisa2: TSpeedButton;
SBPesquisa3: TSpeedButton;
DBEPaciente: TDBEdit;
DBEFRG: TDBEdit;
DBEFCodDisciplina: TDBEdit;
DBEFCodRequisitante: TDBEdit;
DBEDisciplina: TDBEdit;
DBERequisitante: TDBEdit;
Panel1: TPanel;
BTNCommit: TBitBtn;
BTNRollback: TBitBtn;
BTNSair: TBitBtn;
Panel2: TPanel;
BTNIniciar2: TBitBtn;
BTNInserir2: TBitBtn;
BTNCancelar2: TBitBtn;

```
BTNRemover: TBitBtn;
DBEFValorTotal: TDBEdit;
Label7: TLabel;
RBAula: TRadioButton;
RBCirurgia: TRadioButton;
DBEFTipoSaida: TDBEdit;
SpeedButton1: TSpeedButton;
procedure BTNIniciarClick(Sender: TObject);
procedure BTNInserirClick(Sender: TObject);
procedure BTNCommitClick(Sender: TObject);
procedure BTNRollbackClick(Sender: TObject);
procedure BTNSairClick(Sender: TObject);
procedure FormShow(Sender: TObject);
procedure FormClose(Sender: TObject; var Action: TCloseAction);
procedure BTNCancelarClick(Sender: TObject);
procedure SPDataClick(Sender: TObject);
procedure MCDataClick(Sender: TObject);
procedure MCDataDblClick(Sender: TObject);
procedure SBPesquisa1Click(Sender: TObject);
procedure SBPesquisa2Click(Sender: TObject);
procedure SBPesquisa3Click(Sender: TObject);
procedure RBAulaClick(Sender: TObject);
procedure RBCirurgiaClick(Sender: TObject);
procedure BTNIniciar2Click(Sender: TObject);
procedure BTNInserir2Click(Sender: TObject);
procedure BTNCancelar2Click(Sender: TObject);
procedure BTNRemoverClick(Sender: TObject);
procedure EIFCodProdutoClick(Sender: TObject);
procedure EIFCodProdutoKeyUp(Sender: TObject; var Key: Word;
  Shift: TShiftState);
procedure EIFQtdSaidaKeyUp(Sender: TObject; var Key: Word;
  Shift: TShiftState);
procedure setDBG;
procedure verificaEstoque;
```

```
    procedure SpeedButton1Click(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  F_Cad_Saida_Farmacia: TF_Cad_Saida_Farmacia;
  cont: String;
  cont2: Integer;

implementation

uses U_DataModule, U_Pesquisa, Math;

{$R *.dfm}

procedure TF_Cad_Saida_Farmacia.FormShow(Sender: TObject);
begin
  BTNIniciar.Default:=TRUE;
  DM.IBQFarmacia.Active:=TRUE;
//  DM.IBQFarmacia.Last;
  DM.IBQItensFarmacia.Active:=TRUE;
//  DM.IBQItensFarmacia.Last;
  DM.IBQPacientes.Active:=TRUE;
//  DM.IBQPacientes.Last;
  DM.IBQRequisitantes.Active:=TRUE;
//  DM.IBQRequisitantes.Last;
  DM.IBQDisciplinas.Active:=TRUE;
//  DM.IBQDisciplinas.First;
  MCData.Date:=Date;
end;
```

```
procedure TF_Cad_Saida_Farmacia.FormClose(Sender: TObject; var Action:
TCloseAction);
begin
  DM.IBQFarmacia.First;
  DM.IBQItensFarmacia.First;
{  while not DM.IBQFarmacia.Eof do
  begin
    DM.IBQItensFarmacia.Close;
    DM.IBQItensFarmacia.SQL.Text:='SELECT * FROM itens_saidas WHERE
cod_saida LIKE :cs';

DM.IBQItensFarmacia.ParamByName('cs').AsString:=DM.IBQFarmaciaCOD_SAIDA.
AsString;
    DM.IBQItensFarmacia.Open;
    if (DM.IBQItensFarmaciaCOD_SAIDA.AsString <>
DM.IBQFarmaciaCOD_SAIDA.AsString) then
    begin
      DM.IBQFarmacia.Delete;
      DM.IBTrans.Commit;
      DM.IBQFarmacia.Open;
      DM.IBQFarmacia.First;
    end;
    if DM.IBQFarmacia.RecordCount > 1 then
      DM.IBQFarmacia.Next
    else
      DM.IBQFarmacia.Last;
  end;}
  DM.IBQPacientes.Active:=FALSE;
  DM.IBQPacientes.Close;
  DM.IBQRequisitantes.Active:=FALSE;
  DM.IBQRequisitantes.Close;
  DM.IBQDisciplinas.Active:=FALSE;
  DM.IBQDisciplinas.Close;
end;
```

```
procedure TF_Cad_Saida_Farmacia.BTNIniciarClick(Sender: TObject);
begin
  DM.IBQFarmacia.Insert;

  DBEFValorTotal.Text:='0';
  DBEFRG.Text:=DBERG.Text;
  RBAula.Checked:=TRUE;
  RBAula.Enabled:=FALSE;
  DBEFCodDisciplina.Text:=DBECodDisciplina.Text;
  DBEFCodRequisitante.Text:=DBECodRequisitante.Text;
  DBEFDia.Text:=DateToStr(MCData.Date);

  GBInfoSaida.Enabled:=TRUE;
  BTNIniciar.Enabled:=FALSE;
  BTNInserir.Enabled:=TRUE;
  BTNCancelar.Enabled:=TRUE;

  if BTNCommit.Enabled = TRUE then
    BTNCommit.Enabled:=FALSE;
  if BTNRollback.Enabled = TRUE then
    BTNRollback.Enabled:=FALSE;

  BTNIniciar.Default:=FALSE;
  BTNInserir.Default:=TRUE;
end;

procedure TF_Cad_Saida_Farmacia.BTNInserirClick(Sender: TObject);
begin
  DM.IBQFarmacia.Post;

  GBInfoSaida.Enabled:=FALSE;

  cont:='1';
```

```
  cont2:=1;

  BTNInserir.Enabled:=FALSE;
  BTNInserir.Default:=FALSE;
  BTNCancelar.Enabled:=FALSE;
  BTNCommit.Enabled:=TRUE;
  BTNRollback.Enabled:=TRUE;
end;

procedure TF_Cad_Saida_Farmacia.BTNCancelarClick(Sender: TObject);
begin
  DM.IBQFarmacia.Cancel;
  GBInfoSaida.Enabled:=FALSE;
  BTNIniciar.Enabled:=TRUE;
  BTNInserir.Enabled:=FALSE;
  BTNCancelar.Enabled:=FALSE;
  BTNIniciar.Default:=TRUE;
  BTNInserir.Default:=FALSE;
end;

procedure TF_Cad_Saida_Farmacia.BTNCommitClick(Sender: TObject);
begin
  if MessageDlg('Ao comfirmar você estára adicionando todas as informações
fornecidas até esse momento na tabela, deseja continuar?', mtConfirmation, [mbYes,
mbNo],0) = mrYes then
  begin
    if cont = '1' then
    begin
      BTNIniciar.Enabled:=FALSE;
      BTNIniciar.Default:=FALSE;
      BTNIniciar2.Default:=TRUE;
      BTNIniciar2.Enabled:=TRUE;
    end
    else
```

```
    begin
      BTNIniciar.Enabled:=TRUE;
      BTNIniciar.Default:=TRUE;
      BTNIniciar2.Default:=FALSE;
      BTNIniciar2.Enabled:=FALSE;
      BTNRemover.Enabled:=FALSE;
    end;
    if cont2 = 0 then
{    begin
      DM.IBQProdutos.Edit;

DM.IBQProdutosQUANTIDADE_TOTAL.AsString:=IntToStr((StrToInt(DM.IBQProdut
osQUANTIDADE_TOTAL.AsString))-(StrToInt(EIFQtdSaida.Text)));
      DM.IBQProdutos.Post;
    end;}
    cont:='0';
    BTNCommit.Enabled:=FALSE;
    BTNRollback.Enabled:=FALSE;
    DM.IBTrans.Commit;
    DM.IBTrans.StartTransaction;
    DM.IBQFarmacia.Open;
    DM.IBQFarmacia.Last;
    DM.IBQItensFarmacia.Open;
//    DM.IBQItensFarmacia.Last;
    DM.IBQPacientes.Open;
    DM.IBQDisciplinas.Open;
    DM.IBQRequisitantes.Open;
    DM.IBQProdutos.Open;
    setDBG;
  end;
end;

procedure TF_Cad_Saida_Farmacia.BTNRollbackClick(Sender: TObject);
begin
```

```
  if MessageDlg('Ao voltar, todas informações fornecidas des da última comfirmação
serão perdidas, deseja voltar?', mtConfirmation, [mbYes, mbNo],0) = mrYes then
  begin
    if cont2 = 1 then
    begin
      BTNIniciar.Default:=TRUE;
      BTNIniciar.Enabled:=TRUE;
      BTNInserir.Default:=FALSE;
      BTNCommit.Enabled:=FALSE;
      BTNRollback.Enabled:=FALSE;
      DM.IBQFarmacia.Close;
      DM.IBTrans.Rollback;
    end;
    if cont2 = 0 then
    begin
      BTNIniciar2.Default:=TRUE;
      BTNInserir2.Default:=FALSE;
      BTNCommit.Enabled:=FALSE;
      BTNRemover.Enabled:=FALSE;
      BTNRollback.Enabled:=FALSE;
      DM.IBQItensFarmacia.Close;
      DM.IBTrans.Rollback;
    end;
    DM.IBQFarmacia.Open;
    DM.IBQFarmacia.Last;
    DM.IBQItensFarmacia.Open;
    DM.IBQItensFarmacia.Last;
    DM.IBQDisciplinas.Open;
    DM.IBQRequisitantes.Open;
    DM.IBQPacientes.Open;
    LValorTotal.Caption:='0,00';
  end;
end;
procedure TF_Cad_Saida_Farmacia.BTNSairClick(Sender: TObject);
```

```
begin
  Self.Close;
end;

procedure TF_Cad_Saida_Farmacia.SPDataClick(Sender: TObject);
begin
  if MCData.Visible = FALSE then
    MCData.Show
  else
    MCData.Hide;
end;

procedure TF_Cad_Saida_Farmacia.MCDataClick(Sender: TObject);
begin
  DBEFDia.Text:=DateToStr(MCData.Date);
end;

procedure TF_Cad_Saida_Farmacia.MCDataDblClick(Sender: TObject);
begin
  MCData.Hide;
end;

procedure TF_Cad_Saida_Farmacia.SBPesquisa1Click(Sender: TObject);
begin
  F_Pesquisa.setOP(1);
  F_Pesquisa.CBPaciente.Checked:=TRUE;
  F_Pesquisa.DBGPesquisa.DataSource:=DM.DSPacientes;
  F_Pesquisa.Show;
end;

procedure TF_Cad_Saida_Farmacia.SBPesquisa2Click(Sender: TObject);
begin
  F_Pesquisa.setOP(1);
  F_Pesquisa.CBDisciplina.Checked:=TRUE;
```

```
  F_Pesquisa.DBGPesquisa.DataSource:=DM.DSDisciplinas;
  F_Pesquisa.Show;
end;

procedure TF_Cad_Saida_Farmacia.SBPesquisa3Click(Sender: TObject);
begin
  F_Pesquisa.setOP(1);
  F_Pesquisa.CBRequisitante.Checked:=TRUE;
  F_Pesquisa.DBGPesquisa.DataSource:=DM.DSRequisitantes;
  F_Pesquisa.Show;
end;

procedure TF_Cad_Saida_Farmacia.RBAulaClick(Sender: TObject);
begin
  DBEFTipoSaida.Text:=RBAula.Caption;
  RBAula.Enabled:=FALSE;
  RBCirurgia.Enabled:=TRUE;
end;

procedure TF_Cad_Saida_Farmacia.RBCirurgiaClick(Sender: TObject);
begin
  DBEFTipoSaida.Text:=RBCirurgia.Caption;
  RBCirurgia.Enabled:=FALSE;
  RBAula.Enabled:=TRUE;
end;

procedure TF_Cad_Saida_Farmacia.BTNIniciar2Click(Sender: TObject);
begin
  DM.IBQItensFarmacia.Insert;
  DM.IBQFarmacia.Edit;
  GBItensSaida.Enabled:=TRUE;
  BTNIniciar2.Enabled:=FALSE;
  BTNInserir2.Enabled:=TRUE;
  BTNCancelar2.Enabled:=TRUE;
```

```
  if BTNCommit.Enabled = TRUE then
    BTNCommit.Enabled:=FALSE;
  if BTNRollback.Enabled = TRUE then
    BTNRollback.Enabled:=FALSE;
  BTNIniciar2.Default:=FALSE;
  EIFCodProduto.SetFocus;
end;

procedure TF_Cad_Saida_Farmacia.BTNInserir2Click(Sender: TObject);
begin
  DBEFTipoSaida.Text:='Saída';
  DBEIFCodSaida.Text:=DBTFCodSaida.Caption;
  DBEIFCodProduto.Text:=EIFCodProduto.Text;
  DBEIFQtd.Text:=EIFQtdSaida.Text;

LValorTotal.Caption:=FloatToStr(StrToFloat(LValorTotal.Caption)+(StrToFloat(DM.IB
QProdutosPRECO_VENDA.AsString)*StrToFloat(EIFQtdSaida.Text)));
  LValorTotal.Caption:=FloatToStr(RoundTo(StrToFloat(LValorTotal.Caption),-2));
  DBEFValorTotal.Text:=LValorTotal.Caption;

  DM.IBQItensFarmacia.Post;
  DM.IBQFarmacia.Post;

  GBItensSaida.Enabled:=FALSE;

  cont:='2';
  cont2:=0;

  BTNIniciar2.Enabled:=TRUE;
  BTNInserir2.Enabled:=FALSE;
  BTNCancelar2.Enabled:=FALSE;
  BTNCommit.Enabled:=TRUE;
  BTNRollback.Enabled:=TRUE;
  BTNIniciar2.Default:=TRUE;
```

```
  verificaEstoque;
  DM.IBQProdutos.Edit;

DM.IBQProdutosQUANTIDADE_TOTAL.AsString:=IntToStr((StrToInt(DM.IBQProdut
osQUANTIDADE_TOTAL.AsString))-(StrToInt(EIFQtdSaida.Text)));
  DM.IBQProdutos.Post;
end;

procedure TF_Cad_Saida_Farmacia.BTNCancelar2Click(Sender: TObject);
begin
  DM.IBQItensFarmacia.Cancel;
  GBItensSaida.Enabled:=FALSE;
  BTNIniciar2.Enabled:=TRUE;
  BTNInserir2.Enabled:=FALSE;
  BTNCancelar2.Enabled:=FALSE;
  BTNIniciar2.Default:=TRUE;
  BTNInserir2.Default:=FALSE;
  if BTNCommit.Enabled = FALSE then
    BTNCommit.Enabled:=TRUE;
  if BTNRollback.Enabled = FALSE then
    BTNRollback.Enabled:=TRUE;
end;

procedure TF_Cad_Saida_Farmacia.BTNRemoverClick(Sender: TObject);
begin
  DM.IBQItensFarmacia.Open;
  try
    DM.IBQItensFarmacia.Delete;
  finally
    if BTNRollback.Enabled = FALSE then
      BTNRollback.Enabled:=TRUE;
    if BTNCommit.Enabled = FALSE then
      BTNCommit.Enabled:=TRUE;
    with DBGItensFarmacia.DataSource.DataSet,DBGItensFarmacia do
```

```
    begin
      if(Trim(FieldByName('COD_SAIDA').Text) = '') then
        BTNRemover.Enabled:=FALSE;
    end;
  end;
end;

procedure TF_Cad_Saida_Farmacia.EIFCodProdutoClick(Sender: TObject);
begin
  DM.IBQProdutos.Active:=FALSE;
  EIFCodProduto.Clear;
end;

procedure TF_Cad_Saida_Farmacia.EIFCodProdutoKeyUp(Sender: TObject;
  var Key: Word; Shift: TShiftState);
begin
  if EIFCodProduto.Text<>'' then
  begin
    DM.IBQProdutos.Close;
    DM.IBQProdutos.SQL.Text:='SELECT * FROM produtos WHERE cod_produto
LIKE :cp';
    DM.IBQProdutos.ParamByName('cp').AsString:=EIFCodProduto.Text;
    DM.IBQProdutos.Open;
    DM.IBQProdutos.Active:=TRUE;
  end
  else
  begin
    DM.IBQProdutos.Close;
    DM.IBQProdutos.SQL.Text:='SELECT * FROM produtos';
    DM.IBQProdutos.Open;
  end;
  DM.IBQProdutos.Edit;
end;
procedure TF_Cad_Saida_Farmacia.EIFQtdSaidaKeyUp(Sender: TObject;
```

```
  var Key: Word; Shift: TShiftState);
begin
  DM.PontoVirgula(EIFQtdSaida.Text);
  if (trim(EIFQtdSaida.Text)) <> '' then
  begin
    if (StrToFloat(EIFQtdSaida.Text) >
StrToFloat(DM.IBQProdutosQUANTIDADE_TOTAL.AsString)) then
      EIFQtdSaida.Text:=DM.IBQProdutosQUANTIDADE_TOTAL.AsString;
  end;
end;

procedure TF_Cad_Saida_Farmacia.setDBG;
begin
  DM.IBQItensFarmacia.Close;
  DM.IBQItensFarmacia.SQL.Text:='SELECT * FROM itens_saidas WHERE
cod_saida LIKE :cs';
  DM.IBQItensFarmacia.ParamByName('cs').AsString:=DBTFCodSaida.Caption;
  DM.IBQItensFarmacia.Open;
end;

procedure TF_Cad_Saida_Farmacia.verificaEstoque;
begin
  if StrToInt(EIFQtdSaida.Text) =
StrToInt(DM.IBQProdutosQUANTIDADE_TOTAL.AsString) then
     showmessage('produto "'+DM.IBQProdutosNOME.AsString+'" em falta,
reabastecer o estoque')
  else
    if ((StrToInt(DM.IBQProdutosQUANTIDADE_TOTAL.AsString))-
(StrToInt(EIFQtdSaida.Text))) <= 10 then
      showmessage('produto "'+DM.IBQProdutosNOME.AsString+'" com pouca
quantidade, precisará reabastecer em breve');
end;

procedure TF_Cad_Saida_Farmacia.SpeedButton1Click(Sender: TObject);
```

```
begin
  F_Pesquisa.setOP(4);
  F_Pesquisa.CBProduto.Checked:=TRUE;
  F_Pesquisa.DBGPesquisa.DataSource:=DM.DSProdutos;
  F_Pesquisa.Show;
  DM.IBQItensFarmacia.Cancel;
end;

end.
```

Código-Fonte Após as Refatorações

```
{
  Autor: Rodolfo Caceraghi dos Santos
  Formulário de repositório com as funções básicas para INSERÇÃO de um cadastro
}

unit U_Cad_Saida_Farmacia;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, StdCtrls, Buttons, ExtCtrls, ComCtrls, Mask, DBCtrls, Grids,
  DBGrids;

type
  TF_Cad_Saida_Farmacia = class(TForm)
    PCadSaidaFarmacia: TPanel;
    GBSaidaFarmacia: TGroupBox;
    POpcoesNota: TPanel;
    BTNIniciarNota: TBitBtn;
    BTNInserirNota: TBitBtn;
    BTNCancelarNota: TBitBtn;
    GBInfoSaida: TGroupBox;
    DBECodRequisitante: TDBEdit;
    DBECodDisciplina: TDBEdit;
    Label4: TLabel;
    Label3: TLabel;
    DBERG: TDBEdit;
    Label2: TLabel;
    Label1: TLabel;
    SBPesquisaRG: TSpeedButton;
    Label10: TLabel;
    Label9: TLabel;
```

Label8: TLabel;
DBEFDia: TDBEdit;
Label6: TLabel;
SPData: TSpeedButton;
MCData: TMonthCalendar;
GBItensSaida: TGroupBox;
LValorTotal: TLabel;
Label11: TLabel;
Label5: TLabel;
Label14: TLabel;
Label15: TLabel;
DBEIFProduto: TDBEdit;
Label16: TLabel;
DBGItensFarmacia: TDBGrid;
DBTFCodSaida: TDBText;
EIFCodProduto: TEdit;
EIFQtdSaida: TEdit;
SBPesquisaCodDisciplina: TSpeedButton;
SBPesquisaCodRequisitante: TSpeedButton;
DBEPaciente: TDBEdit;
DBEFRG: TDBEdit;
DBEFCodDisciplina: TDBEdit;
DBEFCodRequisitante: TDBEdit;
DBEDisciplina: TDBEdit;
DBERequisitante: TDBEdit;
Panel1: TPanel;
BTNCommit: TBitBtn;
BTNRollback: TBitBtn;
BTNSair: TBitBtn;
POpcoesItens: TPanel;
BTNIniciarItens: TBitBtn;
BTNInserirItens: TBitBtn;
BTNCancelarItens: TBitBtn;
BTNRemoverItens: TBitBtn;

```
  DBEFValorTotal: TDBEdit;
  Label7: TLabel;
  RBAula: TRadioButton;
  RBCirurgia: TRadioButton;
  SBPesquisaCodProduto: TSpeedButton;
  procedure BTNIniciarNotaClick(Sender: TObject);
  procedure BTNInserirNotaClick(Sender: TObject);
  procedure BTNCommitClick(Sender: TObject);
  procedure BTNRollbackClick(Sender: TObject);
  procedure BTNSairClick(Sender: TObject);
  procedure FormShow(Sender: TObject);
  procedure FormClose(Sender: TObject; var Action: TCloseAction);
  procedure BTNCancelarNotaClick(Sender: TObject);
  procedure SPDataClick(Sender: TObject);
  procedure MCDataClick(Sender: TObject);
  procedure MCDataDblClick(Sender: TObject);
  procedure SBPesquisaRGClick(Sender: TObject);
  procedure SBPesquisaCodDisciplinaClick(Sender: TObject);
  procedure SBPesquisaCodRequisitanteClick(Sender: TObject);
  procedure RBAulaClick(Sender: TObject);
  procedure RBCirurgiaClick(Sender: TObject);
  procedure BTNIniciarItensClick(Sender: TObject);
  procedure BTNInserirItensClick(Sender: TObject);
  procedure BTNCancelarItensClick(Sender: TObject);
  procedure BTNRemoverItensClick(Sender: TObject);
  procedure EIFCodProdutoClick(Sender: TObject);
  procedure EIFCodProdutoKeyUp(Sender: TObject; var Key: Word;
    Shift: TShiftState);
  procedure EIFQtdSaidaKeyUp(Sender: TObject; var Key: Word;
    Shift: TShiftState);
  procedure setDBG;
  procedure verificaEstoque;
  procedure SBPesquisaCodProdutoClick(Sender: TObject);
private
```

```
    { Private declarations }
  public
    { Public declarations }
  end;

var
  F_Cad_Saida_Farmacia: TF_Cad_Saida_Farmacia;
  UltimoRegistroInserido, TipoSaida: string;

implementation

uses U_DataModule, U_Pesquisa, Math;

{$R *.dfm}

procedure TF_Cad_Saida_Farmacia.FormShow(Sender: TObject);
begin
  DM.IBQFarmacia.Active:=TRUE;
  DM.IBQItensFarmacia.Active:=TRUE;
  DM.IBQPacientes.Active:=TRUE;
  DM.IBQRequisitantes.Active:=TRUE;
  DM.IBQDisciplinas.Active:=TRUE;

  MCData.Date:=Date;
end;

procedure TF_Cad_Saida_Farmacia.FormClose(Sender: TObject; var Action:
TCloseAction);
begin
  DM.IBQFarmacia.First;
  DM.IBQItensFarmacia.First;
  DM.IBQPacientes.Active:=FALSE;
  DM.IBQPacientes.Close;
  DM.IBQRequisitantes.Active:=FALSE;
```

```
  DM.IBQRequisitantes.Close;
  DM.IBQDisciplinas.Active:=FALSE;
  DM.IBQDisciplinas.Close;
end;

procedure TF_Cad_Saida_Farmacia.BTNIniciarNotaClick(Sender: TObject);
begin
  DM.IBQFarmacia.Insert;

  DBEFValorTotal.Text:='0';
  DBEFRG.Text:=DBERG.Text;
  RBAula.Checked:=TRUE;
  RBAula.Enabled:=FALSE;
  DBEFCodDisciplina.Text:=DBECodDisciplina.Text;
  DBEFCodRequisitante.Text:=DBECodRequisitante.Text;
  DBEFDia.Text:=DateToStr(MCData.Date);

  GBInfoSaida.Enabled:=TRUE;
  BTNIniciarNota.Enabled:=FALSE;
  BTNInserirNota.Enabled:=TRUE;
  BTNCancelarNota.Enabled:=TRUE;

  if BTNCommit.Enabled  and BTNRollback.Enabled then
  begin
    BTNCommit.Enabled:=FALSE;
    BTNRollback.Enabled:=FALSE;
  end;

  BTNIniciarNota.Default:=FALSE;
  BTNInserirNota.Default:=TRUE;

end;

procedure TF_Cad_Saida_Farmacia.BTNInserirNotaClick(Sender: TObject);
```

```
begin
  DM.IBQFarmacia.Post;

  GBInfoSaida.Enabled:=FALSE;

  UltimoRegistroInserido:='saida';

  BTNInserirNota.Enabled:=FALSE;
  BTNInserirNota.Default:=FALSE;
  BTNCancelarNota.Enabled:=FALSE;

  BTNCommit.Enabled:=TRUE;
  BTNRollback.Enabled:=TRUE;
end;

procedure TF_Cad_Saida_Farmacia.BTNCancelarNotaClick(Sender: TObject);
begin
  DM.IBQFarmacia.Cancel;
  GBInfoSaida.Enabled:=FALSE;

  BTNIniciarNota.Enabled:=TRUE;
  BTNInserirNota.Enabled:=FALSE;
  BTNCancelarNota.Enabled:=FALSE;
  BTNIniciarNota.Default:=TRUE;
  BTNInserirNota.Default:=FALSE;
end;

procedure TF_Cad_Saida_Farmacia.BTNCommitClick(Sender: TObject);
begin
  if MessageDlg('Ao confirmar você estará adicionando todas as informações
fornecidas até esse momento na tabela, deseja continuar?', mtConfirmation, [mbYes,
mbNo],0) = mrYes then
  begin
    if UltimoRegistroInserido = 'saida' then
```

```
    begin
      BTNIniciarNota.Enabled:=FALSE;
      BTNIniciarNota.Default:=FALSE;

      BTNIniciarItens.Default:=TRUE;
      BTNIniciarItens.Enabled:=TRUE;
    end
    else if UltimoRegistroInserido='item' then
    begin
      BTNIniciarNota.Enabled:=TRUE;
      BTNIniciarNota.Default:=TRUE;

      BTNIniciarItens.Default:=FALSE;
      BTNIniciarItens.Enabled:=FALSE;
      BTNRemoverItens.Enabled:=FALSE;
    end;

    BTNCommit.Enabled:=FALSE;
    BTNRollback.Enabled:=FALSE;

    DM.IBTrans.Commit;
    DM.IBTrans.StartTransaction;

    DM.IBQFarmacia.Open;
    DM.IBQFarmacia.Last;
    DM.IBQItensFarmacia.Open;
    DM.IBQPacientes.Open;
    DM.IBQDisciplinas.Open;
    DM.IBQRequisitantes.Open;
    DM.IBQProdutos.Open;
    setDBG;
  end;
end;
```

```
procedure TF_Cad_Saida_Farmacia.BTNRollbackClick(Sender: TObject);
begin
  if MessageDlg('Ao voltar, todas informações fornecidas desde a última confirmação
serão perdidas, deseja voltar?', mtConfirmation, [mbYes, mbNo],0) = mrYes then
  begin
    if UltimoRegistroInserido = 'saida' then
    begin
      BTNIniciarNota.Default:=TRUE;
      BTNIniciarNota.Enabled:=TRUE;
      BTNInserirNota.Default:=FALSE;
    end
    else if UltimoRegistroInserido = 'item'then
    begin
      BTNIniciarItens.Default:=TRUE;
      BTNInserirItens.Default:=FALSE;
      BTNRemoverItens.Enabled:=FALSE;
    end;

    BTNCommit.Enabled:=FALSE;
    BTNRollback.Enabled:=FALSE;

    DM.IBQFarmacia.Close;
    DM.IBTrans.Rollback;

    DM.IBQFarmacia.Open;
    DM.IBQFarmacia.Last;
    DM.IBQItensFarmacia.Open;
    DM.IBQItensFarmacia.Last;
    DM.IBQDisciplinas.Open;
    DM.IBQRequisitantes.Open;
    DM.IBQPacientes.Open;
    LValorTotal.Caption:='0,00';
  end;
end;
```

```
procedure TF_Cad_Saida_Farmacia.BTNSairClick(Sender: TObject);
begin
  Self.Close;
end;

procedure TF_Cad_Saida_Farmacia.SPDataClick(Sender: TObject);
begin
  if MCData.Visible = FALSE then
    MCData.Show
  else
    MCData.Hide;
end;

procedure TF_Cad_Saida_Farmacia.MCDataClick(Sender: TObject);
begin
  DBEFDia.Text:=DateToStr(MCData.Date);
end;

procedure TF_Cad_Saida_Farmacia.MCDataDblClick(Sender: TObject);
begin
  MCData.Hide;
end;

procedure TF_Cad_Saida_Farmacia.SBPesquisaRGClick(Sender: TObject);
begin
  F_Pesquisa.setOP(1);
  F_Pesquisa.CBPaciente.Checked:=TRUE;
  F_Pesquisa.DBGPesquisa.DataSource:=DM.DSPacientes;
  F_Pesquisa.Show;
end;

procedure TF_Cad_Saida_Farmacia.SBPesquisaCodDisciplinaClick(Sender:
TObject);
```

```
begin
  F_Pesquisa.setOP(1);
  F_Pesquisa.CBDisciplina.Checked:=TRUE;
  F_Pesquisa.DBGPesquisa.DataSource:=DM.DSDisciplinas;
  F_Pesquisa.Show;
end;

procedure TF_Cad_Saida_Farmacia.SBPesquisaCodRequisitanteClick(Sender:
TObject);
begin
  F_Pesquisa.setOP(1);
  F_Pesquisa.CBRequisitante.Checked:=TRUE;
  F_Pesquisa.DBGPesquisa.DataSource:=DM.DSRequisitantes;
  F_Pesquisa.Show;
end;

procedure TF_Cad_Saida_Farmacia.RBAulaClick(Sender: TObject);
begin
  TipoSaida:=RBAula.Caption;
  RBAula.Enabled:=FALSE;
  RBCirurgia.Enabled:=TRUE;
end;

procedure TF_Cad_Saida_Farmacia.RBCirurgiaClick(Sender: TObject);
begin
  TipoSaida:=RBCirurgia.Caption;
  RBCirurgia.Enabled:=FALSE;
  RBAula.Enabled:=TRUE;
end;

procedure TF_Cad_Saida_Farmacia.BTNIniciarItensClick(Sender: TObject);
begin
  DM.IBQItensFarmacia.Insert;
  DM.IBQFarmacia.Edit;
```

```
  GBItensSaida.Enabled:=TRUE;

  BTNIniciarItens.Enabled:=FALSE;
  BTNIniciarItens.Default:=FALSE;
  BTNInserirItens.Enabled:=TRUE;
  BTNCancelarItens.Enabled:=TRUE;

  if BTNCommit.Enabled and BTNRollback.Enabled then
  begin
    BTNCommit.Enabled:=FALSE;
    BTNRollback.Enabled:=FALSE;
  end;


  EIFCodProduto.SetFocus;

end;

procedure TF_Cad_Saida_Farmacia.BTNInserirItensClick(Sender: TObject);
begin
  DM.IBQItensFarmaciaTIPO.AsString:='Saída';
  DM.IBQItensFarmaciaCOD_SAIDA.AsString:=DBTFCodSaida.Caption;
  DM.IBQItensFarmaciaCOD_PRODUTO.AsString:=EIFCodProduto.Text;
  DM.IBQItensFarmaciaQUANTIDADE.AsString:=EIFQtdSaida.Text;

LValorTotal.Caption:=FloatToStr(StrToFloat(LValorTotal.Caption)+(StrToFloat(DM.IB
QProdutosPRECO_VENDA.AsString)*StrToFloat(EIFQtdSaida.Text)));
  LValorTotal.Caption:=FloatToStr(RoundTo(StrToFloat(LValorTotal.Caption),-2));
  DM.IBQFarmaciaVALOR_TOTAL_SAIDA.AsString:=LValorTotal.Caption;

  DM.IBQItensFarmacia.Post;
  DM.IBQFarmacia.Post;

  GBItensSaida.Enabled:=FALSE;
```

```
  UltimoRegistroInserido:='item';

  BTNIniciarItens.Enabled:=TRUE;
  BTNIniciarItens.Default:=TRUE;
  BTNInserirItens.Enabled:=FALSE;
  BTNCancelarItens.Enabled:=FALSE;

  BTNCommit.Enabled:=TRUE;
  BTNRollback.Enabled:=TRUE;

  verificaEstoque;

  DM.IBQProdutos.Edit;

DM.IBQProdutosQUANTIDADE_TOTAL.AsString:=IntToStr((StrToInt(DM.IBQProdut
osQUANTIDADE_TOTAL.AsString))-(StrToInt(EIFQtdSaida.Text)));
  DM.IBQProdutos.Post;
end;

procedure TF_Cad_Saida_Farmacia.BTNCancelarItensClick(Sender: TObject);
begin
  DM.IBQItensFarmacia.Cancel;
  GBItensSaida.Enabled:=FALSE;

  BTNIniciarItens.Enabled:=TRUE;
  BTNInserirItens.Enabled:=FALSE;
  BTNCancelarItens.Enabled:=FALSE;
  BTNIniciarItens.Default:=TRUE;
  BTNInserirItens.Default:=FALSE;

  if not BTNCommit.Enabled and not BTNRollback.Enabled then
  begin
    BTNCommit.Enabled:=TRUE;
```

```
    BTNRollback.Enabled:=TRUE;
  end;
end;

procedure TF_Cad_Saida_Farmacia.BTNRemoverItensClick(Sender: TObject);
begin
  DM.IBQItensFarmacia.Open;
  try
    DM.IBQItensFarmacia.Delete;
  finally
  if not BTNCommit.Enabled and not BTNRollback.Enabled then
  begin
    BTNCommit.Enabled:=TRUE;
    BTNRollback.Enabled:=TRUE;
  end;
    with DBGItensFarmacia.DataSource.DataSet,DBGItensFarmacia do
    begin
      if(Trim(FieldByName('COD_SAIDA').Text) = '') then
        BTNRemoverItens.Enabled:=FALSE;
    end;
  end;
end;

procedure TF_Cad_Saida_Farmacia.EIFCodProdutoClick(Sender: TObject);
begin
  DM.IBQProdutos.Active:=FALSE;
  EIFCodProduto.Clear;
end;

procedure TF_Cad_Saida_Farmacia.EIFCodProdutoKeyUp(Sender: TObject;
  var Key: Word; Shift: TShiftState);
begin
  if EIFCodProduto.Text<>'' then
  begin
```

```
    DM.IBQProdutos.Close;
    DM.IBQProdutos.SQL.Text:='SELECT * FROM produtos WHERE cod_produto
LIKE :cp';
    DM.IBQProdutos.ParamByName('cp').AsString:=EIFCodProduto.Text;
    DM.IBQProdutos.Open;
    DM.IBQProdutos.Active:=TRUE;
  end
  else
  begin
    DM.IBQProdutos.Close;
    DM.IBQProdutos.SQL.Text:='SELECT * FROM produtos';
    DM.IBQProdutos.Open;
  end;
  DM.IBQProdutos.Edit;
end;

procedure TF_Cad_Saida_Farmacia.EIFQtdSaidaKeyUp(Sender: TObject;
  var Key: Word; Shift: TShiftState);
begin
  DM.PontoVirgula(EIFQtdSaida.Text);
  if (trim(EIFQtdSaida.Text)) <> '' then
  begin
    if (StrToFloat(EIFQtdSaida.Text) >
StrToFloat(DM.IBQProdutosQUANTIDADE_TOTAL.AsString)) then
      EIFQtdSaida.Text:=DM.IBQProdutosQUANTIDADE_TOTAL.AsString;
  end;
end;

procedure TF_Cad_Saida_Farmacia.setDBG;
begin
  DM.IBQItensFarmacia.Close;
  DM.IBQItensFarmacia.SQL.Text:='SELECT * FROM itens_saidas WHERE
cod_saida LIKE :cs';
  DM.IBQItensFarmacia.ParamByName('cs').AsString:=DBTFCodSaida.Caption;
```

```
  DM.IBQItensFarmacia.Open;
end;

procedure TF_Cad_Saida_Farmacia.verificaEstoque;
begin
  if StrToInt(EIFQtdSaida.Text) =
StrToInt(DM.IBQProdutosQUANTIDADE_TOTAL.AsString) then
     showmessage('produto "'+DM.IBQProdutosNOME.AsString+'" em falta,
reabastecer o estoque')
  else
    if ((StrToInt(DM.IBQProdutosQUANTIDADE_TOTAL.AsString))-
(StrToInt(EIFQtdSaida.Text))) <= 10 then
      showmessage('produto "'+DM.IBQProdutosNOME.AsString+'" com pouca
quantidade, precisará reabastecer em breve');
end;

procedure TF_Cad_Saida_Farmacia.SBPesquisaCodProdutoClick(Sender: TObject);
begin
  F_Pesquisa.setOP(4);
  F_Pesquisa.CBProduto.Checked:=TRUE;
  F_Pesquisa.DBGPesquisa.DataSource:=DM.DSProdutos;
  F_Pesquisa.Show;
  DM.IBQItensFarmacia.Cancel;
end;

end.
```

APÊNDICES

Código-Fonte Após as Refatorações

```
unit U_Cad_Saida_Farmacia;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, StdCtrls, Buttons, ExtCtrls, ComCtrls, Mask, DBCtrls, Grids,
  DBGrids;

type
  TF_Cad_Saida_Farmacia = class(TForm)
    PCadSaidaFarmacia: TPanel;
    GBSaidaFarmacia: TGroupBox;
    POpcoesNota: TPanel;
    BTNIniciarNota: TBitBtn;
    BTNInserirNota: TBitBtn;
    BTNCancelarNota: TBitBtn;
    GBInfoSaida: TGroupBox;
    DBECodRequisitante: TDBEdit;
    DBECodDisciplina: TDBEdit;
    Label4: TLabel;
    Label3: TLabel;
    DBERG: TDBEdit;
    Label2: TLabel;
    Label1: TLabel;
    SBPesquisaRG: TSpeedButton;
    Label10: TLabel;
    Label9: TLabel;
    Label8: TLabel;
    DBEFDia: TDBEdit;
```

Label6: TLabel;
SPData: TSpeedButton;
MCData: TMonthCalendar;
GBItensSaida: TGroupBox;
LValorTotal: TLabel;
Label11: TLabel;
Label5: TLabel;
Label14: TLabel;
Label15: TLabel;
DBEIFProduto: TDBEdit;
Label16: TLabel;
DBGItensFarmacia: TDBGrid;
DBTFCodSaida: TDBText;
EIFCodProduto: TEdit;
EIFQtdSaida: TEdit;
SBPesquisaCodDisciplina: TSpeedButton;
SBPesquisaCodRequisitante: TSpeedButton;
DBEPaciente: TDBEdit;
DBEFRG: TDBEdit;
DBEFCodDisciplina: TDBEdit;
DBEFCodRequisitante: TDBEdit;
DBEDisciplina: TDBEdit;
DBERequisitante: TDBEdit;
Panel1: TPanel;
BTNCommit: TBitBtn;
BTNRollback: TBitBtn;
BTNSair: TBitBtn;
POpcoesItens: TPanel;
BTNIniciarItens: TBitBtn;
BTNInserirItens: TBitBtn;
BTNCancelarItens: TBitBtn;
BTNRemoverItens: TBitBtn;
DBEFValorTotal: TDBEdit;
Label7: TLabel;

```
  RBAula: TRadioButton;
  RBCirurgia: TRadioButton;
  SBPesquisaCodProduto: TSpeedButton;
  procedure BTNIniciarNotaClick(Sender: TObject);
  procedure BTNInserirNotaClick(Sender: TObject);
  procedure BTNCommitClick(Sender: TObject);
  procedure BTNRollbackClick(Sender: TObject);
  procedure BTNSairClick(Sender: TObject);
  procedure FormShow(Sender: TObject);
  procedure FormClose(Sender: TObject; var Action: TCloseAction);
  procedure BTNCancelarNotaClick(Sender: TObject);
  procedure SPDataClick(Sender: TObject);
  procedure MCDataClick(Sender: TObject);
  procedure MCDataDblClick(Sender: TObject);
  procedure SBPesquisaRGClick(Sender: TObject);
  procedure SBPesquisaCodDisciplinaClick(Sender: TObject);
  procedure SBPesquisaCodRequisitanteClick(Sender: TObject);
  procedure RBAulaClick(Sender: TObject);
  procedure RBCirurgiaClick(Sender: TObject);
  procedure BTNIniciarItensClick(Sender: TObject);
  procedure BTNInserirItensClick(Sender: TObject);
  procedure BTNCancelarItensClick(Sender: TObject);
  procedure BTNRemoverItensClick(Sender: TObject);
  procedure EIFCodProdutoClick(Sender: TObject);
  procedure EIFCodProdutoKeyUp(Sender: TObject; var Key: Word;
    Shift: TShiftState);
  procedure EIFQtdSaidaKeyUp(Sender: TObject; var Key: Word;
    Shift: TShiftState);
  procedure setDBG;
  procedure verificaEstoque;
  procedure SBPesquisaCodProdutoClick(Sender: TObject);
private
  { Private declarations }
public
```

```
    { Public declarations }
  end;

var
  F_Cad_Saida_Farmacia: TF_Cad_Saida_Farmacia;
  UltimoRegistroInserido, TipoSaida: string;

implementation

uses U_DataModule, U_Pesquisa, Math;

{$R *.dfm}

procedure TF_Cad_Saida_Farmacia.FormShow(Sender: TObject);
begin
  DM.IBQFarmacia.Active:=TRUE;
  DM.IBQItensFarmacia.Active:=TRUE;
  DM.IBQPacientes.Active:=TRUE;
  DM.IBQRequisitantes.Active:=TRUE;
  DM.IBQDisciplinas.Active:=TRUE;

  MCData.Date:=Date;
end;

procedure TF_Cad_Saida_Farmacia.FormClose(Sender: TObject; var Action:
TCloseAction);
begin
  DM.IBQFarmacia.First;
  DM.IBQItensFarmacia.First;
  DM.IBQPacientes.Active:=FALSE;
  DM.IBQPacientes.Close;
  DM.IBQRequisitantes.Active:=FALSE;
  DM.IBQRequisitantes.Close;
  DM.IBQDisciplinas.Active:=FALSE;
```

```
  DM.IBQDisciplinas.Close;
end;

procedure TF_Cad_Saida_Farmacia.BTNIniciarNotaClick(Sender: TObject);
begin
  DM.IBQFarmacia.Insert;

  DBEFValorTotal.Text:='0';
  DBEFRG.Text:=DBERG.Text;
  RBAula.Checked:=TRUE;
  RBAula.Enabled:=FALSE;
  DBEFCodDisciplina.Text:=DBECodDisciplina.Text;
  DBEFCodRequisitante.Text:=DBECodRequisitante.Text;
  DBEFDia.Text:=DateToStr(MCData.Date);

  GBInfoSaida.Enabled:=TRUE;
  BTNIniciarNota.Enabled:=FALSE;
  BTNInserirNota.Enabled:=TRUE;
  BTNCancelarNota.Enabled:=TRUE;

  if BTNCommit.Enabled  and BTNRollback.Enabled then
  begin
    BTNCommit.Enabled:=FALSE;
    BTNRollback.Enabled:=FALSE;
  end;

  BTNIniciarNota.Default:=FALSE;
  BTNInserirNota.Default:=TRUE;

end;

procedure TF_Cad_Saida_Farmacia.BTNInserirNotaClick(Sender: TObject);
begin
  DM.IBQFarmacia.Post;
```

```
  GBInfoSaida.Enabled:=FALSE;

  UltimoRegistroInserido:='saida';

  BTNInserirNota.Enabled:=FALSE;
  BTNInserirNota.Default:=FALSE;
  BTNCancelarNota.Enabled:=FALSE;

  BTNCommit.Enabled:=TRUE;
  BTNRollback.Enabled:=TRUE;
end;

procedure TF_Cad_Saida_Farmacia.BTNCancelarNotaClick(Sender: TObject);
begin
  DM.IBQFarmacia.Cancel;
  GBInfoSaida.Enabled:=FALSE;

  BTNIniciarNota.Enabled:=TRUE;
  BTNInserirNota.Enabled:=FALSE;
  BTNCancelarNota.Enabled:=FALSE;
  BTNIniciarNota.Default:=TRUE;
  BTNInserirNota.Default:=FALSE;
end;

procedure TF_Cad_Saida_Farmacia.BTNCommitClick(Sender: TObject);
begin
  if MessageDlg('Ao confirmar você estará adicionando todas as informações
fornecidas até esse momento na tabela, deseja continuar?', mtConfirmation, [mbYes,
mbNo],0) = mrYes then
  begin
    if UltimoRegistroInserido = 'saida' then
    begin
      BTNIniciarNota.Enabled:=FALSE;
```

```
    BTNIniciarNota.Default:=FALSE;

    BTNIniciarItens.Default:=TRUE;
    BTNIniciarItens.Enabled:=TRUE;
   end
   else if UltimoRegistroInserido='item' then
   begin
    BTNIniciarNota.Enabled:=TRUE;
    BTNIniciarNota.Default:=TRUE;

    BTNIniciarItens.Default:=FALSE;
    BTNIniciarItens.Enabled:=FALSE;
    BTNRemoverItens.Enabled:=FALSE;
   end;

   BTNCommit.Enabled:=FALSE;
   BTNRollback.Enabled:=FALSE;

   DM.IBTrans.Commit;
   DM.IBTrans.StartTransaction;

   DM.IBQFarmacia.Open;
   DM.IBQFarmacia.Last;
   DM.IBQItensFarmacia.Open;
   DM.IBQPacientes.Open;
   DM.IBQDisciplinas.Open;
   DM.IBQRequisitantes.Open;
   DM.IBQProdutos.Open;
   setDBG;
  end;
end;

procedure TF_Cad_Saida_Farmacia.BTNRollbackClick(Sender: TObject);
begin
```

```
  if MessageDlg('Ao voltar, todas informações fornecidas desde a última confirmação
serão perdidas, deseja voltar?', mtConfirmation, [mbYes, mbNo],0) = mrYes then
  begin
    if UltimoRegistroInserido = 'saida' then
    begin
      BTNIniciarNota.Default:=TRUE;
      BTNIniciarNota.Enabled:=TRUE;
      BTNInserirNota.Default:=FALSE;
    end
    else if UltimoRegistroInserido = 'item'then
    begin
      BTNIniciarItens.Default:=TRUE;
      BTNInserirItens.Default:=FALSE;
      BTNRemoverItens.Enabled:=FALSE;
    end;

    BTNCommit.Enabled:=FALSE;
    BTNRollback.Enabled:=FALSE;

    DM.IBQFarmacia.Close;
    DM.IBTrans.Rollback;

    DM.IBQFarmacia.Open;
    DM.IBQFarmacia.Last;
    DM.IBQItensFarmacia.Open;
    DM.IBQItensFarmacia.Last;
    DM.IBQDisciplinas.Open;
    DM.IBQRequisitantes.Open;
    DM.IBQPacientes.Open;
    LValorTotal.Caption:='0,00';
  end;
end;

procedure TF_Cad_Saida_Farmacia.BTNSairClick(Sender: TObject);
```

```
begin
  Self.Close;
end;

procedure TF_Cad_Saida_Farmacia.SPDataClick(Sender: TObject);
begin
  if MCData.Visible = FALSE then
    MCData.Show
  else
    MCData.Hide;
end;

procedure TF_Cad_Saida_Farmacia.MCDataClick(Sender: TObject);
begin
  DBEFDia.Text:=DateToStr(MCData.Date);
end;

procedure TF_Cad_Saida_Farmacia.MCDataDblClick(Sender: TObject);
begin
  MCData.Hide;
end;

procedure TF_Cad_Saida_Farmacia.SBPesquisaRGClick(Sender: TObject);
begin
  F_Pesquisa.setOP(1);
  F_Pesquisa.CBPaciente.Checked:=TRUE;
  F_Pesquisa.DBGPesquisa.DataSource:=DM.DSPacientes;
  F_Pesquisa.Show;
end;

procedure TF_Cad_Saida_Farmacia.SBPesquisaCodDisciplinaClick(Sender:
TObject);
begin
  F_Pesquisa.setOP(1);
```

```
  F_Pesquisa.CBDisciplina.Checked:=TRUE;
  F_Pesquisa.DBGPesquisa.DataSource:=DM.DSDisciplinas;
  F_Pesquisa.Show;
end;

procedure TF_Cad_Saida_Farmacia.SBPesquisaCodRequisitanteClick(Sender:
TObject);
begin
  F_Pesquisa.setOP(1);
  F_Pesquisa.CBRequisitante.Checked:=TRUE;
  F_Pesquisa.DBGPesquisa.DataSource:=DM.DSRequisitantes;
  F_Pesquisa.Show;
end;

procedure TF_Cad_Saida_Farmacia.RBAulaClick(Sender: TObject);
begin
  TipoSaida:=RBAula.Caption;
  RBAula.Enabled:=FALSE;
  RBCirurgia.Enabled:=TRUE;
end;

procedure TF_Cad_Saida_Farmacia.RBCirurgiaClick(Sender: TObject);
begin
  TipoSaida:=RBCirurgia.Caption;
  RBCirurgia.Enabled:=FALSE;
  RBAula.Enabled:=TRUE;
end;

procedure TF_Cad_Saida_Farmacia.BTNIniciarItensClick(Sender: TObject);
begin
  DM.IBQItensFarmacia.Insert;
  DM.IBQFarmacia.Edit;
  GBItensSaida.Enabled:=TRUE;
```

```
  BTNIniciarItens.Enabled:=FALSE;
  BTNIniciarItens.Default:=FALSE;
  BTNInserirItens.Enabled:=TRUE;
  BTNCancelarItens.Enabled:=TRUE;

  if BTNCommit.Enabled and BTNRollback.Enabled then
  begin
    BTNCommit.Enabled:=FALSE;
    BTNRollback.Enabled:=FALSE;
  end;


  EIFCodProduto.SetFocus;

end;

procedure TF_Cad_Saida_Farmacia.BTNInserirItensClick(Sender: TObject);
begin
  DM.IBQItensFarmaciaTIPO.AsString:='Saída';
  DM.IBQItensFarmaciaCOD_SAIDA.AsString:=DBTFCodSaida.Caption;
  DM.IBQItensFarmaciaCOD_PRODUTO.AsString:=EIFCodProduto.Text;
  DM.IBQItensFarmaciaQUANTIDADE.AsString:=EIFQtdSaida.Text;

LValorTotal.Caption:=FloatToStr(StrToFloat(LValorTotal.Caption)+(StrToFloat(DM.IB
QProdutosPRECO_VENDA.AsString)*StrToFloat(EIFQtdSaida.Text)));
  LValorTotal.Caption:=FloatToStr(RoundTo(StrToFloat(LValorTotal.Caption),-2));
  DM.IBQFarmaciaVALOR_TOTAL_SAIDA.AsString:=LValorTotal.Caption;

  DM.IBQItensFarmacia.Post;
  DM.IBQFarmacia.Post;

  GBItensSaida.Enabled:=FALSE;

  UltimoRegistroInserido:='item';
```

```
  BTNIniciarItens.Enabled:=TRUE;
  BTNIniciarItens.Default:=TRUE;
  BTNInserirItens.Enabled:=FALSE;
  BTNCancelarItens.Enabled:=FALSE;

  BTNCommit.Enabled:=TRUE;
  BTNRollback.Enabled:=TRUE;

  verificaEstoque;

  DM.IBQProdutos.Edit;

DM.IBQProdutosQUANTIDADE_TOTAL.AsString:=IntToStr((StrToInt(DM.IBQProdut
osQUANTIDADE_TOTAL.AsString))-(StrToInt(EIFQtdSaida.Text)));
  DM.IBQProdutos.Post;
end;

procedure TF_Cad_Saida_Farmacia.BTNCancelarItensClick(Sender: TObject);
begin
  DM.IBQItensFarmacia.Cancel;
  GBItensSaida.Enabled:=FALSE;

  BTNIniciarItens.Enabled:=TRUE;
  BTNInserirItens.Enabled:=FALSE;
  BTNCancelarItens.Enabled:=FALSE;
  BTNIniciarItens.Default:=TRUE;
  BTNInserirItens.Default:=FALSE;

  if not BTNCommit.Enabled and not BTNRollback.Enabled then
  begin
    BTNCommit.Enabled:=TRUE;
    BTNRollback.Enabled:=TRUE;
  end;
```

```
end;

procedure TF_Cad_Saida_Farmacia.BTNRemoverItensClick(Sender: TObject);
begin
  DM.IBQItensFarmacia.Open;
  try
    DM.IBQItensFarmacia.Delete;
  finally
  if not BTNCommit.Enabled and not BTNRollback.Enabled then
  begin
    BTNCommit.Enabled:=TRUE;
    BTNRollback.Enabled:=TRUE;
  end;
    with DBGItensFarmacia.DataSource.DataSet,DBGItensFarmacia do
    begin
      if(Trim(FieldByName('COD_SAIDA').Text) = '') then
        BTNRemoverItens.Enabled:=FALSE;
    end;
  end;
end;

procedure TF_Cad_Saida_Farmacia.EIFCodProdutoClick(Sender: TObject);
begin
  DM.IBQProdutos.Active:=FALSE;
  EIFCodProduto.Clear;
end;

procedure TF_Cad_Saida_Farmacia.EIFCodProdutoKeyUp(Sender: TObject;
  var Key: Word; Shift: TShiftState);
begin
  if EIFCodProduto.Text<>'' then
  begin
    DM.IBQProdutos.Close;
```

```
    DM.IBQProdutos.SQL.Text:='SELECT * FROM produtos WHERE cod_produto
LIKE :cp';
    DM.IBQProdutos.ParamByName('cp').AsString:=EIFCodProduto.Text;
    DM.IBQProdutos.Open;
    DM.IBQProdutos.Active:=TRUE;
  end
  else
  begin
    DM.IBQProdutos.Close;
    DM.IBQProdutos.SQL.Text:='SELECT * FROM produtos';
    DM.IBQProdutos.Open;
  end;
  DM.IBQProdutos.Edit;
end;

procedure TF_Cad_Saida_Farmacia.EIFQtdSaidaKeyUp(Sender: TObject;
  var Key: Word; Shift: TShiftState);
begin
  DM.PontoVirgula(EIFQtdSaida.Text);
  if (trim(EIFQtdSaida.Text)) <> '' then
  begin
    if (StrToFloat(EIFQtdSaida.Text) >
StrToFloat(DM.IBQProdutosQUANTIDADE_TOTAL.AsString)) then
      EIFQtdSaida.Text:=DM.IBQProdutosQUANTIDADE_TOTAL.AsString;
  end;
end;

procedure TF_Cad_Saida_Farmacia.setDBG;
begin
  DM.IBQItensFarmacia.Close;
  DM.IBQItensFarmacia.SQL.Text:='SELECT * FROM itens_saidas WHERE
cod_saida LIKE :cs';
  DM.IBQItensFarmacia.ParamByName('cs').AsString:=DBTFCodSaida.Caption;
  DM.IBQItensFarmacia.Open;
```

```
end;

procedure TF_Cad_Saida_Farmacia.verificaEstoque;
begin
  if StrToInt(EIFQtdSaida.Text) =
StrToInt(DM.IBQProdutosQUANTIDADE_TOTAL.AsString) then
     showmessage('produto "'+DM.IBQProdutosNOME.AsString+'" em falta,
reabastecer o estoque')
  else
    if ((StrToInt(DM.IBQProdutosQUANTIDADE_TOTAL.AsString))-
(StrToInt(EIFQtdSaida.Text))) <= 10 then
      showmessage('produto "'+DM.IBQProdutosNOME.AsString+'" com pouca
quantidade, precisará reabastecer em breve');
end;

procedure TF_Cad_Saida_Farmacia.SBPesquisaCodProdutoClick(Sender: TObject);
begin
  F_Pesquisa.setOP(4);
  F_Pesquisa.CBProduto.Checked:=TRUE;
  F_Pesquisa.DBGPesquisa.DataSource:=DM.DSProdutos;
  F_Pesquisa.Show;
  DM.IBQItensFarmacia.Cancel;
end;

end.
```