Henrique Antonácio Povedano

Especificação e desenvolvimento de software de gestão para empresa de entregas sustentáveis

Trabalho de formatura apresentado à Escola Politécnica da Universidade de São Paulo para obtenção do Diploma de Engenheiro de Produção

São Paulo
2015

Henrique Antonácio Povedano

Especificação e desenvolvimento de software de gestão para empresa de entregas sustentáveis

Trabalho de formatura apresentado à Escola Politécnica da Universidade de São Paulo para obtenção do Diploma de Engenheiro de Produção

Orientador: Prof. Dr. Álvaro Hernandez

São Paulo
2015

Catalogação-na-publicação

Povedano, Henrique A.
Especificação e desenvolvimento de software de gestão para empresa de entregas sustentáveis / H. A. Povedano -- São Paulo, 2015.
118 p.

Trabalho de Formatura - Escola Politécnica da Universidade de São Paulo. Departamento de Engenharia de Produção.

1.Tecnologia da Informação I.Universidade de São Paulo. Escola Politécnica. Departamento de Engenharia de Produção II.t.

AGRADECIMENTOS

À minha família, meus amigos e minha namorada pelo apoio ao longo de meus estudos.

À Escola Politécnica, por ter me proporcionado experiências de valor inestimável para minha formação pessoal, acadêmica e profissional.

Ao meu orientador, Prof. Dr. Álvaro Hernandez, por sua paciência, exigência e divertidas reuniões.

À equipe da Courrieros, em especial ao meu amigo de longa data André Biselli, pelo apoio ao trabalho.

RESUMO

Este trabalho busca resolver as dificuldades de gestão de entregas, recursos humanos e de vendas enfrentados pela empresa Ecolivery Courrieros, do ramo de entregas por bicicletas. Após uma revisão da literatura que forneceu bases para a escolha da metodologia e tecnologia de desenvolvimento, o autor construiu um sistema para a plataforma Windows em C# e SQL-SERVER utilizando o método de engenharia de software Praxis. O novo programa resolve os principais problemas que atacou e dá mais transparência aos resultados da Courrieros, qualidade fundamental dado que se encontra em processo de captação de investimentos.

Palavras-chave: Praxis, Tecnologia da Informação, C#, SQL-SERVER.

ABSTRACT

This paper aims to solve the sales, human-resources and delivery management problems faced by Ecolivery Courrieros, a bike-delivery service company based in São Paulo, Brazil. After a literature review which supported the methodology and technology choices, the author developed a C#, SQL-SERVER based solution for the Windows operating system. The custom-built program addressed the main difficulties the company faced and gives its sales data more transparency, a fundamental attribute given its current search for venture-capital investment.

Key-words: SQL-Server, C#, Praxis, IT

LISTA DE TABELAS

LISTA DE FIGURAS

LISTA DE QUADROS

LISTA DE ABREVIATURAS E SIGLAS

OEA – Organização dos Estados Americanos

RUP – Rational Unified Process

AWS – Amazon Web Services

MS – The Microsoft Corporation

UML – Unified Modeling Language

GUI – Graphical User Interface, Interface Gráfica

DoD – Departmento of Defense, equivalente ao Ministério da Defesa nos EUA

OOA – Object-oriented Analysis

OOD - Object-oriented Design

OOP – Object-oriented Programming

VS – Microsoft Visual Studio

DB – Banco de Dados

PRO – Departamento de Engenharia de Produção da Escolha Politécnica da USP

TDD – Test-driven Development

SUMÁRIO

Parte I – Introdução e definição do problema

1. Introdução

A eficiência e confiabilidade trazidas por um sistema digital geração de demonstrativos contábeis são parte fundamental do sucesso de pequenas e médias empresas, como sugere o estudo de Tim Mills-Gronninger (Gronninger, 2011, p. 1).

Em particular, empresas recém-criadas podem enfrentar maiores dificuldades do que as mais bem estabelecidas no mercado: a falta de experiência de seus funcionários e a escassez de capital para investimento em sistemas de TI podem ser fatores que dificultam a decisão de compra de um software adequado para a empresa. Segundo Noah Abelson, CEO e fundador da empresa de marketing digital Shareroot, startups sofrem por utilizar ferramentas não adaptadas ao forte crescimento que muitas vezes encontram, como o gerenciador de planilhas Excel. (Abelson, 2015)

Este trabalho se insere neste contexto: A empresa Ecolivery Courrieros Entregas Ecológicas, fundada em 2012, passa por uma fase de grande crescimento em sua receita, o que atraiu potenciais investidores interessados no desempenho da companhia. A firma não conta, porém, com um sistema confiável de controle de custos e despesas, o que torna a apresentação dos resultados aos potenciais capitalistas menos confiável aos olhos destes.

Através de minha amizade com dois dos fundadores da empresa, fui convidado a desenvolver meu trabalho de formatura com a Courrieros, com o tema de especificação e produção de um software de gestão para a companhia.

A resolução do problema começa com uma breve apresentação sobre a empresa, em especial sobre seu sistema de gestão de entregas atual, que apresenta os problemas que devem ser resolvidos por este TF.

2. Apresentação da empresa

2.1. História

A empresa Ecolivery Courrieros, Ltda. é uma empresa de entregas por motos elétricas e bicicletas fundada em outubro de 2012 pelos amigos André Biselli, Victor Castello Branco e Stefano Cappanari, que decidiram explorar com um produto ambientalmente sustentável o mercado de entregas na cidade de São Paulo. Estabelecida originalmente na Avenida Pedroso de Morais, em Pinheiros, a Courrieros está sediada atualmente na Rua Lisboa, no mesmo bairro, de onde atende o centro expandido da capital.

A empresa passa por um período de forte crescimento desde sua fundação: o número de funcionários-entregadores passou de 10 em outubro de 2013 para 26 em março de 2015, e o número de entregas cresceu aproximadamente 200% de março de 2014 a junho de 2015.

A empresa baseia seu modelo de negócios em duas frentes principais: terceirização de ciclistas para estabelecimentos comerciais como cartórios ou restaurantes, e entregas avulsas, e atende o centro expandido.

2.2. Modelo de negócios

Como mencionado acima, a empresa atende seus clientes de duas formas: por entregas avulsas ou por contratos de terceirização de serviços. A primeira consiste em efetuar entregas individuais e, na segunda, o cliente assina um contrato com a Courrieros em que esta se compromete a deixar à disposição do contratante um entregador por um determinado período por dia. A Courrieros rentabiliza esta operação cobrando mais do cliente do que paga ao ciclista, que continua sendo funcionário da Ecolivery.

A participação de cada um dos dois modelos na receita da empresa varia significativamente com a época do ano, mas podemos observar uma tendência de maior participação dos contratos de terceirização com o passar do tempo.

2.3. Clientes

A empresa já atendeu mais de 450 clientes ao longo de sua existência e conta atualmente com 80 clientes regulares, i.e., que fazem uso dos serviços de entrega ao menos uma vez por mês. O tamanho e segmento de atuação dos clientes são muito variados: A empresa atende desde o Banco Itaú BBA, maior banco de investimentos da América Latina, até o pequeno restaurante Piadina, localizado na região do Itaim Bibi. Além disso, a Courrieros é responsável por efetuar as entregas do tipo *same-day* da empresa de *e-commerce* Netshoes, maior empresa do segmento no Brasil.

A Figura 1 mostra os principais clientes da Courrieros. Nota-se a diversidade dos ramos de atuação.

Figura 1 – Principais clientes da Courrieros

Fonte: www.courrieros.com/site/clientes

2.4. Reconhecimento do mercado: Presença na mídia e premiações

A empresa já recebeu prêmios significativos pelo apelo ambientalmente responsável de seu modelo de negócios. Dentre estes podemos citar como o mais significativo o ECO-Challenge das Américas (PEPSICO, página única), organizado pelo OEA (Organização dos Estados Americanos) e pela PEPSICO, empresa do ramo alimentício dona de companhias com *Lay's* e Pepsi. O prêmio reconhece a startup mais sustentável das Américas e premia a vencedora com US$ 5 000,00.

Além deste prêmio, a empresa já foi alvo de reportagens de grandes periódicos nacionais. Recentemente foi capa do caderno Mercado MPME do jornal A Folha de São Paulo graças ao viés sustentável do negócio, em matéria sobre empreendedorismo social no Brasil. A figura 2 mostra um dos sócios-fundadores, André Biselli, e um de seus entregadores, em foto desta reportagem.

Figura 2 - Sócio e entregador da Courrieros em matéria Folha de São Paulo

Fabio Braga/Folhapress

O entregador Fabrício Santana e o dono da Courrier, André Bicelli

Fonte: A Folha de São Paulo, 19 de Abril de 2015

2.5. Dados operacionais da empresa

No mês de maio de 2015 a empresa teve um faturamento de R$ 73 000,00, e efetuou aproximadamente 3000 entregas - o número exato não é conhecido pois ciclistas alocados não são obrigados a reportar suas entregas. No mês em questão, a empresa contava com 28 entregadores, sendo 26 ciclistas e 2 motoqueiros, que efetuam as entregas de longa distância com motos elétricas. A terceirização de ciclistas corresponde em média a 75% do faturamento da empresa (R$ 56 000,00 em maio 2015), e as entregas avulsas são responsáveis pela fatia restante.

O custo fixo médio por ciclista em período integral para o mês de outubro de 2014 é mostrado na tabela 1 Chama atenção o fato de o salário representar apenas 50% do custo médio de um ciclista, além do alto custo de manutenção e vales pagos pela empresa. Trata-se, porém, de um valor abaixo da média brasileira, como mostram Souza et al. em pesquisa da Fundação Getúlio Vargas (2012, p. 4), que indica que os desembolsos de uma empresa brasileira com seus funcionários chegam a 255% do salário mensal do trabalhador.

Tabela 1 – Custo fixo por ciclista

Componente	Valor
Salário	R$ 1.001,00
Vales	R$ 350,00
Seguros	R$ 150,00
13º	R$ 83,42
Manutenção + Depreciação	R$ 200,00
Férias	R$ 27,53
Uniforme	R$ 40,00
FGTS	R$ 120,12
Total	R$ 1.972,06

Fonte: Courrieros

O custo fixo por ciclista é um dado de grande relevância para a Courrieros, pois indica o preço mínimo que esta deve cobrar por seus serviços de terceirização (sem levar em conta custos de oportunidade) ou a receita média mensal que um trabalhador deve gerar em entregas avulsas para que consiga pagar por seus custos.

Após esta breve imagem da história e das operações da empresa, o trabalho irá se concentrar em definir o problema que a empresa enfrenta com seus sistemas de gestão e controle de entregas, fonte de contratempos que este projeto tratará de evitar.

3. O problema

Como foi apresentado acima, a Courrieros apresenta atualmente forte crescimento de suas receitas, o que atraiu um grupo de investidores interessados em se tornar sócios da empresa. Esse crescimento trouxe também dificuldades no controle financeiro, o que torna a prestação de contas para com os novos sócios mais difícil, particularmente devido à falta de informações confiáveis sobre os dados operacionais e financeiros da empresa.

O objetivo desta seção é delimitar o problema enfrentado pela empresa, com foco no sistema de informações gerenciais usado atualmente e em como este causa a falta de confiabilidade nos dados da Courrieros.

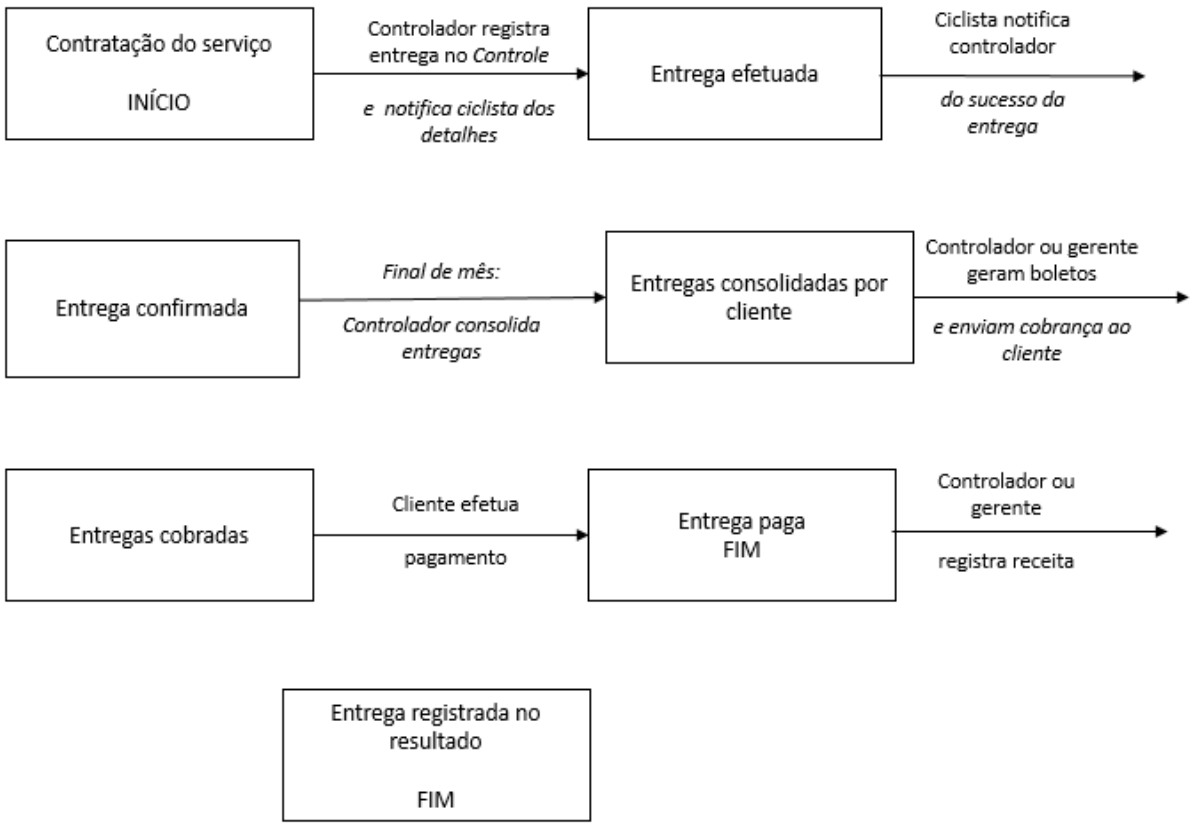
3.1. O sistema de informações contábeis atual

O sistema de informações contábeis atual é baseado totalmente em uma planilha Excel, chamada pelos funcionários da Courrieros de *Controle.* Nela são registradas todas as informações das entregas contratadas, como nome do cliente contratante, preço do serviço e endereço de retirada, entre outros.

A empresa possui uma planilha para cada mês, ao final do qual é calculado o resultado com base nos registros do controle. A planilha também é usada para a cobrança de clientes, que recebem boletos gerados manualmente segundo as informações da planilha.

A figura 3 ilustra o ciclo de uma entrega, desde sua contratação até sua cobrança, ao final do mês. Nela, eventos ou acontecimentos são representados por retângulos, enquanto atividades ou ações são indicadas por setas.

Figura 3 - Ciclo de vida de uma entrega



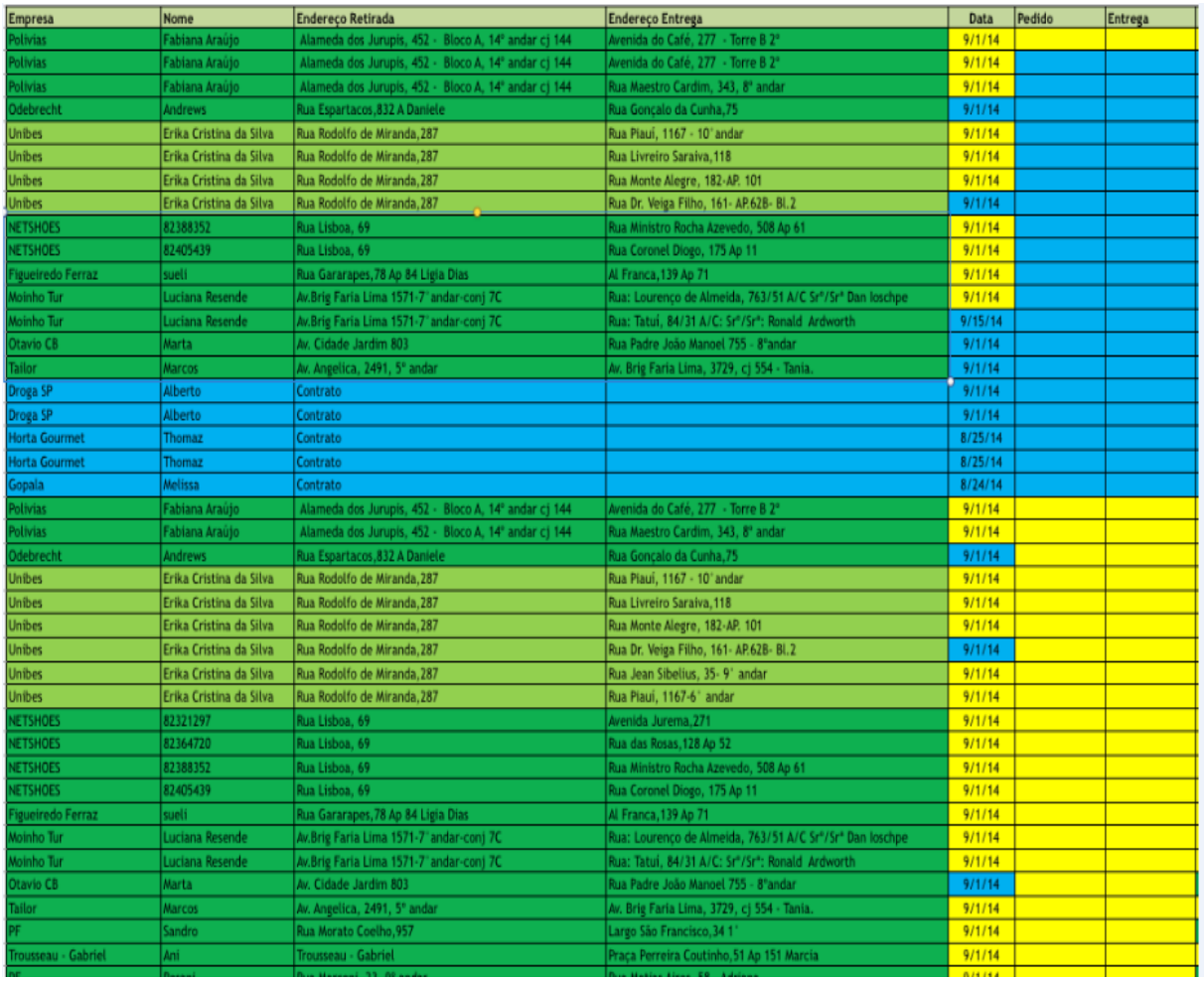
Fonte: Do autor

É importante notar que todas as atividades do ciclo são executadas manualmente por funcionários da empresa ou clientes, mesmo aquelas que possuem grande potencial para automação, como consolidação de entregas, registro de receitas ou geração de boletos.

Parte da planilha de controle é mostrada na figura 4. É importante mencionar que a planilha não possui mecanismos que validem a entrada de dados: É possível registrar uma entrega para um cliente que não existe, assim como é possível atribuir dois nomes para o mesmo cliente. Além disso, a planilha permite operações de exclusão e edição sem confirmação por parte do usuário.

É possível notar algumas possíveis fontes de erro: Nomes de empresas cliente são misturados com os de funcionários dos clientes, e a planilha não detecta telefones de contato inseridos como nomes de contatos, o que nona linha para o cliente NETSHOES.

Figura 4 - Planilha de controle atual da Courrieros



Empresa	Nome	Endereço Retirada	Endereço Entrega	Data	Pedido	Entrega
Polivias	Fabiana Araújo	Alameda dos Jurupis, 452 - Bloco A, 14° andar cj 144	Avenida do Café, 277 - Torre B 2°	9/1/14		
Polivias	Fabiana Araújo	Alameda dos Jurupis, 452 - Bloco A, 14° andar cj 144	Avenida do Café, 277 - Torre B 2°	9/1/14		
Polivias	Fabiana Araújo	Alameda dos Jurupis, 452 - Bloco A, 14° andar cj 144	Rua Maestro Cardim, 343, 8° andar	9/1/14		
Odebrecht	Andrews	Rua Espartacos,832 A Daniele	Rua Gonçalo da Cunha,75	9/1/14		
Unibes	Erika Cristina da Silva	Rua Rodolfo de Miranda,287	Rua Piauí, 1167 - 10° andar	9/1/14		
Unibes	Erika Cristina da Silva	Rua Rodolfo de Miranda,287	Rua Livreiro Saraiva,118	9/1/14		
Unibes	Erika Cristina da Silva	Rua Rodolfo de Miranda,287	Rua Monte Alegre, 182-AP. 101	9/1/14		
Unibes	Erika Cristina da Silva	Rua Rodolfo de Miranda,287	Rua Dr. Veiga Filho, 161- AP.62B- Bl.2	9/1/14		
NETSHOES	82388352	Rua Lisboa, 69	Rua Ministro Rocha Azevedo, 508 Ap 61	9/1/14		
NETSHOES	82405439	Rua Lisboa, 69	Rua Coronel Diogo, 175 Ap 11	9/1/14		
Figueiredo Ferraz	sueli	Rua Gararapes,78 Ap 84 Ligia Dias	Al Franca,139 Ap 71	9/1/14		
Moinho Tur	Luciana Resende	Av.Brig Faria Lima 1571-7° andar-conj 7C	Rua: Lourenço de Almeida, 763/51 A/C Sr°/Sr° Dan loschpe	9/1/14		
Moinho Tur	Luciana Resende	Av.Brig Faria Lima 1571-7° andar-conj 7C	Rua: Tatuí, 84/31 A/C: Sr°/Sr°: Ronald Ardworth	9/15/14		
Otavio CB	Marta	Av. Cidade Jardim 803	Rua Padre João Manoel 755 - 8°andar	9/1/14		
Tailor	Marcos	Av. Angelica, 2491, 5° andar	Av. Brig Faria Lima, 3729, cj 554 - Tania.	9/1/14		
Droga SP	Alberto	Contrato		9/1/14		
Droga SP	Alberto	Contrato		9/1/14		
Horta Gourmet	Thomaz	Contrato		8/25/14		
Horta Gourmet	Thomaz	Contrato		8/25/14		
Gopala	Melissa	Contrato		8/24/14		
Polivias	Fabiana Araújo	Alameda dos Jurupis, 452 - Bloco A, 14° andar cj 144	Avenida do Café, 277 - Torre B 2°	9/1/14		
Polivias	Fabiana Araújo	Alameda dos Jurupis, 452 - Bloco A, 14° andar cj 144	Rua Maestro Cardim, 343, 8° andar	9/1/14		
Odebrecht	Andrews	Rua Espartacos,832 A Daniele	Rua Gonçalo da Cunha,75	9/1/14		
Unibes	Erika Cristina da Silva	Rua Rodolfo de Miranda,287	Rua Piauí, 1167 - 10° andar	9/1/14		
Unibes	Erika Cristina da Silva	Rua Rodolfo de Miranda,287	Rua Livreiro Saraiva,118	9/1/14		
Unibes	Erika Cristina da Silva	Rua Rodolfo de Miranda,287	Rua Monte Alegre, 182-AP. 101	9/1/14		
Unibes	Erika Cristina da Silva	Rua Rodolfo de Miranda,287	Rua Dr. Veiga Filho, 161- AP.62B- Bl.2	9/1/14		
Unibes	Erika Cristina da Silva	Rua Rodolfo de Miranda,287	Rua Jean Sibelius, 35- 9° andar	9/1/14		
Unibes	Erika Cristina da Silva	Rua Rodolfo de Miranda,287	Rua Piauí, 1167-6° andar	9/1/14		
NETSHOES	82321297	Rua Lisboa, 69	Avenida Jurema,271	9/1/14		
NETSHOES	82364720	Rua Lisboa, 69	Rua das Rosas,128 Ap 52	9/1/14		
NETSHOES	82388352	Rua Lisboa, 69	Rua Ministro Rocha Azevedo, 508 Ap 61	9/1/14		
NETSHOES	82405439	Rua Lisboa, 69	Rua Coronel Diogo, 175 Ap 11	9/1/14		
Figueiredo Ferraz	sueli	Rua Gararapes,78 Ap 84 Ligia Dias	Al Franca,139 Ap 71	9/1/14		
Moinho Tur	Luciana Resende	Av.Brig Faria Lima 1571-7° andar-conj 7C	Rua: Lourenço de Almeida, 763/51 A/C Sr°/Sr° Dan loschpe	9/1/14		
Moinho Tur	Luciana Resende	Av.Brig Faria Lima 1571-7° andar-conj 7C	Rua: Tatuí, 84/31 A/C: Sr°/Sr°: Ronald Ardworth	9/1/14		
Otavio CB	Marta	Av. Cidade Jardim 803	Rua Padre João Manoel 755 - 8°andar	9/1/14		
Tailor	Marcos	Av. Angelica, 2491, 5° andar	Av. Brig Faria Lima, 3729, cj 554 - Tania.	9/1/14		
PF	Sandro	Rua Morato Coelho,957	Largo São Francisco,34 1°	9/1/14		
Trousseau - Gabriel	Ani	Trousseau - Gabriel	Praça Perreira Coutinho,51 Ap 151 Marcia	9/1/14		

Fonte: Courrieros

Mais um exemplo de possível fonte de erro é a não obrigatoriedade de preenchimento de alguns dados, como o horário de entrega, o que pode levar a desentendimentos entre a empresa e seus fregueses: Diversas vezes o cliente assume que sua entrega será realizada assim que retirada, enquanto na realidade o ciclista pode só vir a finalizar o serviço horas depois, frustrando as expectativas do contratante. O *controle* se propõe a atacar este problema com sua coluna “Entrega”, que indica o horário esperado de finalização, mas esta informação quase nunca é preenchida, como mostra a figura 5.

Por último, uma situação mencionada com frequência é a dupla cobrança, que ocorre quando o cliente paga pelo serviço à vista mas é cobrado por tal à prazo também. A

principal causa-raiz deste erro é o não-preenchimento ou edição involuntária da coluna “Pago à vista”, também mostrada na figura 5.

Figura 5 - Parte da planilha de controle com a coluna "Entrega" em branco

Empresa	Nome	Endereço Retirada	Endereço Entrega	Data	Pedido	Entrega	Pago à Vista
Unibes	Erika Cristina da Silva	Rua Rodolfo de Miranda,287	Rua Critios 57 apt 141 (Alberto Coury)	9/3/14			
Odebrecht - Gonçalo da Cunh	Andrews	Rua Lemos Monteiro,120 19° Luana Timpano	Rua Gonçalo da Cunha,75 Francineide	9/3/14			
Prisma	Marcelo	Av. Brig. Faria Lima, 1.903 - cj 44	Rua Libero Badaró, 293 - 29° andar A/C SINTIA	9/3/14	9:30		
PF	Diego Fresco	Rua Caconde 125, 01425-011, Retirada na portaria	Av. Brigadeiro Luis Antonio 2367, 16o andar	9/3/14			R$ 23.00
May	Mayra	Al. Campinas, 1027	Rua Moacir, 395, cj 52, Gabriela	9/3/14	retirar 9h30		
Casa do Brincar	Isabel	Rua Ferreira de Araujo 388	Rua Combatente do Gueto 186 - Scheila	9/3/14		9:00	
MensMarket	Leandro	Rua Lisboa, 69	Rua Barão de Tatuí, 371, Vila Buarque, Ap 02	9/3/14			
Serge Ferrari	Cintia	Alameda Santos, 1800 - 4° andar - Sala 405 - com Cintia	Av. Brig Luiz Antonio, 2050 - 15° andar. A/C: Depto Contabil até as	9/3/14		até as 11h.	
Serge Ferrari	Cintia	Alameda Santos, 1800 - 4° andar - Sala 405 - com Cintia	Rua Dr. Seidel, 425 1° Andar - Vila Leopoldina;	9/3/14			
NETSHOES	82200309	Rua Lisboa, 69	Rua Itambé,367	9/3/14			
NETSHOES	82485220	Rua Lisboa, 69	Avenida Vieira de Carvalho,197 Ap	9/3/14			
NETSHOES	82488323	Rua Lisboa, 69	Avenida Europa, 149	9/3/14			
NETSHOES	82499976	Rua Lisboa, 69	Avenida Indianópolis, 2665	9/3/14			
NETSHOES	82509243	Rua Lisboa, 69	Rua Pamplona,1808 Ap 21	9/3/14			

Fonte: Courrieros

Apesar de possuir os defeitos mencionados, o gerente financeiro da empresa afirma que o *controle* possui algumas vantagens fundamentais que a mantêm em uso:

- Fácil uso: Somente um conhecimento básico de planilhas Excel é necessário para operar o documento de forma eficiente e eficaz
- Facilidade de acesso a dados passados: A interface gráfica do MS Excel permite que dados de períodos anteriores sejam facilmente acessados sem a necessidade de executar buscar em algum banco de dados
- Flexibilidade: Fácil inserção e exclusão de colunas ou dados com mudanças nas operações da empresa
- Facilidade back-up de dados: Versão salva diretamente no aplicativo de gestão de arquivos DropBox, o que elimina a necessidade de back-up periódicos
- Rapidez em análises ad-hoc: Mecanismos de filtro e tabelas dinâmicas permitem a geração de indicadores de performance com facilidade e rapidez

3.2. Problemas segundo o gerente financeiro

Em um primeiro momento, buscou-se definir os problemas contábeis enfrentados pela empresa na visão do gerente financeiro da Courrieros, André Biselli. Foram apontados os seguintes desafios a serem superados:

- **Falta de confiabilidade dos dados do sistema financeiro atual:** Planilhas Excel não oferecem mecanismo de validação de dados confiável, o que gera a presença de muitas informações erradas ou mal digitadas na planilha de controle. O prejuízo de tais imprecisões é evidente: Cobrança errada de clientes, esquecimento de entregas e imprecisão de indicadores de desempenho
- **Falta de homogeneidade nos dados atuais:** O sistema de controle por planilhas Excel, por não permitir validação de dados, leva a um problema de referência de nomes: Clientes diferentes recebem o mesmo nome na planilha e o mesmo cliente recebe dois nomes diferentes. Isso pode levar a cobranças erradas, além de gerar prejuízo a eventuais análises de vendas por cliente: o volume de entregas por mês por contratante pode ser super ou subestimado.
- **Dificuldade ou impossibilidade de geração de relatórios gerenciais:** Apesar de o Microsoft Excel permitir o uso de filtros, gráficos e a geração de tabelas dinâmicas, estas são operações sujeitas a erro e que exigem certa habilidade informática não possuída pelo operador da planilha.
- **Falta de integração dos diversos sistemas de controle:** Os sistemas de controle de entregas, de gerenciamento de recursos humanos, de controle de gastos e de geração de relatórios contábeis não possuem atualmente nenhum tipo de integração: A entrada de dados para um sistema é feita manualmente segundo a saída de algum outro sistema. Consequências disso são a ineficiência na execução das funções de cada sistema e a também a possível imprecisão nos dados transferidos entre aplicativos.

A solução a ser desenvolvida deve atacar estes problemas, mantendo, porém, as vantagens enumeradas na subseção 3.1. Pode-se agora definir um critério de sucesso do projeto, que será a capacidade da solução de resolver os problemas

citados e de manter os pontos positivos da planilha de controle. O projeto perfeito ataca todas as desvantagens do sistema atual sem comprometer seus benefícios.

Antes de iniciar a resolução do problema, porém, uma revisão bibliográfica guiará o autor em sua busca por uma metodologia e tecnologia de desenvolvimento de programas.

Parte II – Revisão Bibliográfica

1. Introdução

Nesta parte do trabalho são analisados artigos científicos e publicações relevantes ao tema de desenvolvimento de software para pequenas empresas afim de criar uma base de conhecimento para a resolução do problema.

A revisão se divide em três partes: na primeira são pesquisados métodos para a especificação de software. Na segunda parte estuda-se as soluções disponíveis no mercado como possíveis soluções, e por fim são analisadas as tecnologias (linguagens de programação) de desenvolvimento de software mais adaptadas ao problema em questão.

2. Estudo dos paradigmas de desenvolvimento de software

Nesta seção serão estudados os principais paradigmas de desenvolvimento de software existentes atualmente. O objetivo deste estudo é determinar qual destes será adotado para resolver o problema descrito em detalhes na seção acima com maior chance de sucesso.

É importante esclarecer em que sentido o conceito de paradigma é utilizado no contexto deste trabalho. Thomas Kuhn (1962, prefácio) definiu, no contexto das ciências, um paradigma como sendo “uma tradição coerente na pesquisa científica”, tradição esta que envolve técnicas, conhecimentos e prioridades de pesquisa. Para este trabalho, a definição de Kuhn é usada com uma ligeira modificação. Um paradigma é definido aqui como “uma tradição coerente de desenvolvimento de software”. Esta tradição consiste do uso de ferramentas, técnicas, conhecimento e prioridades para alcançar o sucesso no desenvolvimento de um programa de computador.

Para poder efetuar comparações entre os paradigmas, é necessário estabelecer critérios segundo os quais estes serão comparados. Para isso, um modelo de projetos de desenvolvimento de software é definido abaixo em função de suas atividades fundamentais. Este servirá como base para executar as comparações entre paradigmas e a consequente escolha do mais adaptado para resolver o problema definido acima.

2.1.1. Um modelo de projetos de desenvolvimento de software

Nesta subseção é estabelecido um modelo de projetos de software em função de atividades fundamentais. Um modelo é uma representação simplificada da realidade com uma finalidade. A simplificação adotada aqui é a de descrever um projeto como um conjunto de atividades executadas com maior ou menor prioridade e intensidade, e a finalidade é a de comparar diferentes paradigmas.

2.1.1.1. Atividades fundamentais de um projeto de desenvolvimento de software

Estabelecimento dos requisitos

Nesta atividade é estabelecido o que o software deve fazer, ou mais concretamente, qual problema ele resolverá. Diversas ferramentas podem ser usadas determinar e expressar esses requisitos, como os Diagramas de Caso de Uso e Diagrama de Contexto.

Determinação da arquitetura do sistema

Esta atividade determina a estrutura do sistema em um alto nível de abstração. Exemplo de estruturas de alto nível são Banco de Dados, Classes do programa e Interfaces, que podem ser representadas por Diagramas de Classe, Lista de Interfaces, entre outros. É importante frisar que a arquitetura do sistema é independente da linguagem de programação utilizada para construir o programa, e inclui as atividades conhecidas como Análise e Desenho, explicadas mais abaixo.

Implementação (escrita do código)

Nesta atividade, o código necessário para que o sistema funcione é escrito. O código depende da escolha de uma linguagem de programação específica (Java, C++, C).

Testes

Nesta atividade é controlada a qualidade do software e de seus componentes, ou seja, a adequação das funções do software a seus requisitos estabelecidos previamente. Exemplos de testes são testes de integração e testes de unidade.

Manutenção

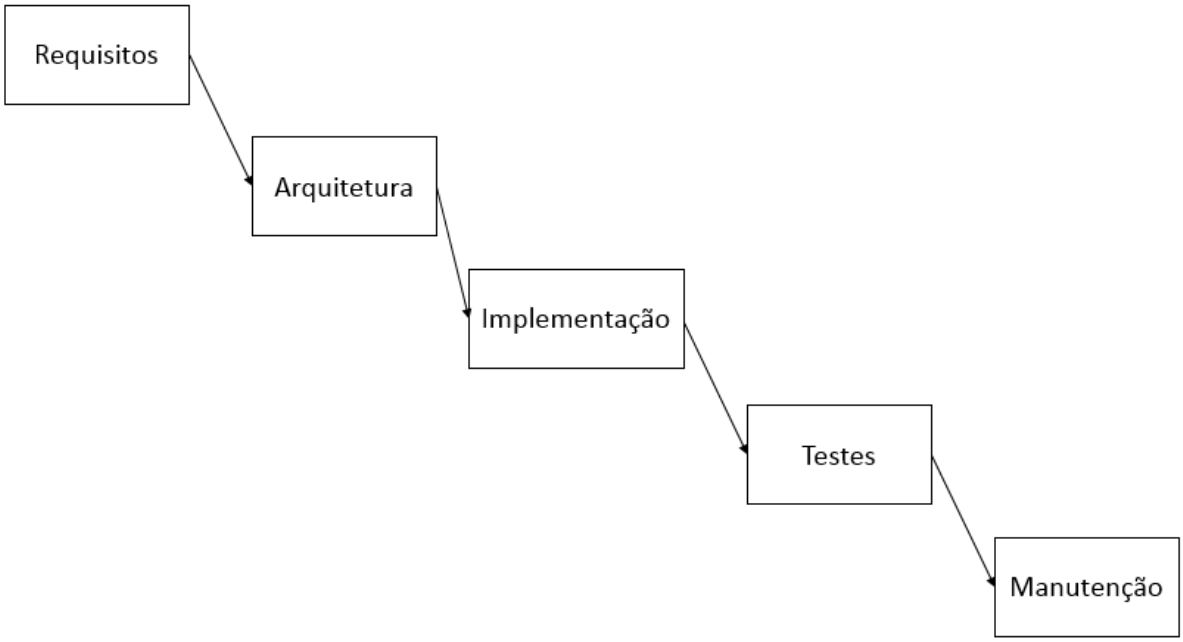
A manutenção consiste em manter o software em funcionamento ao longo de seu uso. Para isso o programa já deve estar em uso pelo usuário final.

2.1.2. Paradigmas existentes e sua relação com o modelo definido

2.1.2.1. Paradigma Waterfall

Os métodos pertencentes ao paradigma *Waterfall* possuem como característica principal a execução sequencial (em série) das atividades fundamentais descritas acima, como mostra a figura 6.

Figura 6 - Atividades do paradigma waterfall



Fonte: Elaborada pelo autor

Neste paradigma, uma atividade só é iniciada quando a anterior foi terminada. Trata-se do primeiro paradigma utilizado em desenvolvimento de software, com seu uso instituído no final da década de 1960. Atualmente, variações deste paradigma são normalmente utilizados na indústria de construção civil e de manufatura de produtos, como explica Rajlich (2006, pág. 69).

Por se tratar de um processo sequencial, o paradigma Waterfall apresenta baixa robustez a mudanças nos requisitos. Devido à sua execução em série, cada etapa é baseada totalmente das saídas da etapa anterior para gerar seus resultados. Se

houver mudanças significativas em saídas de etapas já executadas, há o risco de que todas as etapas que a seguem devam ser significativamente retrabalhadas.

Devido à alta volatilidade nos requisitos projetos de software, métodos baseados no paradigma Waterfall podem apresentar baixas taxas de sucesso para projetos de software. Uma ilustração concreta desta volatilidade é encontrada em Cusumano e Shelby (1997, p. 56): Segundo os autores, a lista de especificação de requisitos pode mudar em até 30% em projetos de software.

2.1.2.2. Paradigma Ágil

Em resposta à falta de robustez a mudança encontrada em processos do tipo Waterfall, processos mais resistentes a mudanças nos requisitos foram popularizados nos anos 1990 (Agile Allliance, [2001?]). Uma série destes processos pode ser agrupada sob o paradigma ágil, formalizada em 2001 no *Agile Manifesto* (Agile Alliance, 2001):

> *We are uncovering better ways of developing software by doing it and helping others do it. Through this work we have come to value:*
>
> *Individuals and interactions over Processes and tools*
>
> *Working software over Comprehensive documentation*
>
> *Customer collaboration over Contract negotiation*
>
> *Responding to change over Following a plan*
>
> *That is, while there is value in the items on the right, we value the items on the left more*

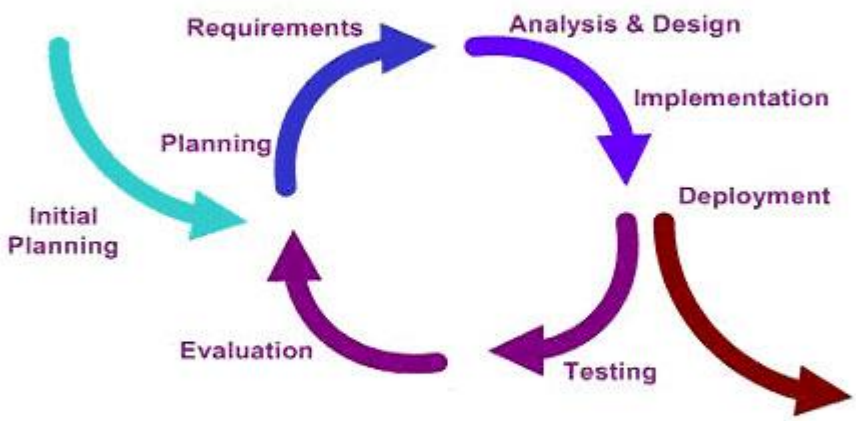
Os versos mais significativos do manifesto para o entendimento da filosofia ágil são o quarto e o sexto, que tratam das atividades de implementação e estabelecimento de requisitos do software: métodos ágeis favorecem a resposta a mudanças a seguir um plano e software que funcione a documentação extensiva.

O paradigma ágil é baseado no desenvolvimento incremental do produto final, gerando produtos intermediários em ciclos de desenvolvimento. Um ciclo é uma execução sequencial de cada uma das atividades acima descritas. Cada ciclo tem

duração de uma a quatro semanas, em geral, e o produto de cada ciclo é um programa utilizável, mesmo que este não satisfaça todos os requisitos finais do produto.

O fluxo de atividades de processos ágeis pode ser representado pela figura 7.

Figura 7 - Atividades do paradigma ágil



Fonte: eclipesource.com

Traduzindo as atividades requirements, analysis & design, implementation, testing e evaluation por requisitos, arquitetura, implementação, teste e avaliação, podemos perceber que um ciclo do paradigma ágil corresponde ao processo completo do paradigma Waterfall. A atividade de avaliação corresponde à análise do produto final de cada ciclo, que serve como entrada para determinar os requisitos do próximo ciclo. Uma vez que a avaliação afirmar que o produto está pronto, parte-se para a entrega (deployment).

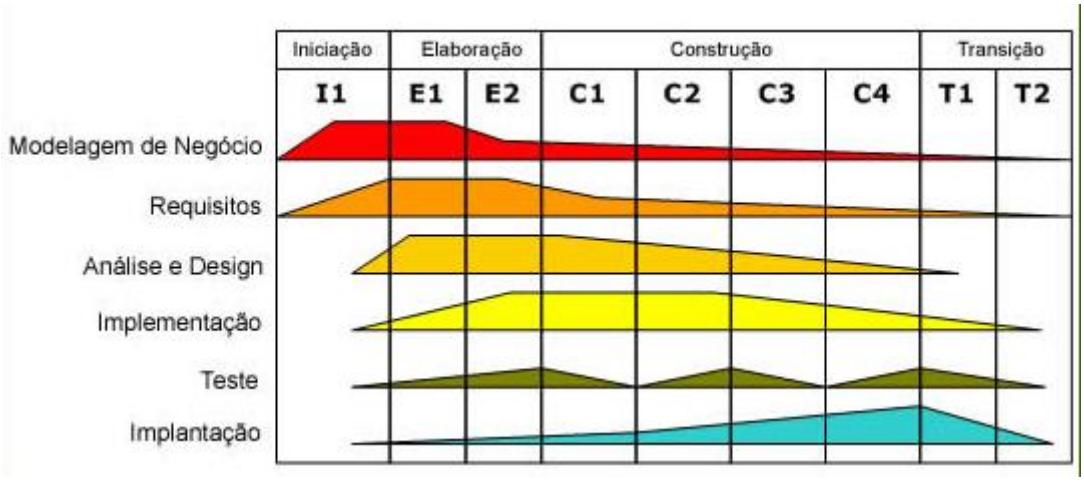
Desvantagens do paradigma ágil são sua difícil implementação em corporações, como mencionado em Barlow et al. (2011, p. 25) e a dificuldade de adaptação que muitos desenvolvedores enfrentam ao tentar mudar de processos baseados no paradigma Waterfall. Além disso, é intuitivo pensar que a filosofia incremental pode enfrentar dificuldades para produzir programas de grande complexidade e com ênfase em performance, devido a dificuldades de alterar a arquitetura significativamente entre ciclos. Uma solução proposta para esta última desvantagem é a geração de produtos que ataquem os maiores riscos previstos do projeto no início deste, para que não surjam dificuldades insuperáveis no final do projeto somente.

2.1.3. Paradigma de protótipos

O paradigma de protótipos mistura características dos dois paradigmas anteriores. Nele, há interações ou ciclos como em métodos ágeis, mas nem todas as atividades são executadas em cada ciclo, pois cada iteração possui uma ênfase em certa atividade. No início, forte prioridade é dada à definição de requisitos em detrimento da implementação, que adquire tração conforme o projeto avança.

Outra diferença fundamental em comparação com processos ágeis é a de que nem todos ciclos culminam com entregas de produtos prontos e os que terminam não satisfazem necessariamente todos os requisitos definidos até então.

Figura 8 – Ciclos de um método de protótipos



Fonte: infoescola.com

A figura 8 mostra o ciclo de um projeto feito com um método de protótipos. Nota-se que as cinco atividades fundamentais definidas acima estão presentes no método mostrado na figura 7 (a atividade de Modelagem de Negócios corresponde à caraterização da empresa cliente e não constitui formalmente uma etapa do desenvolvimento do software), mas que a intensidade com que cada uma delas aparece em cada ciclo (indicado na figura 8 por I1, E1, etc.) varia conforme o projeto avança em direção à transição. Há, portanto, diferenças fundamentais entre métodos de protótipo e métodos ágeis: Nos últimos, as atividades fundamentais estão presentes em cada ciclo em igual intensidade e cada iteração produz um programa completo segundo os requisitos estabelecidos no ciclo em questão. Já nos métodos de protótipo, as atividades são distribuídas de maneira heterogênea nos diferentes ciclos, chegando a ser completamente excluída de alguns, e os protótipos produzidos nunca respeitarão completamente os requisitos definidos até então, mas se focarão nos aspectos mais críticos e de maior risco.

Além do RUP, existem diversos outros métodos que possuem as características do paradigma de protótipos. Dentre estes, é possível destacar o processo Praxis, desenvolvido por Wilson de Pádua Paulo Filho, exposto em seu livro *Engenharia de Software* (Pádua Filho, 2003).

Críticas ao paradigma de protótipos incluem as desvantagens das duas metodologias citadas acima, dado que este inclui características de ambos. Notavelmente, pode-se citar o possível apego de programadores a protótipos desenvolvidos e a decorrente resistência em evoluir em direção ao produto final, e o gasto excessivo de tempo no desenvolvimento de protótipos que serão eventualmente descartados.

Vantagens incluem o contato do cliente com software desde cedo e a possível aprendizagem decorrente das diversas versões descartáveis produzidas antes de do produto final.

A literatura indica que o uso do paradigma de protótipos é mais adaptado a aplicações que oferecem intensa iteração com o usuário e onde equipes do projeto não possuem todo o conhecimento para gerar uma solução satisfatória no início do projeto.

2.1.4. Conclusões sobre o estudo dos paradigmas de desenvolvimento de software

Foram estudados nesta seção três paradigmas de desenvolvimento de software com o intuito de guiar a escolha do método para resolver o problema exposto acima. Para poder comparar os paradigmas entre si, foi estabelecido um modelo que descreve todos os métodos de desenvolvimento em função da presença de cinco atividades fundamentais presentes em qualquer projeto de TI. Na seção de resolução do problema deste trabalho um dos três paradigmas será escolhido para servir como base da solução a ser apresentada.

3. Estudo de soluções de software de contabilidade disponíveis no mercado

Nesta seção são estudados os programas de contabilidade disponíveis no mercado, com dois objetivos principais. Primeiramente, visa-se determinar se alguma das soluções encontradas pode resolver o problema descrito acima. Em segundo lugar, esta revisão objetiva também aprender as principais características dos programas disponíveis para eventualmente aplicá-las em uma solução individualizada produzida pelo autor.

3.1.1. Características relevantes para a escolha de um software

A base para este estudo é o trabalho de Elikai et al. (2007) em que são apresentados os principais fatores que devem ser levados em conta na escolha de um software de contabilidade.

Serão analisadas as seguintes dimensões dos produtos, que foram consideradas as três mais importantes segundo Elikai (2007, p. 27):

- Funcionalidades ou capacidades, incluindo flexibilidade de uso e possibilidade de customização
- Custo de aquisição e manutenção
- Compatibilidade com outros sistemas

Estabilidade financeira do produtor e suporte ao cliente não são consideradas aqui por ser tratarem dos fatores menos importantes encontrados no trabalho acima.

Elikai et al. também encontraram as dimensões mais importantes dentro de cada categoria. Por exemplo, foi determinado que os quesitos funcionalidade é o mais importante para os usuários de software de contabilidade. Mais quais são as funções mais importantes dentro desta categoria? Abaixo são listados os fatores mais relevantes das dimensões acima escolhidas, filtrados pelo gerente financeiro da Courrieros, que excluiu alguns itens e detalhou outros para tornar a análise dos programas mais objetiva.

- Funcionalidades ou capacidades
    - Funcionalidades: Variedade de relatórios gerados, possibilidade de backup dos dados
    - Flexibilidade de uso: Número máximo de clientes ou funcionários suportados, tamanho máximo do total de dados armazenados, possibilidade de gerenciar mais de uma empresa
    - Usabilidade: Facilidade de uso
    - Segurança: Segurança dos dados e permissões de acesso diferenciadas por usuário
- Custo de manutenção e aquisição
    - Custo total de uso excluindo treinamentos por um ano de software
- Compatibilidade com outros sistemas
    - Compatibilidade com diferentes versões do MS Windows
    - Compatibilidade com processadores com capacidade intermediária de processamento (1,5 – 1,9GHz)
    - Compatibilidade com softwares do MS Office para importação e exportação de dados

Agora que já são conhecidas as características importantes de um software de contabilidade, é necessário decidir quais serão os produtos analisados.

3.1.2. Escolha e descrição dos programas analisados

Aqui serão escolhidos os produtos que serão alvo da análise. Esta decisão será tomada de maneira pouco formal, mas eficiente e, espera-se, efetiva: Serão buscados na internet comparativos de softwares de contabilidade para pequenas empresas, e então selecionados os produtos com mais forte presença nos artigos encontrados. Serão usadas fontes de boa reputação no mercado de softwares, mas não será feita uma análise dos métodos de escolha dos produtos comparados.

Business Insider (Angeles, 2011, p. 1)

- Quickbooks Online
- Wave Accounting

- Freshbooks

PCMAG (Wilson, 2011, p. 1)

- Quickbooks Premier
- Wave Accounting
- Freshbooks
- Billing Boss
  Intuit Billing Manager
- Outright
- Zoho Invoice

Forbes (Marks, 2014, p. 1)

- Quickbooks Online
- Xero
- Cheqbook
- Kashoo
- Wave
- Zoho Books
- FreshBooks

Accounting Software Review (2014, p. 1)

- Quickbooks Pro
- Sage
- AccountEdge
- Cougar Mountain
- CYMA
- Bookkepper
- Business2Go
- Express Account

Uma simples contagem dos programas presentes nas listas acima permite chegar aos seguintes softwares para análise: Wave Accounting, Freshbooks, Quickbooks Online

e Zoho Account. Abaixo é apresentada uma breve descrição de cada um dos programas.

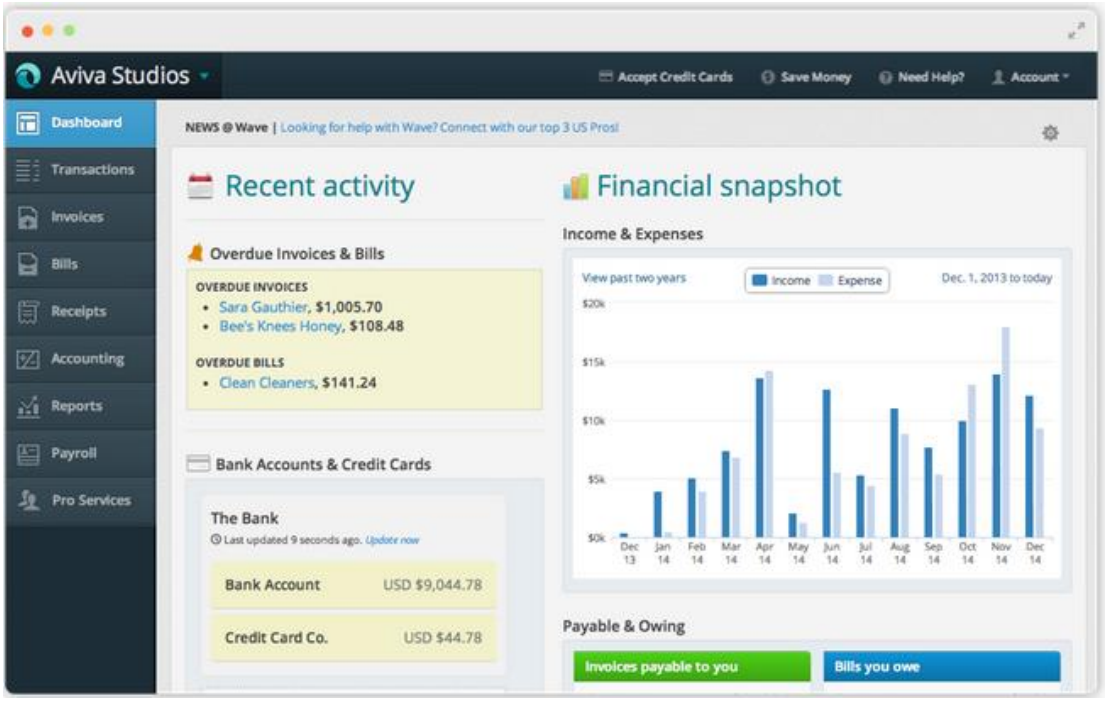
O Wave Accounting é uma solução de software de contabilidade da Wave Apps, empresa que oferece produtos de pagamentos e geração de boletos para pequenas empresas. Trata-se de um programa totalmente gratuito para o usuário, que em troca deve aceitar receber propagandas em alguns momentos quando está usando o programa. Foi projetado para empresas com 9 funcionários ou menos, profissionais independentes ou consultores, e não apresenta uma grande riqueza de funcionalidades, sendo voltado majoritariamente para facilitar as operações de cobrança e pagamentos.

O aplicativo Quickbook Pro é produzido pela reputada empresa Intuit, gigante do mercado e pioneira na produção de programas para gestão financeira pessoal, com valor de mercado de aproximadamente 26 bilhões de dólares. O Quickbook Pro é um software de contabilidade com um grande número de funcionalidades como relatórios variados, serviços de pagamento e cobrança e cálculo de valor agregado para questões fiscais.

O Zoho Books oferece serviços semelhantes aos do Wave Apps, mas não conta com propagandas e é portanto pago. Pode ser integrado com serviços de pagamento online como PayPal para facilitar a cobrança de clientes e conta também com um módulo de sincronização bancária que permite criar uma imagem do saldo da conta corrente do usuário dentro do programa e classificar despesas para melhor gestão de gastos.

Por último, o Freshbooks é um serviço online projetado especialmente para donos de empresas sem experiência em contabilidade. Não oferece um grande número de funcionalidades, mas foi eleito pelo site accounting-software-review como o melhor programa na categoria de aplicativos de contabilidade para pequenas empresas.

Figura 9 - Tela inicial do Wave Accounting



Fonte: www.waveapps.com

Figura 10 - Tela de sincronização de conta correte Zoho Books

Fonte: www.zoho.com

4. Estudo das tecnologias de desenvolvimento

Nesta seção serão analisadas as principais tecnologias de desenvolvimento de software disponíveis no mercado. Por tecnologias entende-se a linguagem de programação usada para o criar o produto.

Uma restrição é colocada sobre as tecnologias: Estas devem poder ser utilizadas em plataformas Windows, presentes nos computadores da Courrieros, e devem permitir a escrita de arquivos em formato .xls no sistema operacional, para permitir a geração de relatórios Excel.

Para estudo, serão consideradas as linguagens C#, VB, VBA e C++. A imposição desta restrição de linguagens analisadas é simples: São as únicas que possibilitam o desenvolvimento de aplicações com interface gráfica para o sistema operacional Windows, como afirma a própria Microsoft. (Chapter 3, p. 1). Esta seção é baseada em artigo de autoria da Microsoft intitulado “Choosing Windows Development Technologies”, que compara as cinco possibilidades de linguagens segundo os aspectos de performance e produtividade do programador. ([201-], p. 1). Esta parte da revisão da literatura é também dividida segundo estes critérios. A linguagem VBA será deixada de lado por enquanto, pois não permite o desenvolvimento de aplicações independentes para Windows.

4.1. Eficiência do programa

A linguagem C++ é a de maior performance dentre todas. A necessidade de gerenciar memória em aplicações desta tecnologia é ao mesmo tempo uma benção e uma maldição: O ganho em performance é considerável, mas a necessidade de gerenciar os objetos não usados pelo código pode gerar sobreuso de recursos e levar eventualmente à falha do programa.

C# e VB são as próximas na fila em termos de performance do programa escrito, sem nenhuma vantagem significativa para algum dos lados, como afirma Jerry Nixon,

desenvolvedor da MS, em artigo sobre o assunto (sem data, p. 1), e a escolha entre as duas seria baseada em preferências de cada programador.

4.2. Produtividade do programador

Este aspecto representa o quão rápido um mesmo programador com conhecimentos iguais de cada linguagem consegue escrever um programa com as mesmas funcionalidades, em cada uma das tecnologias escolhidas.

A produtividade depende de diversos fatores, ainda segundo Nixon: a sintaxe da linguagem, a disponibilidade de recursos de ajuda para uma das tecnologias e o estado dos compiladores disponíveis para cada uma das opções. Na comparação VB – C#, a ganhadora é C#, segundo Nixon. O desenvolvedor afirma que, apesar de VB apresentar sintaxe menos densa e mais produtiva, a disponibilidade de recursos de ajuda online é muito maior para C#, o que a torna vencedora.

Na comparação entre C# e C++, a clara vencedora é C#: Segundo Bill Kratochvill, escrevendo na revista de desenvolvedores da Microsoft, a linguagem C# foi desenvolvida para favorecer produtividade sobre performance e é fortemente integrada com o compilador Visual Studio (2014, p.1), o que significa menor necessidade de digitar os nomes de variáveis e menos tempo corrigindo erros de sintaxe e lógica da aplicação. Conclui-se que C# é a mais produtiva das linguagens estudadas.

4.3. Comparação VBA – Outras linguagens e VSTO

A linguagem VBA é de natureza particular, pois só pode ser utilizada em uma aplicação do Microsoft Office (como Excel ou Power Point). O VBA é considerado como alternativa para o desenvolvimento da aplicação Courribilidade pois esta poderia ser baseada totalmente em Excel, que atuaria tanto como base de dados como interface de exibição de relatórios e outras métricas. Soluções desenvolvidas em outro ambiente (como o compilador Visual Studio) são independentes do Microsoft Office e autossuficientes, mas possuem o ponto negativo de possuírem integração mais lenta

e difícil com este. O quadro 1, de autoria da Microsoft mostra as principais diferenças entre as duas alternativas.

Quadro 1 - Comparação VBA x VSTO (Outras linguagens)

VBA	**C#, VB ou C++ (Visual Studio Tools for Office)**
Código presente no arquivo da suíte Office em questão	Código separado do documento Office, sua existência independe deste
Trabalha com elementos do Office e APIs de VBA	Pode trabalhar com elementos do Office e o framework .NET.
Projetado para um desenvolvimento de soluções simples	Projetado para alta segurança e a possibilidade de usar o Visual Studio par a desenvolvimento
Funciona bem com soluções que requerem alta integração com produtos do Office	Funciona bem com soluções que requerem recursos do Visual Studio e do framework .NET
Possui limitações de segurança e distribuição para empresas	Projetado para uso em empresas

Fonte: Microsoft Development Network

Com a leitura do quadro, percebe-se que a linguagem VBA é construída para atuar principalmente sobre algum produto específico do Office, função que esta desempenha de maneira eficiente. Para produtos que visam comunicar com os programas desta suíte, mas que também objetivam oferecer outras funcionalidades independentes, projetos no Visual Studio são mais adaptados.

Uma das grandes vantagens do VBA, como mostrado no quadro 1, é a comunicação simples e eficiente com elementos que compõe um documento Office. O VBA possibilita, por exemplo, fazer referência direta a uma célula do Excel ou linha do Word do código, o que torna a experiência de programação intuitiva e mais produtiva. Isso não era possível usando C# ou outras linguagens do Visual Studio até 2003, quando a Microsoft lançou o produto Visual Studio Tools for Office, que permite a integração do código escrito em C# ou VB com elementos do Office como células ou colunas do Excel.

Esta solução torna o uso do Visual Studio superior ao VBA para uma grande parte de aplicações, pois a vantagem de comunicação com elementos *core* dos documentos Office que possuía o VBA passou a ser compartilhada com Visual Basic e C#.

Ainda há uso para o VBA, em especial porque as ferramentas do VSTO são pagas e só estão disponíveis para usuários da versão *Professional* do Visual Studio, enquanto todos os programas Office possuem modos de desenvolvedor que aceitam código VBA sem custos adicionais. Além disso, pode ser interessante entregar um documento do Office com código embutido ao invés de ter de produzir dois programas separados. Por último, o VBA é a única linguagem que permite a gravação de macros, além de permitir análises *ad-hoc* de maneira rápida, o que não é possível para programas do Visual Studio, que exigem a escrita um programa completo para interagir com documentos do Office.

4.4. Conclusão sobre as tecnologias disponíveis

A análise das linguagens de programação disponíveis para Windows tratou da performance e da produtividade do programador para quatro tecnologias disponíveis: C++, C#, VB e VBA. Esta última foi analisada separadamente pelas limitações do tipo de aplicações que pode produzir.

Os resultados da revisão mostram que, em termos de performance do programa, C++ é largamente superior a C# ou VB, sendo que entre estas últimas não há diferença apreciável de desempenho.

Em termos de produtividade do programador, C# se destaca sobre VB, que se destaca sobre C++. Este resultado se deve principalmente à disponibilidade de recursos de ajuda online existentes para a tecnologia C#, largamente superior aos de VB. Em termos de produtividade intrínseca à linguagem, não há diferença significativa entre VB e C#, e estas duas são superiores neste quesito a C++, muito devido à necessidade de gerenciar memória que esta apresenta.

Parte III – Resolução do problema

1. Introdução

Esta parte do trabalho trata da resolução do problema apresentado na introdução, e é estruturada em duas seções lógicas principais.

Em um primeiro momento serão escolhidas e descritas as ferramentas usadas para solucionar a questão proposta. Isto se traduz em escolher se um novo sistema será desenvolvido ou se uma solução disponível no mercado será usada, decidir sobre o melhor método de desenvolvimento de software, que será escolhido com base nos estudos sobre metodologia feitos na revisão da literatura e finalmente selecionar a tecnologia para implementação do Courribilidade.

Na sequência, passa-se à aplicação destas ferramentas e à apresentação dos resultados com estas obtido.

2. Escolha e descrição dos métodos e tecnologias

Nesta seção são escolhidos a metodologia, o método que a implementa e a tecnologia a serem usados para solucionar o problema.

A metodologia é selecionada dentre as três estudadas na seção precedente. Uma vez tomada esta decisão, um método baseado nesta será selecionado. É importante frisar que podem existir diversos métodos que sigam a mesma metodologia, e que a escolha no contexto desse trabalho se fará baseada principalmente na disponibilidade de material didático sobre o método.

Trata-se de um critério de escolha simples e que ignora provavelmente características individuais de cada método, as decidiu-se por este caminho pela crença de que a decisão da metodologia é de maior importância.

Já a escolha da tecnologia será baseada no estudo das linguagens de programação disponíveis para construir aplicações Windows, a saber C#, C++, VB e VBA. A decisão será tomada tentando alinhar restrições esperadas do produto, como requisitos de performance e tempo limite para desenvolvimento, com as características de cada linguagem.

2.1. Desenvolver ou comprar o produto?

Neste item é tomada a decisão da estratégia de resolução do problema. A definição deste último deixa claro que se trata de um problema de software. Para resolvê-lo, deve-se primeiramente decidir se um programa será comprado do mercado ou desenvolvido do zero.

Esta escolha será baseada na capacidade de cada uma das alternativas em produzir as características necessárias para resolver o problema. Para isso, é necessário revisitar a definição deste. Voltando à parte I, nota-se que o problema pode ser descrito por 4 falhas do sistema atual:

- Falta de confiabilidade dos dados do sistema: Valores de propriedades incorretos devido à falta de validação de dados
- Falta de coerência dos dados atuais: Problema de referência a entidades do sistema como nome de empresas, funcionários ou endereços
- Dificuldade de geração automática de relatórios
- Falta de integração entre os diversos sistemas

Uma simples matriz de decisão será usada para a escolha. Todos os critérios são considerados como igualmente importantes, e as notas de cada alternativa para cada critério variam de 1 a 3. O software de mercado utilizado para a comparação foi o QuickBooks Pro, julgado pelo autor como o mais completo dentre os encontrados. Os resultados são mostrados na tabela 2.

Tabela 2 – Comparação Desenvolvimento Próprio x Compra de software

	Confiabilidade Dos dados	**Coerência Dos dados**	**Geração de relatórios**	**Integração entre**	**Nota Final**
Desenvolvimento Próprio	3	3	3	3	12
Compra de software existente	3	3	2	2	10

Fonte: Do autor

A alternativa de desenvolvimento próprio é a vencedora, pois oferece melhor possibilidade de geração de relatórios e de integração de sistemas. Estas notas de devem principalmente à possibilidade de personalização que o desenvolvimento próprio oferece, uma vez que softwares de prateleira oferecem pouca ou nenhuma flexibilidade quanto às suas funcionalidades.

É importante notar que não foi usado um critério de custo de aquisição ou de manutenção para a decisão. Isto é uma particularidade do contexto em que este o problema e este trabalho se inserem: Primeiramente, os benefícios estimados pelo cliente possuem valor largamente superior ao preço cobrado pelo mais caro dos softwares encontrados na revisão da literatura, e este seria, portanto, irrelevante. Em segundo lugar, o custo para a Courrieros no caso desenvolvimento de

desenvolvimento próprio é nulo. Trata-se claramente de uma particularidade do trabalho, que não pode ser extrapolada para casos de necessidades futuras.

2.2. Escolha da metodologia e método

2.2.1. Método de escolha: Matriz de Decisão

O método de decisão usado será o de Matriz de Decisão. Trata-se de um método simples e de fácil aplicação, e deve-se questionar se não é uma escolha simplista. Mas uma breve revisão da literatura nos mostra que não existem métodos de escolha de metodologia mais sofisticados do que o bom senso, como indica Glass (2004, p. 19). Isso significa que o trabalho de classificar metodologias e problemas de desenvolvimento e então determinar qual classe de métodos se adapta melhor a cada classe de problemas ainda não foi feito.

Na impossibilidade de usar métodos mais sofisticados, recorre-se então à matriz de decisão para efetuar a ligação. Mas o problema não está resolvido: que critérios usar na matriz, e como dar a cada um deles pesos diferentes? Este é o objeto dos parágrafos seguintes.

2.2.2. Fatores de decisão

Aqui são definidas características esperadas do produto que guiarão a decisão. A literatura não apresenta materiais satisfatórios sobre o assunto, como mencionou Glass (2004, p. 19). Fontes de menor confiabilidade serão portanto usadas. Danie Krige, funcionário da Amazon Web Services (AWS) sugere a seguinte lista de critérios:

- Conhecimento dos riscos: Os riscos do projeto estão claros?
- Conhecimento do escopo: As fronteiras do projeto estão claras ou existe a possibilidade de o desenvolvedor se perder em detalhes??
- Maturidade dos requisitos: O cliente conhece todos os requisitos que quer?
- Custo de desenvolvimento: Qual é o custo da qualidade e o custo de ownership do projeto?

- Relacionamentos dentro do time: Existe distância geográfica entre os membros do time? O time está focado em trabalho em equipe ou são os membros colaboradores individuais para o projeto?
- Flexibilidade dos stakeholders: Os stakeholders conseguem adaptar-se ao projeto e ao time?
- Experiência do time: Qual é o nível de capacidade de cada membro do time?
- Comprometimento dos clientes: Há alguém diretamente responsável pelo projeto na empresa cliente?
- Documentação: É necessária documentação a ser entregue ao cliente?
- Complexidade do programa: Quão complexo é o programa em termos de inovação?
- Patamar mínimo de tamanho do programa: Quão alto é este patamar?
- Estabilidade do ambiente do programa: Quão estável é este ambiente?
- Tamanho do time: Qual é o tamanho do time?
- Familiaridade com a aplicação: Trata-se de uma aplicação familiar ou totalmente nova?
- Linguagens de programação a serem usadas: depende-se de capacidades escassas para completar o projeto?
- Método de contrato: Preço fixado por entrega de produto ou determinado pelo tempo e pelo custo?
- Influência de terceiros quanto à escolha da metodologia: Existe pressão de terceiras partes para que o time use certo método?

Esta lista será tomada como base e filtrada para o problema que está sendo enfrentado e para o contexto de um trabalho de formatura. Obviamente, critérios de grupo de trabalho não se aplicam, pois o projeto será desenvolvido por uma única pessoa. Dimensões de custo também não serão consideradas, assim como questões contratuais. Obtêm-se então a seguinte lista, com critérios semelhantes agrupados.

- Capacidade de lidar com incertezas: A metodologia permite lidar com riscos ainda desconhecidos?

- Capacidade de lidar com mudanças de escopo: Os métodos permitem que se mude radicalmente o escopo do projeto sem impor grandes mudanças na arquitetura do projeto?
- Robustez quanto a mudança nos requisitos: A metodologia tem espaço para mudanças nos requisitos sem requerer grandes retrabalhos?
- Capacidade de permitir aprendizado durante o processo de desenvolvimento: A metodologia permite que o programador aprenda sobre as tecnologias envolvidas durante o projeto?
- Rastreabilidade e documentação: A metodologia permite que cada elemento do sistema pronto possua uma causa de existência bem definida nos requisitos? As decisões são bem documentadas?

Graduar cada um dos elementos da lista acima de 1 a 5, sendo 5 o valor que indica maior intensidade, como mostrado na escala abaixo, permitirá determinar o quão importante cada dimensão é para o projeto. Esta importância representará um problema que deverá ser endereçado pela metodologia.

- 1: Irrelevante ou não presente no projeto
- 2: De pequena importância ou presença esporádica no projeto
- 3: De importância ou presença a no projeto
- 4: De forte presença ou importância no projeto
- 5: De importância crucial para o projeto

Por exemplo, espera-se que o projeto tenha requisitos estáveis, por se tratar essencialmente de um sistema de gerenciamento de banco de dados. Pode-se, portanto, classificar o critério Volatilidade de requisitos como tendo intensidade 1, dado que este fator não está presente no projeto. Trata-se de uma dimensão que as metodologias estudadas não terão que resolver, e o fato de uma destas poder endereçar melhor este desafio é totalmente irrelevante.

Por outro lado, o autor não possui fortes conhecimentos prévios sobre a linguagem de programação usada para implementação da solução: pode-se dar intensidade 4 ou 5 para este critério pois a aprendizagem da tecnologia durante o processo certamente

estará presente nesse problema. Isso significa que esta é uma dimensão prioritária na escolha da abordagem de resolução do problema. Esta avaliação dos critérios nada mais é do que a definição dos pesos da matriz de decisão.

Tabela 3 – Pesos da matriz de decisão

Critério	Peso
Resiliência a riscos desconhecidos	2
Volatilidade de requisitos	4
Volatilidade da arquitetura	2
Necessidade de documentação	4
Aprendizado durante o processo	5

Fonte: Do autor

2.2.3. Adaptação das metodologias aos critérios: Notas da Matriz de Decisão

Aqui são definidos os graus de alinhamento das metodologias com os critérios apresentados acima, ou, em outras palavras, as notas de cada uma delas na matriz de decisão. Será usada uma escala de 1 a 5, onde 5 representa a mais alta adaptação.

Como exemplo, considerando o alinhamento entre Volatilidade de Requisitos e a metodologia ágil, a nota é 5, pois, como foi exposto acima, métodos ágeis são adaptados a projetos de alta volatilidade de requisitos. Processos do tipo Waterfall possuem pouca robustez para lidar com requisitos variáveis e recebem nota 1 para esta dimensão. Juntando as notas das alternativas com os pesos dados a cada um dos critérios obtêm-se a matriz de decisão, representada na tabela 4. A Metodologia vencedora é a metodologia de protótipos, ganhadora por pouco sobre os métodos ágeis, como indicado na linha Resultado Final da matriz.

Tabela 4 – Matriz de decisão da metodologia (Continua)

Critério	Peso	Waterfall	Agile	Protótipos
Resiliência a riscos desconhecidos	2	1	5	3
Volatilidade de requisitos	4	1	5	4

Conclusão

Critério	Peso	Waterfall	Agile	Protótipos
Volatilidade da arquitetura	2	1	2	4
Necessidade de documentação	3	5	3	5
Aprendizado durante o processo	5	1	4	4
Resultado Final		28	63	65

Fonte: Do autor

Processo Waterfall são pouquíssimos adaptados ao tipo de projeto que será desenvolvido, em especial pois apresentam baixíssima resiliência a mudanças e assumem um conhecimento prévio das tecnologias que o autor não pode oferecer.

Entre a metodologia *Ágil* e o *Processo Unificado*, o resultado foi praticamente um empate. Métodos ágeis se mostram mais adaptados a lidar com volatilidade de requisitos, mas produzem menos documentação e são, portanto, menos propensos a oferecer rastreabilidade.

Prevaleceram os métodos do tipo UP, que se mostraram equilibrados em todos os critérios escolhidos. Como já foi mencionado na revisão da literatura, este equilíbrio é característica fundamental dos métodos UP, que se localizam no meio do espectro de agilidade, oferecendo elementos dos dois extremos.

Uma vez escolhida a metodologia, é necessário escolher o método que a implementa. Trata-se de uma decisão de caráter tático, em oposição à escolha que acaba de ser feita, que possui características estratégicas, mas ainda de suma importância, em especial para métodos de protótipos, que possuem uma grande variedade de métodos disponíveis no mercado. A subseção seguinte tratará deste assunto.

2.3. Escolha e descrição do método

2.3.1. Escolha do método

A escolha do método será baseada em um único critério, extremamente simples, mas na opinião do autor muito efetivo do ponto de vista prático: O número de projetos

desenvolvidos com a ajuda de cada método acessíveis ao autor. Para determinar esta disponibilidade, o autor atuou em duas frentes.

Primeiramente, efetuou-se uma pesquisa sobre os trabalhos de formatura publicados no site do departamento de Engenharia de Produção da Escola Politécnica da USP, buscando por temas semelhantes ao desta obra, a saber, especificação e desenvolvimento de softwares, e tomou nota dos métodos usados pelos alunos.

Em seguida, foram buscados na Biblioteca da Engenharia Elétrica e da Engenharia de Produção títulos que descrevessem métodos de protótipos, em especial métodos do tipo *Unified Process*. Foram pesquisadas obras que mostrassem, além dos conceitos teóricos dos processos, exemplos de uso destes, para tornar a aplicação do método por parte do autor mais simples e eficaz.

Após esta rápida busca por um método que implemente as ideias da filosofia de protótipos, decidiu-se por utilizar o método Praxis, do Engenheiro Wilson de Pádua Paula Filho. O Praxis foi utilizado em maior ou menor intensidade em todos os trabalhos de formatura do PRO que lidaram com especificação de software nos últimos três anos, tendo sido usado em sua totalidade por Michailovici (2012) e Sumares (2012), e a Biblioteca da Produção possui literatura satisfatória sobre o processo.

2.3.2. Breve descrição do Método Praxis

O Praxis é um método baseado em iterações. Cada iteração compreende um ciclo completo das atividades fundamentais da engenharia de software apresentadas na seção 2 da revisão da literatura deste trabalho (especificação de requisitos, implementação, etc). Estas tarefas fundamentais recebem o nome de fluxos. Dizer “Tarefa de Especificação de Requisitos” é totalmente equivalente a dizer “Fluxo de Requisitos”.

Cada iteração pode ser considerada como um mini-projeto *Waterfall*, cujo produto final é um protótipo. Quando o protótipo satisfizer todos os requisitos definidos pelo cliente, o projeto é considerado terminado.

Desta descrição deve-se depreender dois aspectos principais do Praxis (e de qualquer outro método de protótipos). Primeiramente, os fluxos serão executados em iterações, e existirão iterações até que o produto final for considerado satisfatório. Em segundo lugar, o produto final só é atingido na última iteração. Esta é uma diferença fundamental dos métodos ágeis para o Praxis: Enquanto esses produzem um programa pronto para uso ao final de cada *sprint*, o Praxis não visa obter um software funcional antes do fim. A finalidade de protótipos é a de provar um conceito ou superar um risco, e não de prover funcionalidade, e por isso muitos são descartados após sua produção

Mas qual é a estrutura básica de uma iteração? Qual deve ser o fluxo com maior intensidade de trabalho em cada uma delas? O que esperar como resultado de cada ciclo? As respostas são apresentadas abaixo.

2.3.2.1. Estrutura de uma iteração

A imagem da capa da terceira edição do livro Engenharia de Software (Pádua Filho, 2010) traduz de maneira simples e expressiva todo o processo Praxis, e por isso está representada na figura 11. Uma iteração é a execução sequencial de cada um dos fluxos mostrados na ilustração, com duração média de uma semana. A curta duração é fundamental, pois força os engenheiros a priorizarem suas atividades e impossibilita o dispêndio excessivo de tempo com uma só atividade, o que aproximaria o processo da metodologia *Waterfall* e eliminaria suas vantagens.

2.3.2.2. Objetivos de uma iteração

É fundamental que cada iteração possua um objetivo, sem o qual não há como medir seu sucesso. Para atribuir metas ao trabalho, o Praxis agrupa iterações em fases, e define a meta de cada fase como sendo um documento que formaliza os resultados atingidos pelas iterações. Como exemplo, a terceira fase do método possui como *deliverable* o documento ERSw (especificação dos requisitos do software) totalmente completo. Quando todas as seções deste documento tiverem sido preenchidas corretamente, a quarta fase terá início.

O número de repetições é então variável: Devem ser feitas tantas iterações quanto forem necessárias para atingir o objetivo da fase, respeitando o limite de duração de cada ciclo. De maneira geral, uma fase precisa de três a cinco iterações para atingir seus objetivos (Pádua Filho, 2003, p. 1)

2.3.2.3. Quanto tempo atribuir a cada fluxo em uma iteração?

Chega-se ao problema de priorização de uso do tempo por fluxo. Quanto esforço deve ser dado ao fluxo de implementação na primeira fase do projeto? Provavelmente não muito, dado que pouco se conhece sobre o sistema a ser desenvolvido. Por outro lado, o tempo gasto com testes ao final do projeto deve proporcionalmente grande, uma vez que todos os requisitos já foram estabelecidos.

Como exemplo, o quadro 2 mostra as tarefas da fase de Ativação, a primeira do método. Nota-se que há três atividades ligadas ao fluxo de requisitos, mas somente uma (opcional) de implementação. É de se esperar que a proporção de tarefas de implementação cresça com a maturidade do produto.

Quadro 2 – Tarefas da fase de Ativação

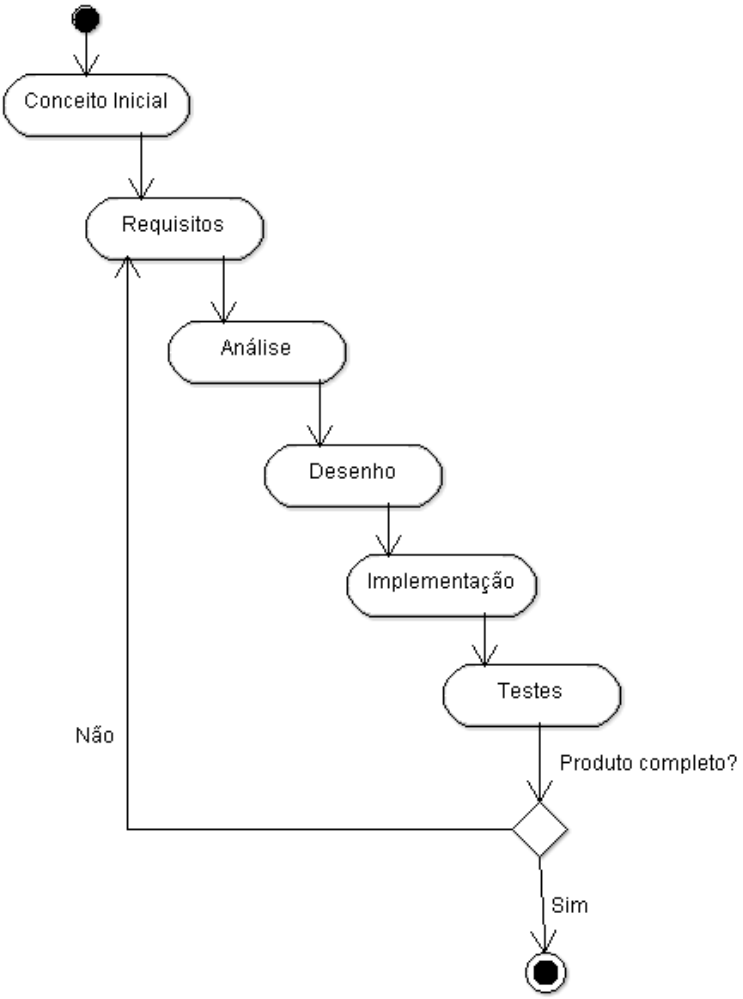
Fluxo	**Tarefa**
Requisitos	Determinação do contexto
Requisitos	Definição do escopo
Requisitos	Definição dos requisitos (preliminar)
Análise	(Não há atividades)
Desenho	Esboço arquitetônico
Implementação	Prototipagem inicial (se necessário)
Testes	(Não há atividades)

Fonte: Adaptado de Pádua Filho (2003)

A figura 12, já apresentada na revisão bibliográfica, mostra a intensidade de cada um dos fluxos durante as fases, cujos nomes estão indicados na primeira linha. (Por ser baseada no processo RUP, as fases têm nomes diferentes)

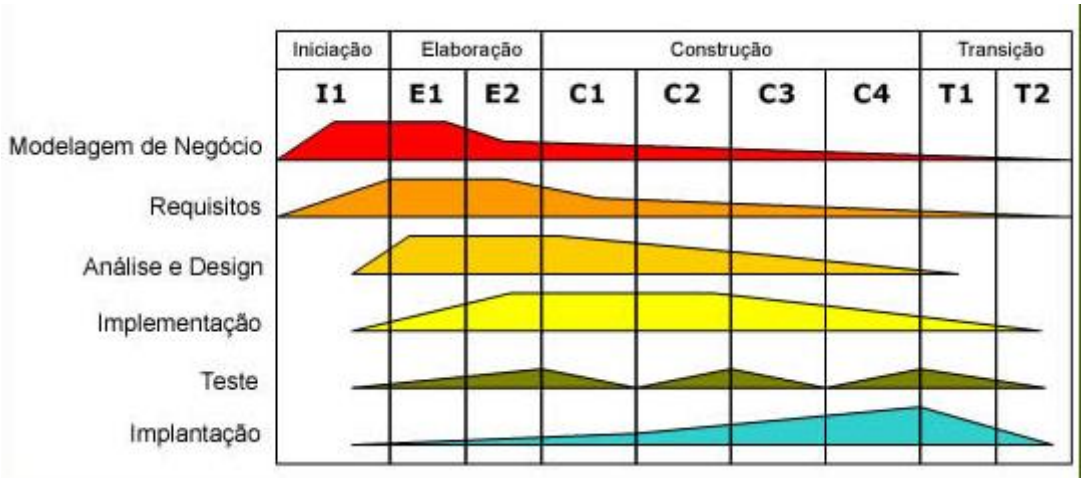
Para solucionar o problema de priorização, o Praxis define scripts para cada uma das fases. Scripts são roteiros que descrevem as tarefas a serem executadas em cada iteração daquela fase. Se as tarefas definidas forem em sua maioria pertencentes ao fluxo de especificação de análise, então a fase será focada neste aspecto.

Figura 11 - Atividades no Praxis



Fonte: Adaptado de Pádua Filho (2010)

Figura 12 - Distribuição da intensidade de trabalho com o tempo no Praxis



Fonte: infoescola.com

Agora que já se conhece um pouco mais sobre o Praxis, este será usado para resolver o problema da Courrieros. As seções de 3 a 5 tratam da aplicação do método no contexto da criação do software Courribilidade, começando pela especificação de requisitos.

3. Especificação dos requisitos

3.1. Requisitos na Engenharia de Software

Assim como em qualquer projeto de engenharia, requisitos são a base sobre a qual se desenvolve um projeto de software. Nesta disciplina, requisitos podem ser divididos em requisitos funcionais e não-funcionais. Requisitos funcionais indicam o que o software deve fazer, enquanto os não-funcionais definem restrições sobre como estas funções podem ser executadas. Por exemplo, o produto deste trabalho, o programa Courribilidade, possui como requisito funcional executar o backup de dados essenciais no software *Dropbox*. Um requisito não-funcional ligado a este requisito funcional pode ser que o tempo para executar a gravação dos dados no servidor não pode superar 5 minutos.

Como afirma Pádua Filho (2003, p. 87), uma boa engenharia de requisitos é de fundamental importância para garantir o sucesso de um projeto de software. Mesmo na aplicação de métodos ágeis, é fundamental que os requisitos do ciclo de desenvolvimento em questão sejam precisamente definidos para que este possa ser bem-sucedido. Ainda de acordo com o mesmo autor (2003, p. 90), especificações de requisitos apresentam certas características que definem sua qualidade. São estas:

- **Correção**: Os requisitos da especificação são realmente requisitos desejados no produto final
- **Precisão:** Os requisitos apresentam definição precisa e objetiva, sem ambiguidades ou margens para interpretação subjetiva
- **Completude:** Nenhuma funcionalidade pode ser ignorada
- **Consistência:** Requisitos não apresentam conflitos entre si, é possível implementar todos os requisitos ao mesmo tempo
- **Priorização:** Cada requisito deve ser classificado de acordo com sua
- **Verificabilidade:** Todos os requisitos definidos na especificação devem ser verificáveis, i.e., deve ser possível determinar se certo requisito foi implementado

- **Mutabilidade:** É possível mudar requisitos da especificação sem grande esforço, mantendo consistência e completude
- **Rastreabilidade:** Cada requisito deve poder ser rastreado desde suas consequências e permitir o rastreamento de suas origens

3.2. Requisitos no Praxis

3.3. Estrutura desta seção

A estrutura desta seção baseia-se no papel dos requisitos no processo Praxis. Em um primeiro momento serão apresentados os benefícios que são esperados pela gerência da Courrieros do sistema desenvolvido.

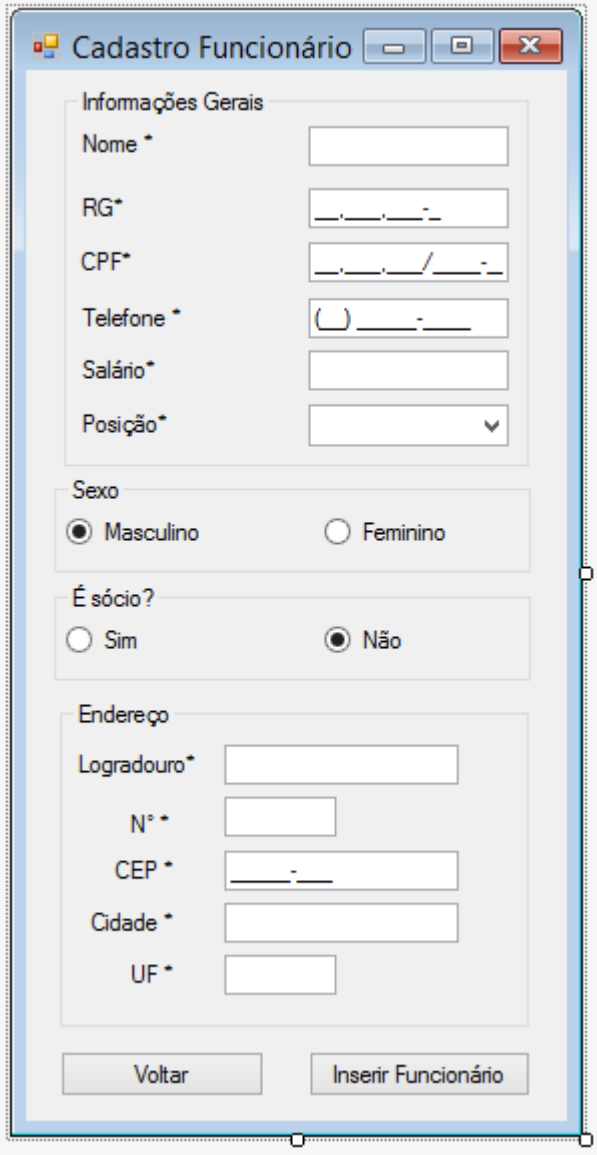
Em seguida são apresentados os casos de uso do sistema, que são apresentados em um diagrama de contexto UML. Nesta seção está a base sobre a qual será construída o sistema, pois aí são definidas e detalhadas formalmente todas as funcionalidades que são esperadas do sistema.

Na sequência, parte-se para a enumeração e desenho das interfaces do Courribilidade. Existem dois tipos de interface segundo o processo de desenvolvimento adotado: Interfaces de usuário e interfaces externas. Interfaces de usuário são aquelas que os atores confrontam quando do uso do sistema. Um bom exemplo é a tela de cadastro de funcionários, reproduzida na figura 13. Interfaces externas são aquelas usadas pelo sistema para se relacionar com seu ambiente. Um exemplo claro é a comunicação com o aplicativo Excel.

Por último são apresentados os requisitos não-funcionais do Courribilidade. Se os casos de uso representam o O QUÊ do sistema, os requisitos não-funcionais são o COMO. Exemplo de um requisito deste tipo é a velocidade com que o sistema deve gerar relatórios em formato Excel.

É importante notar que a estrutura aqui apresentada é adaptada do Praxis, mas não segue estritamente a lógica por ele proposta para facilitar o entendimento do texto.

Figura 13 - Cadastro de funcionário



Fonte: Do autor

3.4. Resultado do fluxo de requisitos

3.4.1. Benefícios esperados do sistema

Os benefícios esperados do Courribilidade são definidos na primeira fase do Praxis, a de ativação, e é o primeiro resultado concreto do fluxo de requisitos. Trata-se de uma enumeração das melhorias esperadas do sistema pelo cliente do projeto, e, portanto, totalmente correlacionado com os problemas que este cliente enfrenta. No caso da

Courrieros, esta correlação é evidente quando se compara a seção de definição do problema com a o quadro 3.

Quadro 3 – Benefícios esperados do sistema

Número de ordem	Benefício	Valor para o cliente
1	Maior precisão nos dados operacionais e financeiros	Essencial
2	Maior agilidade na geração de relatórios	Essencial
3	Relatórios mais inteligíveis	Desejável
4	Maior precisão na avaliação dos entregadores	Essencial
5	Maior abrangência dos relatórios de venda por cliente	Essencial
6	Maior agilidade na geração de relatórios de venda por cliente	Essencial
7	Maior espectro de indicadores de avaliação de entregadores	Desejável
8	Maior espectro de indicadores de avaliação de clientes	Desejável
9	Geração automática de notas fiscais	Opcional
10	Geração automática de notas fiscais	Opcional
11	Comunicação com o aplicativo da Courrieros	Opcional
12	Geração automática de protocolos de entrega	Opcional
13	Georeferenciamento de entregas	Desejável

Fonte: Do autor

É importante notar também que todos os benefícios possuem também uma classificação de valor para o cliente, que indica o quanto cada melhoria é importante para a empresa contratante. No caso da Courrieros, os benefícios ligados à correção e precisão dos dados são os de maior valor, enquanto que os benefícios ligados à

integração com outros sistemas, como emissão de notas fiscais ou comunicação com o aplicativo são considerados pelo gerente financeiro da Courrieros como opcionais apenas.

A definição de importância para os benefícios se mostra essencial em projetos em que os recursos de desenvolvimento são escassos, pois ajuda a priorizar funcionalidades e a entregar o máximo de valor com o tempo e capital disponível.

3.4.2. Casos de uso e atores do sistema

Nesta subseção são definidos os casos de uso do sistema, i.e., as representações das funções do produto na forma escrita ou diagramática, assim como os atores que sobre eles agem. Atores são representações das classes de usuários que agem sobre o sistema, e modelam os papéis que cada usuário tem no seu relacionamento com o sistema. Um usuário humano pode ter mais de um papel e ser representado por mais de um ator. Por exemplo, o sr. Biselli, gerente financeiro da Courrieros, pode ora agir sobre o sistema como administrador deste, adicionando ou excluindo usuários, ora registrar entregas e, portanto, agir como controlador.

A lista de casos de uso é gerada iterativamente no Praxis, começando na fase de Concepção e terminando na primeira iteração da fase de Levantamento de Requisitos. O resultado final das atividades é mostrado nesta subseção.

Quadro 4 - Casos de uso do Courribilidade (Continua)

Número de ordem	Caso de uso	Descrição
1	Inserir usuário	Inserir usuário que terá acesso ao Courribilidade
2	Apagar usuário	Apagar usuário que tem acesso ao Courribilidade
3	Editar usuário	Alterar dados de usuário que tem acesso ao Courribilidade
4	Inserir cliente	Inserir cliente na base de dados do Courribilidade

Continuação

Número de ordem	Caso de uso	Descrição
5	Apagar cliente	Apagar cliente presente na base de dados do Courribilidade
6	Gerar relatório de clientes	Gerar relatório gerencial de clientes
7	Editar cliente	Alterar dados de cliente presenta na base de dados do sistema
8	Inserir entrega	Inserir na base de dados de uma entrega
9	Editar entrega	Editar informações de uma entrega presenta na base de dados
10	Apagar entrega	Remover da base de dados entrega lá presente
11	Gerar relatório de entregas	Gerar relatório gerencial de entregas/vendas
12	Inserir funcionário	Inserir funcionário no banco de dados do sistema
13	Apagar funcionário	Remover funcionário do banco de dados do sistema
14	Editar funcionário	Editar informações presentes no banco de dados sobre um dado funcionário
15	Gerar relatório de funcionário	Gerar relatório gerencial sobre funcionários
16	Gerar relatório de resultados	Gerar relatório de resultados econômicos de um dado período
17	Gerar relatório de fluxo de caixa	Gerar relatório de fluxo de caixa de um dado período
18	Gerar balanço patrimonial	Gerar relatório de balanço patrimonial da empresa
19	Georeferenciamento de entregas	Cálculo das coordenadas geográficas de cada ponto de coleta e entrega
20	Emissão de nota fiscal	Emissão de nota fiscal para cliente

Conclusão

Número de ordem	Caso de uso	Descrição
21	Backup automático de dados	Backup de dados relevantes para sistema de armazenamento externo (cloud ou não)

Fonte: Do autor

É importante frisar que os casos de uso são a tradução em funções do sistema dos benefícios esperados. É fácil notar isto: No quadro 4 acima observamos o caso de uso *Geração de relatório de entregas*, que é claramente decorrente dos benefícios 5 e 6.

Quadro 5 - Enumeração e descrição dos atores do sistema

Número de ordem	Ator	Definição
1	Gerente	Funcionário responsável pela administração da empresa
2	Controlador de entregadores	Funcionário responsável pela alocação de ciclistas para cada entrega e pelo registro de entregas no livro de controle da empresa
3	Aplicativo	Aplicativo de gestão de entregas e rastreamento de ciclistas
4	Windows	Sistema operacional onde será instalado o programa
5	Sócio	Sócio da empresa Courrieros
6	Entregador	Funcionário que executa entregas
7	Sócio-Investidor	Investidor que não trabalha na empresa
8	Administrador do sistema	Funcionário responsável pela manutenção do Courribilidade

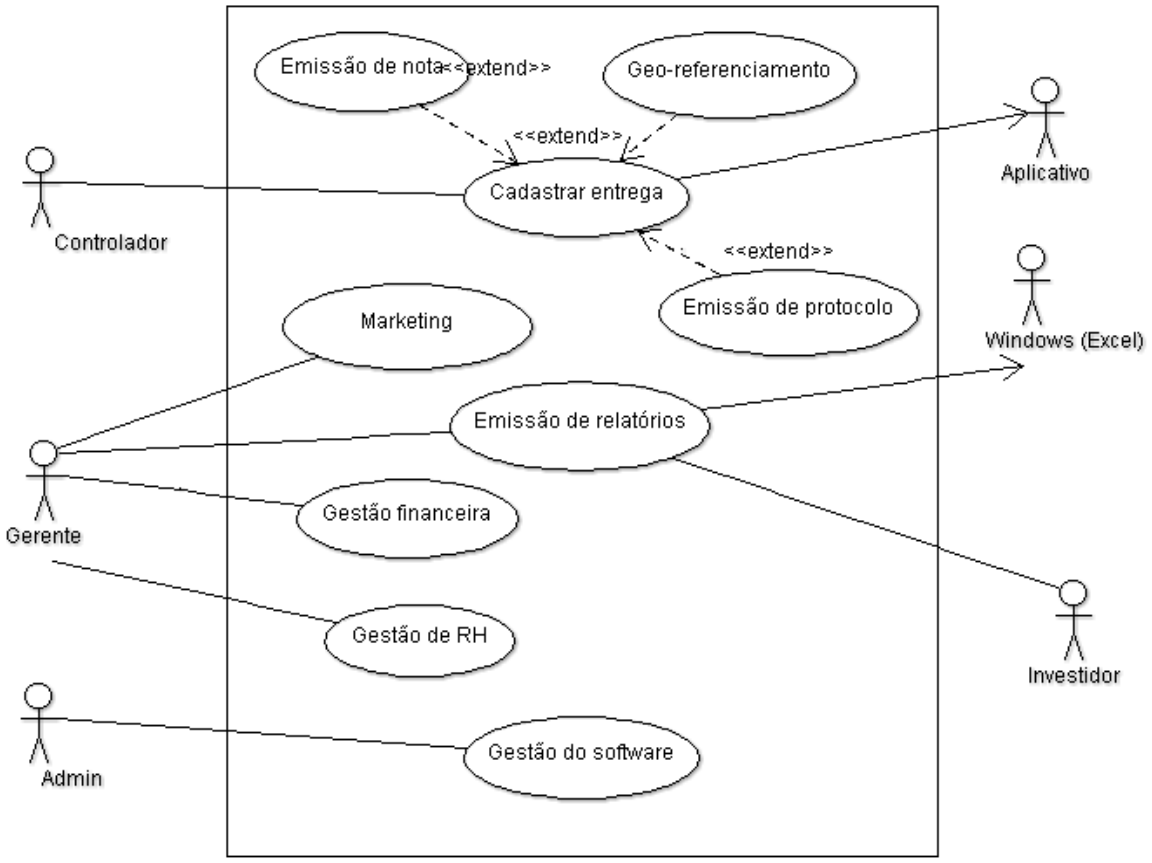
Fonte: Do Autor

Cada caso de uso é executado por um ator (como mostrado no quadro 5), que possui também características de uso particulares, o que influencia a realização das interfaces e dos fluxos de caso de uso. O controlador provavelmente usará o Courribilidade várias vezes ao dia. Um sócio-investidor, porém, deve atuar sobre o sistema uma ou duas vezes ao mês. É natural esperar, por exemplo, que um ator com baixas habilidades em informática apresente dificuldades para utilizar uma interface gráfica com muitas opções ou funções de natureza complexa, da mesma maneira que alguém que interage pouco com o sistema pode requerer uma GUI (*Graphical User Interface*) intuitiva dado que não conseguirá memorizar os comandos do sistema com pouca frequência de uso.

Casos de uso podem ser agrupados em classes, que indicam funcionalidades semelhantes entre si ou que implementam o mesmo benefício. Por exemplo, seria possível agrupas os casos de *Inserir, Apagar e Editar usuário* na classe *Gestão de Usuários.* Isso torna na opinião do autor os casos de uso mais compreensíveis do ponto de vista das melhorias trazidas, mas dificulta o detalhamento destes, atividade executada mais adiante. Este texto irá utilizar as duas denominações, dependendo do contexto.

Casos de uso podem ser apresentados de forma mais visual usando diagramas UML. Para isto, é possível usar o diagrama de contexto, que Pádua Paula descreve como uma figura que mostra as interfaces do sistema com seu ambiente (2003, p. 100) e as funcionalidades de que estas interfaces participam. Trata-se de uma visão de alto nível do sistema. A figura 14 mostra o diagrama de contexto do Courribilidade.

Figura 14 - Diagrama de contexto do Courribilidade



Fonte: Do Autor

Nota-se que foi usada a denominação de classes para os casos de uso, o que favorece o entendimento das funcionalidades que são executadas por cada um dos atores e torna o diagrama mais sintético.

O diagrama de contexto fornece uma visão geral sobre o sistema, mas não fornece detalhes sobre como as funcionalidades devem ser implementadas pelo programador ou utilizadas pelo usuário. Isso é feito no detalhamento dos casos de uso, apresentado abaixo. Mais uma vez, nem todos os casos de uso são detalhados neste documento, pois isto tornaria o texto de difícil leitura.

3.4.2.1. Detalhamento dos casos de uso

Detalhamentos seguem uma estrutura bem definida: primeiramente são expostas as pré-condições para que o usuário possa ter acesso ao caso de uso, como possuir certas permissões de uso ou estar em algum modo do sistema. Em seguida é mostrado o passo a passo que o ator deve seguir para executar a função, e por fim são detalhados sub-fluxos que podem estar contidos nos detalhes de execução da função.

Abaixo são apresentados os detalhamentos para os casos de uso de inserção de usuário no Courribilidade e de geo-referenciamento de entregas.

3.4.2.2. Detalhamento Caso de Uso *Inserção de usuários*

3.4.2.2.1. Pré-condições

1. O usuário deve estar registrado como administrador do sistema

3.4.2.2.2. Detalhamento do fluxo

1. O administrador clica no botão “Gestão de Usuários”, dentro do item “Usuários” na barra de menu principal
2. O Courribilidade Exibe o painel de gestão de usuários, ocupando integralmente a tela do sistema
3. O administrador clica no botão “Inserir usuário” da tela de gestão de usuários
4. O Courribilidade exibe a tela de inserção de usuários e bloqueia a execução de outras funcionalidades do programa enquanto o usuário não finalizar o processo de inserção
5. O administrador preenche os dados do novo usuário e clica no botão inserir usuário
6. O Courribilidade verifica a validade de todas as informações; Caso haja pelo menos uma informação incorreta, executa-se o subfluxo alerta de erro de dados incorretos
7. Caso todas as informações estejam corretas, o Courribilidade cria na base de dados o usuário com as informações dadas pelo usuário

8. O Courribilidade exibe uma mensagem de sucesso, fecha o formulário de inserção de usuários e desbloqueia a tela de gestão de usuários.

3.4.2.2.3. Detalhamento do subfluxo alerta de dados incorretos

1. O Courribilidade exibe uma mensagem de erro com uma lista das informações incorretamente digitadas ou faltantes
2. O usuário clica em OK
3. O usuário é redirecionado para a tela de inserção de usuários

3.4.2.3. Detalhamento do caso de uso Georeferenciamento de entregas

3.4.2.3.1. Pré-condições

1. O usuário deve possuir permissão para executar esta tarefa

3.4.2.3.2. Detalhamento

1. O gerente seleciona na barra de menu principal o botão “Georeferenciamento”
2. O Courribilidade exibe no painel esquerdo a tela de controle de georeferenciamento e no painel direito o painel de geração de relatório de georeferenciamento pelo banco de dados
3. Caso o gerente deseje um relatório com dados externos:
    a. O gerente clica no botão correspondente no painel esquerdo
    b. O Courribilidade exibe a tela correspondente à geração de relatórios com dados externos
    c. O gerente carrega o arquivo de dados externos
    d. O Courribilidade exibe mensagem de erro ou sucesso conforme o resultado da leitura do arquivo
    e. Em caso de sucesso, o gerente clica no botão gerar relatório
    f. O Courribilidade começa a gerar o relatório e exibe mensagem de sucesso quando terminado
4. Caso o gerente deseje gerar relatório com dados de entregas do banco de dados:

a. O gerente preenche os campos de busca necessários para gerar seus resultados
b. O Courribilidade executa a busca no banco de dados e mostra painel com resultados no painel direito
c. O gerente clica em gerar relatório
d. O Courribilidade gera relatório e avisa o gerente quando do sucesso da geração deste

Detalhamentos são construídos em comunicação com o usuário e são a base para a implementação do sistema. Lendo-os com atenção, já é possível visualizar como será o fluxo de execução de cada uma das funcionalidades pelo usuário e como deverão ser dispostas as GUI para satisfazer cada um dos requisitos. É possível também entender como o detalhamento é fundamental para uma construção de manual do sistema.

3.5. Interfaces do sistema

Após a descrição dos casos de uso, o Praxis foca na definição das interfaces do sistema. Interfaces são os canais através dos quais o sistema interage com seu ambiente. As telas vistas pelo usuário quando do uso do programa são claramente interfaces, assim como as funções que se comunicam com o programa Excel. O Praxis prevê quatro tipos de interface: de usuário, de software e hardware.

Interfaces de usuário são os canais que o usuário usa para se comunicar com o sistema, e são principalmente compostas pelas telas. Interfaces de software são meios de comunicação com outros programas, como as funções de leitura e escrita sobre arquivos do sistema operacional usadas para gerar relatórios Excel. Interfaces de hardware são ligações com recursos como memória ou câmeras de vídeo, que não se aplicam ao Courribilidade, enquanto que interfaces de comunicação são os protocolos não-usuais de comunicação com o meio externo. Como o Courribilidade não faz uso de interfaces não convencionais, estas não serão tratadas.

3.5.1. Interfaces de usuário

O quadro 6 mostra as interfaces de usuário do Courribilidade. É importante mencionar que as interfaces foram agrupadas por benefício, para diminuir seu número e tornar sua leitura mais fácil. Por exemplo, existem diversas telas ligadas à gestão de RH, como edição de funcionários ou geração de relatórios, mas somente uma interface, a tela de gestão de RH, foi mostrada aqui.

Quadro 6 - Interfaces de usuário do sistema (Continua)

Número de ordem	**Nome**	**Ator**	**Caso de uso**	**Descrição**
1	Tela inicial	Todos	Login de usuário	Interface para a autenticação do usuário
2	Tela de gestão de entregas	Gerente e controlador de ciclistas	Gestão de vendas	Tela onde entregas são cadastradas no banco de dados
3	Tela de geração de relatórios gerenciais	Gerente, sócio-investidor e sócio	Gestão de resultados	Interface onde são gerados relatórios de resultados financeiros
4	Tela de gestão de RH	Gerente de RH e sócio	Gestão de RH	Interface onde são cadastrados, consultados e e deletados os funcionários da
5	Tela de gestão de clientes	Gerente de marketing, sócio-investidor e sócio	Gestão de clientes	Interface onde são cadastrados, consultados e deletados clientes
6	Tela de gestão backup	Gerente do sistema	Backup de dados	Interface onde o backup de dados é executado
7	Tela de geração de relatório de resultados	Gerente, sócio-investidor, sócio	Geração de relatórios	Interface onde são gerados relatórios gerenciais como relatório de resultados e relatório de clientes

Número de ordem	Nome	Ator	Caso de uso	Descrição
8	Tela de emissão de nota fiscal	Gerente	Emissão de nota fiscal	Interface onde é gerada nota fiscal
9	Tela de georeferenciamento	Gerente	Georeferenciamento	Tela onde são gerados relatórios de georeferenciamento

Fonte: Do autor

3.5.2. Interfaces de software

As interfaces de software são menos numerosas do que as de usuário, e são mostradas no quadro 7.

Quadro 7 - Interfaces de software do Courribilidade

Número de ordem	Nome	Ator	Caso de uso	Descrição
1	Conexão com o sistema operecional	Sistema operacional	Geração de relatórios	A
2	Conexão com o serviço de backup de dados	Sistema de backup de dados	Backup de dados	Tabelas do banco de dados que serão salvas no sistema de backup
3	Comunicação com o aplicativo	Aplicativo Courribilidade	Comunicação com aplicativo	Dados em formato JSON ou XML
4	Comunicação com o banco de dados	SQL Server	Todos os casos que envolverem leitura/escrita do banco de dados	Comunicação através da linguagem SQL-Server

Fonte: Do autor

3.6. Requisitos não-funcionais

Requisitos funcionais, como mencionado na introdução desta seção, são as condições de performance das funcionalidades. Mais claramente, são restrições que as funções do sistema devem obedecer para serem aceitas pelo cliente.

As principais restrições em um sistema de informação são relativas à eficiência de cálculo dos algoritmos, números máximo de transações aceitáveis por unidade de tempo e tempo de resposta para uma dada requisição (estes dois últimos cruciais para sistemas conectados à internet).

Como o Courribilidade não será um software web e não trabalhará com algoritmos de computação complexos, as necessidades não-funcionais são poucas e simples, e são mostrados no quadro 8. Estes foram levantados com o gerente financeiro, sr. André Biselli, na primeira reunião de requisitos e mantiveram-se os mesmos até o final do projeto.

Quadro 8 - Requisitos não-funcionais do sistema

Requisito	**Descrição**
Query unitária ao banco de dados	Uma query com retorno de uma linha da tabela não pode demorar mais do que 0,5 segundos
Query com retornos múltiplos ao banco de dados	Uma query com retorno de mais de uma linha da tabela não pode demorar mais do que 5 segundos
Tempo de geração de um relatório qualquer	A geração e gravação em disco de um relatório qualquer não pode demorar mais do que 30 segundos
Tempo de backup do banco de dados	O backup de dados para a nuvem não pode demorar mais do que 2 minutos

Fonte: Do autor

Todas as restrições levantadas tratam de tempo de resposta ou de execução de alguma funcionalidade do sistema. Foram definidas de maneira subjetiva, e representam o maior tempo que o sr. Biselli disse estar disposto a esperar para cada uma das operações. Não é sempre o caso para requisitos deste tipo: muitas vezes a

não satisfação de um requisito não-funcional implica na perda de funcionalidade de um caso de uso. Para sistemas web, por exemplo, se um website não responder dentro de um certo intervalo de tempo definido pelo navegador, este envia um erro ao usuário, que não consegue fazer uso da página desejada, mesmo que esta esteja pronta para oferecer seus serviços.

4. Análise

Esta seção trata da análise do projeto Courribilidade. O fluxo de análise no Praxis visa três objetivos principais, segundo Pádua Paula (2003, p. 121): Primeiramente, busca-se modelar os conceitos trazidos pelo levantamento de requisitos de forma precisa. Em seguida, verificar se os requisitos determinados anteriormente possuem a qualidade necessária. Por último, procura-se detalhar estes requisitos para torna-los compreensíveis para os desenvolvedores.

De maneira mais clara, a análise busca modelar os conceitos do domínio do problema. Como já foi visto antes, modelar é simplificar com um objetivo. Busca-se, portanto, simplificar os requisitos de forma a tornar a sua implementação possível.

O fluxo de análise não é exclusivo do desenvolvimento de software, mas é característico da engenharia de sistemas como menciona o *Department of* Defense (DoD), o departamento de defesa americano (2001, p. 35). Ainda segundo o DoD, existem diversos tipos de técnicas de análise. (2001, p. 38), cada uma adaptada para o tipo de sistema em projeto. O Praxis utiliza a Análise Orientada a Objetos, ou Object-oriented Analysis, muito comum para o projeto de softwares.

4.1. Objetos

Objetos em linguagens de programação a estes orientados são estruturas de dados que contém informações, na forma de atributos, e código para manipular dados, também chamados de métodos. Objetos podem controlar o acesso a seus atributos e métodos através de permissões, e podem interagir entre si através de mensagens emitidas por um de seus métodos. Linguagens de programação orientadas a objeto consideram que todos os componentes de um programa são objeto, e o programa funciona através da comunicação seus diversos objetos. (Booth et al., 2007, p. 29).

Objetos são, de maneira mais intuitiva, cápsulas de dados e métodos, que só expõe ao mundo exterior o que decidem mostrar. Esse tipo de estrutura de dados permite aos desenvolvedores criar programas altamente modularizados e escaláveis.

4.2. Análise orientada a objetos

Grady Booch, um dos criadores da UML e guru da análise orientada a objetos, fornece uma definição clara e concisa dessa (2007, p. 42):

> *A análise orientada a objetos é o método de análise que examina os requisitos do ponto de vista das classes e objetos encontrados no vocabulário do domínio do problema*

Percebe-se, portanto, que a análise está intimamente ligada com o vocabulário usado para definir o problema. Ora, no Praxis, o problema é definido no levantamento de requisitos. Portanto, a análise se concentrará nos requisitos levantados, mais especificamente no fluxo dos requisitos.

4.3. Análise no Praxis

A análise no Praxis é executada em seis atividades distintas, indicadas no quadro 9. Mais uma vez, é importante mencionar que estas atividades não foram feitas na sequência exata em que aparecem aqui. Cada uma é parte de uma iteração que contém atividades de vários fluxos (como fluxo de Requisitos ou de Testes), mas para tornar esta seção mais compreensível, o autor decidiu apresentar todas as etapas do fluxo de análise na mesma seção, seguindo a ordem em que estas são apresentadas no quadro 9.

Quadro 9 - Atividades da análise no Praxis

Número de ordem	Atividade	Descrição sucinta
1	Identificação das classes	Identifica as classes do produto baseada no detalhamento dos casos de uso
2	Organização das classes	Organiza as classes em pacotes lógicos
3	Identificação dos relacionamentos	Determina os relacionamentos de vários tipos que podem existir entre os objetos das diversas classes identificadas
4	Identificação dos atributos	Levanta os atributos que correspondem a propriedades que fazem parte do conceito expresso pela classe
5	Realização dos casos de uso	Verifica os fluxos dos casos de uso, representando-os através de diagramas de interação
6	Revisão da análise	Valida o esforço da análise e o correspondente esforço dos requisitos

Fonte: Pádua Filho (2003, p. 124)

4.4. Classes chave identificadas

Como foi dito de maneira superficial acima, a identificação de classes chave é feita pela análise do detalhamento dos casos de uso, responsabilidade do fluxo de requisitos. O Praxis sugere marcar o nome do produto no estilo **negrito sublinhado**, o de atores em sublinhado e o de outros substantivos em sublinhado duplo. Todas as palavras marcadas com sublinhado são potenciais classes.

O trabalho de identificação de classes é essencial para o sucesso do projeto, e pode usar uma parte considerável dos recursos disponíveis para a empreitada, como afirma Pádua Filho (2003, p. 120). Mesmo para um sistema simples como o Courribilidade, consome-se quantidade considerável de tempo para sua execução. No corpo do texto são apresentados os resultados da análise feita sobre os detalhes dos casos de uso apresentados anteriormente.

4.4.1. Substantivos em inserção de usuário

1. O administrador clica no botão “Gestão de Usuários”, dentro do item “Usuários” na barra de menu principal
2. O **Courribilidade** exibe o painel de gestão de usuários, ocupando integralmente a tela do sistema
3. O administrador clica no botão “Inserir usuário” da tela de gestão de usuários
4. O **Courribilidade** exibe a tela de inserção de usuários e bloqueia a execução de outras funcionalidades do programa enquanto o administrador não finalizar o processo de inserção
5. O administrador preenche os dados do novo usuário, a saber, nome de usuário, senha e a identificação do funcionário (chave estrangeira) e clica no botão inserir usuário
6. O **Courribilidade** verifica a validade de todas as informações; Caso haja pelo menos uma informação incorreta, executa-se o subfluxo alerta de erro de dados incorretos
7. Caso todas as informações estejam corretas, o **Courribilidade** cria na base de dados o usuário com as informações dadas pelo administrador

O **Courribilidade** exibe uma mensagem de sucesso, fecha o formulário de inserção de usuários e desbloqueia a tela de gestão de usuários.

4.4.2. Substantivos em georeferenciamento de entregas

1. O gerente seleciona na barra de menu principal o botão “Georeferenciamento”
2. O Courribilidade exibe no painel esquerdo a tela de controle de georeferenciamento e no painel direito o painel de geração de relatório de georeferenciamento pelo banco de dados
3. Caso o gerente deseje um relatório com dados externos:
    a. O gerente clica no botão correspondente no painel esquerdo
    b. O Courribilidade exibe a tela correspondente à geração de relatórios com dados externos
    c. O gerente carrega o arquivo de dados externos

   d. O Courribilidade exibe mensagem de erro ou sucesso conforme o resultado da leitura do arquivo
   e. Em caso de sucesso, o gerente clica no botão gerar relatório
   f. O Courribilidade começa a gerar o relatório e exibe mensagem de sucesso quando terminado
4. Caso o gerente deseje gerar relatório com *dados de entregas* do banco de dados:
   a. O gerente preenche os campos de busca necessários para gerar seus resultados
   b. O Courribilidade executa a *busca* no banco de dados e mostra painel com resultados no painel direito
   c. O gerente clica em gerar relatório
   d. O Courribilidade gera relatório e avisa o gerente quando do sucesso da geração deste

4.5. Determinação das classes pelas classes candidatas

Uma vez determinadas as classes candidatas, executa-se analisa-se com mais cuidado estas para determinar quais são potenciais classes. O resultado é mostrado no quadro 10

Quadro 10 - Resultado da análise dos casos de uso para determinação de classes (Continua)

Classe candidata	**Análise**
Barra de menu	Provável classe
Usuário	Provável classe
Dados	Atributo da classe usuário
Alerta	Provável classe
Botão Georeferenciamento	Provável classe
Painel esquerdo/direito	Provável classe
Georeferenciamento	Operação
Senha	Provável atributo da classe usuário
Funcionário	Provável classe
Nome de usuário	Provável atributo da classe usuário

Conclusão

Classe candidata	Análise
Banco de dados	Entidade externa, não tratada (ao contrário dos resultados de pesquisas e escritas, que serão tratados)
Relatório	Entidade de implementação
Tela	Provável classe
Dados externos	Entidade externa, não tratada
Mensagem de erro/sucesso	Provável classe
Entrega	Provável classe
Dados de entrega	Atributo da classe candidata entrega

Fonte: Do autor

Vale a pena notar alguns fatos sobre a análise: Elementos de interface de usuário aparecem com alta frequência. Como já estão implementados como classes na linguagem C#, não é necessário se preocupar com eles por ora. Além disso, algumas classes que são parte do projeto não são facilmente identificadas através da marcação dos substantivos. São as classes de controle ou controladores, que tratam de coordenar a execução de uma funcionalidade, e geralmente são associados a cada caso de uso. Por se tratarem de classes muito abstratas, podem ser omitidas

Excluindo os itens de interface gráfica e os atores e adicionando os resultados das outras análises obtemos as seguintes classes mostradas na figura 15 para o Courribilidade:

Figura 15 - Classes do Courribilidade

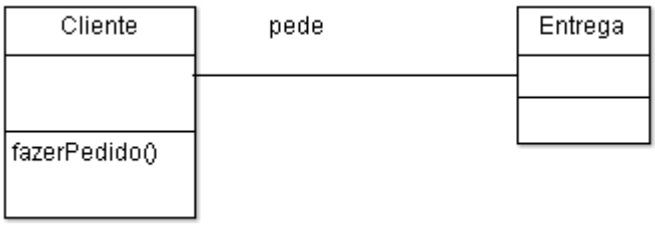
Fonte: Do autor

O Praxis sugere também que sejam mostradas as classes de fronteira, i.e., aquelas através das quais ocorre a comunicação do programa com o mundo exterior. Como o Courribilidade possui muitas telas, isto não será mostrado no corpo do texto.

4.6. Relações entre as classes

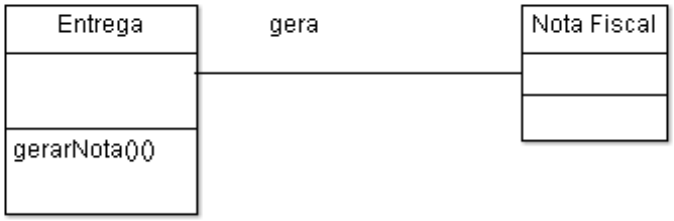
Uma vez estabelecidas as classes, passa-se à determinação das relações entre elas. Relacionamentos são conexões lógicas entre as classes do sistema. Podem ser de diversos tipos, e os principais são os relacionamentos de associação e os de agregação. Um exemplo claro de associação é a relação existente entre uma entrega e um cliente. Não pode existir entrega sem haver cliente, então é possível que haja uma relação de associação entre estas duas classes. Relacionamentos de agregação são conexões do tipo “parte-todo” entre as classes. Por exemplo, é possível pensar na classe endereço como propriedade da classe cliente. Um endereço é portanto parte de um cliente em particular, e o relacionamento entre eles é do tipo *agregação.* No Praxis, estas relações são representadas através de diagramas de classe. Como nas seções anteriores, só são dois dos relacionamentos obtidos.

Figura 16 - Relacionamento de Cliente-Entrega



Fonte: Do autor

Figura 17 - Relacionamento Entrega-Nota Fiscal

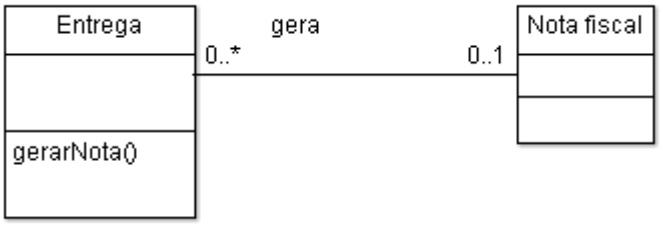


Fonte: Do autor

Relacionamentos na linguagem UML possuem diversos tipos de complexidade e nível de detalhes. É possível, por exemplo, indicar a multiplicidade das associações, que representa o número de elementos envolvidos na relação.

A figura 18 mostra as etiquetas de multiplicidade 0..* para as entregas e 0..1 para nota fiscal. Deve-se ler o relacionamento assim: Da esquerda para a direita, uma entrega gera zero ou uma nota fiscal (pode-se não gerar nota fiscal pois o recibo é emitido manualmente ou por outro sistema); Da direita para a esquerda, uma nota fiscal pode ser emitida por nenhuma ou várias entregas (uma nota fiscal não necessariamente é referente a uma entrega, e uma nota pode conter várias entregas).

Figura 18 - Relacionamento com direção e multiplicidade



Fonte: Do autor

Pádua Filho (2003, p. 132) sugere que os detalhamentos de multiplicidade e direção dos relacionamentos sejam deixados para a fase de desenho, e este trabalho segue esta recomendação.

4.7. Realização dos casos de uso

A identificação das classes e dos relacionamentos deve permitir realizar os casos de uso identificados no fluxo de requisito. Por realização, entende-se, no caso de sistemas orientados a objetos, a tradução dos detalhamentos dos casos de uso em interações entre os objetos encontrados na análise.

Os relacionamentos entre objetos se dão por mensagens, ou seja, invocação dos métodos ou procedimentos da classe receptora por parte da classe emissora. Um exemplo no contexto deste trabalho: O objeto *Nota Fiscal* pede ao objeto *Entrega* o valor de sua propriedade *Preço.* Neste caso, o emissor é a *Nota Fiscal* e o receptor, que possui o método, *Entrega.*

No Praxis, assim como em todos os métodos que utilizam a UML, representa-se os casos de uso através dos diagramas de interação, que podem ser do tipo sequência ou de colaboração. Diagramas de sequência enfatizam a ordem cronológica das operações executadas sobre o sistema. Por isso, representam uma visão mais próxima do sistema do detalhamento de casos de uso. O termo “mais próximo do sistema” deve ser entendido aqui como representando maior proximidade as classes que serão implementadas no sistema. Diagramas de colaboração dão ênfase aos relacionamentos entres as classes existentes no sistema. Foi decidido mostrar o

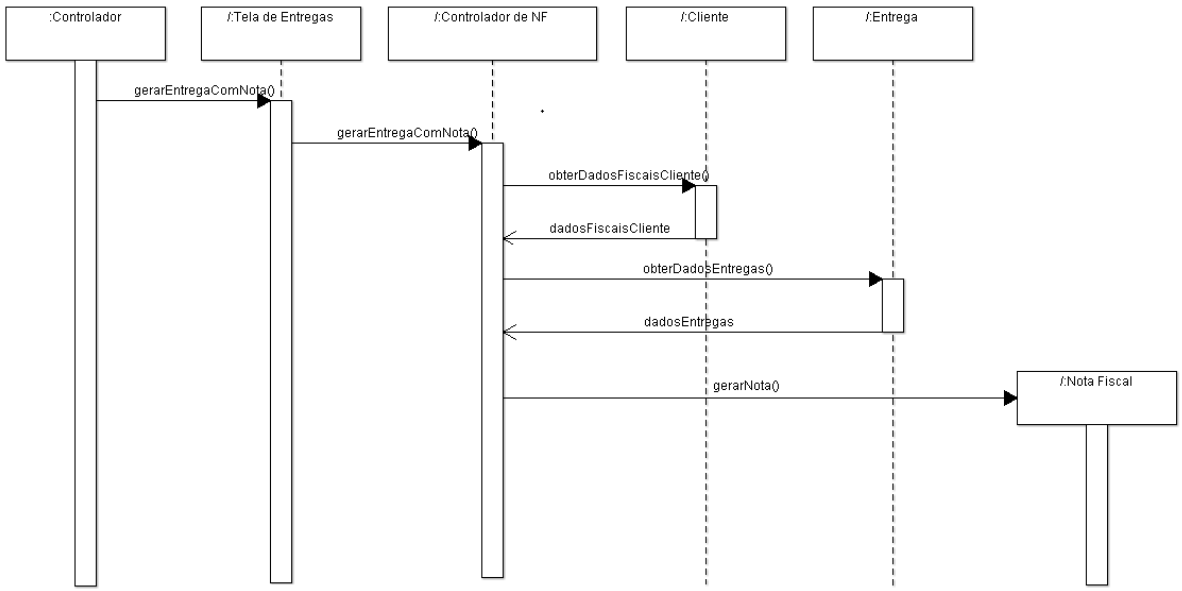
diagrama de sequência para o caso de uso escolhido pois este representa de maneira mais inteligível as operações e a lógica da funcionalidade.

4.7.1. Diagramas de sequência

A figura 19 mostra o diagrama de sequência para o caso de uso de emissão de uma nota fiscal. O diagrama da tem o objetivo de apresentar as operações necessárias para a emissão de uma nota fiscal na ordem cronológica em que acontecem. Ele permite também a visualização de todas as classes envolvidas e quando estas são invocadas pelo sistema.

Os retângulos na parte superior representam as classes que participam do processo. As linhas (pontilhadas ou cheias) que descem das classes até a parte baixa da figura são as linhas de vida de cada classe, e representam o período de tempo pelo qual a classe está presente na memória do programa. O usuário Controlador está presente desde o início, como indica sua linha do tempo integralmente cheia. Já a classe Nota Fiscal só é criada ao fim do processo, pela classe “Controlador de NF”, a classe que é responsável pela coordenação do caso de uso em questão.

Figura 19 - Diagrama de sequência para inserção de cliente



Fonte: Do autor

É importante notar que alguns objetos são efêmeros, i.e., são criados somente para efeito de realização do caso de uso, e são destruídos logo depois. É o caso do objeto da classe cliente: Ele surge para fornecer seus dados, e uma vez que estes são obtidos, a classe não é mais necessária e é inutilizada.

4.8. Identificação dos atributos

Como mencionado no início desta seção, atributos são propriedades que definem uma classe. No escopo deste trabalho, uma entrega é definida por um endereço de retirada e de entrega, horários para entrega e retirada, um cliente, um entregador e um preço. Uma vez definidos estes sete valores, uma entrega está perfeitamente definida para o Courribilidade. Assim como, para outros sistemas, um usuário pode ser definido perfeitamente por sua senha e login, e talvez sua data de nascimento.

4.8.1. Método de identificação de atributos

O Praxis sugere a seguinte sequência de atividades para determinar os atributos de uma classe (Pádua Filho, 2003, p. 143):

1. Listar as propriedades de uma classe que sejam relevantes para o problema em questão. É importante manter em mente o *tradeoff* entre generalidade da classe (e sua consequente facilidade para reuso em outros projetos e módulos do mesmo sistema) e a objetividade dos atributos. No caso do sistema aqui desenvolvido, atributos serão tão objetivos quanto possível, dada a baixa probabilidade de redesenho do sistema.
2. Localizar nos documentos de requisitos atributos que possam ter sido ignorados em “1.”
3. Evitar atributos que sejam relevantes somente para implementação, como indicadores de capacidade de uma variável ou semelhantes.

4.8.2. Atributos de Funcionário, Cliente e Entrega

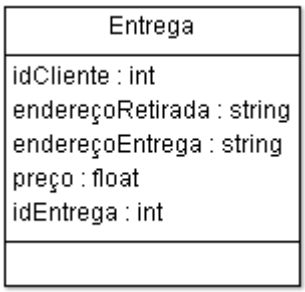
Para ilustrar o conceito de atributos foram escolhidas as classes Funcionário, Cliente (Figura 21) e Entrega (Figura 20). Com estas será possível explicar o conceito de

herança e de como uma classe pode ter como atributo elementos de outras classes, o que é ligado à multiplicidade das relações de associação entre elas.

É interessante notar dois pontos: Primeiramente, os atributos de uma classe sempre possuem um tipo de dado, e este tipo pode ser outra classe. Por exemplo, a classe entrega possui preço como atributo, do tipo double, e um endereço de retirada do tipo Endereço.

Em segundo lugar, nota-se que alguns atributos fundamentais da classe Funcionário (Figura 21) não estão mostrados. Pode-se, à primeira vista, acreditar que há um erro, dado funcionários parecem não ter nomes. Mas um exame mais cuidadoso do diagrama mostra que a classe Funcionário herda alguns atributos de PessoaFísicaOuJurídica, dentre eles, o string *nome*. A hereditariedade permite que classes se especializem em certas funções enquanto guardam comportamentos mais gerais herdados de suas mães.

Figura 20 - Atributos da classe Entrega



Fonte: Do autor

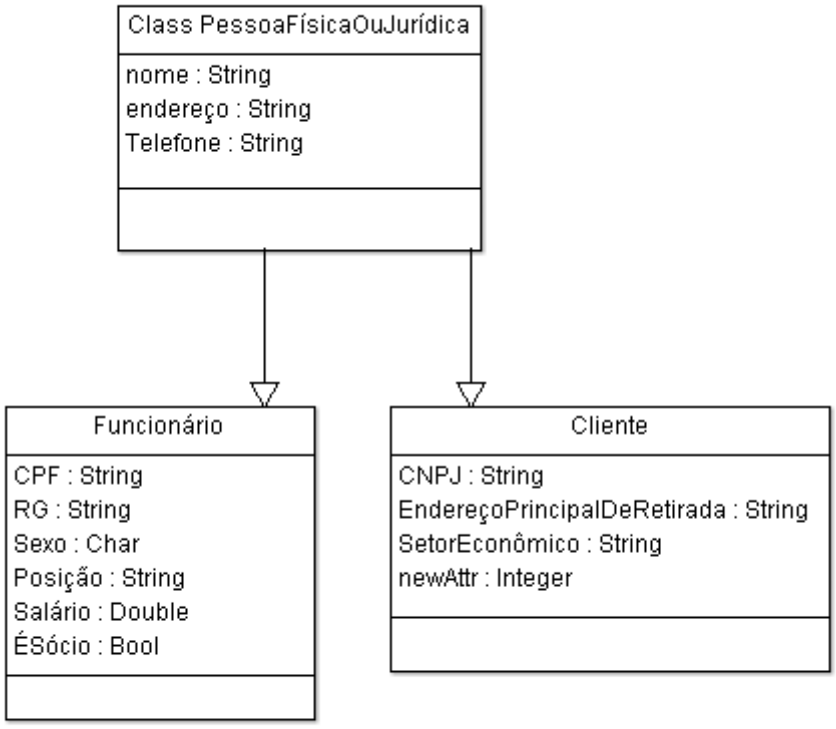
4.8.3. Identificação das heranças

A capacidade de herdar atributos de classes “pai” é um dos blocos construtores de linguagens orientadas a objetos. Herança significa que uma classe mais geral gera filhos mais especializados, que herdam de seus pais algumas características e métodos fundamentais e possuem por sua parte algo que os torna diferentes de seus genitores.

Um exemplo claro no contexto do Courribilidade é a classe Funcionário, que possui atributos em comum com a classe Pessoa Física ou Jurídica. Mais claramente, a classe Pessoa Física ou Jurídica é uma classe que possui os atributos Nome, Telefone e Endereço, que podem ser atribuídos tanto a empresas (Clientes, no caso do Courribilidade) quanto para pessoas físicas (Funcionários, neste contexto).

Algumas heranças relevantes para o Courribilidade são mostradas na figura 21. É importante notar que a classe Pessoa Física ou Jurídica é abstrata, i.e., não pode ser instanciada. Isso significa que não pode existir no sistema um objeto que pertença a esta classe, pois esta não possui todos os atributos ou métodos necessários para executar algum caso de uso. Já a classe *Funcionário* é perfeitamente completa para os fins do sistema e pode portanto ser instanciada. Ela herda de sua mãe os atributos desta e se especializa com propriedades como Salário e Posição, enquanto sua irmã *Cliente* possui atributos como Setor Econômico ou CNPJ que não são cabíveis para uma pessoa física.

Figura 21 - Exemplo de heranças no Courribilidade



Fonte: Do autor

4.9. Conclusões da análise

O fluxo de análise do Praxis visa traduzir os elementos do sistema, identificados durante as atividades de requisitos, para uma linguagem orientada a objetos. Este modelo do sistema em termos de objetos é mais próximo da linguagem de programação utilizada para implementação do sistema. Esta seção representa então um passo em direção à implementação do sistema.

Se o sistema está mais próximo de tomar forma, ainda é preciso algum trabalho antes de começar a codifica-lo. Mais especificamente, é preciso detalhar as interfaces e classes do produto.

5. Desenho

5.1. Introdução

O fluxo de desenho, em alguns processos chamado de design ou projeto, tem por objetivo, segundo Pádua Filho, definir uma estrutura para o produto que possibilite sua implementação, satisfazendo os requisitos definidos anteriormente.

É importante não confundir a fase de desenho com a fase de análise: a primeira é centrada na descrição de uma solução, enquanto a última busca definir o problema com uma linguagem mais adaptada à engenharia de software.

O quadro 11 nos mostra uma comparação feita por Pádua Filho entre os dois fluxos do Praxis, o que esclarece os papéis e características de cada um deles.

Quadro 11 – Comparação Desenho x Análise

Modelo de Análise	**Modelo de desenho**
Descreve o problema	Descreve uma solução
Conceitual (não trata de implementação)	Físico (base da implementação)
Suporta vários possíveis desenhos	Específico em relação a uma implementação
Classes estereotipadas conceituais	Pode conter classes estereotipadas
Pouco formal e pouco detalhado	Muito formal e detalhado
Poucos pacotes lógicos	Muitos pacotes lógicos

Fonte: Wilson Pádua Paula Filho (2003, p. 149)

5.2. Atividades do fluxo de Desenho no Praxis

O fluxo de desenho no Praxis é que contém o maior número de atividades. Começa-se pelo desenho arquitetônico, que soluciona aspectos estratégicos do produto como estrutura do sistema e escolha das tecnologias mais adequadas.

Passando pelo desenho das interfaces e o detalhamento definitivos dos casos de uso, o Desenho do sistema termina com a realização destes últimos, i.e., como os objetos das classes definidas colaborarão entre si par realizar as funcionalidades esperadas

do sistema. O quadro 12 exibe as atividades na sequência em que estas são realizadas nas interações do Praxis.

Quadro 12 - Atividades de Desenho no Praxis

Número de Ordem	Atividade	Descrição sucinta
1	Desenho arquitetônico	Resolve aspectos estratégicos de desenho externo e interno com
2	Desenho das interfaces	Desenha em detalhes as interfaces de usuário em seu ambiente definitivo de implementação
3	Detalhamento dos casos de uso	Resolve os detalhes dos fluxos anteriormente definidos
4	Desenho das entidades	Desenho das classes de entidades
5	Desenho da persistência	Desenho das camadas de persistência (Banco de Dados)
6	Realização dos casos de uso	Já descrito na análise

Fonte: Adaptado de Pádua Filho (2003, p.1)

5.3. Desenho arquitetônico

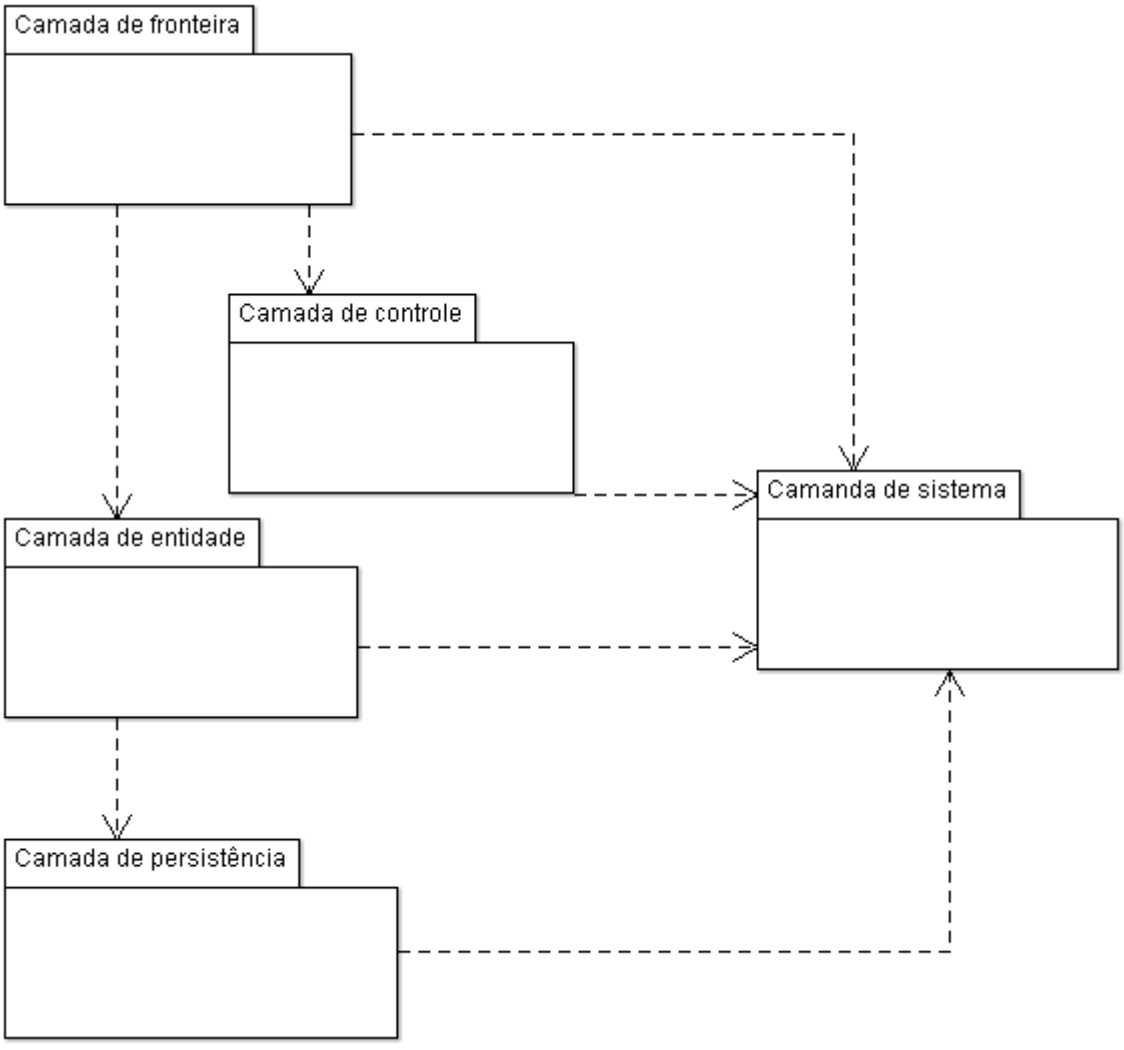
A primeira atividade de desenho é o desenho arquitetônico, como mostrado no quadro 12. Trata-se, fundamentalmente, de dividir o sistema em subsistemas e componentes com menor nível de abstração, como indica Pádua Filho (2003, p. 152). O objetivo desta modelagem é tornar a implementação e consequente satisfação dos requisitos funcionais e não-funcionais mais simples e sistemática.

A UML modela sistemas como conjuntos de subsistemas, e estes são por sua vez modelados por pacotes lógicos de desenho. Um pacote lógico nada mais é do que um agrupamento de classes (tudo é uma classe em OOP) em conjuntos com alta coesão interna e baixo acoplamento externo.

5.3.1. Arquitetura no Praxis

O Praxis utiliza uma arquitetura própria, comum a sistemas orientados a objetos, mostrada na figura 22. A principal vantagem desta estrutura é a promoção da separação entre a lógica da aplicação que será desenvolvida, a implementação e o domínio de aplicação, o que permite a fácil reutilização das classes envolvidas no programa, uma vez que estas estão segregadas de sua utilização efetiva. O quadro 13 torna o papel de cada uma das camadas mais claro.

Figura 22 - Arquitetura base do Praxis



Fonte: Adaptado de Pádua Filho (2003, p. 153)

É possível também definir naturezas para cada uma das camadas. A natureza nada mais é do que maneira de agrupar camadas (que por si já são agrupamentos) de acordo com o papel que cada uma desempenha na resolução do problema. O Praxis

considera três tipos de natureza: De aplicação, de domínio e de implementação. Cada uma destes é explicada em maiores detalhes nos parágrafos abaixo.

Classes de fronteira são aquelas que se comunicam com o ambiente do programa. As mais comuns serão, portanto, as interfaces de usuário, mas podem existir canais de troca de mensagens com o sistema operacional ou outros elementos do ambiente. A camada de controle inclui os controladores definidos na fase de desenho, e são normalmente ligadas à implementação de cada caso de uso. Tanto a camada de fronteira quanto a camada de controle fazem parte do grupo chamado de “camadas de aplicação”, pois são específicos do sistema que está sendo desenvolvido e não podem ser reaproveitadas sem antes serem retrabalhadas.

Camadas de domínio incluem somente a camada de entidade, que, como o nome sugere, contém as classes que representam as entidades ou elementos concretos do sistema, como a classe *Nota fiscal*, U*suário* ou *Entrega*. Por se tratarem de objetos concretos que não possuem o viés forte de nenhuma aplicação específica, podem ser reutilizados em outras soluções que as requeiram.

Por último, o Praxis define as camadas de implementação como aquelas que podem ser reaproveitadas para resolver problemas de outros domínios, ou seja, são logicamente independentes do problema que está sendo resolvido. Elas são a camada de persistência, que garante que as informações de objetos das camadas de domínio possam ser salvas na memória de maneira permanente (i.e., independentemente de aplicação estar ativa ou não), e a camada de sistema, que contém os serviços compartilhados por todas as outras camadas da aplicação. Como exemplo desta última, pode-se citar bibliotecas matemáticas ou de conexão com a internet.

Quadro 13 - Classes do Praxis e suas naturezas (Continua)

Natureza	**Camada**	**Descrição**
Camadas de aplicação	Camada de fronteira	Classes que implementam a interface do produto; Ex.: Tela de inserção de usuário

Conclusão

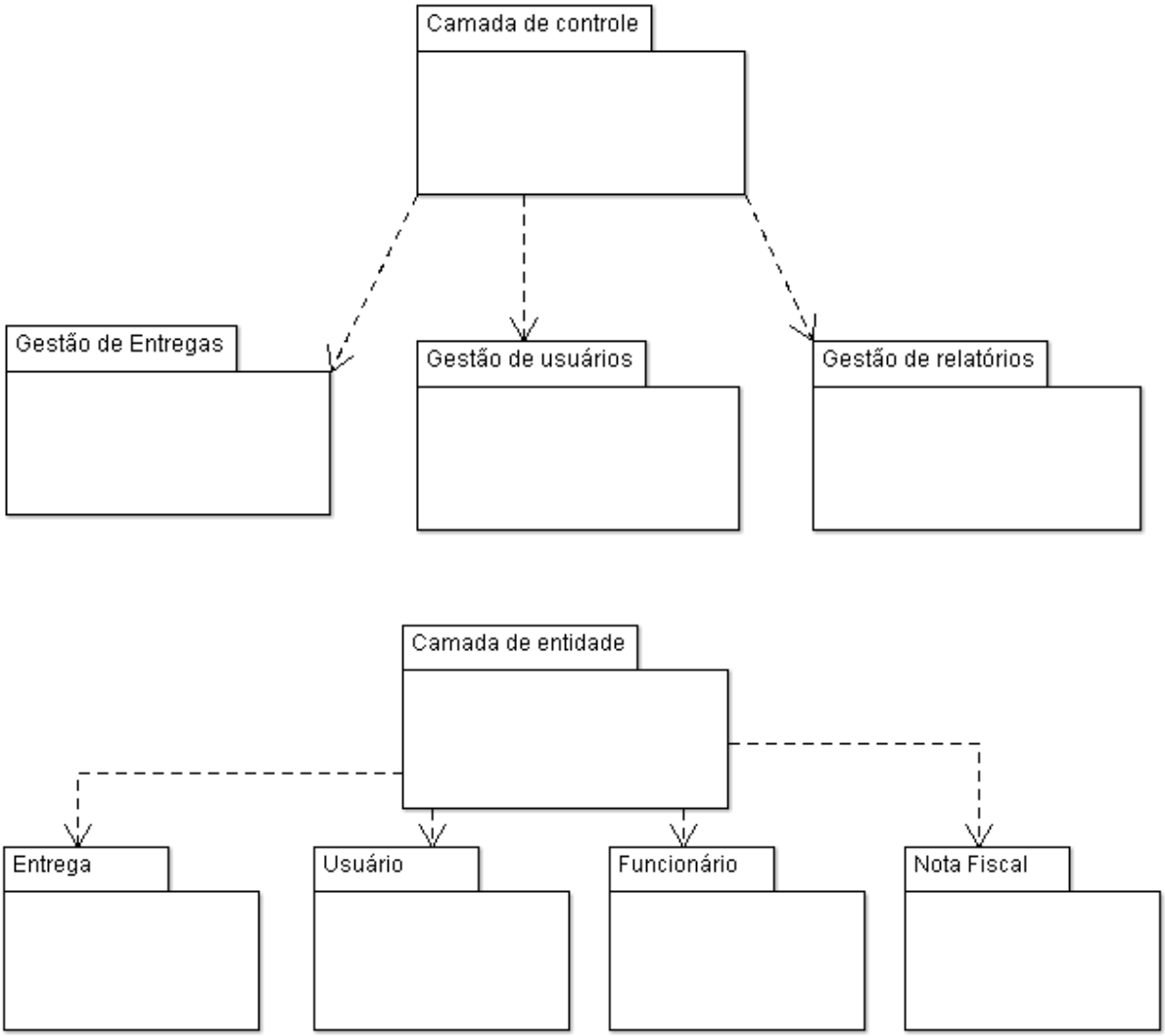
Natureza	Camada	Descrição
Camadas de aplicação	Camada de controle	Classes que implementam aspectos específicos da lógica da aplicação, como fluxos de casos de uso; Ex.: Controlador de inserção de entrega
Camadas de domínio	Camada de entidade	Classes que implementam as entidades concretas do sistema; Ex.: Usuário, Entrega
Camadas de implementação	Camada de persistência	Classes que garantem a persistência de dados de objetos da camada de entidade; Ex.: Microsoft SQL Server
Camadas de implementação	Camada de sistema	Classes que oferecem serviços comuns a todas as classes do programa. Ex.: System.Net.HTTP, System.Math para C#

Fonte: Adaptado de Pádua Filho, exemplos do autor (2003, p. 153)

5.3.2. Definição das camadas

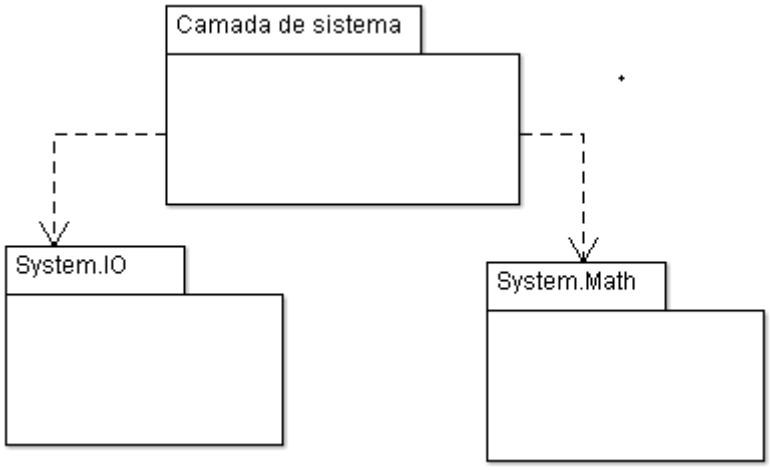
Uma vez definida a estrutura das camadas, passa-se à definição dos pacotes lógicos que as compõem, mostradas nas figuras 23 e 24. O Courribilidade, apesar de ser um sistema relativamente simples, possui diversos pacotes e sua apresentação integral no corpo deste trabalho tornaria o texto de leitura difícil e fastidiosa. Por isso, são apresentados nesta subseção somente alguns conjuntos.

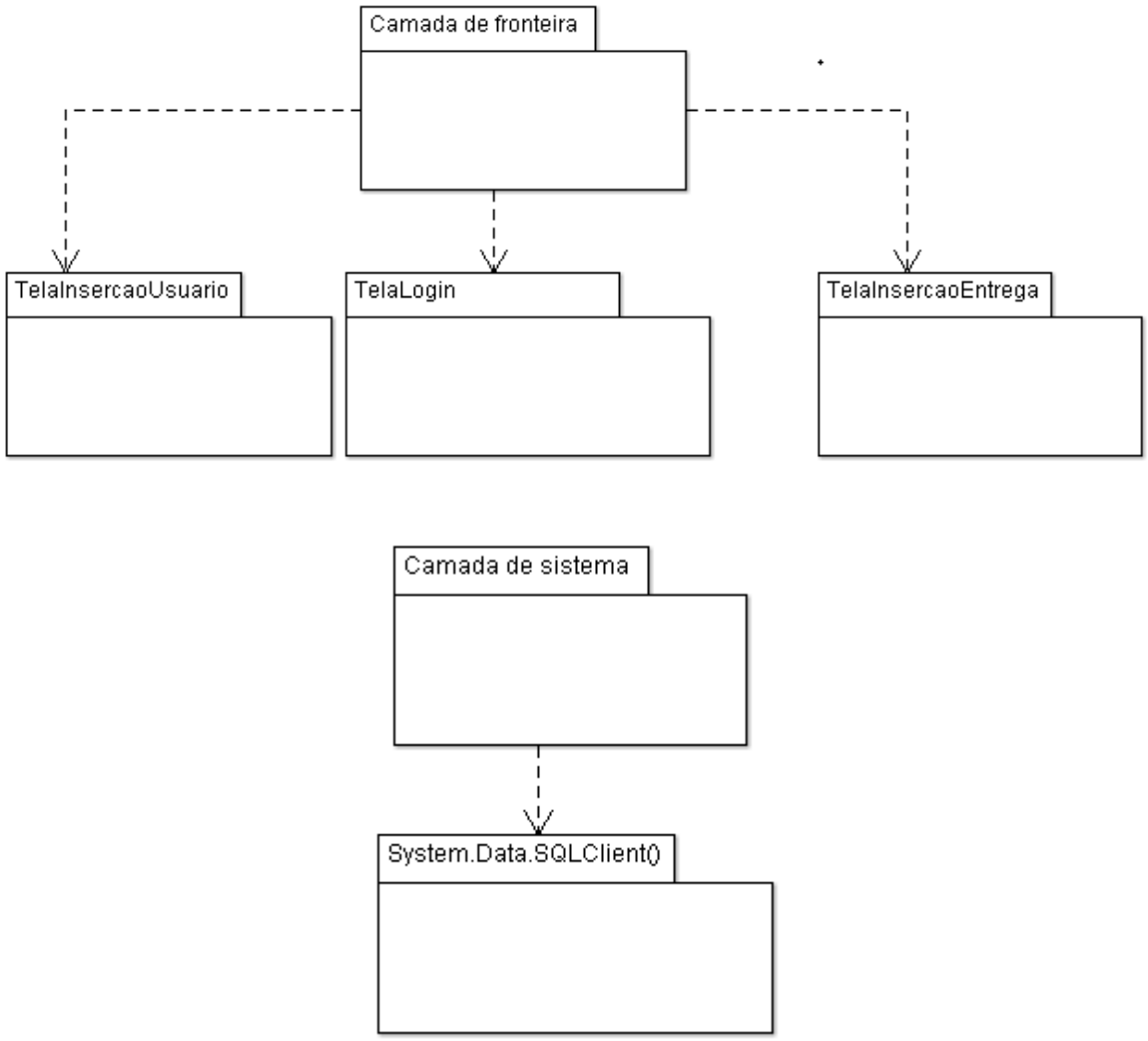
Figura 23 - Alguns pacotes das camadas de controle e entidade



Fonte: Do autor

Figura 24 - Classes das camadas de sistema, fronteira e persistência





Fonte: Do autor

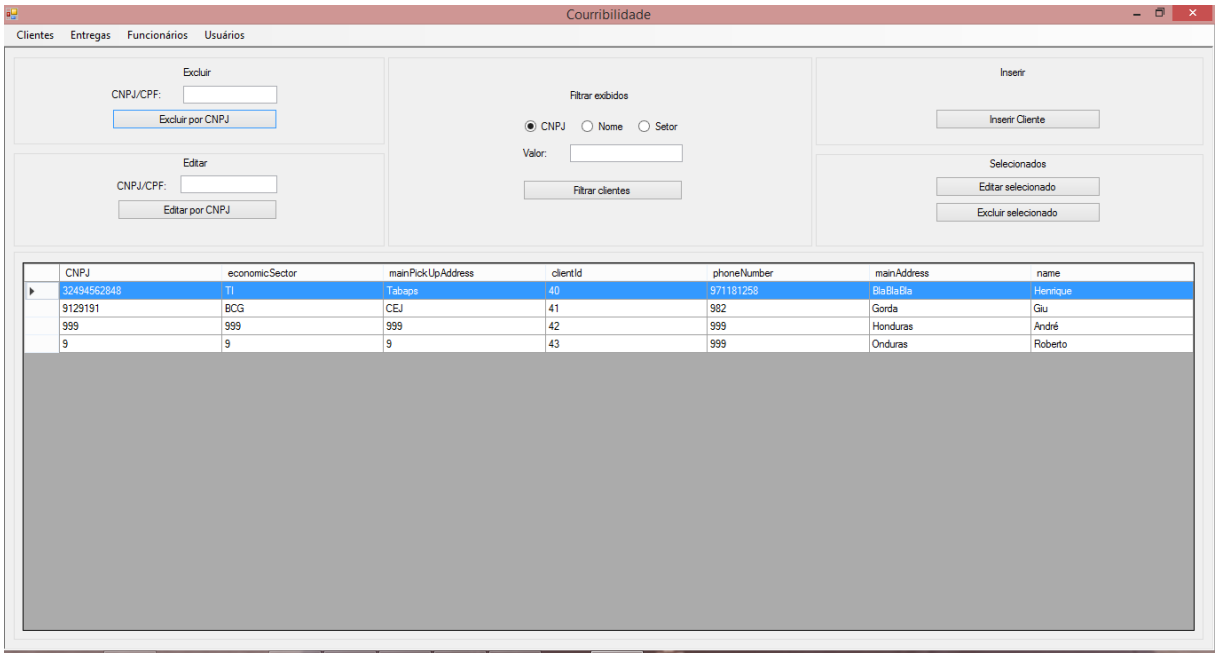
5.4. Desenho das interfaces

Esta atividade consiste do desenho externo e interno das interfaces de usuário. Por desenho externo deve-se entender o layout gráfico das interfaces, ou seja, como o usuário final enxergará o canal em questão. O desenho interno nada mais é do que o código por trás do desenho externo. Mais claramente, trata-se da tradução em linguagem de programação dos elementos presentes no desenho externo assim como de suas funcionalidades esperadas.

Estes conceitos são exemplificados na figura 25 que mostra a tela de gestão de Clientes, ou mais precisamente, o desenho externo desta. É através desta que o

usuário do sistema executará todas as operações relacionadas a instâncias da classe *Cliente*, como inserção (canto direito superior), edição (canto esquerdo) ou exclusão (canto esquerdo).

Figura 25 - Tela de controle de relatórios



Fonte: Do autor

É importante mencionar que o desenho das interfaces pode consumir muito tempo se não for feito com ferramentas especializadas, pois requer a definição manual da posição de cada elemento da interface, além da implementação do tratamento de eventos como cliques e cliques duplos. Felizmente, a Escola Politécnica da USP disponibiliza para seus alunos a ferramenta Visual Studio Ultimate (VS), que permite que o programador se preocupe com o desenho externo das telas enquanto o programa gera automaticamente o código correspondente aos elementos da interface, como posição dos botões e esqueletos dos métodos para tratar os eventos que cada elemento pode trazer. O código da figura 26 foi totalmente gerado pelo VS, bastando ao autor detalhar as funções criadas pelo software.

Figura 26 - Desenho interno da interface Tela de Gestão de Clientes

```
3 references
public partial class MDIFrame : Form
{
    private int childFormNumber = 0;

    1 reference
    public MDIFrame()...

    1 reference
    private void MDIFrame_Load(object sender, EventArgs e)...

    1 reference
    public MdiClient GetMdiClientWindow()...

    1 reference
    private void resizeChildren(int hDrec,int wDecr)...

    1 reference
    private void MDIFrame_SizeChanged(object sender, EventArgs e)...

    1 reference
    private void gestãoDeClientesToolStripMenuItem_Click(object sender, EventArgs e)...

    0 references
    private void voltarPáginaInicialToolStripMenuItem_Click(object sender, EventArgs e)...

    1 reference
    private void relatóriosToolStripMenuItem_Click(object sender, EventArgs e)...


}
```

Fonte: Do autor

Assim como nas seções anteriores, mostrar o detalhamento de todas as interfaces foge ao escopo do corpo do texto.

5.5. Detalhamento dos casos de uso

O detalhamento dos casos de uso para o fluxo de Desenho concentra-se em descrever como cada funcionalidade será realizada em termos das interfaces desenhadas anteriormente. Usa-se o modelo de análise como base da lógica dos fluxos, mas desta vez estão presentes os elementos da camada de fronteira relevantes para o caso de uso. Esta subseção apresenta o detalhamento do caso de uso “Criação de cliente”.

5.5.1. Detalhamento da “Criação de cliente”

5.5.1.1. Breve descrição do estado das interfaces

O acesso à funcionalidade de criação de clientes é feito através da barra de menu principal. Para tal, o usuário deve clicar sobre o botão “Clientes” e, em seguida, sobre o item “Cadastrar cliente” na barra que aparece. Se o painel esquerdo contiver o painel de gestão de clientes, deve ser mantido em seu estado. Caso contrário, deve exibir esta tela. O painel direito deve mostrar a tela de inserção de clientes, com os campos em branco.

5.5.1.2. Fluxo principal de “Criação de clientes”

1. O gerente clica no botão “Gestão de Clientes”, dentro do item “Clientes” da barra de menu principal
2. O Courribilidade exibe a tela de gestão de cliente (Figura 25)
3. O gerente clica no botão “Inserir Cliente” no canto superior direito da tela do sistema
4. O Courribilidade exibe a tela de inserção de clientes com todos os campos inicialmente em branco.
5. O Gerente de Vendas preenche os campos de *Nome*, *CNPJ/CPF*, *Endereço*, *Endereço de Retirada, Telefone* e *Setor Econômico* do cliente
6. O Gerente clica sobre o botão *Criar Cliente*
7. O Courribilidade executa o subfluxo Inserir Cliente no Banco de Dados, responsabilidade da classe *GestorDeClientes*
8. O Courribilidade exibe uma mensagem de sucesso de inserção no banco de dados e o gerente é redirecionado para a tela de gestão de clientes

5.5.1.3. Subfluxo Inserir Cliente no Banco de Dados

1. Se algum dos campos contiver informações incompletas ou com formato incorreto, ou estiver vazia, o Courribilidade emite mensagem de erro pedindo para eu o usuário verifique os dados
2. Quando os dados estiverem corretos, o Courribilidade instancia um objeto da classe *ClientManager*, que lida com operações de banco de dados da classe cliente

3. O objeto instanciado chama seu método insertClient(data), com *data* representado as informações inseridas no banco de dados do cliente
4. O sistema tenta inserir o novo cliente no banco de dados. Caso haja algum erro na operação, o Courribilidade emite uma mensagem de erro com o erro SQL no corpo
5. Caso haja sucesso, o sistema emite uma mensagem de sucesso com o nome do cliente, fecha a janela de inserção de usuários e desbloqueia o acesso à tela de gestão de clientes

5.6. Desenho das entidades

O desenho das entidades nada mais é do que um detalhamento e refinamento das classes obtidas durante a Análise. Pode-se criar novas classes a partir das anteriormente definidas, agrupar classes existentes em uma nova entidade, etc., como afirma Pádua Filho (2003, p. 161).

Em particular, é necessário detalhar os relacionamentos obtidos no fluxo anterior. Isso significa determinar a direção de alguns relacionamentos. Como mencionado na seção anterior, a direção indica que é possível obter o elemento destino através do elemento origem. A figura 27 ilustra o caso mais concretamente: É possível obter um cliente através de uma entrega, mas para obter as entregas de um cliente é necessário realizar uma pesquisa no banco de dados.

Figura 27 - Relacionamento direcional Entrega-Cliente

Fonte: Do autor

5.7. Desenho da persistência

O desenho da persistência consiste em determinar o esquema do banco de dados que será utilizado. Pode-se usar um banco de dados orientado a objetos, que torna o trabalho de determinação da estrutura deste praticamente inexistente: Basta criar uma coleção (termo para conjunto de objetos) por classe e preenche-la com os objetos desejados.

O uso de bancos de dados não-relacionais (ou NoSQL, no jargão de programadores) está se popularizando enormemente devido à sua alta performance para aplicativos da web e sua capacidade de crescer em volume sem prejudicar a performance, como afirma um paper da MongoDB (2015, p. 1). Este tipo de estrutura de dados é muito simples, o que permite queries com altíssima eficiência. Além disso, não há necessidade de tratar os resultados de uma busca nas coleções, uma vez que este já vem formatado como objeto da linguagem desejada.

Porém, o uso de DBs relacionais ainda é predominante sobre seus concorrentes, principalmente para sistemas empresariais, como afirma o mesmo paper em sua introdução (2015, pág. 1). Juntando a isso o fato de o aluno possuir conhecimentos prévios sobre este tipo de estrutura de armazenagem, esta foi a tecnologia esperada para implementar a persistência do Courribilidade.

O Praxis sugere as seguintes práticas, a serem seguidas para o uso de banco de dados SQL (relacionais):

- Cada classe da camada de entidades deve ser representada por uma tabela
- Atributos simples devem ser representados por uma coluna
- Atributos complexos por diversas colunas ou tabelas relacionadas adicionais
- A coluna da chave primária deverá ser um indicador do objeto salvo, que pode ser
    - Um atributo deste
    - Um número designado pelo sistema e sem nenhum significado concreto

Como exemplo, é mostrada no quadro 14 uma representação da tabela de Clientes, segundo o Praxis. Alguns atributos (como telefone) foram deixados de lado para permitir sua representação no espaço de uma página. É importante notar que o atributo endereço é em si uma classe, e é representado, portanto, por seu id na tabela Endereços.

Quadro 14 - Representação da Tabela de Clientes no Banco de Dados

CNPJ: Id	**Nome**	**Endereço**	**Telefone**	**Setor Econômico**	**Endereço de retirada principal**
32494564312	Courri Corp.	0xx1343	Elis	345	0xx32
32432352213	GreenDel	0xx3435	Maria	346	0xx1234
12345678909	EcoLiv	0xx4355	João	123	0xx123

Fonte: Do autor (sem significado prático)

5.8. Conclusão do Desenho

As atividades do fluxo de Desenho permitem que se chegue a um modelo da solução na forma de um software programado em linguagem orientada a objetos. Falta, porém, transformar esta simplificação em um programa que execute todas as suas funcionalidades. Este é o papel da implementação, que toma como entrada o documento construído no fluxo que a precede e o transforma em código-fonte e consequentemente em software funcional. A seção 6, a última antes da conclusão do trabalho, tratará das atividades deste fluxo.

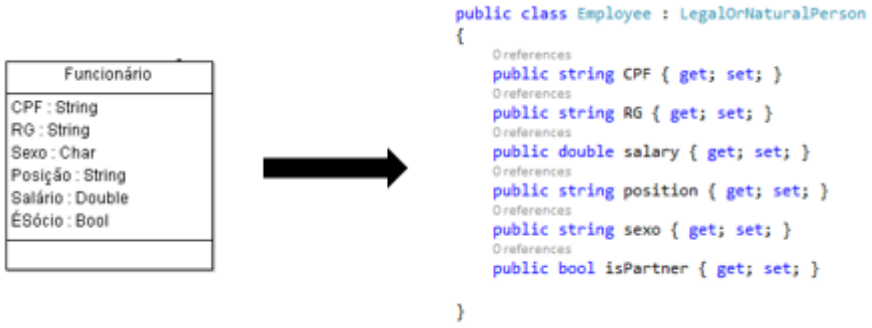
6. Implementação

O fluxo de implementação possui 5 atividades definidas no Praxis: Desenho detalhado, Codificação, Inspeção, Testes de unidade e Integração. Somente a atividade de codificação será apresentada neste trabalho, pois as outras, excetuando-se os testes de unidade, não produzem documentação relevante para o trabalho e são executadas naturalmente durante a codificação.

6.1. Codificação

A codificação é a atividade central da implementação e consiste na transformação do modelo de Desenho definido anteriormente em código-fonte que pode ser compilado ou interpretado pelo sistema operacional e que resulta em software funcional. A figura 28 ilustra de maneira gráfica a ideia desta fase: Como entrada tem-se um diagrama de classe e como saída uma classe em C#.

Figura 28 - Ideia fundamental da Implementação



Fonte: Do autor

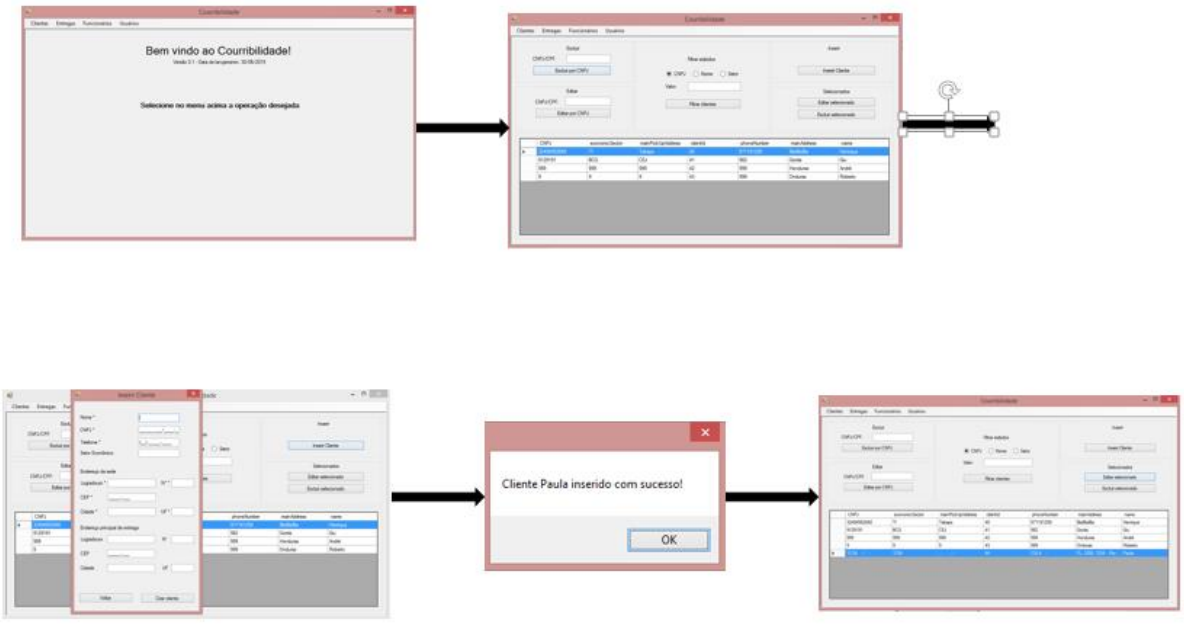
6.1.1. Exemplo de codificação: Inserção de clientes

Para ilustrar o processo de implementação, será tomado como exemplo o caso de uso de inserção de Clientes, pois a codificação deste é relativamente complexa e ilustra bem o processo de construção do sistema. A sequência de interfaces enxergadas pelo usuário é mostrada na figura 29.

Inicialmente, o usuário está na página inicial e clica em “Gestão de clientes” no menu “Clientes”. O Courribilidade deve direcioná-lo para a tela de gestão correspondente. Este redirecionamento é feito pelo método gestãoClientesMenuStrip_Click() da classe MDIFrame, a tela principal. A implementação deste método é mostrada na figura 30.

O usuário pode agora visualizar todos os clientes do banco. Para prosseguir, deve clicar em “Inserir cliente”. O sistema deve criar uma nova interface de inserção e exibi-a. Isto é implementado pelo método insertBtn_Click(), membro da classe ClientViewer, chamado quando do clique do botão “Inserir Cliente”.

Figura 29 - Sequência de interfaces do caso de uso "Inserção de usuário"



Fonte: Do autor

A seguir, após o preenchimento dos dados, o sistema deve validá-los e inserir o novo cliente no banco de dados. Esta funcionalidade é responsabilidade do método createClient_Click(), da classe *insertClientForm*, e de insertClient() da classe *clientManager*. O primeiro reage ao clique validando os dados e o segundo chamando a função de inserção de clientes no banco de dados.

Sem erros de validação, o sistema deve mostrar uma mensagem de sucesso e retornar à tela *clientViewer*, responsabilidades do método *MessageBox.Show()* e *this.Close(),* ambos mostrados no códido do formulário *insertClientForm* (Figura 32).

Figura 30 - Código de direcionamento à página de Gestão de Clientes

```
1 reference
private void gestãoDeClientesToolStripMenuItem_Click(object sender, EventArgs e)
{
    foreach (var child in this.MdiChildren)
    {
        child.Close();
    }
    ClientViewer clientViewer = new ClientViewer();
    clientViewer.MdiParent = this;
    clientViewer.Dock = DockStyle.Fill;
    clientViewer.Show();
}
```

Fonte: Do autor

Figura 31 - Código da aparência da tela de gestão de clientes

```
namespace Courri_Forms
{
    4 references
    partial class ClientViewer
    {
        /// <summary> ...
        private System.ComponentModel.IContainer components = null;

        /// <summary> ...
        19 references
        protected override void Dispose(bool disposing)...

        #region Windows Form Designer generated code

        /// <summary> ...
        1 reference
        private void InitializeComponent()
        {
            this.clientGridView = new System.Windows.Forms.DataGridView();
            this.label1 = new System.Windows.Forms.Label();
            this.filterTB = new System.Windows.Forms.TextBox();
            this.label4 = new System.Windows.Forms.Label();
            this.label5 = new System.Windows.Forms.Label();
            this.label8 = new System.Windows.Forms.Label();
            this.filterBtn = new System.Windows.Forms.Button();
            this.deleteCNPJTB = new System.Windows.Forms.MaskedTextBox();
            this.label6 = new System.Windows.Forms.Label();
            this.label7 = new System.Windows.Forms.Label();
            this.editCNPJTB = new System.Windows.Forms.MaskedTextBox();
            this.deleteByCNPJBtn = new System.Windows.Forms.Button();
            this.editByCNPJBtn = new System.Windows.Forms.Button();
            this.editSelectedBtn = new System.Windows.Forms.Button();
            this.deleteSelectedBtn = new System.Windows.Forms.Button();
```

Fonte: Do autor

Figura 32 - Implementação dos comportamentos de inserção de clientes

```
private void insertBtn_Click(object sender, EventArgs e)
{
    InsertClientForm = new InsertClientForm();
    insertClientForm.ShowDialog();
    loadGridView();
}


private void createClient_Click(object sender, EventArgs e)
        {
            string mainAddress = mainAddressStreetTB.Text + ", " +
mainAddressNumberTB.Text + ", " +
                mainAddressZipCodeTB.Text + ", " + mainAddressCityTB.Text + "-" +
mainAddressStateTB.Text;
            string mainPickUpAddress = mainPickUpAddressStreetTB.Text + ", " +
mainPickUpAddressNumberTB.Text + ", " +
                mainPickUpAddressZipCodeTB.Text + ", " + mainPickUpAddressCityTB.Text
+ "-" + mainPickUpAddressStateTB.Text;
            if (string.IsNullOrWhiteSpace(mainAddress) ||
string.IsNullOrWhiteSpace(phoneNumberTB.Text)
                || string.IsNullOrWhiteSpace(mainAddressStreetTB.Text) ||
string.IsNullOrWhiteSpace(mainAddressNumberTB.Text)
                || string.IsNullOrWhiteSpace(mainAddressZipCodeTB.Text) ||
string.IsNullOrWhiteSpace(mainAddressStateTB.Text))
            {
                MessageBox.Show("Um ou mais campos obrigatórios foram deixados em
branco!");
            }
            else
            {
                if (this.clientId <= 0)
                {
                    try
                    {
                        ClientManager clientManager = new ClientManager();
                        clientManager.insertClient(nameTB.Text, phoneNumberTB.Text,
mainAddress, CNPJTB.Text, economicSectorTB.Text, mainPickUpAddress);
                        MessageBox.Show("Cliente " + nameTB.Text + " inserido com
sucesso!");
                        this.Close();
                    }
                    catch (Exception es)
                    {
                        MessageBox.Show("Ocorreu um erro!\n" + es.Message);
                    }
                }
                else
                {
                    ClientManager clientManager = new ClientManager();
                    clientManager.updateClient(this.clientId, this.nameTB.Text,
this.phoneNumberTB.Text, this.mainAddress, this.CNPJTB.Text,
this.economicSectorTB.Text, this.mainPickUpAddress);
                    MessageBox.Show("Cliente " + nameTB.Text + " editado com
sucesso!");

                    this.Close();
                }
            }
        }
```

Fonte: Do autor

6.2. Conclusão da implementação

A implementação é a parte que mais consumiu recursos ao autor, devido principalmente à falta de conhecimentos prévios sobre a linguagem C# e à dificuldade de se desenhar interfaces de usuário que reajam bem a todos os tamanhos de janela. Seus objetivos foram, porém, alcançados, e o produto final implementa com sucesso os casos de uso considerados prioritários pelo cliente, em especial os referentes à integridade e validação dos dados.

Antes de passar à conclusão do trabalho, é importante mencionar as atividades propostas no Praxis e não executadas pelo autor.

Primeiramente, têm-se a atividade de testes unitários. Testes unitários visam determinar se uma certa unidade de código (em geral uma função) satisfaz seus requisitos. Este tipo de teste pode ser executado ao final da implementação de todas as funções, caso em que requer planejamento e documentação, ou durante a criação dos métodos, no estilo *Test-driven Development* (TDD). Neste caso, o programador só avança para a implementação de uma outra unidade caso a anterior tenha passado nos testes. Para acelerar a implementação e diminuir a documentação necessária, foi adotada a técnica TDD, o que eliminou a necessidade de um plano formal de testes de unidade.

Em seguida, a inspeção de código, que não foi executada de maneira formal pois requer que os inspetores sejam diferentes dos programadores que escreveram o código, o que é impossível no contexto deste trabalho. Pádua Filho ainda cita alguns estudos que colocam em dúvida a eficácia e eficiência da inspeção como instrumento de garantia de qualidade (2003, p. 209), o que contribuiu para que esta atividade fosse totalmente ignorada.

Na sequência, o desenho detalhado foi totalmente relevado, pois considerou-se que o modelo obtido na fase anterior possuía um grau de detalhe suficiente para permitir a implementação.

Por último, a integração foi feita no estilo bottom-up, sem porém seguir um plano formal de execução, dado que o sistema não possui grandes graus de complexidade.

7. Conclusão e análise crítica do trabalho

Esta seção comporta uma análise crítica dos resultados do trabalho, e é dividida em três partes. Na primeira, analisa-se o programa desenvolvido com base no critério de sucesso definido na introdução deste trabalho. Em seguida, é feita uma crítica do método de escolha da metodologia aplicada, e por último o autor discorre sobre a adaptação do método Praxis para problemas pouco complexos.

7.1. Sucesso do programa

No momento de redação desta seção, o software Courribilidade encontra-se completamente finalizado, i.e., todas as suas funcionalidades foram implementadas e testadas no ambiente de desenvolvimento do autor. Retornando aos problemas encontrados pelo gerente financeiro André Biselli, pode-se analisar a real contribuição do sistema para resolvê-los:

- Falta de confiabilidade e homogeneidade dos dados: Problema totalmente resolvido pelo novo sistema. A validação de dados feita dentro do programa e pelo gerenciador de banco de dados permite garantir que o formato dos dados está correto, além de assegurar o fim do problema de referências incorretas
- Dificuldade ou impossibilidade de geração de relatórios: Problema parcialmente resolvido. A geração de relatórios foi implementada no Courribilidade e permite a criação de documentos gerenciais com rapidez e confiabilidade. Falta porém um instrumento semelhante à Tabela Dinâmica, que permita maior flexibilidade destes
- Falta de integração: Problema parcialmente resolvido. O controle de entregas se faz de maneira muito mais rápida segura com o uso do Courribilidade, o que facilita o processo de cobrança e cálculo de receitas. A ambição de integrar todos os aspectos relevantes da empresa, como comunicação com o aplicativo da empresa não foi implementada

A real medida de sucesso virá porém quando o sistema for completamente implementado na empresa e for avaliado pelos funcionários, em especial quanto à manutenção dos benefícios que a planilha de controle atual possui.

7.2. Crítica à escolha da metodologia

O processo de escolha da metodologia de desenvolvimento foi considerado pelo autor como ineficaz, pois, analisando o andamento do processo de desenvolvimento uma vez terminado este, acredita-se que um método ágil teria sido mais bem adaptado ao problema.

Uma sugestão para trabalhos futuros por parte do autor é testar, durante uma semana, ao menos dois métodos de escolas diferentes e só então tomar uma decisão. A escolha feita para este trabalho foi tomada com base em conceitos puramente teóricos que na prática se mostraram menos relevantes do que o imaginado. É importante enxergar o funcionamento dos métodos na prática para poder fazer uma escolha mais informada.

7.3. Crítica ao método Praxis

Na opinião do autor, métodos do estilo UP, quando aplicados a projetos de complexidade relativamente baixa como o Courribilidade, dão ênfase exagerada na documentação, tomando recursos que poderiam ser aplicados na implementação. Um número considerável de atividades propostas por Pádua Filho foi deixado de lado para acelerar o trabalho de implementação. O código-fonte, quando utiliza boas práticas de formatação e nomeação de variáveis, pode ser em si uma documentação de grande valor.

A documentação não gera valor suficiente na coordenação do projeto para compensar pelo tempo e esforço que consome, pois a gestão do desenvolvimento é em si relativamente simples para projetos com poucos casos de uso e poucas classes de entidade, e pode portanto ser feita ser uso de metodologias especializadas.

É importante frisar que estas colocações só se aplicam para projetos simples e em que não há necessidade de coordenar o trabalho de diversos programadores. Para empreitadas de maior complexidade, o autor imagina que a documentação precisa e a maior ênfase no planejamento dos métodos UP pode trazer valor ao trabalho.

Referências bibliográficas

ABELSON, N. **Accounting for startups: depoimento**. Disponível em https://www.xero.com/small-business-guides/cloud-accounting/accounting-for-startups. Acesso: 15 de abr. 2015. [S.l.]. Entrevista concedida ao site Xero.

AGILE ALLIANCE. **What is Agile Software Development?** Disponível em http://www.agilealliance.org/the-alliance/the-agile-manifesto. Acesso em 20 de março de 2015.

AGILE ALLIANCE. **The Agile Manifesto**. Disponível em http://www.agilealliance.org/the-alliance/the-agile-manifesto. Acesso em: 20 de março 2015.

ALBAHARI. **A comparative overview of C#**. Disponível em http://genamics.com/developer/csharp_comparative.htm. Acesso em: 1° de junho de 2015.

ANGELES, S. **Best Accounting Software**. Disponível em: http://www.businessnewsdaily.com/7543-best-accounting-software.html. Acesso em: 30 de março, 2015.

BARLOW, J. **Overview and Guidance on agile development in large organizations**. Communications of the Association for Information Systems. [S.l], Vol. 29, No. 1, janeiro 2006, pp. 25-44

BOOCH, G. et al. **Object-oriented analysis and design**. 3rd Edition. Upper Saddle River: Addison-Wesley, 2007.

CUSUMANO, M.; SELBY, R. **How Microsoft Bulds Software**. Communications of the ACM. New York, Vol. 40, No. 6, junho de 1997, pp. 53-61

DEPARTMENT OF DEFENSE. **Systems Engineering Fundamentals**. Supplementary text. Fort Belvoir, Virginia: Defense Acquisition University Press. Jan 2001.

GLASS, R. **Matching methodology to problem domain**. Communications of the ACM. New York, Vol. 47, No. 5, maio de 2004, pp. 19-21

GRONINGER, T. **Accounting chores: doing more with less, and yet…** The NonProfit Times. Morris-Plains, NJ, 2011, Vol. 25, No. 1, janeiro 2011, p. 19

KUHN, T. **The structure of scientific revolutions**. 2nd Edition, Enlarged. Chicago: The University of Chicago Press. 1962

KRATOCHVIL, W. **Windows Store C++ for C# developers**. MSDN Magazine. [S.l.], abril 2014.

KRIGE, Danie. **Selection criteria for a development methodology**. Disponível em https://www.linkedin.com/grp/post/3774402-254265649. Acesso em: 27 de mar, 2015.

MARKS, G. **And the best small business cloud accounting software is…** Disponível em http://www.forbes.com/sites/quickerbettertech/2014/05/05/and-the-best-small-business-cloud-accounting-software-is/. Acesso em: 4 de maio de 2015.

THE MICROSOFT CORPORATION. **Chapter 3: Choosing Windows develpment technologies**. Disponível em: https://msdn.microsoft.com/en-us/library/windows/desktop/ff795785.aspx. Acesso em: 15 de abril de 2015.

THE MICROSOFT CORPORATION. **Choosing the programming language**. Disponível em https://msdn.microsoft.com/en-us/library/windows/apps/dn263221.aspx. Acesso em: 20 de mar 2015.

MONGODB. **Top 5 considerations when evaluating NoSQL Databases**. New York: MongoDB, fevereiro de 2015.

NIXON, Jerry. **Choosing between Visual Basic and C#**. Disponível em http://blog.jerrynixon.com/2014/01/choosing-between-visual-basic-and-c.html. Acesso em 20 de maio de 2015.

PAULA FILHO, W. **Engenharia de Software**. 1ª Edição. Rio de Janeiro: LTC Editora, 2003.

PAULA FILHO, W. **Engenharia de Software**. 3ª Edição. Rio de Janeiro: LTC Editora, 2010.

PEPSICO BRASIL. **Prêmio internacional Eco-Challenge reconhece equipe brasileira por iniciativa sustentável**. Disponível em http://www.pepsico.com.br/premio-internacional-eco-challenge-reconhece-equipe-brasileira-por-iniciativa-sustentavel. Acesso em: 20 de outubro de 2014.

RAJLICH, V. **Changing the paradigm of software engineering**. Communications of the ACM. New York. Vol. 2, No. 8, agosto de 2006, pp. 67-70

SOUZA, A. et al. **Custo do Trabalho no Brasil: Proposta de uma nova metodologia de mensuração**. São Paulo: FGV C-Micro, maio de 2012.

WILSON, J. **The Best Free Small Business Accounting Software**. Disponível em http://www.pcmag.com/article2/0,2817,2382514,00.asp. Acesso em: 31 de maio, 2015.