CENTRO UNIVERSITÁRIO FACVEST
CURSO DE CIÊNCIA DA COMPUTAÇÃO
TRABALHO DE CONCLUSÃO DE CURSO

DESENVOLVIMENTO DE SOFTWARE: SEPARANDO INTERFACE DAS REGRAS DE NEGÓCIO

Área: Engenharia de Software

GUILHERME MATOS OLIVEIRA

LAGES (SC), DEZEMBRO DE 2012.

CENTRO UNIVERSITÁRIO FACVEST
CURSO DE CIÊNCIA DA COMPUTAÇÃO
TRABALHO DE CONCLUSÃO DE CURSO

DESENVOLVIMENTO DE SOFTWARE: SEPARANDO INTERFACE DAS REGRAS DE NEGÓCIO

Área: Engenharia de Software

GUILHERME MATOS OLIVEIRA

Projeto apresentado à Banca Examinadora do Trabalho de Conclusão do Curso de Ciência da Computação para análise e aprovação.

LAGES (SC), DEZEMBRO DE 2012.

GUILHERME MATOS OLIVEIRA

DESENVOLVIMENTO DE SOFTWARE: SEPARANDO INTERFACE DAS REGRAS DE NEGÓCIO

Trabalho de Conclusão de Curso apresentado à banca examinadora e ao Centro Universitário FACVEST como parte dos requisitos para a obtenção de bacharel em Ciência da Computação.

Prof. Msc. Márcio José Sembay

Lages, SC _____/_____/2012. Nota _____ _____

_____

Prof. Coordenador do Curso de Ciência da Computação

LAGES (SC), DEZEMBRO DE 2012

EQUIPE TÉCNICA

Acadêmico

Guilherme Matos Oliveira

Professor Orientador

Profº. Márcio José Sembay, Msc.

Coordenador de TCC

Profº. Márcio José Sembay, Msc.

Coordenador do Curso

Profº. Márcio José Sembay, Msc.

Dedico,

À minha mãe e à minha irmã por estarem por perto, por darem essa oportunidade, e por nunca terem deixado de lutar por mim e pela minha educação.

À minha namorada por ter me dado força, apoio, motivação, por ter me ajudado nesse momento difícil, e, acima de tudo, por ter me dado carinho e amor.

AGRADECIMENTOS

Agradeço ao professor/orientador Márcio José Sembay pela sua dedicação, disponibilidade e por ter me ajudado com dicas durante o desenvolvimento deste trabalho.

Agradeço à minha mãe e à minha irmã pela paciência, por entenderem o meu estresse, por me apoiarem e me darem forças, pela oportunidade e pelo incentivo, pois por mais difícil que tenha sido esse caminho, elas sempre conseguiram superar os desafios para me dar uma oportunidade como essa.

Agradeço à minha namorada, que sempre esteve do meu lado, pois sempre conseguia colocar um sorriso no meu rosto, me incentivou, me apoiou e não me deixou desistir.

Agradeço aos meus amigos que estavam por perto, que me suportaram e me ajudaram nessa etapa.

Agradeço ao Kaio Sales, pois foi ele que apresentou a ideia, me ajudou, tirou dúvidas, explicou tecnologias.

Agradeço à NDDigital S/A Software, por ter me dado a oportunidade de aplicar meu trabalho dentro de uma equipe de desenvolvimento.

Agradeço aos outros profissionais que disponibilizaram livros e materiais na internet, e artigos científicos, pois foram essenciais para o desenvolvimento deste trabalho.

“Tecnologia é a habilidade de organizar o mundo de
forma que não tenhamos que senti-lo."

Max Frisch

SUMÁRIO

LISTA DE FIGURAS

LISTA DE QUADROS

LISTA DE ABREVIATURAS

2D	Duas Dimensões.
3D	Três Dimensões.
DB	*Database.*
IEEE	*The Institute of Electrical and Electronics Engineers*.
ISO	*International Organization for Standardization*.
MVC	*Model-View-Controller*.
MVP	*Model-View-Presenter*.
MVVM	*Model-View-ViewModel*.
SGBD	Sistema de Gerenciamento de Banco de Dados
SQL	*Structured Query Language*
TCC	Trabalho de Conclusão de Curso
UML	*Unified Modeling Language*.
VB	*Visual Basic*.
XAML	*Extensible Application Markup Language*.
XML	*Extensible Markup Language*.
WPF	*Windows Presentation Foundation*.

RESUMO

O presente trabalho realiza uma abordagem sobre a importância da Engenharia de Software e a utilização de diversas metodologias de desenvolvimento que juntas auxiliam na criação de sistemas com qualidade. Visto que empresas de desenvolvimento muitas vezes estouram prazos e não criam software com qualidade, são apresentadas algumas tecnologias que, juntas, aumentam a produtividade e minimizam o tempo do desenvolvimento. Qualidade de software é uma das chaves principais para o sucesso de um software, independentemente do tamanho do sistema. Com base em um estudo de caso, pode-se analisar que aplicações bem projetadas e modeladas levam à satisfação do usuário.

Palavras-chave: *Engenharia de software*. *Qualidade de Software. WPF*

ABSTRACT

The current work discusses the importance of software engineering and to use these development's methodologies shall help to create software with quality. Due to many development companies usually reach the product creation's deadline and do not develop a software with quality, some technologies can be presented, and together make the productivity be improved and the development's time reduced. Additionally, quality of software is one of the main keys to the software be successful, independent from de size of the system. Based on case-study, it is possible analyzing and studying that the better designed and modeled the application, the greater costumer's satisfaction will be.

Keywords: *Engineering software*. *Software quality*. WPF.

1 INTRODUÇÃO

O desenvolvimento de software não é uma tarefa fácil, pois envolve muitos processos e necessita de muita colaboração para levar ao sucesso. Porém, é comum encontrarmos sistemas que não são desenvolvidos utilizando técnicas e boas práticas que facilitam a criação, e que possam aperfeiçoar o software levando à qualidade desejada.

Com a aplicação das técnicas de engenharia de software é possível construir um software de maior qualidade, que satisfaça todas as necessidades de clientes e usuários, sem apresentar falhas ou erros, e seja fácil de usar, modificar e manter.

Além disso, utilizando padrões de desenvolvimento com a correta tecnologia, a produtividade pode ser aumentada e o tempo reduzido, atendendo as exigências das empresas, e levando à qualidade ao usuário.

> Projetar software orientado a objeto é difícil, mas projetar software *reutilizável* orientado a objetos é ainda mais complicado. Você deve identificar objetos pertinentes, fatorá-los em classes no nível correto de granularidade, definir as interfaces das classes, as hierarquias de herança e estabelecer as relações-chave entre eles. O seu projeto deve ser especifico para o problema a resolver, mas também genérico o suficiente para atender problemas e requisitos futuros. Também deseja evitar o reprojeto, ou pelo menos minimizá-lo. (GAMMA *et al*., 2000).

Dessa forma, o presente trabalho visa apresentar as melhores técnicas de engenharia de software e alguns padrões de desenvolvimento para a construção de um software com qualidade, e apresentar um sistema onde essas técnicas e práticas foram aplicadas.

O trabalho é dividido em quatro partes, no primeiro capitulo são apresentadas as definições de sistema e sistema da informação, além do ciclo de vida de um software. O segundo capítulo é abordado à engenharia de software, suas técnicas, padrões, processos de software e a linguagem de modelagem UML. No terceiro capitulo é apresentado uma tecnologia, criada pela Microsoft Corporation, conhecida como WPF e seu padrão de arquitetura MVVM, além de outras tecnologias utilizadas no desenvolvimento do software estudado. E por último, é apresentado o estudo de caso, expondo o processo de elaboração do sistema, suas interfaces e diagramas.

1.1 Justificativa

Algumas vezes, as empresas de desenvolvimento de *software* não constroem uma arquitetura adequada para o tamanho do projeto, e isso pode acarretar em uma perca no controle do desenvolvimento quando o devido sistema passa a ser um legado, e pelo período que o desenvolvedor terá para acrescentar as novas demandas, sejam elas internas ou externas. Por esse motivo, foram criados alguns padrões de desenvolvimento de software que servem para auxiliar o desenvolvimento de problemas e/ou necessidades específicas.

Dessa forma, o desenvolvimento de um *software* focado em padrões de projeto deixa o código legível, coerente e preparado para a manutenção do sistema de forma ágil e fácil, pois muitas vezes desenvolvedores deparam-se com situações que já foram resolvidas em projetos anteriores, mas pela deficiência do projeto atual esse problema torna-se complicado e custoso para a empresa.

Por esse motivo, projetar um software deve ser estudado e analisado para que no futuro não gere um retrabalho desnecessário. Com isso, este trabalho tem o objetivo de apresentar um padrão de design de desenvolvimento de software focado na agilidade, manutenibilidade e testabilidade, onde o desenvolvedor consegue criar um sistema bem estruturado e preparado para problemas futuros, ganhando um nível maior de estabilidade, pois além de auxiliar nos processos de desenvolvimento, o software torna-se simples e consegue definir melhor as tarefas dentro de uma equipe de desenvolvimento, separando melhor as atividades, e tendo um tempo de desenvolvimento mais fiel ao cronograma do projeto.

Com isso, o software ganha qualidade pelos processos estarem separados, e por não existir a dependência de projetos, pois a qualidade de software é significativa para o desenvolvimento de um sistema que atenda ao que o mercado exige. Segundo LOBO (2009), o controle de qualidade de software é um dos fatores mais importantes. A qualidade no software permite que fábricas de software criem produtos capazes de atender, da melhor forma, as necessidades dos seus usuários.

1.2 Importância

O desenvolvimento de um software focado na qualidade tem um valor no mercado, e é por esse motivo que este trabalho é importante, pois visa explanar a necessidade de um software ser bem analisado, e mostrar o quão valioso é ter qualidade, pois é assim que um software consegue estar preparado para os problemas e modificações futuras.

Assim como a qualidade de software, os padrões de projeto são essenciais para que esse software seja bem estruturado, pois os padrões de projeto conseguem dar uma direção para os problemas que não costumavam ter uma solução simples, e muitas vezes demorada. Unindo essas duas boas práticas de desenvolvimento junto com um padrão de design criado especificamente para o desenvolvimento de softwares simples, tem-se o projeto ideal para uma equipe de desenvolvimento, onde, no software, existe o baixo acoplamento entre os projetos e todos os processos conseguem ser resolvidos facilmente.

1.3 Objetivos

1.3.1 Objetivo Geral

Explicar a qualidade de software e aplicar o padrão de design de desenvolvimento de software em uma ferramenta de gerência, utilizada na empresa NDDigital Software S/A, visando mostrar a importância de uma arquitetura de software bem definida e analisada para prevenir problemas futuros sem a duplicidade ou a codificação desnecessária.

1.3.2 Objetivo Específico

Os objetivos específicos deste trabalho constituem no estudo de várias tecnologias que fazem um software ter qualidade e o desenvolvimento de um sistema utilizando essas tecnologias. Sendo assim:

1) Apresentar métodos, técnicas e padrões que qualificam um software.
2) Apresentar as tecnologias utilizadas no desenvolvimento do software, que visam auxiliar na criação do produto;
3) Aplicar o padrão de desenvolvimento de software, conhecido como MVVM, em uma ferramenta utilizada na empresa NDDigital Software S/A.

1.4 Metodologia

A Figura 01 mostra as etapas do desenvolvimento deste Trabalho de Conclusão de Curso.

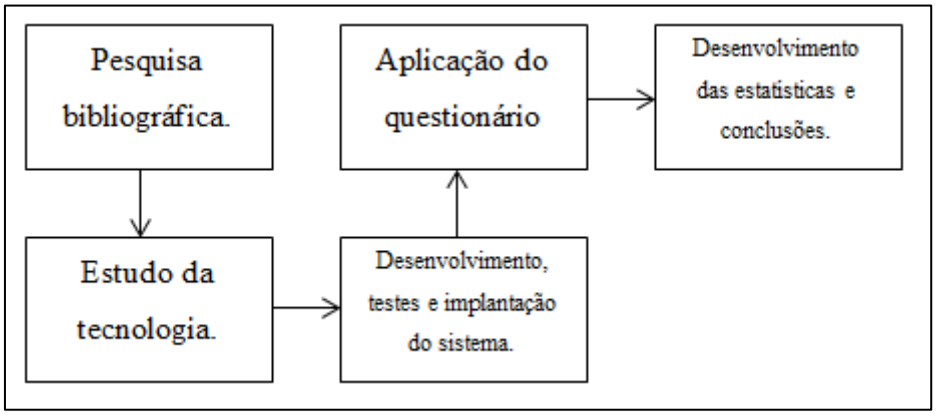

Figura 01: Etapas do Desenvolvimento do TCC.
Fonte: O próprio autor.

1.4.1 Caracterização da Pesquisa

Este trabalho realiza uma pesquisa exploratória, pois o principal objetivo é adquirir conhecimento de uma nova tecnologia e apresenta-la através um estudo de caso aplicado. Com isso, pesquisa exploratória geralmente procura estabelecer critério, métodos e técnicas para a elaboração da pesquisa, além de gerar questionários para uma melhor explanação dos resultados.

> O principal objetivo da pesquisa exploratória é proporcionar maior familiaridade com o objetivo do estudo. Muitas vezes o pesquisador não dispõe de conhecimento suficiente para formular adequadamente um problema ou elaborar de forma mais precisa uma hipótese. Nesse caso, é necessário desencadear um processo de investigação que identifique a natureza do fenômeno e aponte as características essenciais das variáveis que se quer estudar. (HEERDT, 2007).

Por esse motivo, pode-se dizer que a natureza da pesquisa é quanti qualitativa, pois foi aplicado um questionário na equipe de desenvolvimento cujo software foi implantado, e por apresentar técnicas, tecnologias e padrões para o desenvolvimento deste software, o qual foi realizado o estudo de caso.

1.4.2 Coleta e Análise dos Dados

A coleta de dados foi realizada através de um questionário aplicado na equipe de desenvolvimento, explicando o motivo da pesquisa e os critérios para a seleção das perguntas; as perguntas (Anexo A) encontram-se em anexo.

Os recursos utilizados para avaliar o resultado do questionário são os próprios itens da qualidade de software, os quais estão descritos no livro "Guia Prático de Engenharia de Software", de Edson J. R. Lobo, pois visto que o software foi desenvolvido com esse principio, subentende-se que ele tenha qualidade, e também foram retiradas deste site: *http://www2.unemat.br/rhycardo/download/qualidade_em_software.pdf*, as quais foram adaptadas de acordo com a problemática do estudo de caso.

1.4.3 Cronograma

Abaixo, o Quadro 01 mostra o cronograma deste Trabalho de Conclusão de Curso.

ATIVIDADES	AGOSTO	SETEMBRO	OUTUBRO	NOVEMBRO	DEZEMBRO
Pesquisa	X	X			
Revisão Bibliográfica		X	X		
Desenvolvimento			X	X	
Entrega do TCC				X	
Defesa da Banca					X

Quadro 01: Cronograma do TCC
Fonte: O próprio autor.

2 SISTEMA DA INFORMAÇÃO

"A palavra "sistema" possivelmente é o termo mais excessivamente usado e que sofre mais abuso do léxico" (PRESSMAN, 1995), mas o que é um sistema? Muitas vezes apenas trabalhamos em cima de técnicas e esquecemos o que, de certa forma, é importante: as definições. Segundo o *Websters's Dictionary*, a palavra "sistema" é definida como:

> 1. um conjunto ou disposição de coisas relacionado a ponto de formar uma unidade ou um todo orgânico; 2. um conjunto de fatos, princípios, regras etc. classificado e arranjado ordenadamente para mostrar um plano lógico unindo as várias partes; 3. um método ou plano de classificação ou disposição; 4. uma maneira estabelecida de se fazer algo; método; procedimento.

Segundo SCHMITZ (2000), um sistema é um conjunto de partes interligadas ou relacionadas de maneira a formar uma unidade ou um todo orgânico. Dessa forma, podemos entender que um sistema é nada mais e nada menos que uma união de várias "entidades" que, relacionadas, formam um único item para ser utilizado com algum propósito.

Com base nisso, sistemas podem ser projetados para resolver problemas. Dessa forma, WETHERBE (1987) explica que em sistemas projetados ou controlados por pessoas, as entidades são geralmente arranjadas de modo que possam interagir para concluir um ou mais objetivos. Essa ligação entre entidades faz com que o sistema sirva como o intermédio entre o meu problema e a minha solução, onde eu apenas preciso entrar com dados e no final eu vou ter como resultado a minha solução.

Entretanto, sistemas baseados em computadores podem ser chamados de sistemas da informação e, segundo TURBAN, McLEAN e WETHERBE (2004), um sistema da informação baseado em computador (genericamente chamado de sistema da informação) é um método que utiliza tecnologia de computação para executar algumas ou todas as tarefas desejadas. Pode ser composto de apenas um computador pessoal e *software*, ou incluir milhares de computadores de diversos tamanhos com centenas de impressoras e outros equipamentos, bem como redes de comunicação e bancos de dados.

Com isso, podemos levar em consideração o que BATISTA (2004) explica sobre um sistema da informação, o qual é composto por diversos elementos, e a parte de software de um sistema é a principal e a mais eficiente ferramenta para manipular todos os dados gerados no decorrer da utilização.

2.1 Ciclo de Vida do Software

O processo de planejamento de um software deve visar o tempo de vida que esse sistema vai ter, independentemente do porte do sistema eles devem estar preparados para alterações e modificações futuras. Dessa forma, devo concordar com o que SCHMITZ (2000) diz a respeito do ciclo de vida de um sistema:

> Os sistemas da informação modernos têm ficado cada vez mais complexos com o passar do tempo. Modernamente, a construção de um sistema de informação requer o esforço conjugado de muitas pessoas (...). Colocar um grupo de pessoas trabalhando de forma concertada para chegar a um produto de software não é uma tarefa fácil. (...) As estatísticas americanas indicam que cerca de 50% dos projetos de desenvolvimento não atinge seus objetivos, isto é: são entregues fora do prazo, ultrapassam os custos ou não apresentam a qualidade desejada. (SCHMITZ, 2000)

Define-se ciclo de vida do software, o processo que inclui desde a concepção de ideias até a descontinuidade do produto de software. (ISO 12207, 1997).

Dessa forma, "durante o ciclo de vida de software são executados vários processos, sendo que cada um contribui para atingir os objetivos de um estágio do ciclo" (MACHADO, 2002).

Atualmente existem alguns modelos de ciclos de vida do software, os quais eles são categorizados, e são utilizados no desenvolvimento e na manutenção do software. Dentre todas as categorias, segue as mais utilizadas: modelo cascata, desenvolvimento evolucionário, desenvolvimento formal de sistemas, desenvolvimento orientado a reuso, desenvolvimento incremental e desenvolvimento espiral.

Segundo SOMMERVILLE (2003), embora não exista nenhum processo de software "ideal", existem muitas oportunidades de trabalho para melhorá-lo em muitas organizações. Os processos podem incluir técnicas desatualizadas ou podem não tirar vantagem das melhoras práticas da Engenharia de Software.

"A melhoria dos processos de software pode ser implementada de diferentes maneiras. Ela pode ocorrer por meio da padronização de processos, pois é a primeira etapa essencial na introdução de novos métodos, novas técnicas de engenharia de software" (NOGUEIRA, 2009).

2.2 Desenvolvimento de Software

Atualmente, o desenvolvimento de software tem sido de grande importância na nossa sociedade, pois a utilização de computadores só tem aumentado, assim como a utilização de sistemas nas diversas áreas existentes. Porém, muitas vezes os iniciantes em Ciência da Computação confundem o desenvolvimento de software com a programação em si. Desenvolver um software não é só programação, é algo que vai além, pois envolve muitas práticas, padrões e conceitos.

Entretanto, a forma que os iniciantes são introduzidos no meio da programação-desenvolvimento de software não está errada, pois todas as lógicas iniciais e estruturações de dados serão de grande utilidade para o desenvolvimento de software como um todo, porque conforme o nível de lógica vai aumentando, gradativamente a complexidade dos softwares também.

> O sistema é desenvolvido a partir da estaca zero. (...) Primeiramente, são determinadas as necessidades do cliente e, em seguida, é realizada a análise. Quando os artefatos da análise estiverem prontos, é produzido o projeto, que é seguido pela implementação do produto de software completo, que é então instalado nos computadores do cliente. Entretanto, o desenvolvimento de software é consideravelmente diferente na prática por duas razões. Primeiramente, os profissionais de software são seres humanos e, portanto, cometem erros. Em segundo lugar, as necessidades do cliente podem mudar enquanto o produto de software está sendo desenvolvido. (SCHACH, 2009).

Podemos levar em consideração que o desenvolvimento de software pode ser tratado de forma analógica, pois muitas vezes não é necessário criar um projeto, cronograma, vários programadores para um sistema pequeno, e com isso torna-se diferente o desenvolvimento de um sistema grande, pois será necessário acompanhar cronogramas, prazos, programadores, projeto para conseguir chegar ao objetivo final: o software.

3 PROCESSO DE DESENVOLVIMETO DE SOFTWARE

3.1 Engenharia de Software

Ao pensar em qualidade do software e para aumentar a produtividade no processo de desenvolvimento, surgiu a Engenharia de Software, que trata dos aspectos relacionados ao estabelecimento de processos, métodos, técnicas, ferramentas e ambientes de suporte ao desenvolvimento de software.

Podemos dizer que Engenharia de Software é:

> - O estabelecimento e uso de sólidos princípios de engenharia para que se possa obter economicamente um software que seja confiável e que funcione eficientemente em máquinas reais.
> - Descendente da engenharia de sistemas e de hardware. Abrange um conjunto de 3 elementos fundamentais (métodos, ferramentas e procedimentos), que possibilita, ao gerente, o controle do processo de desenvolvimento do software e oferece ao profissional uma base para a construção de software de alta qualidade. (PRESSMAN, 2002)

Antigamente não existia a preocupação ao desenvolver um software, tudo era feito de maneira desordenada sem preocupações da funcionalidade. Com isso, surgiu a necessidade da criação de métodos que permitisse o desenvolvimento de maneira confiável e rápida. Atualmente, empresas de desenvolvimento de software utilizam vários métodos para agilizar a produção do sistema.

Segundo NOGUEIRA (2009), “a engenharia de software é um conjunto de práticas para desenvolvimento de soluções de software, ou seja, um roteiro que pode utilizar diversas técnicas”.

A Engenharia de Software pode ser definida em camadas, na qual temos a qualidade do software como base, logo em seguida os processos atuando como controlador gerencial dos projetos e estabelecendo o contexto que será utilizado no método aplicado.

Os métodos da Engenharia de Software possuem as metodologias e técnicas utilizadas para a construção do software, tais como, análise de requisitos, modelagem de projeto, desenvolvimento do sistema, testes e manutenção. Por último, as ferramentas, como o próprio nome já diz, referem-se aos instrumentos utilizados para obter o produto final, e que forneceram apoio aos métodos e processos.

Figura 02: Camadas da Engenharia de Software
Fonte: Adaptada de Alejandro Fernandez Moraga (2007)

3.1.1 Objetivos da Engenharia de Software

A Engenharia de Software tem como objetivo primário o aprimoramento da qualidade dos produtos de software e o aumento da produtividade dos engenheiros de software, além do atendimento aos requisitos de eficácia e eficiência, ou seja, efetividade (MAFFEO, 1992).

Ao pensar em produtividade e eficiência deve-se pensar em engenharia de software, pois o aprimoramento do software deve acontecer de forma contínua.

Segundo FIORINI:

> Os produtos desenvolvidos e mantidos, seguindo um processo efetivo e segundo preceitos da Engenharia de Software asseguram, por construção, qualidade satisfatória, apoiando adequadamente os seus usuários na realização de suas tarefas, operam satisfatoriamente em ambientes reais e podem evoluir continuamente, adaptando-se a um mundo em constante evolução (FIORINI, 1998).

Dessa forma, é excepcionalmente importe adotar um modelo da Engenharia de Software para gerenciar o desenvolvimento preocupando-se com o futuro do sistema e para alcançar o nível esperado de qualidade e eficiência.

3.1.2 Processo de Software

> Um processo é uma sequência de passos realizados para um dado propósito. Colocado de maneira mais simples, processo é aquilo que você faz. Processo é aquilo que as pessoas estão fazendo, usando procedimentos, métodos, ferramentas e

> equipamentos para transformar matéria-prima (entradas) em produto (saída) que tenha valor para o cliente. (PAULK, 1995)

Podemos, assim, dizer que processos de software são as várias fases necessárias para produzir e manter o software. Individualmente, cada fase necessita de organização e estudo para o desenvolvimento do software.

A partir do momento que é planejado o desenvolvimento de um software, deve-se pensar nos processos que ajudarão a chegar até o produto final. Esse processo deve ser específico para atender as necessidades do desenvolvimento daquele software; construir um software sem uma boa estrutura poderá dar prejuízos ao invés de satisfação.

> Diferentes tipos de sistemas necessitam de diferentes processos de desenvolvimento. (...) O uso de um processo de software inadequado pode reduzir a qualidade ou a utilidade do produto de software a ser desenvolvido e/ou aumentar os custos de desenvolvimento. Este fato leva as organizações que produzem software a usar processos de desenvolvimento que sejam eficientes e que atendam plenamente suas necessidades (SOMMERVILE, 2007).

Por este motivo, existem algumas fases no processo de desenvolvimento de software, e, se seguidas corretamente, elas podem facilitar o desenvolvimento do sistema, levando à qualidade esperada no produto final.

3.1.3 Modelos de Processos de Software

O processo de software pode ser definido como uma coleção de padrões que definem um conjunto de atividades, ações, tarefas de trabalho, produtos de trabalho e comportamentos de trabalho.

Existem vários modelos de processo de desenvolvimento de software que descrevem o fluxo de trabalho desde o projeto até a manutenção. Os principais modelos são:

- Modelo em Cascata
- Modelo incremental
- Modelo evolucionário
- Modelo espiral
- Modelo de desenvolvimento concorrente
- Modelo especializado de processo
- Processo unificado

Todos esses modelos apresentam atividades comuns utilizadas durante o processo de desenvolvimento, tais como: especificação (levantamento de requisitos do software), desenvolvimento (projeto e codificação), validação (testes) e evolução (possível crescimento do software conforme as necessidades do cliente). Por terem atividades parecidas, o que diferencia um modelo de outro é a ênfase à determinada atividade.

3.1.3.1 Modelo Cascata

Para este trabalho, foi escolhido o Modelo em Cascata, conhecido também como *waterfall* ou modelo clássico, por ser o modelo que mais bem se aplica ao desenvolvimento de um software simples. Por ser um dos mais importantes, tornou-se referência para a criação de vários outros modelos, e servindo como base de muitos projetos modernos; este modelo preocupa-se com aquilo que conhecemos como engenharia progressiva de produto de software.

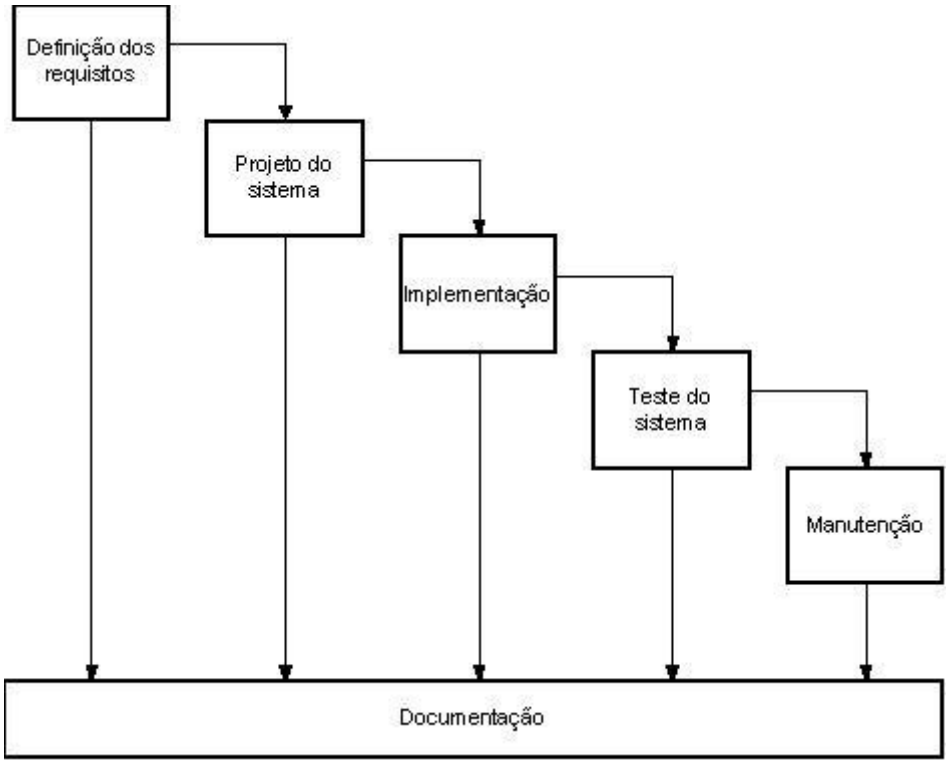

Figura 03: Modelo em Cascata (*Waterfall* ou Clássico)
Fonte: <http://www.api.adm.br/GRS/referencias/t1_g13.modeloCascata.pdf>. Acessado em: 07 nov. 2012.

A ideia deste modelo é que cada etapa siga a outra etapa em sequência. Dessa forma, cada atividade será iniciada assim que a atividade anterior estiver terminada (e aprovada pelo

cliente). Por esse motivo, novas ideias para o sistema não poderão ser aproveitadas, uma vez que utilizando esse modelo não pode voltar atrás para alterar as atividades da etapa anterior. Geralmente, nesse caso, as empresas criam versões com as novas ideias.

Foi utilizado o modelo cascata pelo fato da NDDigital S/A Software trabalhar com esse modelo para a criação das versões dos produtos. Dessa forma, o desenvolvimento do sistema passou por todas as etapas, as quais são definidas neste modelo.

3.1.4 Qualidade de Software

Qualidade pode ser vista como intuitiva, podendo variar de pessoa para pessoa e, por isso, conceituar qualidade pode ser uma tarefa difícil. Segundo o Dicionário Aurélio, qualidade define-se como: "propriedade, atributo ou condição das coisas ou das pessoas capaz de distingui-las das outras e de lhes determinar a natureza". Qualidade pode ser comparada com padrões conhecidos, tais como, tamanho, cor, propriedades elétricas, maleabilidade, etc. Entretanto, é muito mais difícil categorizarmos qualidade em software do que objetos físicos.

> Qualidade no software permite que fábricas de software criem produtos capazes de atender, da melhor forma, as necessidades dos seus usuários. (...) Estes conceitos modelam um software com características de qualidade satisfatórias aos seus usuários e devem ser aplicados e controlados em todas as fases de um método ou processo de desenvolvimento. (LOBO, 2009).

Atingir um alto nível de qualidade de produto ou serviço é o objetivo da maioria das organizações. Atualmente não é mais aceitável entregar produtos com baixa qualidade e reparar os problemas e as deficiências depois que os produtos foram entregues ao cliente. (SOMMERVILLE, 2003).

Para empresas de desenvolvimento de software, criar um sistema com qualidade é difícil, porém se feito da forma correta e respeitando os processos de desenvolvimento pode se tornar fácil, tal como extrair informações relevantes que vão proporcionar o surgimento de novos negócios e a sobrevivência do software no mercado, pois é inviável incorporar qualidade de software no produto final.

Segundo NOGUEIRA:

> Construindo o software com qualidade, cria-se a real possibilidade de extrair de um sistema, informações relevantes que venham não só para contribuir com a decisão, mas para ser um fator de excelência empresarial, permitindo novos negócios, permanência e sobrevivência num mercado atuante. Por tanto, é de suma importância identificar, analisar os riscos que ameaçam o sucesso do projeto bem

> como gerenciá-los para que se possa alcançar os objetivos empresariais. (NOGUEIRA, 2009)

"O controle de qualidade de software é um dos fatores mais importantes em uma fábrica de software" (LOBO, 2009). É feito através de uma série de inspeções, revisões e testes, usados através do ciclo de desenvolvimento para garantir que cada trabalho produzido está de acordo com sua especificação/requerimento. Portanto, o controle de qualidade é parte do processo de desenvolvimento e, como é um processo de *feedback*, ele é essencial para minimizar os defeitos produzidos.

Sendo assim, para um software ter qualidade é necessário implantar alguns conceitos que devem estar garantidos na modelagem do sistema. São eles:

- **Legabilidade**: o sistema tem que estar de acordo com as leis vigentes;
- **Funcionabilidade**: as funcionalidades do sistema devem atender as necessidades do cliente;
- **Segurança**: é importante manter a integridade e a restrição do sistema aos diversos usuários que irão utilizar;
- **Flexibilidade**: o sistema deve ser flexível em relação às várias tecnologias que surgem, dessa forma, o desenvolvimento se torna dinâmico, pois sempre está integrando novas tecnologias;
- **Suporte**: para não deixar o cliente "na mão", é de extrema importância para um software ter uma equipe de suporte treinada para "socorrer" o usuário;
- **Backup**: o sistema deve ter rotinas de backups automática e confiável para garantir a integridade dos dados, que são o maior patrimônio dos clientes;
- **Software intuitivo**: o sistema deve ter uma interface amigável e de fácil compreensão para atender as necessidades do cliente;
- **Praticidade**: está ligado diretamente na facilidade que usuário interage com o sistema; quanto mais fácil, mais benefício trará para o usuário;
- **Eficiência**: este conceito está ligado ao tempo de processamento das funcionalidades do sistema, e quanto recurso da máquina o sistema está utilizando para realizar as devidas tarefas;
- **Manutenibilidade**: segundo IEEE 610.12, manutenibilidade é a facilidade com a qual o sistema ou o componente podem ser modificados para corrigir falhas, melhorar desempenho ou outros atributos, ou adaptar a uma mudança

de ambiente[1]. Também é utilizado para expandir ou contrair o software para satisfazer a novos requisitos. "É muitas vezes quantificada em termos do tempo médio requerido para efetivar a revisão do software para eliminar um erro. Esse atributo é muito significativo para um software, na medida em que a etapa de manutenção pode consumir até 65% do custo total de um produto." (SCHACH, 1992);

- **Testabilidade**: examina as diferentes probabilidades e características comportamentais que levam o código a falhar se alguma coisa estiver incorreta. Segundo IEEE 610.12, testabilidade é o grau que o sistema ou componente facilitam o estabelecimento do critério de teste e o desempenho de teste para determinar se aqueles critérios foram encontrados[2].

3.2 Padrões de Projeto

O termo "padrão de projeto" surgiu na engenharia civil onde sua ideia consiste em descrever um problema para um ambiente e reutilizar a solução. Padrões de projetos também podem ser definidos como uma repetitiva solução para um problema comum. Geralmente, um padrão de projeto não é um design fechado que pode ser transformado diretamente em código. Ele é um modelo indicando como resolver um problema que pode ser usado em várias situações diferentes.

> Os padrões de projeto tornam mais fácil reutilizar projetos e arquiteturas bem-sucedidas. Expressar técnicas testadas e aprovadas as torna mais acessível para os desenvolvedores de novos sistemas. Os padrões de projeto ajudam a escolher alternativas de projetos que tornam um sistema reutilizável e a evitar alternativas que comprometem a reutilização. Os padrões de projeto podem melhorar a documentação e a manutenção de sistemas ao fornecer uma especificação explicita de interações de classes e objetos e o seu objetivo subjacente. Em suma, ajudam um projetista a obter mais rapidamente um projeto adequado. (GAMMA *et al*., 2000).

Segundo ALEXANDER (1977) "cada padrão é uma regra de três partes, que expressa uma relação entre certo contexto, um problema e uma solução". Seguindo a mesma linha de raciocínio, temos um problema, uma solução e um contexto, porém, de acordo com GAMMA *et al*. (2000), existem quatro elementos essenciais em um padrão de projeto. Com isso, para

[1] The ease with which a software system or component can be modified to correct faults, improve performance or other attributes, or adapt to a changed environment.

[2] The degree to which a system or component facilitates the establishment of test criteria and the performance of tests to determine whether those criteria have been met.

entendermos padrão de projeto, teremos que estudar as suas partes: o nome do padrão, o problema, a solução e as consequências desse padrão.

GAMMA *et al*. (2000) dizem que o nome do projeto é uma referência que podemos usar para descrever um projeto de projeto, suas soluções e consequências em uma ou duas palavras. Para o GoF[3] encontrar o nome para os padrões foi uma das partes mais difíceis para o catalogo. O problema descreve em que situação aplicar o padrão, explicando assim o problema em si e o seu contexto. Em alguns padrões poderá estar incluso uma lista de condições específicas que devem ser satisfeitas para fazer sentido incluírem o padrão no desenvolvimento do software. Quanto à solução, GAMMA *et al*. (2000) dizem que ela descreve os elementos que compõe o padrão de projeto, seus relacionamentos, suas responsabilidades e colaborações. A solução em si não é concreta, pois ela pode ser aplicada em diferentes situações. Por esse motivo, as soluções dos padrões de projetos são descritas de uma forma abstrata de um problema e de como os elementos vão resolver isso. As consequências estão relacionadas aos resultados e análises das vantagens e desvantagens da aplicação do padrão.

"Um padrão de projeto nomeia, abstrai e identifica os aspectos-chave de uma estrutura de projeto comum para torná-la útil para a criação de um projeto orientado a objetos reutilizável" (GAMMA *et al*., 2000). É importante ressaltar que cada padrão de projeto focaliza um problema ou tópico particular do projeto orientado a objeto; também descreve em quais situações ele pode ser aplicado, se pode ser aplicado em função de outras restrições e as consequências da aplicação do padrão.

> Os padrões de projeto ajudam a evitar esses problemas ao garantirem que o sistema possa mudar segundo maneiras específicas. Cada padrão de projeto permite a algum aspecto da estrutura do sistema variar independentemente de outros aspectos, desta forma tornando um sistema mais robusto em relação a um tipo particular de mudança (GAMMA *et al*., 2000).

Utilizar padrões de projeto no desenvolvimento do software faz com que o sistema se torne simples e coerente, fácil de utilizar, entender e ao mesmo tempo preparado para manutenções e melhorias; esses padrões são feitos para serem reutilizados em várias partes do código que poderiam acontecer tal problema solucionado.

[3] *GoF*, ou *Gang of Four*, são os quatro autores do livro ***Design Patterns: Elements of Reusable Object-Oriented Software*** (Padrões de Projetos: Soluções Reutilizáveis de Software Orientado a Objetos) que descreveram 23 padrões de projetos e hoje são fortemente conhecidos na comunidade de desenvolvedores.

> Classes que são fortemente acopladas são difíceis de reutilizar isoladamente, uma vez que dependem umas das outras. O acoplamento forte leva a sistemas monolíticos, nos quais você não pode mudar ou remover uma classe sem compreender e mudar muitas outras classes.
> O sistema torna-se uma massa densa difícil de aprender, portar e manter. Um acoplamento fraco aumenta a probabilidade de que uma classe possa ser usada por si mesma e de que um sistema possa ser aprendido, portanto, modificado e estendido mais facilmente. Os padrões de projeto usam técnicas como acoplamento abstrato e projeto em camadas para obter sistemas fracamente acoplados. (GAMMA *et al*., 2000).

Os padrões de projetos são organizados em grupos distintos que visam auxiliar no seu estudo, entendimento e na descoberta de novos. Eles são classificados nos grupos por dois critérios. O primeiro é a finalidade, que define o que o padrão faz. O padrão pode ter finalidade criacional, estrutural ou comportamental. O segundo critério é escopo. Ele especifica se o padrão se aplica a classe ou a objetos. A Figura 04 apresenta os padrões segundo suas classificações.

Classificação dos padrões

		Propósito		
		Criacional	Estrutural	Comportamental
Escopo	Classe	*Factory Method*	*Adapter (classe)*	*Interpreter* *Template Method*
	Objeto	*Abstract Factory* *Builder* *Prototype* *Singleton*	*Adapter (object)* *Bridge* *Composite* *Decorator* *Façade* *Flyweight* *Proxy*	*Chain of Responsability* *Command* *Iterator* *Mediator* *Memento* *Observer* *State* *Strategy* *Visitor*

Figura 04: Classificação dos padrões de projeto.
Fonte: GAMMA *et al*., 2000, p. 26.

Os padrões de criação têm como principal característica abstrair o processo de instanciação, colaborando para um sistema independente de como seus objetos são criados, compostos e representados. Um padrão de criação de classe usa a herança para variar a classe que é instanciada, enquanto que um padrão de criação de objeto delegará a instanciação para

outro objeto. Esses padrões vão se tornando importantes à medida que o sistema vai evoluindo por dependerem mais na composição de objetos do que a herança de classes.

Encapsular a forma que um objeto é instanciado fornece uma grande ferramenta para flexibilizar o sistema. Existem requisitos que exigem que um objeto seja instanciado apenas uma vez. Por exemplo, o pool que controla as conexões de uma aplicação com o banco de dados, o objeto que representa a sessão do usuário, ou mesmo a criação de um objeto de forma dinâmica dependendo apenas das características que deva ter naquele momento.

Para este trabalho foi utilizado o padrão *Singleton*, cuja intenção é ter apenas uma instância de uma classe e ter um ponto global de acesso a ela.

> Como garantimos que uma classe tenha somente uma instância e que essa instância seja facilmente acessível? Uma variável global torna um objeto acessível, mas não impede você de instanciar múltiplos objetos. Uma solução melhor seria tornar a própria classe responsável por manter o controle da sua única instância. A classe pode garantir que nenhuma outra instância seja criada (pela interceptação das solicitações para criação de novos objetos), bem como pode fornecer um meio para acessar sua única instância. (GAMMA *et al*, 2000).

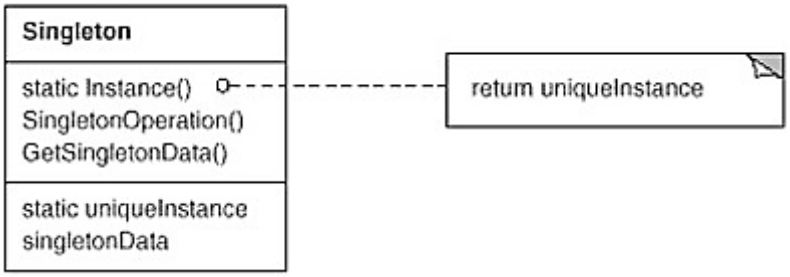

Figura 05: Padrão *Singleton*
Fonte: GAMMA *et al*., 2000, p. 130.

Dessa forma, quem precisar acessar a instância *singleton* só terá acesso através do objeto chamado *Instance*. Segue abaixo um trecho do código onde foi utilizado o padrão.

```
public class NHibernateServiceConfigurator
{
    private static NHibernateServiceConfigurator _instance;

    public static NHibernateServiceConfigurator Instance
    {
       get { return _instance ?? (_instance = new NHibernateServiceConfigurator()); }
    }

    private NHibernateServiceConfigurator() { }
}
```

Quadro 02: Exemplo do padrão *Singleton*.

Fonte: O próprio autor.

É necessário ter uma classe pública, onde será instanciada a propriedade chamada *Instance* do tipo da classe atual. Essa propriedade *Instance* terá que verificar se o objeto está vazio para criar o objeto, caso contrário só retorna o objeto que foi instanciado anteriormente.

"Os padrões estruturais se preocupam com a forma como classes e objetos são compostos para formar estruturas maiores" (GAMMA *et al.*, 2000). Geralmente, as classes utilizam a herança para as suas implementações, e os objetos descrevem outras maneiras para compor novas funcionalidades. Esse padrão é útil para fazer bibliotecas de classes ou, até mesmo, sistemas inteiros desenvolvidos independentemente para trabalharem juntos.

"Os padrões comportamentais se preocupam com algoritmos e a atribuição de responsabilidades entre objetos. Os padrões comportamentais não descrevem apenas padrões de objetos u casses, mas também os padrões de comunicação entre eles." (GAMMA *et al*., 2000). Dessa forma, as classes utilizam a herança para distribuir o comportamento entre classes, e os objetos utilizam a composição de objetos em contrapartida a herança.

Com isso, fazer uso dos padrões de projetos durante o desenvolvimento de software ajuda a construir um software flexível e com documentações reaproveitáveis, pois com eles é possível identificar pontos comuns entre duas soluções diferentes para um mesmo problema, e propicia o desenvolvimento de soluções cada vez melhores e mais eficientes, podendo ser reutilizadas, e auxiliando no avanço do conhecimento humano.

3.3 UML

A UML (*Unified Modeling Language*) "é uma linguagem que utiliza uma notação padrão para especificar, construir, visualizar e documentar sistemas de informação orientados por objetos" (NUNES, 2004).

Essa linguagem disponibiliza uma forma padrão de modelagem de projetos de sistemas, incluindo seus aspectos conceituais, tais como processos de negócios e funções do sistema, além de itens concretos como as classes escritas em determinada linguagem de programação, processos de banco de dados e componentes de software reutilizáveis.

A UML é utilizada pelas fases de análise de requisitos, análise de projeto e oferece a elas cinco tipos de visões que destacam os diferentes aspectos do sistema que está sendo modelado. São elas: visão use-case, visão lógica, visão de componentes, visão de concorrência e visão de organização.

Segundo SILVA (2001), a UML providencia as seguintes particularidades principais:

> - Semântica e notação para tratar um grande número de tópicos atuais de modelação;
> - Semântica para tratar tópicos de modelação futura, relacionados em particular com a tecnologia de componentes, computação distribuída, frameworks e Internet;
> - Mecanismos de extensão de modo a permitir que futuras aproximações e notações de modelação possam continuar a ser desenvolvidas sobre o UML;
> - Semântica e sintaxe para facilitar a troca de modelos entre ferramentas distintas.

Assim como toda linguagem, a UML também é utilizada como base de comunicação. Ela expressa conhecimento em relação ao assunto que está sendo tratado, sendo que este assunto é o software que será desenvolvido. Geralmente, ela é utilizada na modelagem do sistema, é focada justamente no entendimento do assunto na forma de modelos e formas. Também é utilizada para conceber novas ideias durante o planejamento do software. Por ser uma linguagem de modelagem, ela pode ser utilizada para diversos tipos de sistemas, assim como métodos e processos.

Segundo NUNES (2004):

> Pelo fato de utilizar um conjunto de símbolos padrão, a UML funciona como um meio de comunicação entre os diversos elementos envolvidos no processo, utilizadores, gestores e equipes de desenvolvimento. A linguagem pode ser utilizada para documentar o sistema ao longo de todo o ciclo de desenvolvimento, começando com a tarefa inicial de análise dos processos de negócio da organização e prolongando-se até à tarefa de manutenção evolutiva do sistema informático. (NUNES, 2004).

A linguagem tornou-se uma norma industrial no desenvolvimento de softwares orientados a objetos em 1997. Atualmente, é fortemente utilizada nas equipes de desenvolvimento, muitas vezes para explicar processos e descrições mais específicas de atividades sem a necessidade de código-fonte. Além disso, existem nove tipos de diagramas utilizados para descrever o conteúdo em visões diferentes, e são utilizados em combinação para prover todas as visões do sistema. Esses diagramas são: diagrama de caso de uso, diagrama de classe, diagrama de objeto, diagrama de estado, diagrama de sequência, diagrama de colaboração, diagrama de atividade, diagrama de componente e diagrama de execução.

4 WINDOWS PRESENTATION FOUNDATION

"O *Windows Presentation Foundation* (*WPF*) é uma tecnologia da Microsoft para o desenvolvimento de aplicativos e interfaces, acrescentando novos recursos ao .NET Framework[4]" (SONNINO, 2006). O *WPF* veio para mudar um pouco a tecnologia e os componentes de hardware que as máquinas estavam utilizando. Antigamente, quanto melhor a sua placa de vídeo, melhor qualidade suas imagens teriam, e o mesmo aconteciam com os softwares, pois os mesmos utilizavam mais recurso e exigiam mais dos computadores. Atualmente, as aplicações não estão preparadas para altas resoluções de tela, pois são baseadas em pixels. Entretanto, utilizando o *WPF* esse problema é resolvido, pois os aplicativos desenvolvidos são tratados de forma vetorial, permitindo diferentes resoluções e diferentes posições dos objetos, que endereça o pixel do início ao fim do objeto, independente da resolução do monitor.

Dessa mesma forma, como os pixels e a maneira como os objetos são dispostos na tela evoluíram, o usuário também evoluiu, pois houve a necessidade de interfaces mais trabalhadas e mais agradáveis aos clientes; com o *WPF*, é possível desenvolver aplicações com interfaces ricas. A diferença entre o desenvolvimento *Windows Forms* e *WPF* é que o código fonte é separado da interface, permitindo certa independência entre essas duas camadas.

Essa tecnologia foi construída visando algumas funções nativas para permitir ao desenvolvedor fazer a criação de formulários sofisticados, contendo recursos 2D e 3D, animações, vídeos, layouts, documentos, gráficos, segurança, e um poderoso e flexível mecanismo de acesso a dados. Isso faz com que o *WPF* seja conhecido pelo seu novo jeito de desenvolver aplicações sofisticadas, modernas, graficamente ricas, flexíveis e seguras para o usuário.

Segundo SONNINO (2006) as principais características do *WPF* são:

- Flexibilidade da interface, que pode ser independente do código: podemos ter duas apresentações completamente diferentes compartilhando o mesmo comportamento;

[4] .NET Framework é um componente essencial do Windows que visa uma plataforma única para desenvolvimento e execução de aplicações, onde qualquer código gerado pode ser executado em qualquer dispositivo que possua o .NET framework instalado (MICROSOFT CORPORATION, 2008).

- Incorpora todas as funções do .NET 2.0, acrescentando às interfaces novos recursos como 3D, animações, gráficos vetoriais, reconhecimento de voz, layouts avançados, entre outros;
- Leva para o desktop o conceito já existente na Web de separação entre o design e o código, permitindo que a interface seja criada por um designer e o código por um programador especializado, de maneira independente;
- Usa os recursos do sistema operacional, de maneira a otimizar a performance da interface para o hardware do usuário;
- Os controles podem ser personalizados a qualquer nível que se queira, por exemplo, podemos criar um botão não retangular que contém uma animação 3D, sem a necessidade de escrever código para isso;
- É independente de plataforma; o mesmo código-fonte funciona tanto na web quanto para desktop e, no futuro poderá ser distribuído até em sistemas como Mac, Linux e celulares, usando o WPF/E (WPF *Everywhere*, que está em desenvolvimento).

Como a intenção desta tecnologia é separar a interface do código-fonte, também foi separada a forma de como é desenvolvido. A interface é especificada através de um *XML* (abreviação para *Extended Markup Language*), que porventura possui algumas características especiais para o desenvolvimento, chamado de *XAML* (abreviação para *Extensible Application Markup Language*, pronunciada como "zémmel"), onde cada entidade é conhecida como elemento e definida por um par de *tags* de abertura e fechamento (por exemplo, `<TextBox></TextBox>`).

A utilização de uma linguagem declarativa para a definição de interfaces gráficas apresenta uma série de vantagens, tais como:

- Código legível e de fácil compreensão;
- Hierarquia entre os controles facilmente perceptível;
- Código da interface separada da lógica da aplicação; e
- Linguagens de marcação costumam ser mais facilmente processadas e interpretadas por ferramentas de desenvolvimento.

"A utilização de uma linguagem declarativa pode ainda diminuir a quantidade de código escrito pelo desenvolvedor, possibilitando mais tempo para a análise e desenvolvimento da lógica da aplicação" (MacVITTIE, 2006). A Figura 06 apresenta o código fonte escrito em C# criando uma janela e adicionando um controle a ela.

```
System.Windows.Forms.Form mainWindows = new System.Windows.Forms.Form();
System.Windows.Forms.TextBox txtElement = new System.Windows.Forms.TextBox();
txtElement.Text = "Hello World!";
txtElement.ForeColor = Color.DarkRed;
txtElement.Font = new System.Drawing.Font("Microsoft Sans Serif", 14);
mainWindows.Controls.Add(txtElement);
mainWindow.Show();
```

Figura 06: Código fonte na linguagem C# para exibir uma janela *Windows*
Fonte: O próprio autor.

A mesma janela pode ser definida utilizando-se a linguagem *XAML*, como apresentado na Figura 07.

```
<Window xmlns="http://schemas.microsoft.com/winfx/avalon/2005" >
  <TextBox Foreground="DarkRed" FontSize="14" >
    Hello World!
  </TextBox>
</Window>
```

Figura 07: Código fonte na linguagem XAML para exibir uma janela *Windows*
Fonte: O próprio autor.

Para utilizar a linguagem XAML, é necessário apenas definir os elementos da interface, e não será necessário conter nenhum tipo de lógica da aplicação, pois será tratado no código fonte que pode ser escrito utilizando a plataforma Microsoft (C# ou VB.net).

Desta forma, a interface não tem o conhecimento da lógica utilizada, e no código fonte só deve conter informações referentes a validações desses componentes, ou seja, o desenvolvedor não precisa conhecer os detalhes da interface para programar. Ambas as camadas desconhecem uma a outra e estão separadas em arquivos diferentes. Isso faz com que o código fique mais simples e legível.

4.1 Padrão MVVM

Para o NDDigital.ReleaseLog foi adotado o padrão de arquitetura conhecido como MVVM, ou *Model-View-ViewModel*. Este padrão de arquitetura é voltado para aplicações cujo interesse é separar a camada de apresentação da camada de negócio. Além disso, ele é derivado dos padrões MVP (*Model-View-Presenter*) e MVC (*Model-View-Controller*), e este

padrão é exclusivo para aplicações em WPF ou Siverlight[5], pelo fato de utilizarem XAML na construção da interface.

Diferente dos padrões citados acima, as Views no padrão MVVM possuem inteligência e vida própria, pois apesar de ter seus estados manipulados e definidos pela ViewModel, ela é capaz de manipular as propriedades da ViewModel através de *bindings*[6]. Por isso o padrão MVVM (Model-View-ViewModel) é considerado o padrão ideal para aplicações WPF.

Como esse padrão divide o código em três camadas, o desenvolvimento passou a ser mais estruturado e cada camada passou a ter a própria responsabilidade.

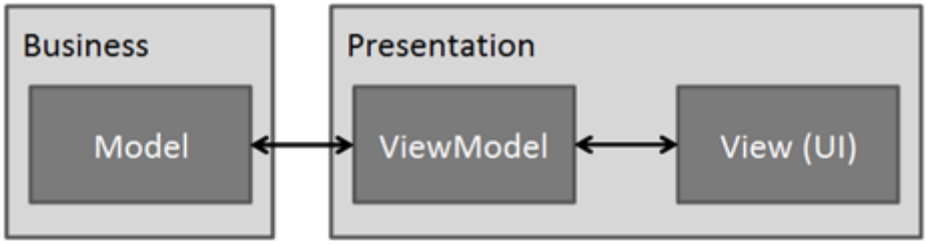

Figura 08: Estrutura MVVM.
Fonte: <http://elemarjr.net/2011/03/08/model-view-viewmodel-mvvm/>. Acessado em: 18 set. 2012.

Sendo assim, as suas camadas são conhecidas como:

- **View:** é composta por classes que representam a interface gráfica exibida ao usuário. Essas classes possuem alguns controles visuais que permitem alterar o estado dos objetos através de algum dispositivo de entrada (como mouse ou teclado). Além disso, podem conter temas, animações, fotos, vídeos e outras funcionalidades interativas com o propósito de apresentação visual. As Views contam com os *bindings*, conectando um componente a uma propriedade do ViewModel. Esses *bindings* só são ativados quando a propriedade *DataContext* da View é preenchida com a classe ViewModel correspondente àquela View. O

[5] Silverlight também é uma tecnologia da Microsoft para desenvolvimento de aplicações web, desktop e para smartphones (online ou off-line).

[6] *Bindings*, também conhecido como *Data Bindings*, permite que as propriedades de uma classe conectem-se diretamente com componentes da interface que trarão dados, independente da forma de entrada, sejam elas arquivos XML ou através de banco de dados. Essa ligação é feita sem que seja necessária a criação de código e pode ser unidirecional (apenas a propriedade da classe é alterada) ou bidirecional (a alteração afeta tanto a propriedade quanto o banco de dados). Como o *binding* é flexível, você pode conectar qualquer componente com qualquer propriedade, o que será necessário é tratar os dados no ViewModel. "Um exemplo de Data Binding pode ser a ligação da propriedade Text de uma TextBox a um campo de uma tabela de um banco de dados. O conteúdo da TextBox é inicializado com o conteúdo do campo e, quando alteramos o valor da TextBox, esta alteração é propagada para o banco de dados. Todo este processo é gerenciado internamente pelo WPF, sem necessidade de criação de código específico para isso." (SONNINO, 2006).

ideal das Views é que no *codebehind*[7] contenha apenas a inicialização do *DataContext*;

- **Model:** é composta pela camada de negócios da aplicação com toda a lógica de programação e acesso ao banco de dados. O proposito do Model é representar os dados, não tendo conhecimento de onde eles serão apresentados para um usuário, nem mesmo se eles serão mostrados. A sua única responsabilidade é a representação;
- **ViewModel:** essa é a camada intermediária entre a View e o Model, responsável por transportados os dados de uma camada à outra. Além disso, é responsável pela lógica de validação e comportamento da interface ou com os use-cases da aplicação; o ViewModel não é o *codebehind* da View. Assim como o Model, o ViewModel desconhece como será a interface. Para maximizar a reutilização deste ViewModel, é importante que ele não referencie nenhuma classes de apresentação/interface, elementos, controles ou comportamentos. Pode-se dizer que a ViewModel é a principal camada da aplicação, pois é ela que vai coordenar as iterações da View com o Model, já que uma camada desconhece a outra. Na maioria das vezes, as ViewsModels utilizam eventos de notificações de mudança de estados, através das interfaces: *INotifyPropertyChanged*, *INotifyCollectionChanged*, para facilitar o preenchimento dos dados da View. Essa câmara é fortemente testável, independentemente da View ou do Model.

Utilizando esse padrão de desenvolvimento, fica muito mais fácil a realização de testes unitários sem precisar da interface, pois não importa como será a interface, o importante é que o sistema funcione e realize os processos corretamente. Esses testes unitário são aplicados diretamente no ViewModel, onde possui acesso a camada do Model, podem também podem ser realizados testes unitário na camada do Model para verificar se os diversos repositórios estão trazendo as informações que deveriam trazer.

[7] *Codebehind* é o termo utilizado para descrever o código associado a View. Utilizando o padrão MVVM, geralmente as Views não possuem codebehind deixando a manipulação da interface para o ViewModel.

4.2 Dependency Injection

Dependency Injection, ou Injeção de Dependências, é um padrão utilizado quando a aplicação necessita de um baixo acoplamento entre os diferentes módulos do sistema. Esse acoplamento é feito através das referências que o projeto tem com outro projeto. Quanto maior for o acoplamento entre os módulos, mais trabalhoso e difícil é a manutenção do software.

Por este motivo, foi desenvolvido um padrão onde tem como principio garantir o baixo acoplamento entre os projetos, visando alguns benefícios como:

- Oferecer reusabilidade de componentes, uma vez que criamos componentes interdependentes, eles podem ser facilmente implementados em sistemas diversos;
- Facilitar a manutenção de sistemas, fazendo com que as manutenções em módulos não afetem o restante do sistema;
- Criar códigos altamente “testáveis”. Uma vez que temos códigos implementados seguindo esse padrão, podemos testá-los mais facilmente utilizando os “mock tests”;
- Criar códigos mais legíveis, o que torna mais fácil a compreensão do sistema como um todo.

No desenvolvimento de uma aplicação, temos que nos certificar que ela vai ser flexível para novas funcionalidades e que ela tenha facilidade nas manutenções, e para conseguirmos fazer esse trabalho corretamente, temos que deixar apenas as dependências necessárias para os objetos realizarem o seu trabalho.

Por esse motivo, temos o padrão de injeção de dependências que ajuda no desenvolvimento, minimizando as referências entre os projetos. Existem quatro tipos de maneiras de implementar injeção de dependências, são elas:

- **Construtor**: implementação da injeção de dependência na definição dos construtores das classes;
- **Getter and Setter**: modo em que implementamos a injeção de dependência na definição dos gets e sets das classes;

- **Interface Implementation**: modo em que se usa a definição de interfaces para realizar a injeção de dependência;
- **Service Locator**: modo em que construímos classes que servem como "localizadoras" de objetos que iremos instanciar em nossas outras classes.

4.2.1 Microsoft Unity

Segundo GAROFALO (2011), Microsoft Unity[8] é uma extensão do padrão de Injeção de Dependências com o qual é possível aplicar injeção de dependência no código usando tanto linguagem de marcação (XML) ou a plataforma .NET (C# ou VB.net). O Unity é conhecido por injetar o código nos construtores, propriedades e métodos.

O Unity possui algumas vantagens no desenvolvimento de aplicações, como por exemplo, simplifica a criação de objetos, especialmente estrutura de objetos hierárquicos e dependentes, a qual simplifica o código da aplicação; suporta adicionar dependências em tempo de execução ou por configuração simplificando o gerenciamento de dependências cruzadas; é compatível com o Service Locator Injection; facilita a arquitetura e o desenvolvimento para implementar facilmente os mais comuns padrões de projetos encontrados em aplicações modernas.

Além disso, Unity pode instanciar várias classes que possuem o construtor público sem precisar mapear aquele tipo em um container, é necessário apenas chamar o método Resolve e especificar a instância padrão daquele tipo que não está registrado, e o container simplesmente chama o construtor daquele tipo e o retorna como resultado.

4.3 Fluent NHibernate

O NHibernate é a versão do framework de mapeamento objeto relacional Hibernate para a plataforma .NET. Ela fornece serviços de mapeamento objeto-relacional, transformando dados de um objeto em uma linha de uma tabela no banco de dados relacional, ou de forma inversa, transformando uma linha da tabela em um objeto para aplicação orientada a objetos. Isso acontece por causa de um mapeamento que deve ser feito no formato XML que é conhecido como HBM, e cada classe tem um HBM XML correspondente.

[8] Microsoft Unity já está adicionado nos frameworks da Microsoft, porém pode também ser feito o download através do endereço: *http://unity.codeplex.com*.

Porém, como esse XML de mapeamento não é avaliado na compilação, é possível alterar o nome de propriedades ou até mesmo de classes e elas não são atualizadas no HBM XML ou no banco de dados. Com isso, foi desenvolvido uma tecnologia que melhora a utilização do NHibernate, chamada de Fluent NHibernate, onde é necessário apenas ter uma configuração convencional para que a aplicação esteja conectando com o banco de dados. Caso alguma propriedade seja alterada, ela será automaticamente renomeada no DB. Para o NDDigital.ReleaseLog foi utilizado o banco de dados conhecido como SQLite[9].

[9] SQLite é um banco de dados que não precisa de um SGBD. Geralmente, ele é recomendado para software que prezam a simplicidade, implementação e manutenção, do que os recursos do próprio gerenciador de banco de dados.

5 ESTUDO DE CASO

5.1 Cenário Atual

A NDDigital S/A Software é uma empresa de desenvolvimento de software e foi fundada em 2003. Reconhecida pela sua inovação, qualidade, integração e serviços de suporte, a empresa tem como objetivo avançar tecnologicamente o conhecimento dos colaboradores para melhorar a eficiência da organização e dos produtos.

Atualmente, a empresa opera no Brasil, América Latina, Estados Unidos da América e Europa; é conhecida como a maior fabricante de software focada em impressão na América Latina.

A empresa possui três verticais. São elas:

- **Impressão:** focada em soluções flexíveis para a gestão de impressão, podendo controlar as impressões, os drives, cotas, links, direcionamento de impressão, e-mails e a não-impressão para haver a redução dos custos das empresas;
- **Documentos Eletrônicos:** as soluções de documentos eletrônicos são focadas nos processos de controle das notas fiscais eletrônica, conhecimento de transporte eletrônico, carta frete eletrônica, nota fiscal de serviço eletrônico;
- **HANT (Engenharias):** trata-se de *e-Procurement* e *Supplies Chain* para construções pesadas e para compra de serviços para transportadoras, reduzindo o custo de compra.

5.2 Problemática

Nas equipes de desenvolvimento da empresa NDDigital S/A é utilizado um sistema, conhecido como Microsoft Team Foundation Server, para registrar e gerenciar as tarefas e os *bugs* das versões dos produtos. Com isso, para ter um controle mais preciso, foi desenvolvido um sistema para controlar e separar os ids de cada versão. Porém, conforme as versões foram surgindo, o software foi perdendo a qualidade e muitas vezes deixando confuso se as versões já haviam sido liberadas, ou em qual versão os ids deveriam ser adicionados.

O sistema atual possui uma interface simples. A Figura 09 mostra a interface da ferramenta que está sendo utilizada atualmente.

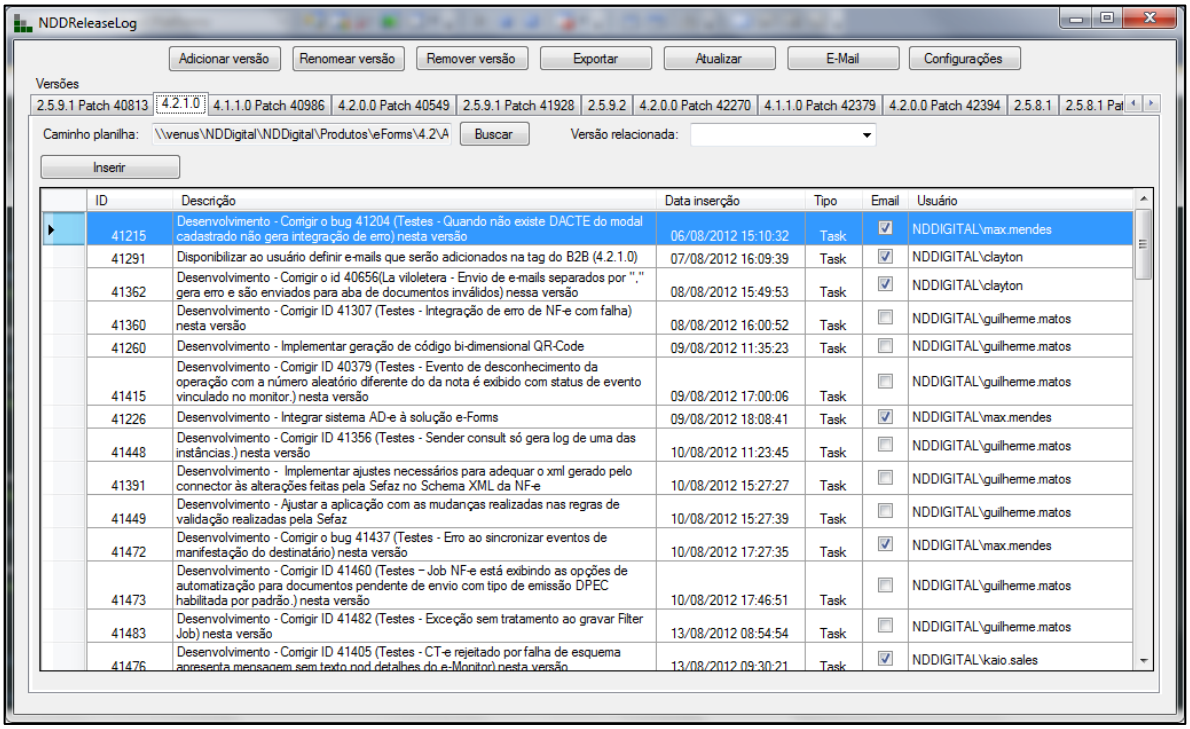

Figura 09: Atual NDDigital.ReleaseLog.
Fonte: O próprio autor.

A fim de solucionar os problemas, foi elaborada uma proposta para melhorar o sistema e adicionar todas as necessidades que não existiam no sistema antigo.

5.3 Visão Geral do Sistema

O novo sistema será responsável por gerenciar as tarefas e *bugs* das versões e dos patches de uma das equipes de desenvolvimento da NDDigital S/A Software, como já era feito na versão anterior. Esse sistema precisa exportar a lista de ids para um arquivo em Excel, e isso só irá acontecer após a versão estar finalizada. Além disso, ele enviará um e-mail para a pessoa responsável de gerar os *builds* da solução contendo o id, o projeto que foi alterado e o(s) módulo(s) que deve(m) ser compilado(s).

Terá uma interface mais simples e limpa, utilizando o padrão Metro UI[10], e por esse motivo, só irá mostrar o que é necessário para o usuário, diferente da versão anterior que mostrava todas as versões e todos os patches e era difícil controlar o arquivo de exportação;

[10] Metro UI é o codinome da nova interface utilizada no Windows 8 pela Microsoft Corporation.

além de ser lento e não ser intuitivo para o usuário. Essa interface foi feita através da ferramenta conhecida como Microsoft Expression Blend 4[11].

O sistema será inteligente o suficiente para criar patches quando for adicionar um id em uma versão que já foi liberada e não houver nenhum patch aberto. E para isso acontecer, foram utilizadas as tecnologias apresentadas anteriormente, para criar um software bem construído, simples e que possa sofrer alterações futuramente sem precisar de um tempo muito grande de desenvolvimento, e sem afetar outras partes do sistema.

[11] Microsoft Expression Blend 4 é um aplicativo multimídia utilizado para sistemas criados em WPF ou Silverlight, e também pode ser utilizado para criar sistemas web, jogos e aplicativos para desktops. O Blend é utilizado para adicionar comportamentos, alterar interface, criar efeitos e mais, no seu projeto. Além do mais, existe a interação do Visual Studio com o Blend 4 para uma melhor experiência para o usuário.

5.4 Diagrama de Entidade Relacionamento

O Diagrama de Entidade Relacionamento é um modelo que descreve os dados do sistema com alto nível de abstração, e tem como objetivo principal mostrar o relacionamento entre as tabelas do sistema.

A Figura 10 mostra o diagrama de Entidade Relacionamento do software. Como o banco de dados é em SQLite, o diagrama foi feito através da ferramenta SQLite Maestro.

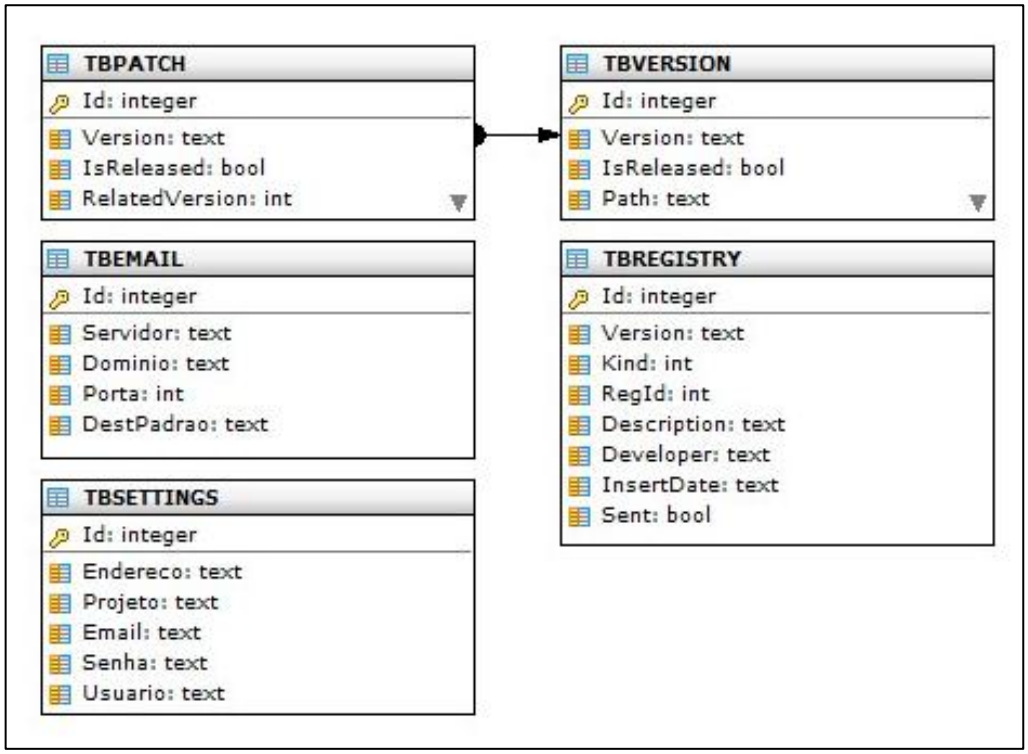

Figura 10: Diagrama de Entidade Relacionamento
Fonte: O próprio autor.

5.5 Diagrama de Classe

Segundo FURLAN (1998) trata-se de uma estrutura lógica estática em uma superfície de duas dimensões mostrando uma coleção de elementos declarativos de modelo como classes, tipos e seus respectivos conteúdos e relações.

A Figura 11 apresenta o diagrama de Classe que feito através da ferramenta Microsoft Visual Studio Professional 2010:

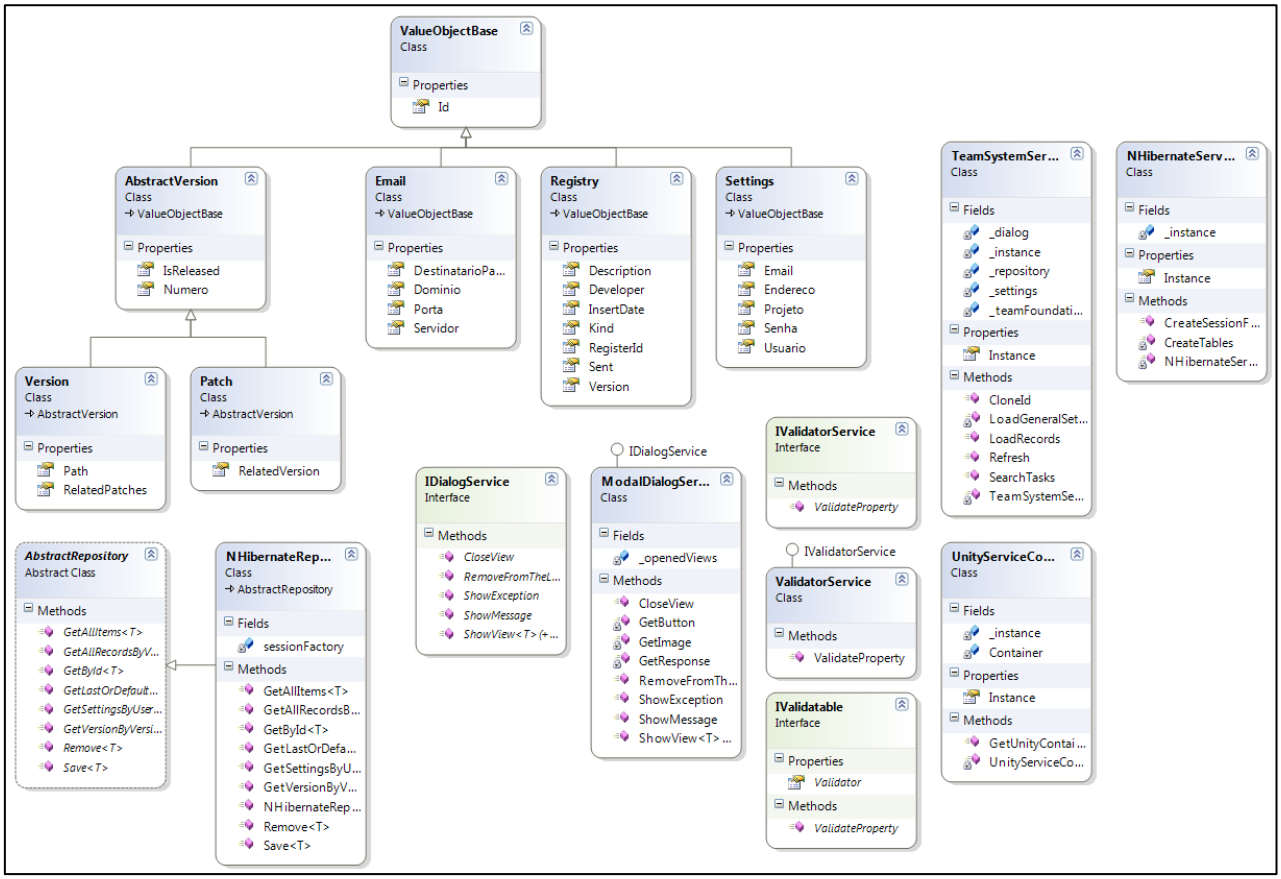

Figura 11: Diagrama de Classe.
Fonte: O próprio autor.

5.6 Diagrama de Atividade

Na UML, um Diagrama de Atividade é utilizado para apresentar uma sequência de atividades, além de mostrar o fluxo de trabalho de um ponto inicial até um ponto final detalhando várias decisões que existem durante essa atividade.

A Figura 12 apresenta o diagrama de Atividades que foi feito através da ferramenta Enterprise Architect 8:

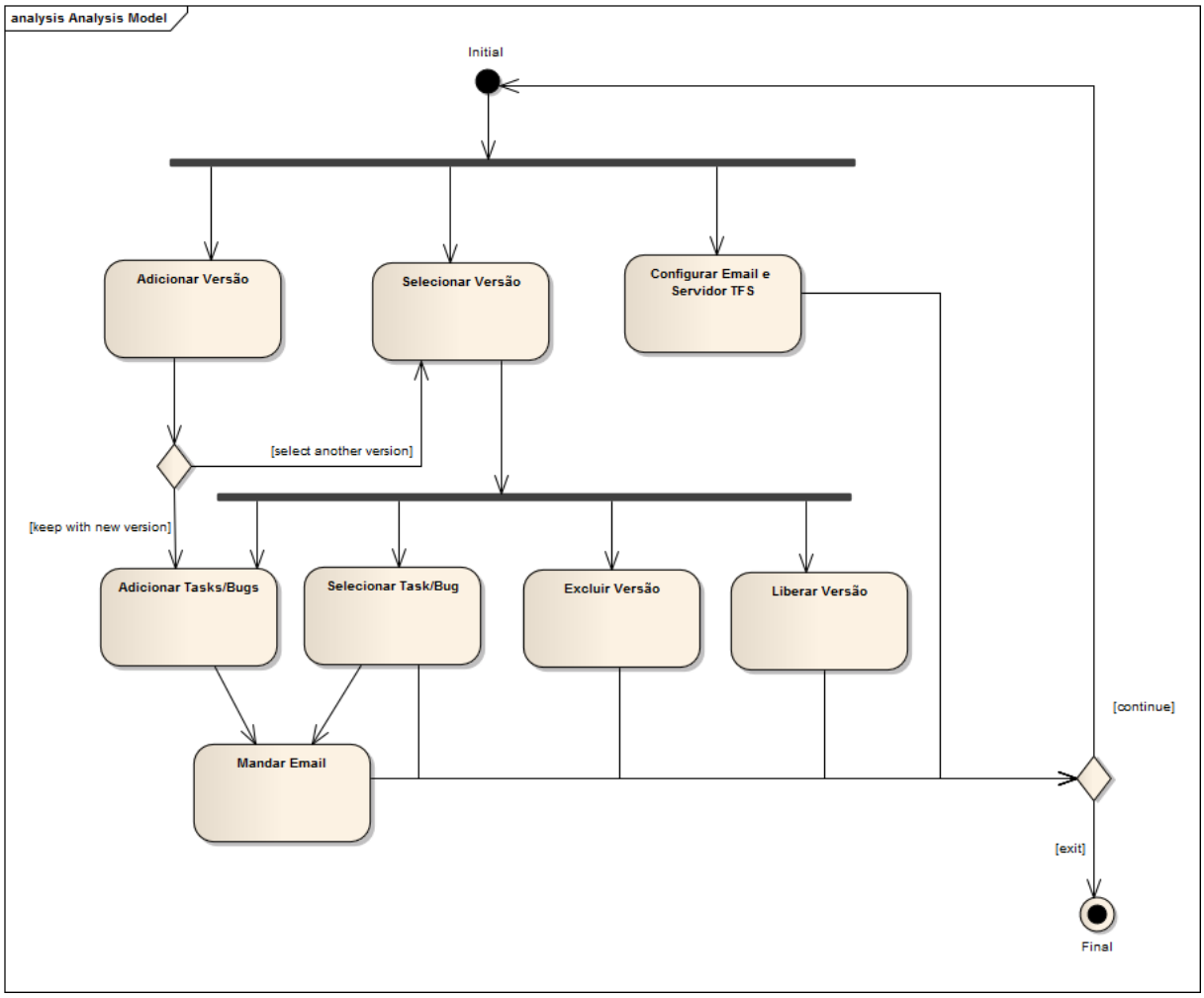

Figura 12: Diagrama de Atividades.
Fonte: O próprio autor.

5.7 Telas do Sistema

A Figura 13 mostra a tela inicial do sistema. Como foi explicado anteriormente, o sistema segue o padrão de interface Metro UI, onde preza a simplicidade e a facilidade das opções. Nessa tela, o usuário poderá adicionar uma nova versão para ter o controle dos ids, poderá configurar o servidor de e-mail, configurar o Team Foundation Server, pode exportar os itens que estão na tabela para um arquivo Excel, pode liberar a versão para não poder mais adicionar Ids nela ou pode pesquisar novos Ids e adicioná-los na versão. Cada versão adicionada poderá ter um patch relacionado.

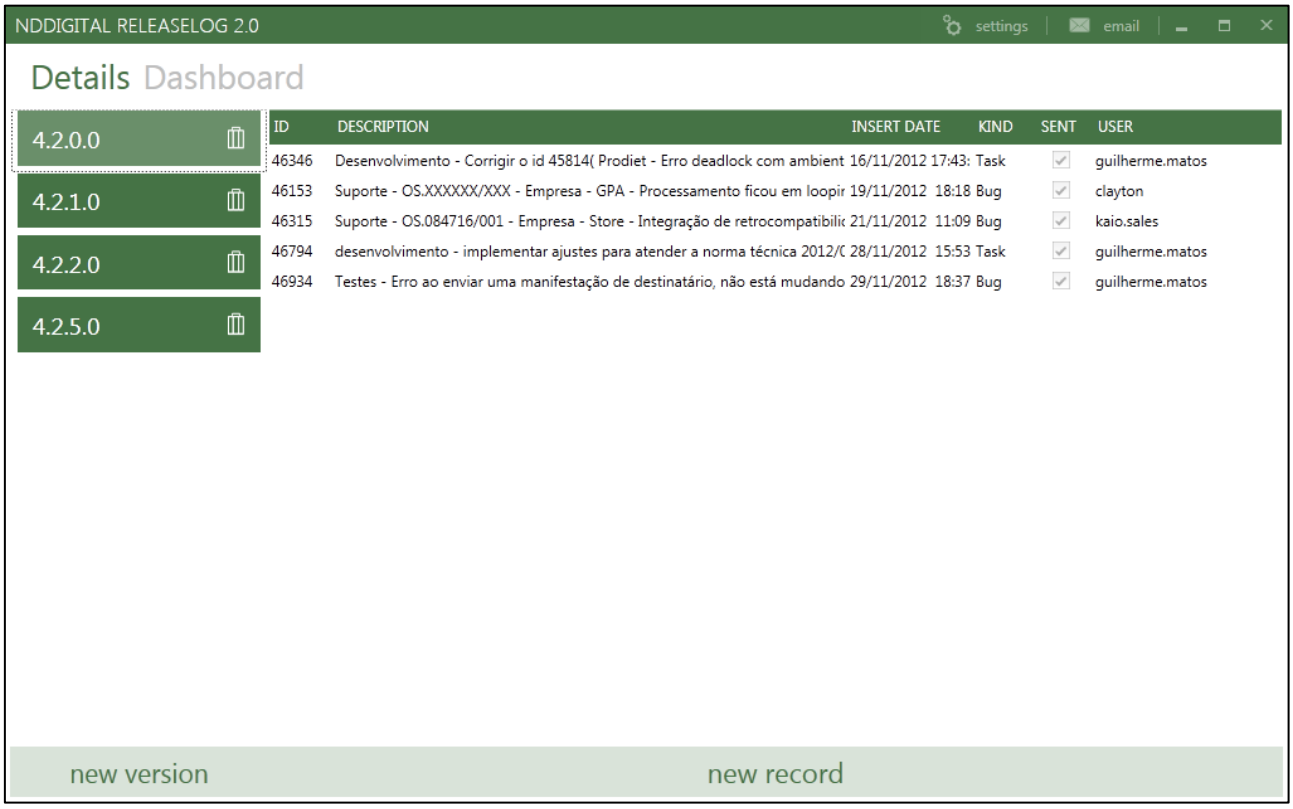

Figura 13: Tela inicial do sistema.
Fonte: O próprio autor.

A Figura 14 mostra a tela para adicionar uma nova versão. Isso vai acontecer quando a equipe de desenvolvimento estiver trabalhando com uma nova versão. Caso essa nova versão for um patch, deverá ser adicionada a versão relacionada.

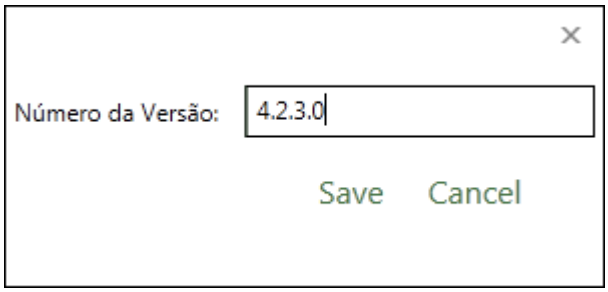

Figura 14: Tela de adição de novas versões.
Fonte: O próprio autor.

A Figura 15 mostra a tela de configuração de e-mail, onde o usuário deverá entrar com os dados de conexão com o servidor de e-mail.

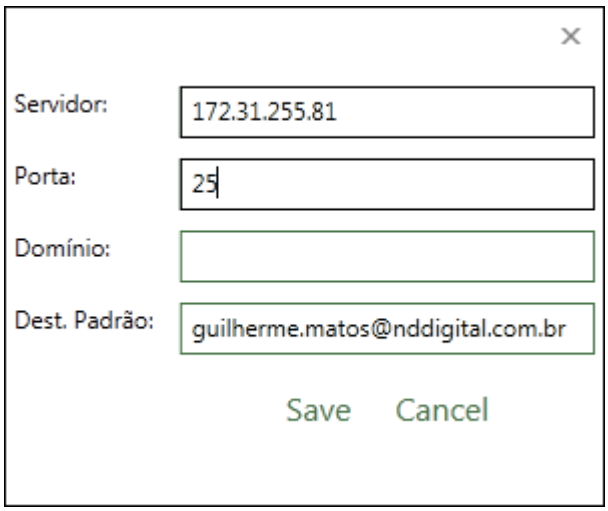

Figura 15: Tela de configuração do servidor de e-mail.
Fonte: O próprio autor.

A Figura 16 mostra a tela das configurações de conexão com o Team Foundation Server. O usuário deverá entrar com o endereço do servidor, o projeto que ele deseja ter o controle dos ids, o usuário e a senha cadastrados no servidor.

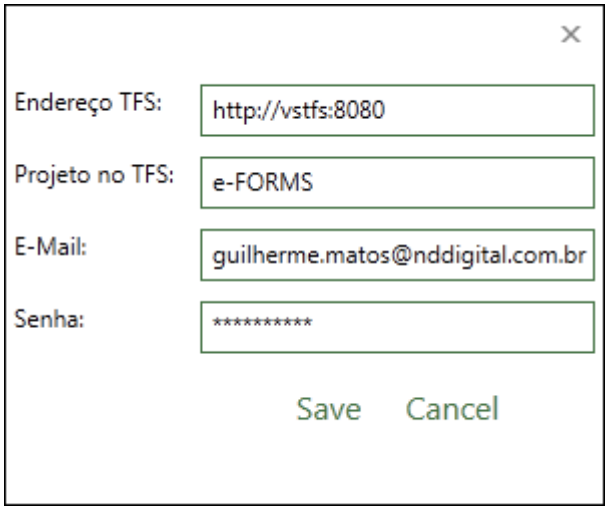

Figura 16: Tela de configuração do Team Foundation Server.
Fonte: O próprio autor.

A Figura 17 mostra a tela de pesquisa de ids no Team Foundation Server. Nela, o usuário é capaz de pesquisar pelo id e adicionar na versão que atual ou em um patch, por exemplo; porém, ao abrir essa tela, irá listar todos os ids que estão assinados com o nome do usuário.

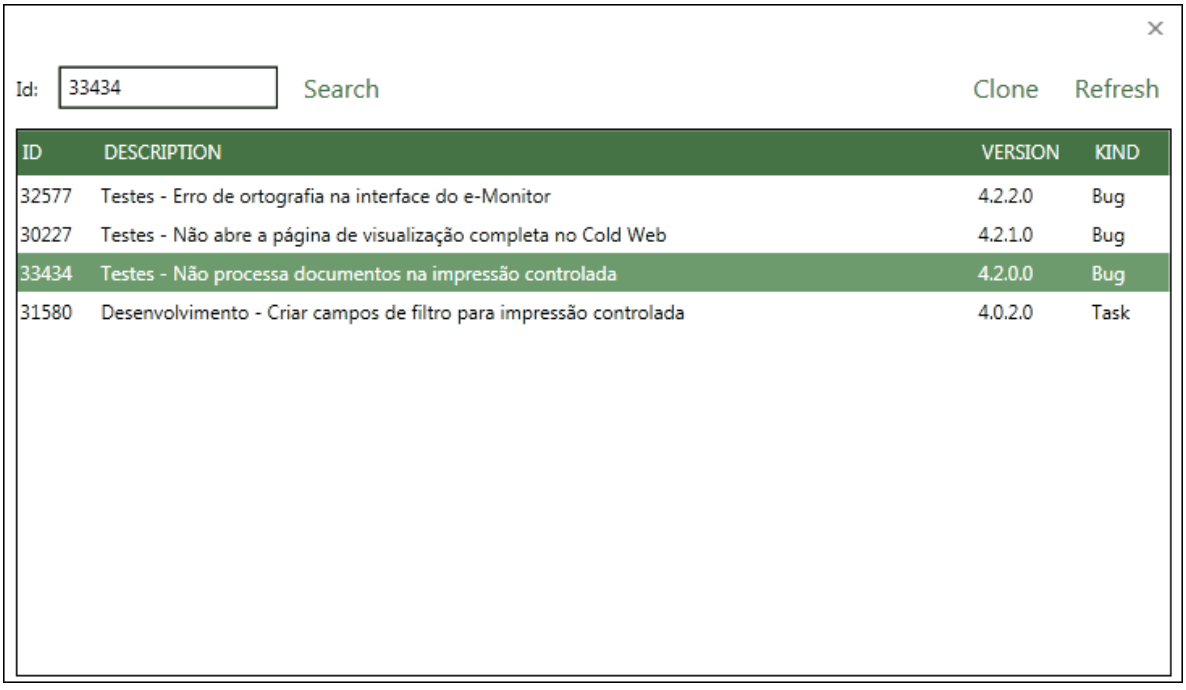

Figura 17: Tela de pesquisa de Ids.
Fonte: O próprio autor.

A Figura 18 mostra a tela de envio de e-mail para o responsável por gerar os *builds* da solução. Nessa tela o usuário só terá que escrever alguma observação, escolher os projetos

alterados e os módulos à compilar, pois as outras opções serão carregadas das configurações feitas anteriormente.

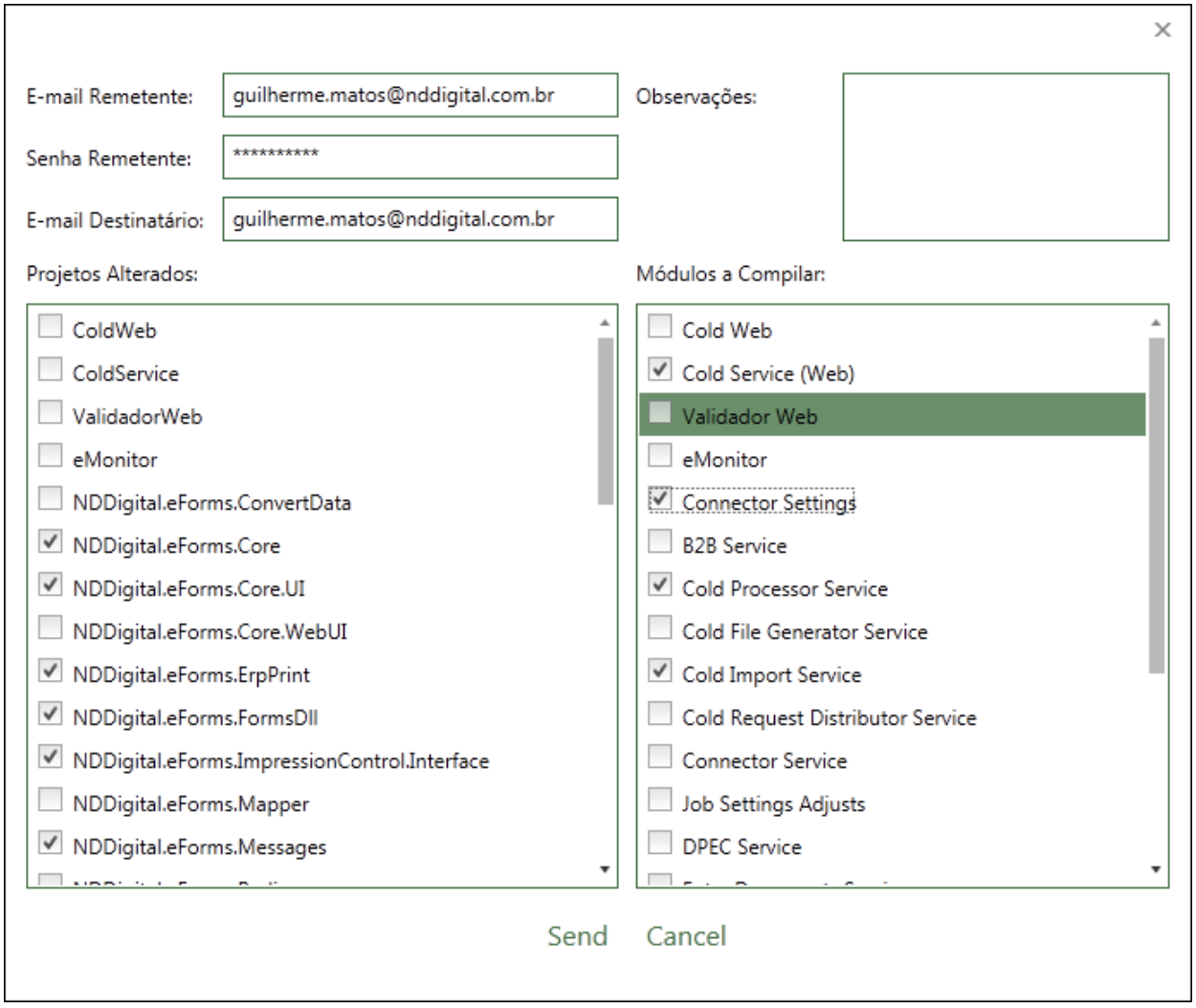

Figura 18: Tela de envio de e-mail.
Fonte: O próprio autor.

6 RESULTADOS

Após ter implantado o software e ter aplicado a avaliação na equipe de desenvolvimento, foi analisado que a nova ferramenta foi bem mais aceita que a antiga. Além disso, pôde ser verificado que por mais que haja diferenças entre os dois softwares, foi mais fácil interagir com essa nossa ferramenta.

A avaliação, a qual está nos anexos (Anexo A), foi realizada com cinco pessoas, as quais possuem em média dois anos de experiência na área de desenvolvimento de software, e foi aplicada duas vezes: uma utilizando o software antigo (Figura 19) e uma utilizando o novo sistema (Figura 20). Ao aplicar o questionário referente ao sistema antigo, teve uma porcentagem aproximada de 49% das respostas "Sim". Já o novo sistema teve uma porcentagem aproximada de 82% de respostas "Sim"[12]. O que pode ser feito para melhorar esse resultado é alterar os recursos do usuário, melhorar a rede do ambiente de trabalho ou aumentar a largura da banda a fim de não ter gargalos na navegação, visto que o software precisa de uma conexão com o servidor para funcionar, e/ou analisar o questionário e verificar os motivos para tentar resolvê-los diretamente no código-fonte.

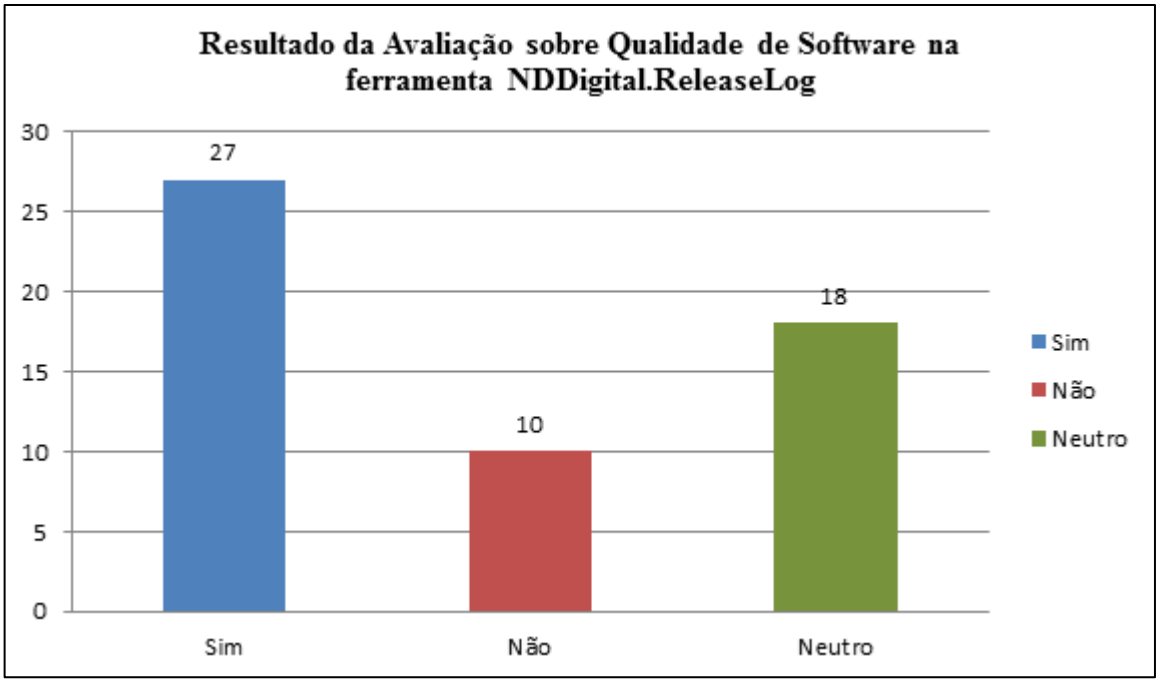

Figura 19: Resultado da avaliação aplicada no software antigo.
Fonte: O próprio autor.

[12] A resposta do questionário que indica que um sistema tem qualidade é "Sim". Quanto maior o número de respostas, mais qualificado é o sistema.

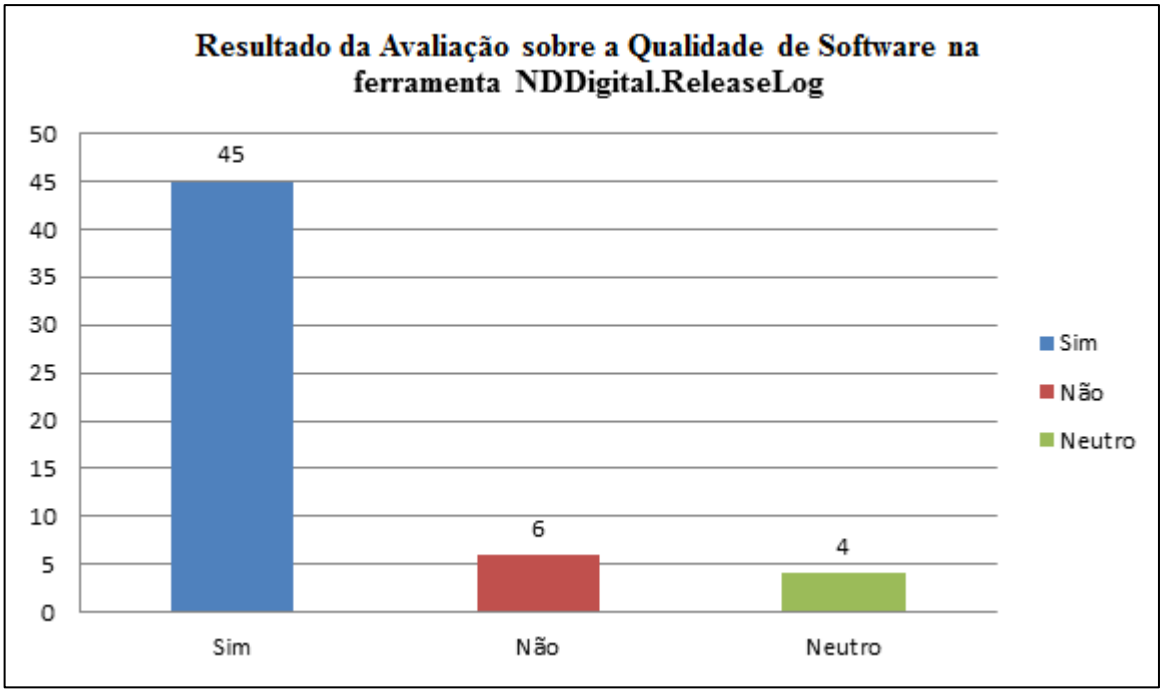

Figura 20: Resultado da avaliação aplicada no novo sistema.
Fonte: O próprio autor.

Dessa forma, pode-se notar que um sistema desenvolvido utilizando boas práticas, técnicas, métodos e tecnologias, consegue satisfazer o cliente nas suas necessidades do dia-a-dia e na resolução dos problemas. Se o usuário vai utilizar o sistema para realizar as suas funções, ele precisa de um sistema confiável, atraente e rápido, onde ele tem certeza que não vai perder suas informações, caso haja alguma falha.

7 CONCLUSÃO

O desenvolvimento de software é considerado uma ciência, e, por isso, deve-se obedecer a princípios, normas, leis e recomendações. Um sistema de software deve ser pensado não apenas na sua elaboração, mas sim durante todo o processo de desenvolvimento do software. Por esse motivo, este trabalho foi desenvolvido com o propósito de realizar um estudo aprofundado sobre tecnologias, técnicas, métodos e padrões que levam à qualidade do software e a satisfação do cliente, garantindo grandes benefícios no processo de manutenção.

No decorrer do trabalho, foram abordados os conceitos de sistema, sistema da informação, o ciclo de vida de um software, engenharia de software, seus modelos de processo, UML, conceitos de qualidade de software, padrões de projeto. Além disso, foi realizado um estudo de caso que envolveu o desenvolvimento de um sistema de gerenciamento de tarefas e *bugs*, e para isso foram apresentadas algumas tecnologias que auxiliaram no desenvolvimento de um software com qualidade.

A qualidade é consequência dos processos, das pessoas e da tecnologia. A relação entre a qualidade do produto e cada um desses fatores é complexa. Por isso, é muito mais difícil controlar o grau de qualidade do produto do que controlar os requisitos. (NOGUEIRA, 2009).

Com este trabalho, pode-se concluir que um bom planejamento, elaboração, estudo e a correta tecnologia fazem com que o software tenha qualidade, a qual muitas vezes é mais importante que o próprio produto; e para que isso aconteça, é importante seguir uma série de etapas. Este trabalho também proporcionou várias oportunidades para conhecer tecnologias diferentes, as quais eu não utilizo no meu dia-a-dia, que auxiliam no desenvolvimento de software com qualidade. Além disso, possibilitou aprofundar o meu conhecimento sobre Engenharia de Software, a qual é uma área extremamente importante e que muitas vezes é a chave principal para conseguir o sucesso de um software, pois é muito mais valioso o investimento no desenvolvimento de um sistema que é vai ser conhecido pela sua qualidade, do que os prazos de liberação deste sistema.

REFERÊNCIAS BIBLIOGRAFICAS

ALEXANDER, Christopher *et al*. **A Pattern Language**. Oxford University Press, New York, 1977.

BATISTA, Emerson de O.. **Sistemas de Informação: O Uso Consciente da Tecnologia para o Gerenciamento**. 1 ed. São Paulo: Saraiva, 2004. 296 p.

FURLAN, José Davi. **Modelagem de Objetos através da UML – *the Unified Modeling Language***. São Paulo: Makron Books, 1998.

GAMMA, Erich *et al*. **Padrões de Projeto: soluções reutilizáveis de software orientado a objetos.** Porto Alegre: Bookman, 2000.

GAROFALO, Raffaele. **Building Enterprise Application with Windows Presentation Foundation and the Model View ViewModel Pattern.** California: O'Reilly Media, 2011.

HEERDT, Mauri Luiz. **Metodologia Científica e da Pesquisa**, 1 ed. Palhoça: UnisulVirtual, 2007. 266 p.

LOBO, Edson Junior Rodrigues. **Guia Prático de Engenharia de Software**. São Paulo: Digerati Books, 2009. 128 p.

MACHADO, Cristina Ângela Filipak. **A-Risk: um método para identificar e quantificar risco de prazo em projetos de desenvolvimento de software**. 2002. 239 f. Dissertação (Mestrado) – Programa de Pós-Graduação em Informática Aplicada, Pontifica Universidade Católica do Paraná, Curitiba.

MacVITTIE, Lory A. **XAML in a Nutshell.** Sebastopol: O'Reilly, 2006.

MICROSOFT CORPORATION. **.NET Framework Overview.** 2008. Disponível em: < http://www.microsoft.com/net/Overview.aspx>. Acessado em: 14 nov 2012.

MAFFEO, Bruno. **Engenharia de Software e Especificação de Sistemas**. Rio de Janeiro, Editora Campus, 1992.

NOGUEIRA, Marcelo. **Engenharia de Software: Um Framework para a Gestão de Riscos em Projetos de Software.** Rio de Janeiro: Editora Ciência Moderna Ltda., 2009. 202 p.

NUNES, Mauro; O'NEILL, Henrique. **Fundamental de UML.** 2 ed. FCA – Editora de Informática, 2004.

PAULK, M.C. *et al*. **The Capatibility Maturity Model – Guidelines for Improving the Software Process.** Addison Wesley, SEI series, 1995.

PRESSMAN, Roger S.. **Engenharia de Software**. São Paulo: Makron Books, 1995. 1056 p.

PRESSMAN, Roger S.. **Engenharia de Software**. 5 ed. Rio de Janeiro: McGraw Hill, 2002. 843 p.

RADDATZ, Jane. **IEEE Standard Glossary of Software Engineering Terminology**. Disponível em: <http://www.idi.ntnu.no/grupper/su/publ/ese/ieee-se-glossary-610.12-1990.pdf>. Acessado em: 09 nov 2012.

SCHMITZ, Eber Assis. **Desenvolvendo Software Orientado a Objeto**. Rio de Janeiro: Brasport, 2000.

SCHACH, Stephen R.. **Engenharia de Software: Os paradigmas clássico & orientado a objetos**. 7 ed. São Paulo: McGraw-Hill, 2009.

SCHACH, S. R. **Pratical Software Engineering.** Irving-Aksen: Homewood, 1992.

SILVA, Alberto; VIDEIRA, Carlos. **UML, Metodologias e Ferramentas CASE**. Lisboa: Centro Atlântico, 2001.

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

SOMMERVILLE, Ian. **Engenharia de Software**. 8ª ed. São Paulo: Addison Wesley, 2007.

SONNINO, Bruno; SONNINO, Roberto. **Introdução ao WPF**. Disponível em: <http://msdn.microsoft.com/pt-br/library/cc564903.aspx>. Acesso em: 14 nov. 2012.

SONNINO, Bruno; SONNINO, Roberto. **Data Bindings com WPF.** Disponível em: < http://www.microsoft.com/brasil/msdn/tecnologias/netframework/DataBinding_WPF.mspx>. Acessado em 15 nov. 2012.

TURBAN, Efraim; McLEAN, Ephraim; WETHERBE, James. **Tecnologia da informação para gestão**. 3 ed. Porto Alegre: Bookman, 2004.

WETHERBE, James C.. **Análise de sistemas para sistemas de informação por computador.** 3 ed. Rio de Janeiro: Campus, 1987.

ANEXOS

ANEXO A

Avaliação de Requisitos de Qualidade de Software Aplicadas na Ferramenta NDDigital.ReleaseLog

Esta avaliação possui como objetivo ser fonte de pesquisa sobre o uso da qualidade de software utilizada na ferramenta NDDigital.ReleaseLog.

Os resultados serão utilizados no Trabalho de Conclusão de Curso do acadêmico Guilherme Matos Oliveira, do curso de Ciência da Computação, do Centro Universitário Facvest.

1. **Qual o seu nome?**
2. **Tempo de Experiência profissional?**
3. **Você tem conhecimento sobre Qualidade de Software?**
4. **Você sabe se foi aplicado algum item de qualidade de software na ferramenta NDDigital.ReleaseLog?**
5. **Com relação ao tempo de inicialização e execução das atividades, responda 'Sim' para pouco demorado, 'Não' para muito demorado e 'Neutro' para indiferente.**
6. **Esta ferramenta consome pouco recurso da máquina?**
7. **É fácil aprender, operar e controlar essa ferramenta?**
8. **Essa ferramenta evita acessos não autorizados dos dados?**
9. **É fácil de realizar a configuração desta ferramenta?**
10. **É capaz de recuperar dados em caso de falha?**
11. **É fácil corrigir uma falha na ferramenta, quando ocorre?**
12. **É fácil de instalar essa ferramenta em outros ambientes?**
13. **É mais fácil utilizar essa ferramenta do que o próprio Team System?**