Faculdades Integradas de Taquara - Faccat
Av. Oscar Martins Rangel, 4.500
Taquara, RS, CEP 95600-000

Curso de Sistemas de Informação

SISTEMA PARA IDENTIFICAÇÃO DE DEFEITOS E PROCESSAMENTO DE DEVOLUÇÃO DE CALÇADOS

Deimon Jonatan Willers
Faculdades Integradas de Taquara – Faccat – Taquara – RS – Brasil
to.dwillers@gmail.com

Marcelo Cunha de Azambuja
Professor Orientador
Faculdades Integradas de Taquara – Faccat – Taquara – RS – Brasil
azambuja@faccat.br

Resumo

Este artigo tem por finalidade apresentar a conceitualização, análise e a metodologia de desenvolvimento de um sistema para identificação de defeitos e processamento de devoluções de calçados. A ferramenta em questão visa minimizar o prejuízo causado pelas devoluções, prejuízos estes enfrentados tanto por fabricantes quanto por franquias. O sistema proposto auxilia no mapeamento de possíveis problemas com fornecedores, setores da empresa ou mesmo em componentes que compõem o calçado. Também visa diminuir o fluxo de produtos para avaliação mediante o preenchimento de um formulário padrão de devolução, juntamente com o envio de imagens, evitando quando possível o incômodo e os custos de transporte.

Palavras-chave: calçados, devolução, defeitos.

SYSTEM FOR IDENTIFICATION OF DEFECTS AND PROCESSING OF RETURN OF SHOES

Abstract

This article aims to present the conceptualization, analysis and the developing methodology of a system to identify defects and processing of return of shoes. The tool in question is intended to minimize the damage caused by the returns faced by both manufacturers and franchises. The proposed system helps in mapping possible problems with suppliers, parts of the company or even components that make up the shoes. It also aims to reduce the flow of products for evaluation by completing a standard devolution formulary, along with sending images of the shoes, avoiding where possible the hassle and cost of shipping.

Key-words: *shoes, returns, defects.*

1. Introdução

Um dos principais problemas enfrentados pela indústria calçadista, tanto para fabricantes quanto para franquias, é o grande volume de devoluções de seus produtos, seja por defeitos estruturais ou nos componentes e matérias-primas que compõem o calçado ou então por descontentamento dos seus consumidores.

O retorno de produtos às fábricas é um desafio enfrentado por todo o segmento. Segundo Rosinha (2008) em uma pesquisa realizada pela ABLAC (Associação Brasileira de Lojistas de Artefatos e Calçados) em 2006, 8 milhões de pares de calçados foram devolvidos, gerando prejuízos da ordem de R$ 360 milhões. Atualmente o problema se tornou ainda maior em função da extrema importância que os fabricantes estão dando às opiniões e avaliações dos seus consumidores, como forma para se destacarem em um mercado cada vez mais competitivo. A partir do momento que o fabricante puder identificar esses problemas, ele poderá então agir objetivamente, refinando o processo de produção e por consequência aumentando a qualidade de seus produtos e diminuindo o número de devoluções.

Outro ponto relevante que deve ser levado em consideração é o fluxo de produtos. O esforço e o custo de transporte exigido para devolver um item defeituoso ao fabricante ou fornecedor muitas vezes superam o preço do próprio produto.

A ideia para o desenvolvimento de um sistema gratuito que facilite as devoluções e auxilie a identificação de defeitos surgiu de uma conversa informal entre o autor e um ex-funcionário de uma empresa de grande porte situada na região do Vale do Paranhana. Ele argumentava que seria de grande ajuda o uso de tal ferramenta na referida empresa, já que todo esforço iniciado na identificação de defeitos apresentados nas devoluções era realizado utilizando planilhas eletrônicas e coleta de dados manual.

Outra solução discutida seria a implementação de um sistema como este aqui apresentado em conjunto com uma ferramenta ERP (*Enterprise Resource Planning* – Sistema de Gestão Empresarial) já existente na empresa em questão. Mas esta não seria a solução ideal, pois acabaria gerando um grande custo de desenvolvimento para a empresa onde trabalhava. Observando esses fatores e também como o problema gera muitos prejuízos às empresas calçadistas em geral, surgiu então a proposta de uma ferramenta genérica, capaz não só de analisar os possíveis fatores envolvidos no processo, mas também de facilitar o processo de devolução.

O sistema desenvolvido e apresentado neste artigo, por definição, possui como escopo principal o mapeamento de possíveis problemas com fornecedores ou em áreas específicas da empresa, como por exemplo, os setores de montagem, corte ou costura, e também em componentes específicos do calçado como a sola, forro ou cabedal.

A identificação de problemas foi disponibilizada na forma de indicadores, entre eles gráficos e relatórios, que segundo Gandin (2002) são sinais para saber se algo que não se pode ver diretamente está presente em uma realidade e que quando estabelecemos uma lista de indicadores para algo determinado, aumenta-se a clareza sobre este algo. A redução do transporte de mercadorias se faz presente no momento que o tráfego físico do produto defeituoso é substituído por imagens do item em questão. O preenchimento do formulário padrão de devolução disponibilizado ao usuário também pode facilitar e reduzir significantemente o tempo total para análise dos defeitos por parte do fabricante, bem como a consequente tomada de decisão quanto à validade da devolução e a autorização da troca da mercadoria por uma nova.

Para o desenvolvimento do sistema, várias tecnologias e ferramentas adicionais foram necessárias, dentre as principais o *framework* e a família de tecnologias .NET da Microsoft. Elas estão descritas na seção 2, ordenadas por afinidade ou por serem tecnologias relacionadas. A seção 3 descreve a metodologia utilizada, tanto na análise do projeto quanto na forma de desenvolvimento. A seção 4 demonstra os resultados obtidos e as principais características do sistema. E a seção 5 apresenta as conclusões do trabalho.

2. Referencial teórico

2.1 Framework .Net

Segundo Minneto (2007), um *framewok* é descrito como uma "base" para a criação de um sistema maior e é também uma coleção de códigos-fonte, classes e funções que facilitam o desenvolvimento. O *framework* escolhido para o desenvolvimento do sistema foi o. NET *Framework* 4 criado pela Microsoft. Ele é descrito como um ambiente gerenciado[1] de execução que provê uma série de serviços para suas aplicações e é constituído de quatro grandes componentes: a CLR (*Common Language Runtime*) que funciona de forma muito semelhante à máquina virtual do Java e é responsável por gerenciar e executar o código gerado a partir da IL (*Intermediate Language*) (MSDN, 2012a). Todo código gerado pelas linguagens .NET não é diretamente transformado em código de máquina para uma determinada arquitetura, mas sim transformado em um *bytecode* (IL) que posteriormente é executado pelo compilador JIT (*Just-in-Time*) e transformado em código nativo, isso faz com que todos os aplicativos gerados por diferentes linguagens sejam compatíveis e executados de maneira uniforme pelo *runtime*. A CLR também é responsável por uma série de detalhes de baixo nível como o gerenciamento de memória, execução de aplicações, controle de *threads*, exceções e verificações de segurança.

[1]Termo atribuido a qualquer código executado pela máquina virtual.

A CTS (*Common Type System*) que descreve todos os tipos de dados conhecidos (classes e estruturas) suportados pelo *framework* e também como são representados na memória do computador, o que garante compatibilidade entre todas as linguagens suportadas bem como a interoperabilidade de tipos (MSDN, 2012a).

A CLS (*Framework Class Library*) define um subconjunto de tipos conhecidos que todas as linguagens suportadas pelo *runtime* .NET devem implementar. Uma linguagem é dada como compatível (*CLS-Compliant*) somente quando respeita as restrições da especificação. Uma vez que a linguagem é compatível com a CLS, todos os seus tipos podem ser consumidos por todas as outras linguagens .NET (TROELSEN, 2010).

Adicionalmente, a BCL (*Base Class Library*) é outro componente importante que oferece (encapsula) uma série de serviços essenciais e também a definição de tipos primitivos como *strings* e inteiros, coleções genéricas, *threads*, *hashes* para criptografia e reflexão. Ela também disponibiliza as operações de escrita e leitura de documentos, incluindo arquivos XML, oferece bibliotecas de código para a comunicação com dispositivos, renderização de gráficos, facilita o acesso a banco de dados, entre outros serviços importantes (TROELSEN, 2010).

A Figura 1 exibe a organização dos componentes e serviços do *framework* .NET:

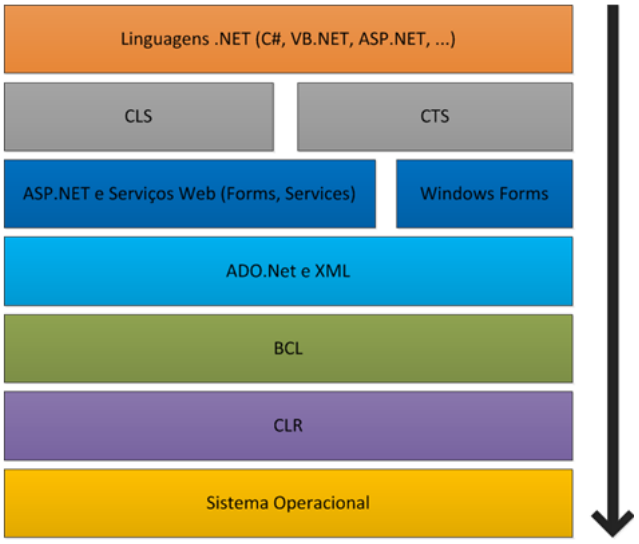

FIGURA 1 – Arquitetura .NET
Fonte: Adaptado de Technet (2006)

2.2 Visual Studio 2010

A IDE (*Integrated Development Environment* - Ambiente Integrado de Desenvolvimento) utilizada para toda a programação do *software* foi o Visual Studio 2010. Além de ser o ambiente de desenvolvimento padrão para todas as linguagens da família .NET, também tem suporte a vários recursos avançados, como um depurador de código (*debugger*) compatível com várias linguagens, incluindo C#, Silverlight e ASP. Também é possível realizar a modelagem UML integrada no

editor, dispensando o uso ou compra de programas adicionais, e também possibilita a execução de rotinas de testes de *software* automatizadas.

2.3 Linguagem C#

O C# (C Sharp) é uma linguagem de programação de uso geral, criada em 2000 especialmente para a plataforma .NET. É baseada na linguagem C, fortemente tipada, orientada a objetos e também orientada a eventos, já que responde a ações iniciadas pelo usuário, como cliques do *mouse* e pressionamento de teclas. O C# também possui um coletor de lixo (*Garbage Collector*), eficaz em desalocar objetos não utilizados.

O principal objetivo da linguagem é ser produtiva e balancear simplicidade, expressividade e performance (ALBAHARI e ALBAHARI, 2012).

2.4 Silverlight

Um dos requisitos do *software* aqui proposto é ser desenvolvido para a plataforma *web*, e para tal objetivo o Silverlight foi utilizado. Ele é descrito como um *plug-in* gratuito para a plataforma .NET e é compatível com múltiplos navegadores, dispositivos e sistemas operacionais (SILVERLIGHT, 2012).

Sendo uma plataforma de desenvolvimento (*framework*), o Silverlight permite a criação de aplicações de *internet* ricas (RIA – *Rich Internet Application*), ou seja, aplicações com características e funcionalidades de sistemas *desktop*, como a riqueza visual e a responsividade a comandos geralmente não obtidos nas aplicações tradicionais em HTML. Por ser diferente da arquitetura padrão cliente-servidor, onde o cliente é geralmente uma página leve e todo o processamento é feito no servidor, o cliente Silverlight é completo no quesito de dependências para seu funcionamento e é naturalmente assíncrono, já que permite efetuar operações sem ter que aguardar uma resposta do servidor.

2.4.1 Formato XAP

Segundo Brown (2012), uma aplicação Silverlight consiste de um arquivo XAP com o código compilado, informações de inicialização, recursos adicionais (fontes, imagens, etc.) e um hospedeiro para o *plug-in*. O hospedeiro para o *plug-in* pode ser entendido como qualquer arquivo que pode ser renderizado em um navegador, como uma página HTML ou ASP.

O XAP gerado nada mais é que um arquivo comprimido no formato ZIP onde as DLLs compiladas, recursos e arquivos adicionais, configurações de serviços w*eb* e um manifesto descrevendo o ponto de entrada do aplicativo e os itens requeridos para o funcionamento são

reunidos. O navegador então só necessita acessar a página onde o aplicativo está hospedado, fazer *download*, inicializar o *plug-in* e entregar o XAP para ser processado e exibido.

2.4.2 CoreCLR

O Silverlight utiliza uma versão simplificada da CLR (*Common Language Runtime*) chamada CoreCLR. É mais enxuta e otimizada que a CLR completa e projetada especificamente para ser utilizada no lado cliente. Ela compartilha algumas partes de código com a implementação completa, como o coletor de lixo, o compilador JIT, gerenciamento de exceções e também a checagem de tipos, o que permite que o *plug-in* completo tenha em torno de 8MB no total (BROW, 2012).

2.4.3 XAML

Diferentemente de outras tecnologias *web*, como o ASP e o PHP, o Silverlight não utiliza a linguagem de marcação HTML e nem folhas de estilo CSS para compor a *interface* visual do usuário. Todos os objetos que compõem a *interface* gráfica são criados declarativamente em um arquivo XAML (*eXtensible Application Markup Language*), o que permite a separação da camada de apresentação da camada de código.

Ghoda e Dalal (2011) definem a XAML como uma linguagem declarativa, baseada em XML que permite uma *interface* de usuário externalizada e desacoplada. Por ser baseada em XML, possibilita a criação e inicialização de objetos .NET por *tags* e elementos, assim qualquer objeto possível de ser criado por código pode ser também instanciado desta forma, como mostra a Figura 2:

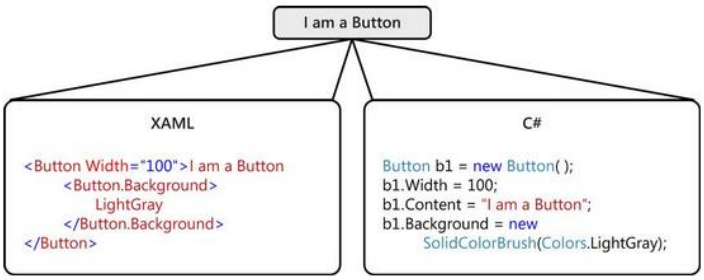

FIGURA 2 – Declaração de objeto em XAML
Fonte: Ghoda e Dalal (2011)

2.5 Banco de Dados

O sistema gerenciador de banco de dados utilizado para a persistência dos dados foi o MySQL Community Edition. O MySQL foi escolhido em lugar do Microsoft SQL Server, sistema gerenciador de banco de dados que acompanha o Visual Studio e é da mesma família de produtos,

pois o software desenvolvido neste trabalho não demanda um banco de dados tão robusto e que exige *hardware* superior, além de ter sido utilizado com sucesso em diversos projetos e por diversas empresas no mundo (MYSQL, 2012a).

O diagrama ER (Entidade Relacionamento) e toda a estrutura do banco de dados, bem como a inserção dos dados iniciais foram feitas com a ferramenta MySQL Workbench (MYSQL, 2012b). O Workbench é um aplicativo visual que provê modelagem de dados, ferramentas de administração, editor SQL, realce de sintaxe, engenharia reversa (diagrama-banco e banco-diagrama), *backup* e restauração de bancos de dados, migração de dados e objetos, suporte a múltiplas configurações, além de ser gratuito e compatível com Windows, Linux e Mac OS (MYSQL, 2012b).

2.6 Entity Framework

O *framework* ORM *(Object-Relational Mapping* - Mapeamento Objeto-Relacional) que foi utilizado para transformar (mapear) as tabelas do banco MySQL em classes para serem utilizadas no *software* foi o Entity Framework (EF) que acompanha o .NET desde a versão 3.5.

Klein (2010) define o Entity Framework como sendo um conjunto de tecnologias que ajuda a preencher o espaço entre o desenvolvimento orientado a objetos e o banco de dados. Este espaço é conhecido como *impedance mismatch*, que pode ser entendido como "problema de adaptação", pois o mapeamento e organização de classes não se relacionam com a organização do modelo relacional. Lerman (2009) complementa a descrição dizendo que um dos principais benefícios do EF é que o desenvolvedor não deve ficar preocupado com estrutura do banco de dados já que todo o acesso a dados é feito a partir de um modelo conceitual que o representa, e destaca também algumas das principais características do *framework*: (i) gera e atualiza automaticamente as classes a partir do modelo, (ii) gerencia toda conectividade com o banco de dados, (iii) disponibiliza uma linguagem de consulta própria que funciona sobre o modelo e que posteriormente é traduzida em consultas SQL compatíveis com o banco de dados alvo, (iv) oferece um mecanismo de acompanhamento de modificações, ou seja, conforme os dados vão sendo modificados pela aplicação o próprio mecanismo se encarrega de enviá-las ao banco de dados.

2.7 Web Services

Diferentemente de aplicações que acessam bancos de dados nativamente, o Silverlight requer que toda a comunicação com o servidor seja feita utilizando um serviço *web* (*web service)*, já que é executado diretamente no navegador do usuário. Um serviço *web* pode ser entendido como uma aplicação ou componente que é executado no servidor e não localmente, ele recebe requisições do cliente (Silverlight), processa e envia o resultado (SHARP, 2010).

Todo serviço publicado define suas operações, tipos de retorno e tipos de dados disponíveis de alguma forma, ou seja, o serviço precisa descrever a si mesmo para que possa ser entendido pelo cliente. Essa definição é feita em um documento WSDL (*Web Services Description Language – Linguagem de Descrição de Serviços Web*), que é escrito em linguagem XML e define um padrão universal (abstrato) para descrever o serviço e suas operações (W3C, 2012). A conversação entre cliente-servidor é feita via envelopes (mensagens) também baseados em XML e trafegados pela rede, normalmente por HTTP/HTTPS, utilizando o protocolo SOAP (*Simple Object Access Protocol*) (LIU, 2010).

2.8 RIA Services

Para simplificar a criação dos serviços *web* necessários para o *software* e também para reduzir o esforço manual despendido na criação dos documentos WSDL, o RIA Services (ou WCF RIA Services) foi utilizado. Ele foi criado sobre outra tecnologia da Microsoft chamada de WCF (*Windows Communication Fundation*) que oferece um modelo de programação que ajuda a construir uma aplicação orientada a serviços e unifica de uma forma composta e extensível as capacidades de sistemas distribuídos (LIU, 2010) e também separa as camadas de segurança e transporte da mensagem (LIKNESS, 2012).

O RIA Services pode ser entendido como um conjunto de *framework* e ferramentas auxiliares, que servem para simplificar a construção de aplicações multicamadas e que provê as seguintes funções: (i) criação automática dos métodos de inserção, leitura, atualização e remoção das entidades (CRUD – *Create, Read, Update e Delete*), (ii) geração e sincronização automática dos objetos *proxy*, que são réplicas das estruturas, métodos e objetos originais, mas sem a implementação dos mesmos no lado cliente, (iii) compartilha as regras de validação, (iv) oferece controles que facilitam a manipulação dos dados pelo cliente, (v) tem integração com o modelo de segurança do ASP (BROWN, 2010). A Figura 4 demonstra a separação entre cliente e servidor e o ponto onde o RIA Services se encaixa:

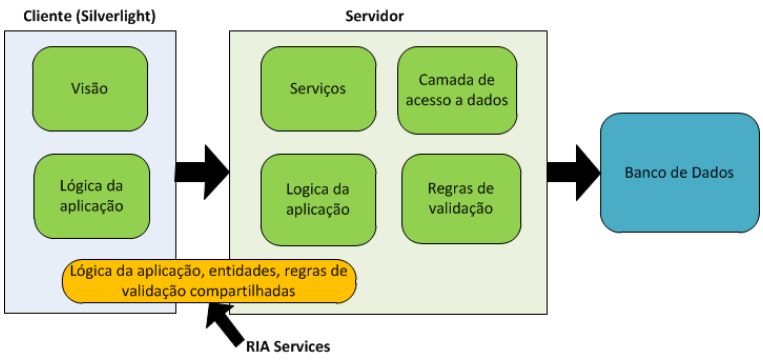

FIGURA 4–Arquitetura do RIA Services
Fonte: Adaptado de Brown (2010)

2.9 Controle de Versão

O sistema de controle de versão (VCS – *Version Control System*) utilizado para gerenciar as modificações nos arquivos, documentos e diretórios do *software* foi o Subversion (APACHE, 2012). Mantido pela fundação Apache, o SVN é um *software* livre e gratuito que permite recuperar, comparar e examinar modificações ao longo do tempo por meio de revisões mantidas no servidor onde foi instalado.

Por ser genérico e não fazer distinção de tipos de arquivos ou tecnologias específicas, o SVN pode ser utilizado em qualquer projeto, incluindo os que utilizam tecnologias proprietárias.

3. Metodologia

Como metodologias de desenvolvimento, foram adotadas partes de dois modelos: o Processo Unificado e o modelo RAD (*Rapid Application Development* – Desenvolvimento Rápido de Aplicação). O Processo Unificado é descrito como uma série de atividades executadas para transformar um conjunto de requisitos do cliente em um *software* (SCOTT, 2003).

O Processo Unificado combina os ciclos de desenvolvimento iterativo e incremental, que segundo Larman (2007) envolvem a imediata programação e testes do sistema, mesmo que parcialmente completo, em ciclos repetidos. Também se considera que normalmente o desenvolvimento começa antes que os requisitos tenham sido definidos em detalhes.

Ainda segundo Larman (2007), o ciclo de desenvolvimento é dividido em uma série de projetos curtos e de duração fixa chamados de iterações. Cada iteração tem suas atividades próprias de análise, projeto, implementação e teste, e como resultado de cada uma dessas iterações é gerado um sistema parcial que pode ser executado e testado.

O modelo RAD, segundo Pressman (2006), é um modelo de *software* incremental, podendo ser considerado uma adaptação do modelo em cascata. Tem ciclos extremamente curtos, com períodos entre 60 a 90 dias e abrange as seguintes etapas: (i) comunicação: entende-se como o momento em que ocorre a comunicação com o cliente e o levantamento de requisitos do *software*, (ii) planejamento: é onde as tarefas são detalhadas e o cronograma do projeto é definido, (iii) modelagem: etapa que abrange a modelagem de dados e processos, (iv) construção: é a etapa onde é o *software* é efetivamente codificado e testado, (v) implantação: etapa onde o *software* é entregue (implantado) e onde o cliente fornece um retorno.

3.1 Análise e Modelagem

Para a modelagem da arquitetura do software, foram utilizados os diagramas propostos pela UML (*Unified Modeling Language* – Linguagem de Modelagem Unificada). A UML pode ser

definida como uma linguagem para documentar projetos de *software*, que pode ser utilizada para visualizar, especificar, construir e documentar elementos de um sistema. E por ser uma linguagem, a UML provê todo um vocabulário e um conjunto de regras (gramática própria) para combinar os elementos conceituais e físicos que representam um sistema e também para auxiliar as pessoas a criar e ler seus documentos (MARTINS, 2007). A ferramenta utilizada para gerar os diagramas foi o Visual Studio 2010, descrita na seção 2.

Antes de iniciar a utilização da UML para gerar os diagramas foi necessário primeiro definir os requisitos funcionais e não-funcionais do sistema. É dito dos requisitos funcionais que eles devem descrever as funcionalidades que o *software* fornece quando estiver pronto e os requisitos não-funcionais são aqueles não associados diretamente às funções desempenhadas pelo produto, mas sim descrevem restrições ao *software* de uma forma geral (KOSCIANSKI e SOARES, 2007).

O Quadro 1 demonstra os requisitos funcionais levantados durante entrevista feita com dois proprietários de empresas especializadas em oferecer serviços e treinamentos focados no comércio varejista no estado de São Paulo, bem como os requisitos iniciais comentados na introdução deste artigo:

Requisitos funcionais:

- O sistema deve funcionar na *web*, em qualquer navegador atual.
- Deve permitir a criação de novos usuários.
- Deve permitir a localização, criação, modificação e exclusão dos seguintes itens:
    - Componentes
    - Defeitos e Defeitos por componentes
    - Fornecedores e Lojas
    - Países, Estados e Cidades
    - Produtos
    - Setores
- Deve permitir ao usuário informar uma nova devolução, contendo os seguintes itens:
    - O requisitante da devolução (próprio usuário logado)
    - O produto em questão (referência)
    - Os defeitos que ocorreram e uma descrição de cada um deles
    - As imagens do produto, sem limite no número de arquivos
- Ao usuário gestor deve permitir a visualização de todas as devoluções realizadas.
- Ao usuário operador deve permitir a visualização somente de suas próprias devoluções.
- Deve permitir a geração dos seguintes gráficos e relatórios, quando possível com seleção de período:
    - Defeitos por componentes, referência, fornecedor, setor e severidade
    - Devoluções por mês e por loja
    - Ranking de produtos mais devolvidos (com imagens)

QUADRO 1 – Requisitos funcionais
Fonte: Autoria própria (2012)

Já o Quadro 2 demonstra os requisitos não-funcionais e os requisitos extras da fase de projeto que não foram aproveitados:

Requisitos não-funcionais:

- Limitar o acesso somente a usuários autenticados.
- O sistema deve ter uma interface com o usuário simplificada, com o mínimo de controles e elementos gráficos que desfoque o usuário do que realmente importa (informação).
- O sistema deve ser de fácil utilização, ou seja, não deve exigir grandes conhecimentos em informática.
- O sistema deve ser ágil na resposta aos comandos (responsivo) e gerar os relatórios e gráficos rapidamente, preferencialmente em não mais que 1 minuto (limite máximo não definido).

Requisitos extras da fase de projeto e que não se mostraram necessários:

- Importação de dados existentes via XML.
    - **Justificativa**: complicado para o usuário final, trabalhoso e declarativo, talvez não exista demanda real (pelo menos inicialmente), banco de dados MySQL aberto que possibilita uma importação rápida pelo Workbench ou por outra ferramenta já existente, possibilidade de exportação de comandos SQL "INSERT" de um banco de dados existente e importação diretamente no MySQL.

QUADRO 2 – Requisitos não-funcionais
Fonte: Autoria própria (2012)

Após todos os requisitos serem definidos e detalhados, o diagrama de casos de uso foi desenvolvido, ver Figura 5, ela exibe uma visão generalizada (de alto nível) do sistema e serve como base para o desenvolvimento das principais funcionalidades. Posteriormente cada caso de uso foi descrito e documentado separadamente para facilitar o entendimento.

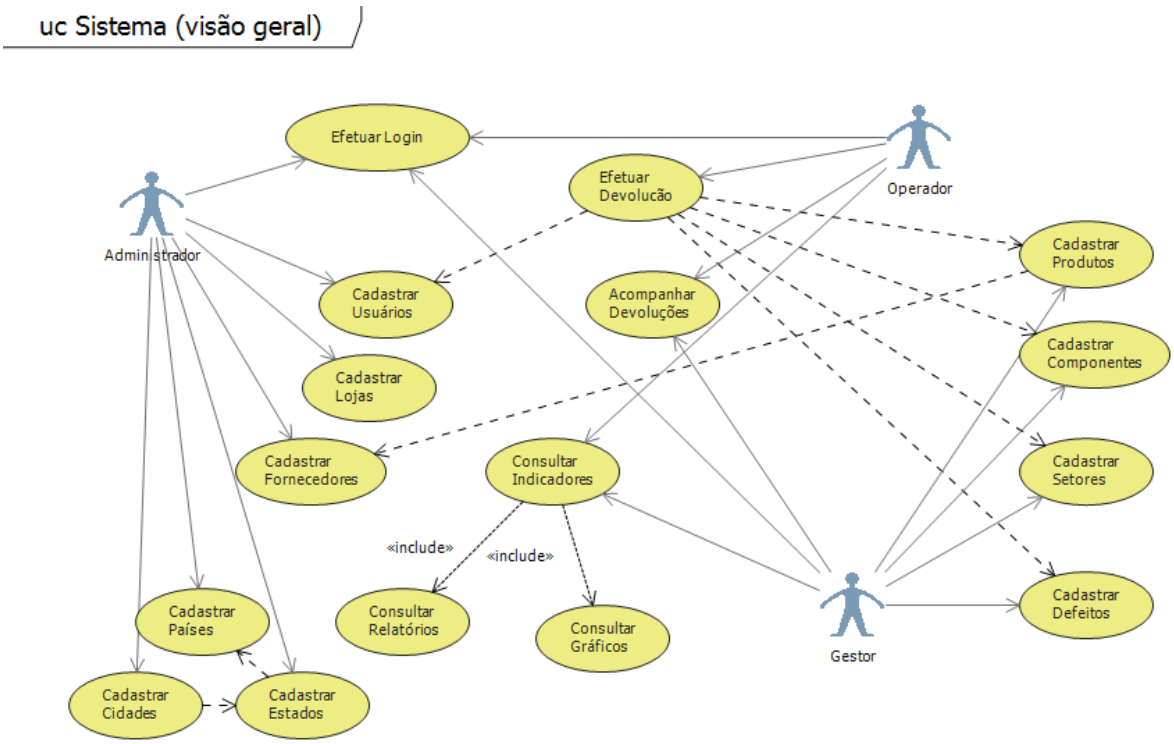

FIGURA 5 – Diagrama de Caso de Uso
Fonte: Autoria própria (2012)

3.1.1 Diagrama Entidade-Relacionamento

Após a descrição de todos os casos de uso e o levantamento de todos os dados necessários, foi criado o modelo ER da base de dados do *software*. Ele é importante, pois especifica as tabelas, campos de dados e relacionamentos entre as entidades. A Figura 6 demonstra o diagrama ER construído com a utilização da ferramenta MySQL Workbench, descrita na seção 2.5:

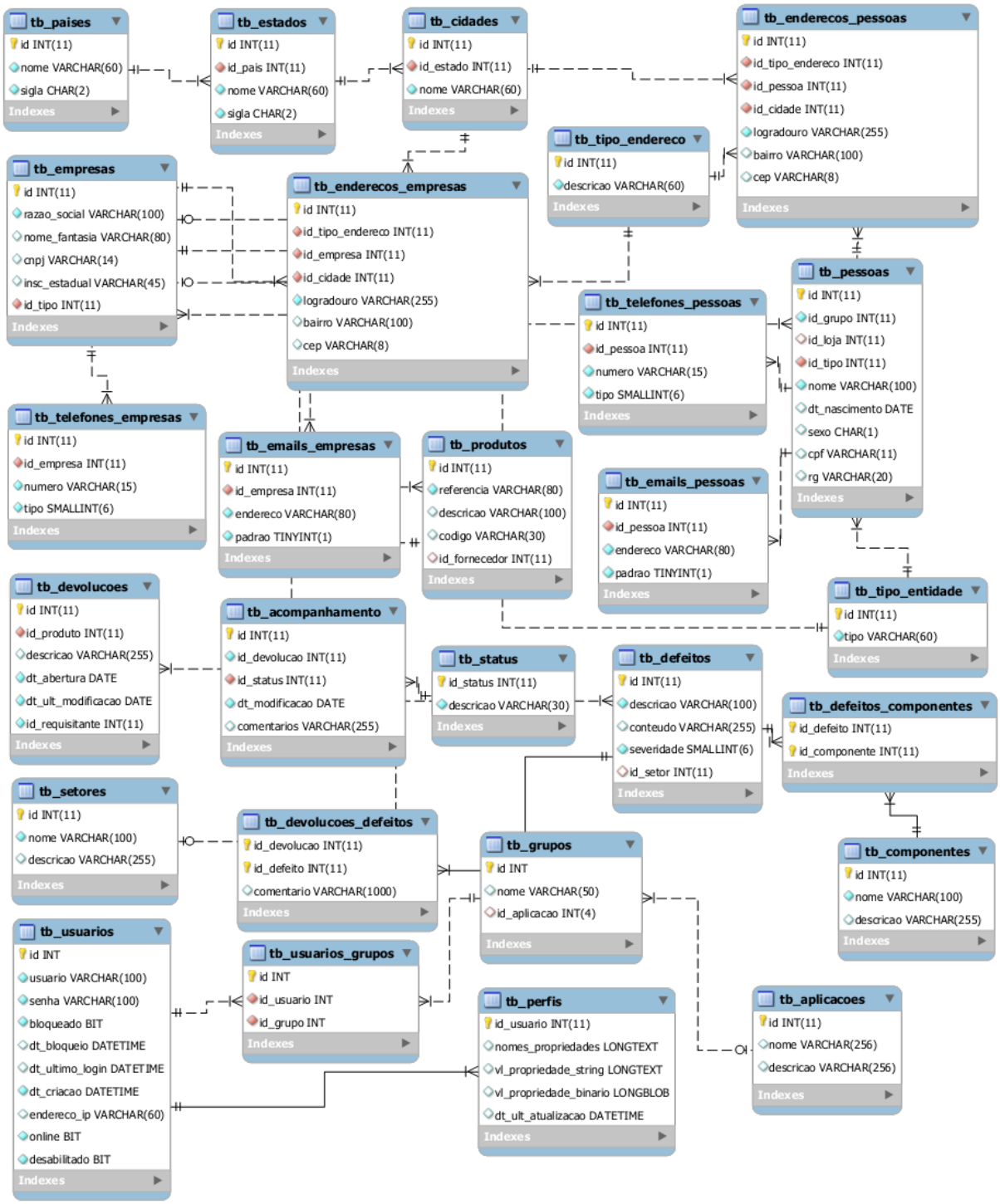

FIGURA 6 – Diagrama Entidade-Relacionamento
Fonte: Autoria própria (2012)

Com o diagrama ER criado, foi possível fazer a engenharia reversa a partir do modelo e gerar o banco de dados físico. Todas as modificações posteriores foram feitas no modelo e sincronizadas sempre que necessário.

3.1.2 Modelo EDM e Diagrama de Classes

Com o banco de dados disponível, foi possível iniciar a geração do modelo conceitual de entidades (*Entity Data Model* - EDM), utilizando o Entity Framework (seção 2.6). A criação do modelo conceitual é feita dentro do próprio Visual Studio, na forma de um *wizard* que simplifica o processo, bastando o desenvolvedor informar a forma de geração – por um banco de dados já existente ou por classes já definidas – a *string* de conexão ao banco de dados e as entidades que deseja importar. Após a criação do EDM, todas as entidades (tabelas) do banco de dados foram transformadas (mapeadas) em objetos e dispostos para uso.

A criação manual dos diagramas para as classes de persistência não foi necessária, já que as ferramentas de diagramação presentes no Visual Studio permitem a geração automática utilizando o modelo conceitual como mostra o exemplo na Figura 7:

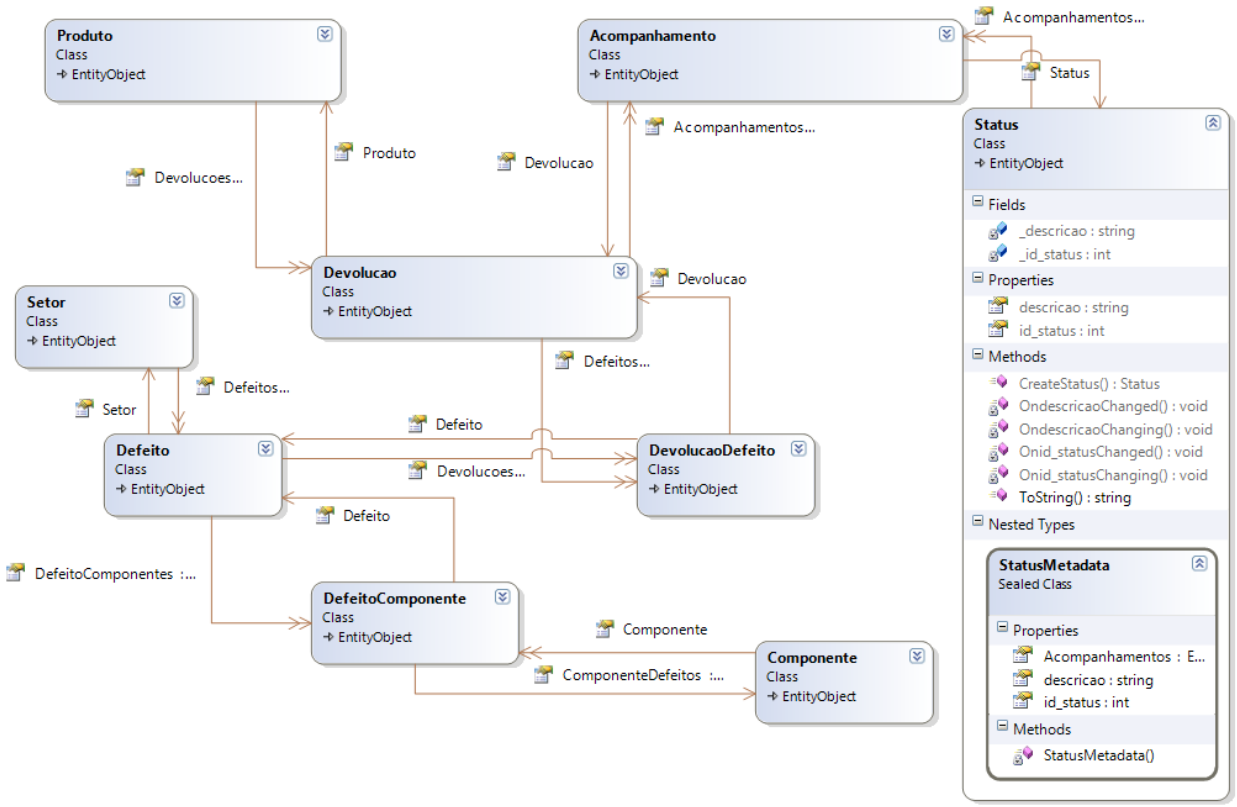

FIGURA 7 – Exemplo de diagrama de classes
Fonte: Autoria própria (2012)

Como se pode observar na Figura 7, a classe de Status não implementa as operações de inserção, localização e exclusão de registros. Isso se deve ao fato dessas operações serem

disponibilizadas pelo contexto de objetos (*ObjectContext*) gerado pelo Entity Framework. O contexto de objetos é a forma primária de interação com os dados das classes geradas no modelo conceitual e encapsula as seguintes funcionalidades: (i) a conexão ao banco de dados, (ii) os metadados que descrevem o modelo e (iii) mantém os estados dos objetos em *cache* e gerencia as instâncias das entidades e seus relacionamentos (MSDN, 2012b).

3.2 Desenvolvimento

Para facilitar o desenvolvimento do software, bem como a futura manutenção e adição de novos recursos, foram adotados dois conceitos distintos: o padrão MVVM e a modularização, descritos a seguir.

3.2.1 MVVM

O padrão de projeto adotado para a codificação do software foi o MVVM (*Model-View-ViewModel*). Introduzido em 2005 pela Microsoft e proposto como uma especialização do padrão PM (*Presentation Model*) e que, do mesmo modo, isola a parte visual (gráfica) da parte lógica de apresentação dos dados (GAROFALO, 2011).

Diferentemente do padrão MVC (*Model-View-Controller*), no MVVM o modelo (*Model*) não notifica a visão (*View*), mas sim o seu modelo-visão (*ViewModel*). A visão por sua vez é um observador do seu modelo-visão, ou seja, no momento que ocorre alguma modificação a visão é notificada imediatamente. A sincronização dos dados entre visão e modelo-visão ocorre por um processo chamado de *Data Binding* (ligação de dados) que estabelece uma ligação entre a interface gráfica e a lógica de apresentação e, caso a ligação seja bem sucedida e as notificações estejam corretas, as modificações são propagadas de forma instantânea (MSDN, 2012c).

A Figura 8 demonstra em detalhes a responsabilidade de cada componente (classe) presente na arquitetura do MVVM. O modelo que fica encarregado somente da lógica de negócio e da manipulação de dados. Já o modelo-visão fica encarregado de gerenciar a lógica de apresentação, os comandos executados na visão, os eventos e também guarda uma referência ao modelo. A visão que mantém a lógica de exibição (XAML) e também todos os componentes gráficos apresentados ao usuário.

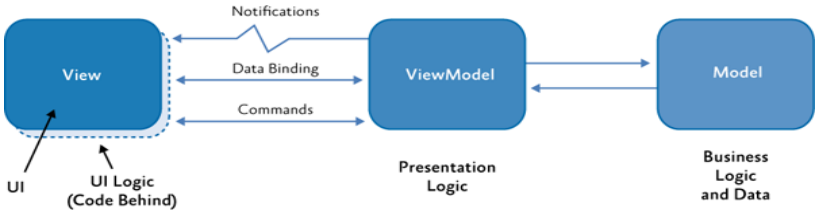

FIGURA 8- Responsabilidades de classes no MVVM
Fonte: MSDN (2012c)

3.2.2 Modularização

Ferrer *et al.* (1999) dizem que um módulo é um grupo de comandos que constitui um trecho de algoritmos, com uma função bem definida e o mais independente possível ao restante do algoritmo. Idealmente cada módulo deve ter um tamanho reduzido e deve definir suas próprias estruturas de dados para que sejam suficientes para atingir seu objetivo. A divisão de um algoritmo em módulos facilita sua elaboração, além de permitir uma melhor documentação e testes mais objetivos.

Wildermuth (2009) complementa a descrição e relaciona a modularização com o Silverlight dizendo que conforme os requerimentos mudam e o projeto vai maturando, é de grande ajuda modificar partes da aplicação sem que as alterações sejam propagadas por todo o sistema. A modularidade permite produzir componentes da aplicação de forma separada e fracamente acoplada e a modificação em uma destas partes não afeta o restante do código.

A Figura 9 demonstra a divisão do *software* em módulos distintos, o principal (base) é o ponto de entrada do sistema e é encarregado de gerenciar os menus e carregar os outros módulos. O módulo de cadastros agrega todos os formulários disponíveis ao usuário, incluindo os formulários de produtos e devoluções. Dentro do módulo de modelo (model) está a camada de persistência de dados e os artefatos gerados pelo Entity Framework. Os relatórios e listagens de defeitos e devoluções ficam no módulo de relatórios. Todos os módulos utilizam de forma direta ou indireta o módulo de funções (lib), nele ficam todas as bibliotecas, funções, recursos, validações, ícones e imagens compartilhadas. Com exceção do próprio módulo de funções, cada um dos outros módulos gera um novo arquivo XAP, que pode ser considerado como um aplicativo separado e que funciona de maneira isolada dos demais.

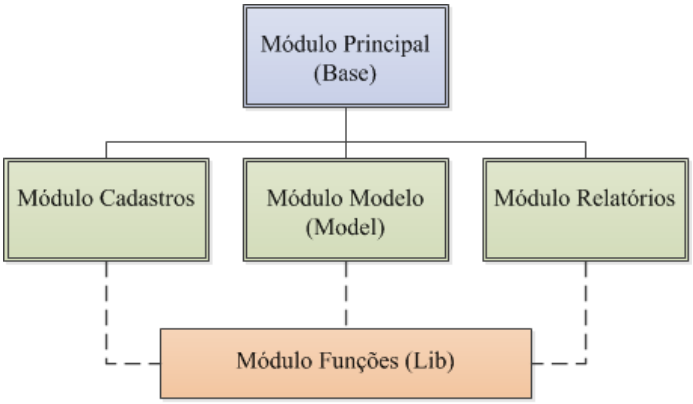

FIGURA 9 – Modularização do sistema
Fonte: Autoria própria (2012)

4. Resultados

Neste capítulo são apresentados os resultados do desenvolvimento do *software*. Como descrito no capítulo 3.2, o sistema foi dividido em módulos e cada um deles engloba um conjunto de funcionalidades.

4.1 Módulo Principal

O módulo principal é ponto de entrada do sistema, nele residem os menus, a lógica de carregamento dos módulos adicionais e do envio dos dados de autenticação para o servidor.

Assim que o sistema é acessado, a tela de login é exibida e, caso os dados informados estejam corretos, o menu principal do aplicativo é carregado e a partir dele o usuário pode acessar os outros módulos, conforme visualizado na Figura 10:

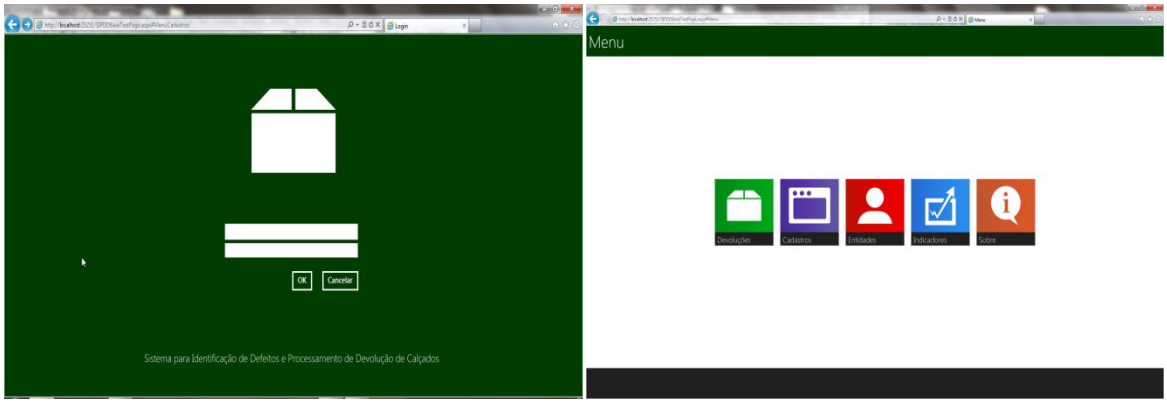

FIGURA 10 - Tela de Login e Menu Principal
Fonte: Autoria própria (2012)

4.2 Módulo de Cadastros

Dentro do menu principal, o usuário tem a opção de acessar o módulo de cadastro. Ele está visualmente divido em três partes, apesar das mesmas integrarem um único módulo: (i) os cadastros gerais, (ii) os cadastros de entidades e (iii) as devoluções. Os formulários de devolução são diferenciados dos demais e serão abordados em um tópico separado. Ao entrar nos cadastros gerais ou de entidades, serão apresentados os formulários especificados nos requisitos do sistema, como mostra a Figura 11:

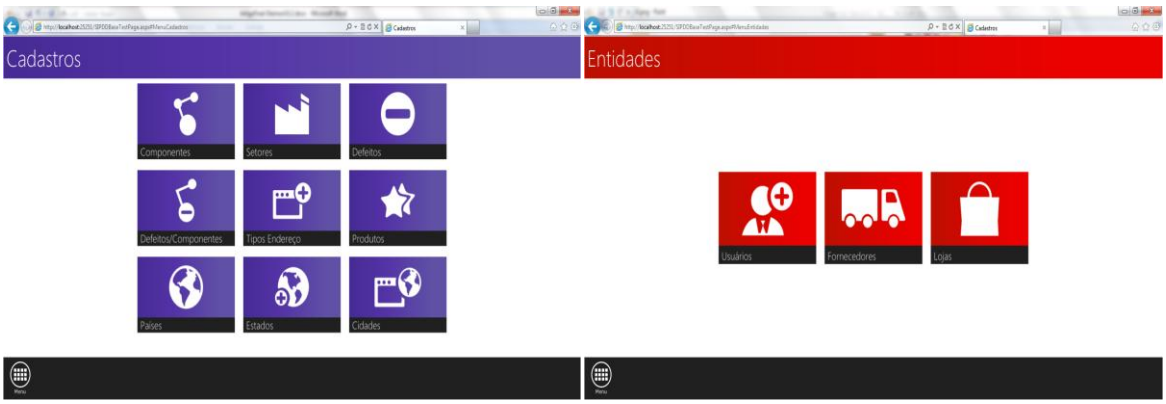

FIGURA 11 - Cadastros Gerais e Cadastros de Entidades
Fonte: Autoria própria (2012)

Todos os formulários utilizados para cadastro de informações têm um padrão uniforme de interface gráfica e funcionamento, para facilitar a utilização e entendimento por parte do usuário. Cada um deles é organizado em quatro seções distintas, identificadas na Figura 12 da seguinte forma:

1) Barra de título do formulário ativo.
2) Grade de localização, para visualizar ou modificar registros já cadastrados.
3) Área de adição e edição de registros, que permite informar as informações de um novo registro ou editar um registro selecionado na grade de localização.
4) Barra de ações.

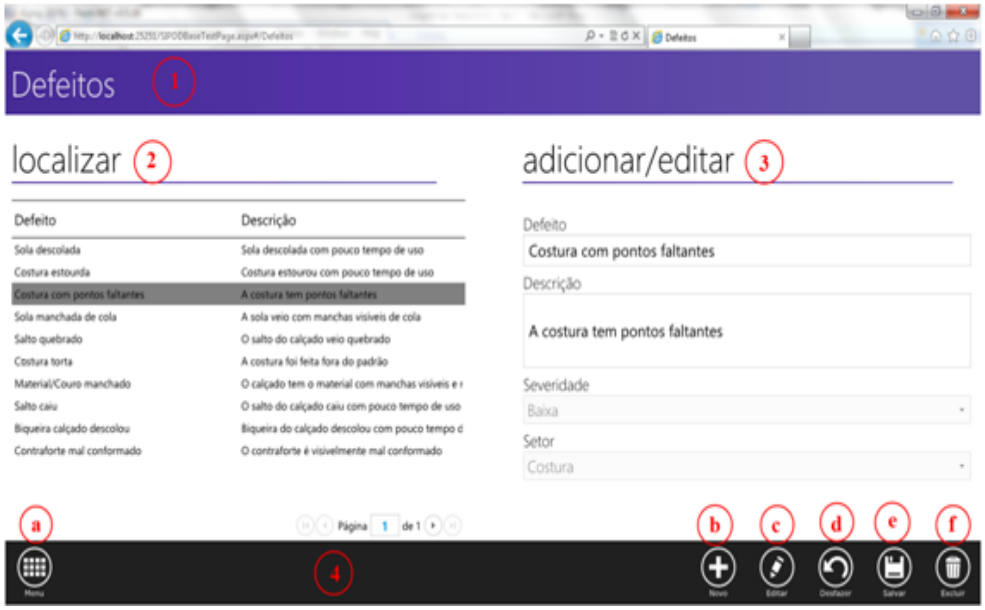

FIGURA 12 - Cadastro de defeitos
Fonte: Autoria própria (2012)

A barra de ações oferece seis funções adicionais, também identificadas na Figura 12 como:

a) Menu: volta ao menu anterior.
b) Novo: cria um novo registro e coloca o formulário em modo de edição.

c) Editar: coloca o formulário em modo de edição.

d) Desfazer: desfaz a inclusão ou alteração do registro.

e) Salvar: valida os dados informados e envia as modificações para o servidor.

f) Excluir: exclui o registro selecionado do banco de dados.

Os outros formulários não serão exibidos neste artigo por serem semelhantes e terem basicamente as mesmas funções.

4.2.1 Devoluções

O menu de devoluções, ver Figura 13, engloba duas atividades distintas: efetuar uma nova devolução e acompanhar as devoluções em andamento. O usuário operador pode acessar o formulário de nova devolução livremente e acompanhar suas próprias devoluções. O usuário gestor pode apenas acompanhar (avaliar) e modificar o status de qualquer devolução efetuada no sistema.

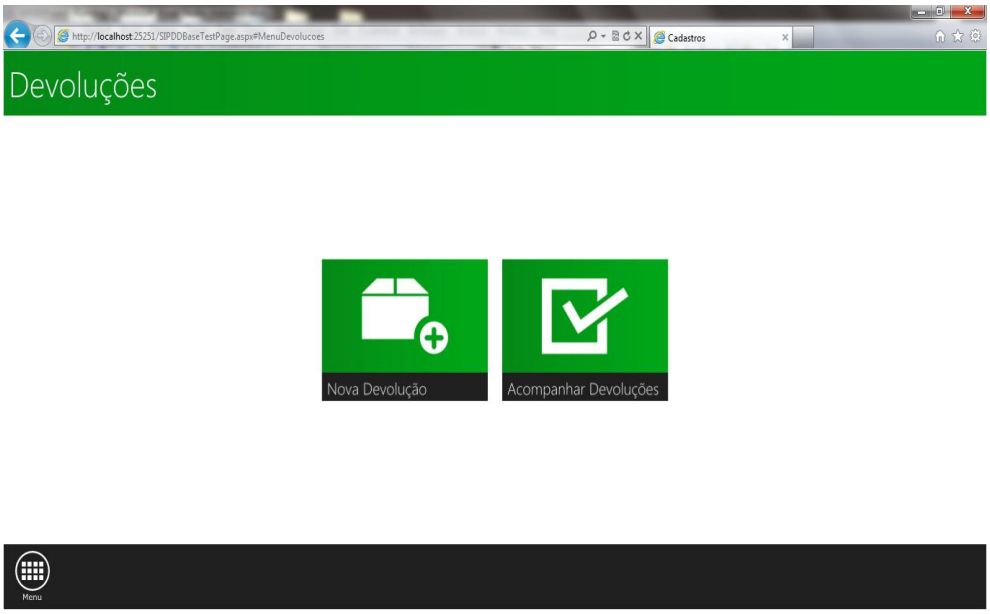

FIGURA 13 - Efetuar e acompanhar devoluções.
Fonte: Autoria própria (2012)

As novas devoluções são realizadas ao completar o formulário padrão, que funciona guiando o usuário pelos passos necessários para concluir o processo com sucesso, sendo identificados na Figura 14 como:

1) Tela de boas vindas.
2) Formulário de defeitos: onde o usuário informa um ou mais defeitos que ocorreram com o produto, juntamente com comentários sobre cada problema.
3) Envio de imagens: exibe um seletor de imagens onde o usuário pode incluir e enviar fotos do produto defeituoso. As fotos serão vistas posteriormente pelo gestor para auxiliar na avaliação.

4) Visão geral: permite ao usuário ver um resumo dos dados informados nos passos anteriores.

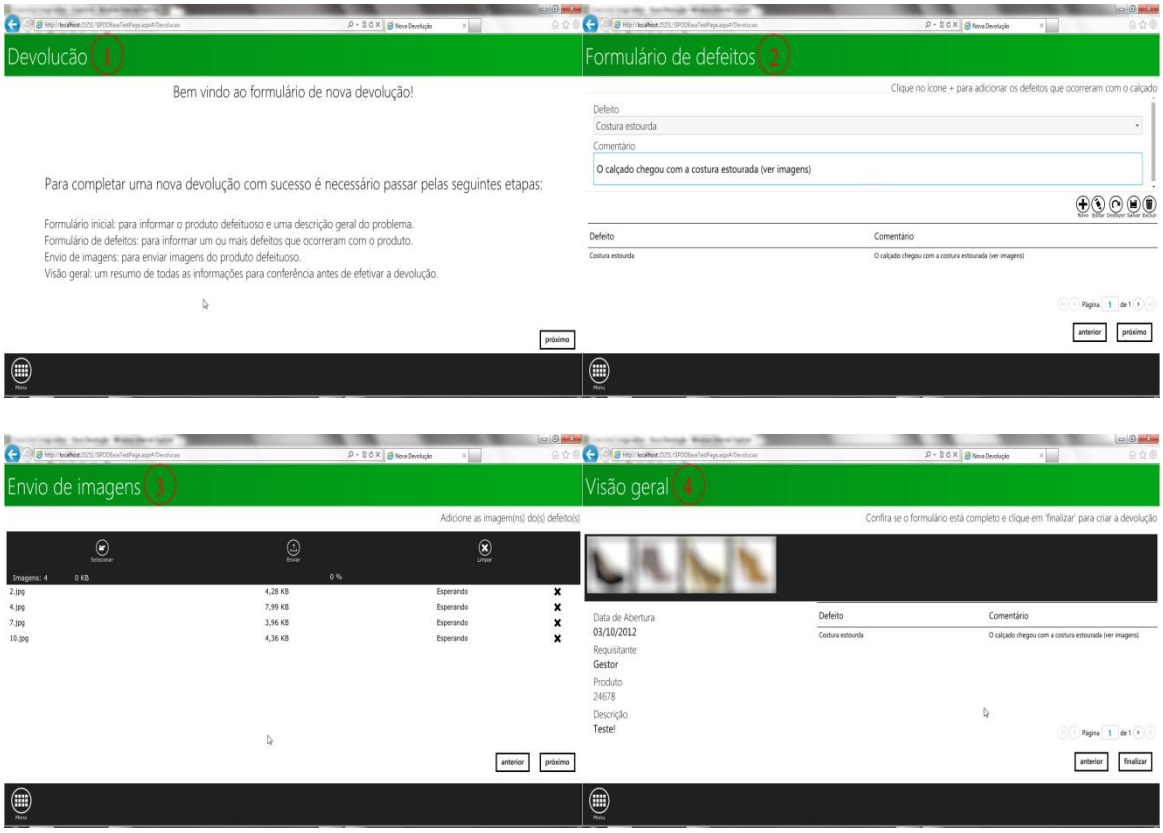

FIGURA 14 – Formulário de devolução.
Fonte: Autoria própria (2012)

Com a devolução criada, o gestor pode posteriormente iniciar o processo de avaliação da requisição consultando o formulário de acompanhamento, ver Figura 15, que exibe as seguintes informações: (i) data de abertura (criação), (ii) produto devolvido, (iii) requisitante, (iv) status da requisição, (v) botão para adicionar novo acompanhamento.

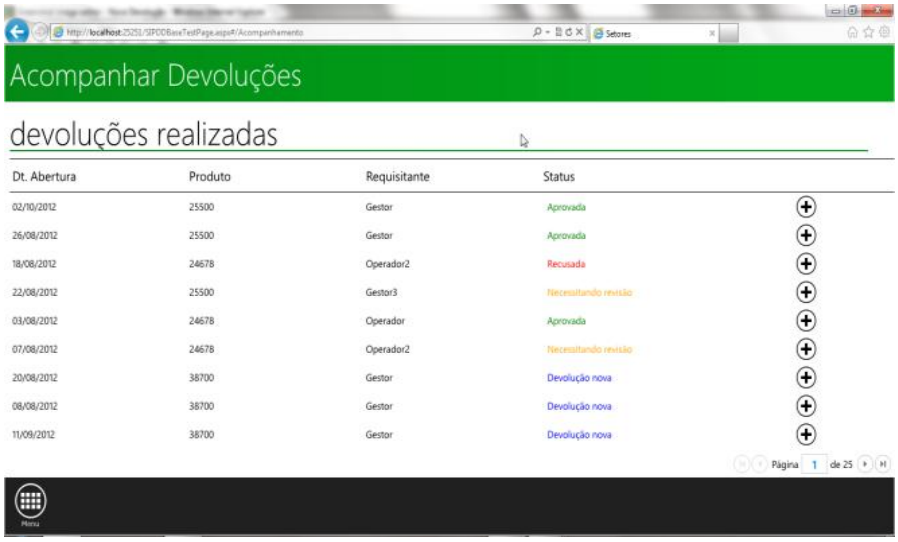

FIGURA 15- Formulário de acompanhamento
Fonte: Autoria própria (2012)

Ao clicar no botão para adicionar um novo acompanhamento, o usuário é direcionado para o formulário de acompanhamentos, onde pode modificar o status da requisição, bem como adicionar um comentário que será exibido ao requisitante e ver a galeria de fotos envidas pelo usuário como mostra a Figura 16.

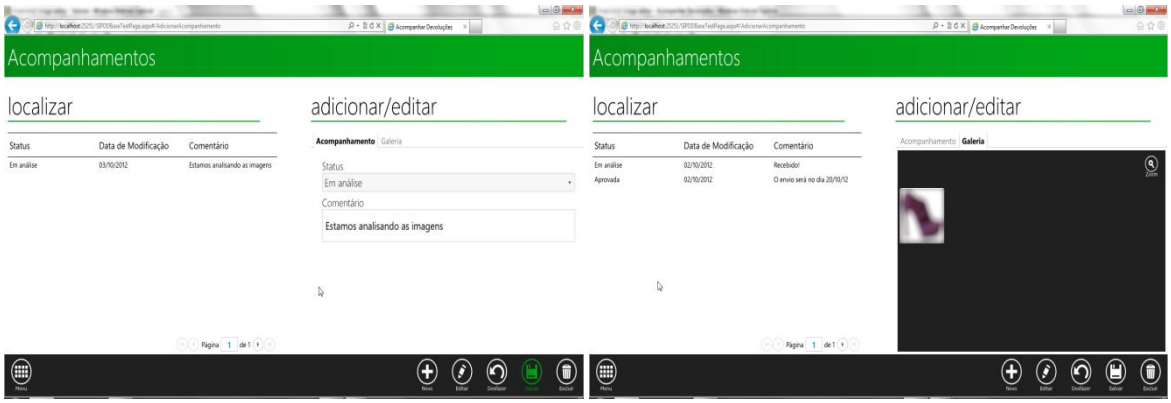

FIGURA 16- Novo acompanhamento e galeria de fotos
Fonte: Autoria própria (2012)

4.3 Módulo de Indicadores

O módulo de indicadores agrega todos os gráficos e relatórios disponíveis, ver Figura 17, limitando a visualização dos dados dependendo do tipo de usuário autenticado no momento. Para usuários do tipo operador alguns relatórios não serão exibidos, pois contém informações pertinentes a fornecedores, lojas ou relacionadas com a produção dos produtos.

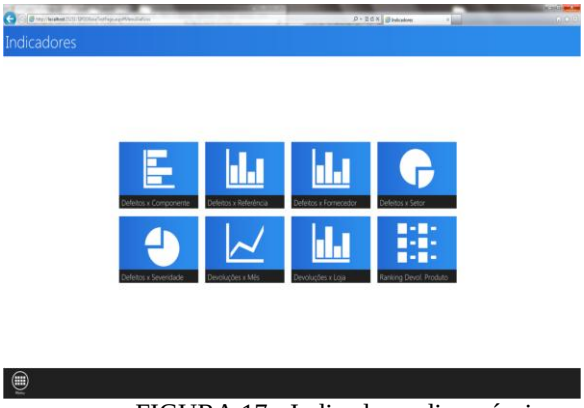

FIGURA 17 - Indicadores disponíveis
Fonte: Autoria própria (2012)

A Figura 18 demonstra os gráficos do número de defeitos por fornecedor e gráfico do número de devoluções por loja. Eles são importantes, pois identificam de forma rápida quais são os fornecedores com maior número de defeitos e quais são as lojas que mais devolvem produtos, e por usarem a visualização em forma de barras, permitem ao usuário ver de forma clara a diferença no

nível entre as entidades relacionadas. Também na Figura 18, no gráfico de defeitos por fornecedor, pode-se notar rapidamente que, dentre os resultados apresentados, o fornecedor 2 está abaixo da média, o que é bom e o fornecedor 1 está muito acima, o que é ruim.

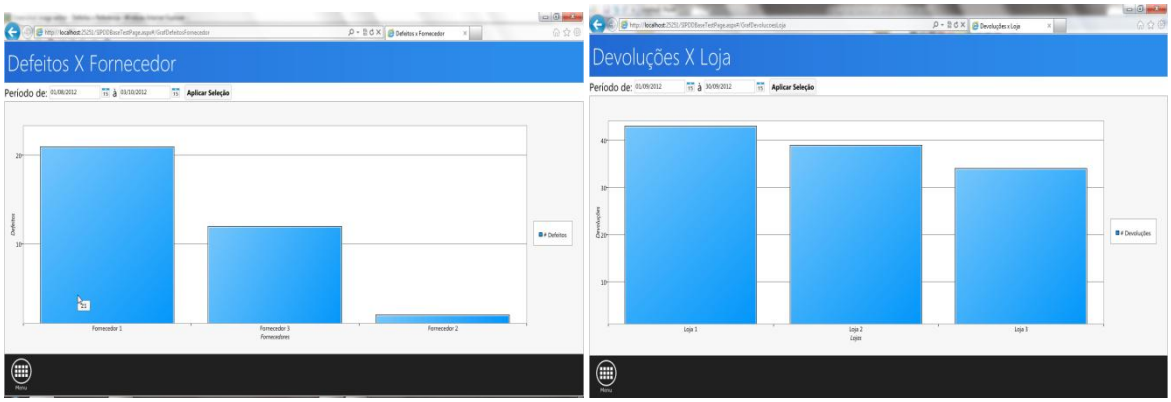

FIGURA 18 – Defeitos x fornecedor e Devoluções x loja
Fonte: Autoria própria (2012)

Os gráficos setoriais exibidos na Figura 19 demonstram com clareza os setores com maior índice de defeitos na empresa, bem como o nível de severidade dos defeitos identificados. Sabendo o setor mais problemático de sua empresa, o gestor pode, de maneira eficiente, otimizar o processo de produção. E conhecendo a severidade dos defeitos, sabe com certa segurança o momento oportuno de tomar uma decisão.

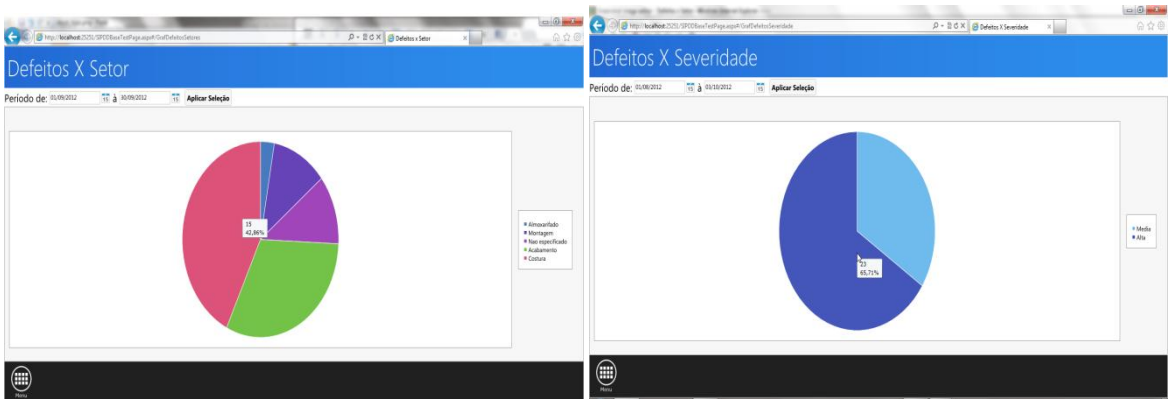

FIGURA 19 – Defeitos x setor e Defeitos x severidade
Fonte: Autoria própria (2012)

O gráfico de devoluções por mês, ver Figura 20, demonstra uma progressão do volume de devoluções realizadas em cada dia do mês corrente ou nos meses anteriores, dependendo da seleção do usuário. Na exibição pode-se notar que no dia 26/09/2012 houve um aumento repentino no número de devoluções, o que poderia indicar uma data especial, como um dia festivo onde as vendas aumentam naturalmente ou mesmo o início das vendas de uma nova modelagem de calçados com propensão a defeitos.

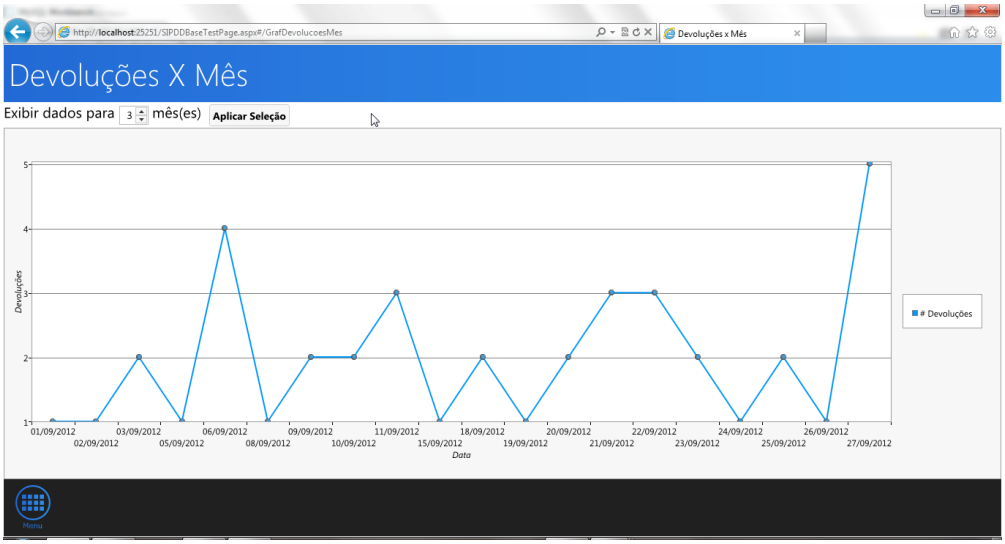

FIGURA 20 – Devoluções x mês
Fonte: Autoria própria (2012)

O gráfico de defeitos por componente, ver Figura 21, também auxilia na otimização da linha de produção, pois exibe o número de defeitos registrados em cada componente do calçado, o que pode demonstrar indícios de matérias-primas de baixa qualidade ou problemas de fabricação.

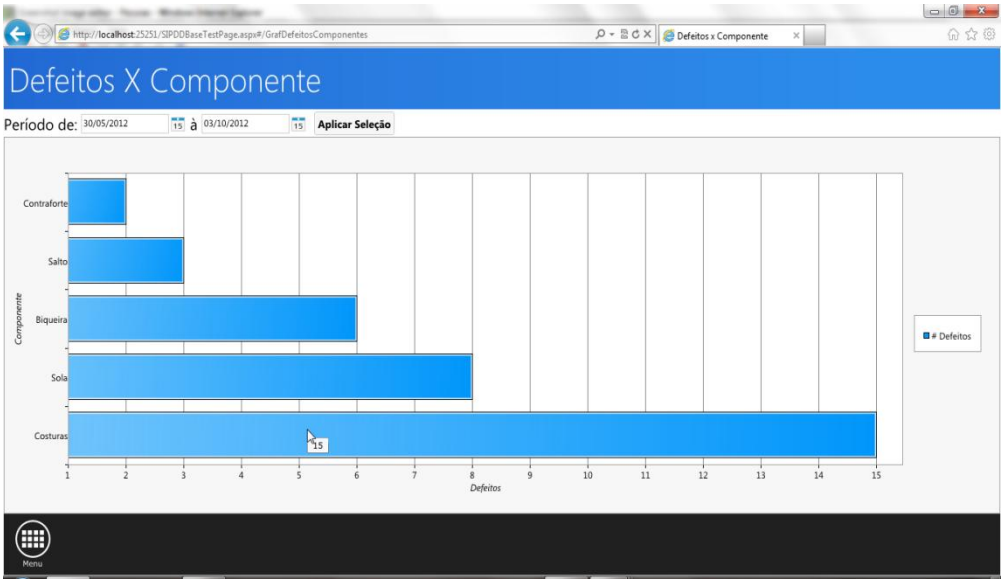

FIGURA 21 – Defeitos x componente
Fonte: Autoria própria (2012)

Os outros indicadores não serão descritos neste artigo por questão de brevidade, mas estarão disponíveis na forma de anexos.

5. Conclusão

No presente artigo foi apresentada a conceitualização, análise e a metodologia aplicada no projeto e desenvolvimento de um sistema para identificação de defeitos e processamento de devolução de calçados. A solução proposta pelo autor visa ajudar as empresas, sejam elas com linha de produção própria ou franqueadoras, a identificar possíveis problemas com seus produtos, setores, matérias-primas ou mesmo fornecedores e lojas. Ao facilitar o acesso à entrada de informações e simplificar o processo de devolução, o gestor pode posteriormente se beneficiar com a padronização da informação, pois é aberta a possibilidade para a geração de indicadores. Estes indicadores exibidos na forma de gráficos e relatórios serão utilizados para observar padrões que demonstrem algum tipo de problema, sejam eles estruturais ou não, como falhas na linha de montagem, matérias-primas de má qualidade, setores que estão aquém do padrão esperado ou até mesmo fornecedores ruins.

Outro ponto relevante é a redução de tráfego físico dos itens defeituosos para serem avaliados para devolução, com o uso de imagens o envio dos produtos fica limitado somente a uma via: o retorno ao requisitante. Além disso, a burocracia necessária para avaliação dos defeitos e o tempo de chegada do produto a fábrica ou franqueadora é reduzido, pois o resultado da avaliação é feito diretamente no sistema, gerando economia de tempo e de recursos importantes.

Identificar problemas assim que eles ocorrem ou em um curto espaço de tempo gera economia para as empresas, gera produtos melhores e clientes satisfeitos. Simplificar o processo de devolução e padronizar as informações evita erros e futuros desentendimentos, além de reforçar a relação de confiança entre fábrica-loja e franqueado-franqueador.

Além da parte funcional e útil do sistema, estão as tecnologias empregadas na sua implementação. As tecnologias e produtos da família .Net da Microsoft, apesar de complexos e em grande número, foram de extrema valia pois permitiram produzir muito em um curto espaço de tempo, além de complementarem umas as outras de forma exemplar. O Silverlight permitiu ao autor realizar sua ideia de uma interface limpa e simplista, porém viva, com cores, animações e transições que chamam a atenção do usuário. A responsividade obtida pelo uso de uma arquitetura em forma de *plug-in* permitiu ao sistema ter um "sentimento" de resposta aos comandos da mesma maneira que uma aplicação *desktop* típica. O Entity Framework em conjunto com o MySQL produziram a rapidez necessária para que os relatórios e gráficos pudessem ser gerados de forma condizente com o restante da aplicação. Igualmente úteis foram as ferramentas oferecidas pelo Visual Studio, tanto para programação quanto para a geração dos diagramas UML.

De certo modo a aplicação não está totalmente completa, pois sempre existe a possibilidade e também a necessidade de serem acrescentados novos relatórios, gráficos e também funcionalidades adicionais. Todo *software* sempre tem espaço para melhorias e novas funções, ele

nunca está completo, seu desenvolvimento é um ciclo contínuo de erros e acertos até se tornar a solução ideal. Apesar disso, dentro do escopo aqui proposto, os resultados mostram que os objetivos do trabalho foram totalmente alcançados.

6. Referências

ABLAC. **Associação Brasileira de Lojistas de Artefatos e Calçados**. Disponível em: <http://www.ablac.com.br/>. Acesso em: 03 ago. 2010.

ALBAHARI J.; ALBAHARI B. **C# Pocket Reference.** United States: O'Reilly, 2012.

APACHE SOFTWARE FOUNDATION. **SubVersion**. Disponível em: <http://subversion.apache.org/>. Acesso em: 14 ago. 2012.

BROWN, P. **Silverlight 5 in Action.** United States: Manning Publications, 2012.

FARRER, H; ET AL. **Algoritmos Estruturados**. 3 ed. Belo Horizonte: LTC, 1999.

GANDIN Danilo. **Indicadores: sinais da realidade no processo de planejamento**. São Paulo: Loyola, 2002.

GAROFALO, R. **Building Enterprise Applications with Windows Presentation Foundation and the Model View ViewModelPattern.** United States: Microsoft Press, 2011.

GHODA, A.; DALAL, M. **XAML Developer Reference**. United States: Microsoft Press, 2011.

KLEIN, S. **Pro Entity Framework 4.0**. United States: Apress, 2010.

KOSCIANSKI, A.; SOARES, M. dos S. **Qualidade de software**. 2 ed. São Paulo: Novatec, 2007.

LARMAN, C. **Utilizando UML e Padrões.** Porto Alegre: Bookman, 2007.

LERMAN, J. **Programming Entity Framework**, United States: O'Reilly, 2009.

LIU, M. **WCF 4.0 Multi-tier Services Development with LINQ to Entities.** United States: Packt Publishing, 2010.

LIKNESS, J. **Designing Silverlight Business Applications: Best Practices for Using Silverlight Effectively in the Enterprise.** United States: Addison-Wesley, 2012.

MARTINS, José Carlos Cordeiro. **Técnicas Para Gerenciamento de Projetos de Software.** Rio de Janeiro: Brasport, 2007.

MSDN. **.NET Framework Conceptual Overview**. 2012a Disponível em: <http://msdn.microsoft.com/en-us/library/zw4w595w(v=vs.100).aspx>. Acesso em: 01 ago. 2012.

MSDN. **ObjectContext Class**. 2012b Disponível em: <http://msdn.microsoft.com/pt-br/library/system.data.objects.objectcontext.aspx>. Acesso em: 07 set. 2012.

MSDN. **Implementing the MVVM Pattern**. 2012c Disponível em: <http://msdn.microsoft.com/en-us/library/gg405484(v=pandp.40).aspx>. Acesso em: 15 ago. 2012.

MYSQL. **MySQL.** 2012a Disponível em: <http://www.mysql.com>. Acesso em: 11 ago. 2012.

MYSQL. **MySQL Workbench.** 2012b Disponível em: <http://http://www.mysql.com/products/workbench/>. Acesso em: 11 ago. 2012.

PRESSMAN, Roger S. **Engenharia de Software**. 6 ed. Rio de Janeiro: McGraw-Hill, 2006.

ROSINHA, D.**Varejo calçadista vem sofrendo com devoluções.** Disponível em: <http://www.exclusivo.com.br/Noticias/53417>. Acesso em: 04 ago. 2010.

SCOTT, K. **O processo unificado explicado**. Porto Alegre: Bookman, 2003.

SHARP, J. **Windows® Communication Foundation 4 Step by Step**. United States: Microsoft Press, 2010.

SILVERLIGHT. **Silverlight.** Disponível em: <http://www.silverlight.net/>. Acesso em: 02 ago. 2012.

TECHNET. **Introduction to .NET.** Disponível em: <http://technet.microsoft.com/en-us/library/bb496996.aspx>. Acesso em: 21 ago. 2012.

TROELSEN, A. **Pro C# 2010 and the .NET 4 Platform.** 5 ed. United States: Apress, 2010.

W3C. **SOAP Version 1.2 Part 1: Messaging Framework (Second Edition)**. Disponível em: <http://www.w3.org/TR/soap12-part1/#intro>. Acesso em: 20 ago. 2012.

WILDERMUTH, S. **Composite Web Apps With Prism**. Disponível em: <http://msdn.microsoft.com/en-us/magazine/dd943055.aspx#id0420088>. Acesso em: 08 fev. 2012.