THIAGO PINTO MOHAMED

MIGRAÇÃO DE UM SISTEMA LEGADO DE COBRANÇA UTILIZANDO METODOLOGIA ÁGIL COM FOCO EM MELHORIA DE PROCESSO

LAVRAS – MG

2019

THIAGO PINTO MOHAMED

MIGRAÇÃO DE UM SISTEMA LEGADO DE COBRANÇA UTILIZANDO METODOLOGIA ÁGIL COM FOCO EM MELHORIA DE PROCESSO

Trabalho de conclusão de curso, na modalidade Relatório Técnico, apresentado à Universidade Federal de Lavras, como parte das exigências do Curso de Engenharia de Controle e Automação, para a obtenção do título de Bacharel.

Prof. DSc. Wilian Soares Lacerda

Orientador

LAVRAS – MG

2019

Ficha catalográfica elaborada pela Coordenadoria de Processos Técnicos

da Biblioteca Universitária da UFLA

Thiago Pinto Mohamed

Migração de um Sistema Legado de Cobrança Utilizando Metodologia Ágil com Foco em Melhoria de Processo / . 1^a ed. rev., atual. e ampl. – Lavras : UFLA, 2019.

52 p. : il.

TCC(graduação)–Universidade Federal de Lavras, 2019.

Orientador: Prof. DSc. Wilian Soares Lacerda.

Bibliografia.

1. Desenvolvimento de Software. 2. Melhoria de processos. 3. Metodologia ágil

Aos meus pais, Ali e Inês, meus irmãos, Enzo e Joselise, e aos meus amigos por todo o apoio e companheirismo até aqui.

AGRADECIMENTOS

Ao professor Wilian pelo apoio e orientação durante o desenvolvimento deste trabalho.

À Universidade Federal de Lavras e ao Departamento de Engenharia, obrigado pela oportunidade.

À minha família, em especial pai e mãe pelo incrível apoio em todas as decisões que tomei. À meus queridos irmãos, por serem as melhores pessoas.

À todos meus amigos e companheiros de curso, em especial Xiadu, Póça, Camis e Igor, por todos os momentos.

À meus colegas de trabalho, em especial Francis, pela ideia e auxílio durante este trabalho.

A todos que de forma direta ou indireta contribuíram para esta realização.

Muito obrigado!

All we have to decide is what to do with the time that is given to us.

Lord of the Rings: The Fellowship of the Ring

RESUMO

Com a grande competitividade do mercado é natural que mudanças ocorram rapidamente, exigindo uma maior agilidade nas respostas das empresas, principalmente da área de TI. Assim como o desenvolvimento de novas soluções sistêmicas é importante, a manutenção de seu ambiente também é de relevância. Um ponto que eleva custos de manutenção são sistemas legados, além de serem relacionados com processos burocráticos, o que torna as atividades muito manuais para os usuários do sistema. Uma das maneiras de melhorar este processo é automatizando rotinas operacionais e repetitivas com *softwares* a fim de otimizar o tempo dos colaboradores de uma empresa. O objetivo deste trabalho é construir uma aplicação Web como solução de um problema envolvendo um sistema legado de cobrança no cartão de crédito, no setor de gestão de frotas em uma empresa de aluguel de carros. Para atingir o objetivo foi utilizado a metodologia Scrum para o desenvolvimento e ferramentas como Visual Studio, ASP.NET MVC Framework, as linguagens C#, Razor, CSS e JavaScript e banco de dados Sybase. Os resultados obtidos foram o aumento da produtividade dos colaboradores envolvidos no processo de cobrança além de redução do custo no médio prazo.

Palavras-chave: Desenvolvimento de Software. Melhoria de processos. Metodologia ágil.

ABSTRACT

With the great competitiveness of the market it is natural that changes occur quickly, requiring a faster response from companies, especially from IT. Just as developing new systemic solutions is important, maintaining your environment is also of relevance. One thing that raises maintenance costs is legacy systems, as well as being related to bureaucratic processes, which makes activities very manual for system users. One way to improve this process is by automating repetitive operational routines with software to optimize the time of a company's employees. The purpose of this paper is to build a web application as a solution to a problem involving a legacy credit card billing system in the fleet management sector of a car rental company. In order to achieve this goal, the Scrum methodology was used for development and tools such as Visual Studio, ASP.NET MVC Framework, C#, Razor, CSS and JavaScript and Sybase database. The results obtained were increased productivity of people involved in the billing process and consequently reduction of operation costs.

Keywords: Software development. Agile methodology. Processes improvement.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

HTML Sigla de *HyperText Markup Language*, expressão inglesa que significa "Linguagem de Marcação de Hipertexto". Consiste em uma linguagem de marcação utilizada para produção de páginas na Web. 20, 21

back-end Parte do *software* que roda no servidor, por isso também é conhecida como *server-side*. Fornece e garante todas as regras de negócio, acesso a banco de dados, segurança e escalabilidade. 42

API *Application Programming Interface*, fornece de rotinas e padrões de programação para acesso a um aplicativo de software ou plataforma baseado na Web. 20, 45

IDE *Integrated Development Environment*, ou Ambiente de Desenvolvimento Integrado, é um programa de computador que reúne ferramentas para auxiliar o desenvolvimento de *softwares*. 20

Query Representa um pedido de uma informação ou de um dado a um banco de dados. Esse pedido também pode ser entendido como uma consulta, uma solicitação ou, ainda, uma requisição. 41

TI Sigla de Tecnologia da Informação. 11, 15

UI *User Interface*, parte visual de um projeto, que fornece meios para que o usuário interaja com a aplicação. 20

SUMÁRIO

1 INTRODUÇÃO

Desde a década de 1990, mudanças são consideradas marcas comerciais do modelo de negócio das empresas dos mais diversos ramos. A frequência com que as mudanças externas ocorrem exige que empresas considerem o gerenciamento delas como uma atividade extremamente importante para a sua manutenção no mercado. Sendo assim, é comum que áreas das empresas estejam tentando responder com velocidade a estas mudança, sendo a TI (Tecnologia da Informação) uma das principais afetadas.

A TI têm papel fundamental em apoiar estas mudanças, visto que a internet é força para distribuir e gerir informações por diversas atividade humanas (CASTELLS, 2009). Deste modo, para uma empresa acompanhar as rápidas mudanças do ambiente, é necessário investimento em TI.

Contudo, apenas a construção de novos sistemas e soluções pela TI não é suficiente para acompanhar este cenário de constantes mudanças. A manutenção de soluções já existentes também é fundamental para garantir que o atendimento das necessidades dos clientes e que o cumprimento de prazos seja cumprida. Esta atividade de preservar o ambiente têm sido um desafio para companhias e segundo GRUBB P.; TAKANG (2003), cerca de 70% dos custos de uma empresa com *software* se concentra em sua manutenção.

Dentre os fatores que contribuem ao elevado percentual gasto com manutenção de sistemas, o principal se deve aos sistema legados. Além de complexa manutenção por razão de suas tecnologias defasadas, estes sistemas geralmente são associados a processos críticos do modelo de negócio das empresas, aumentando ainda mais o esforço de manutenção. Sendo assim, a migração destes sistemas para tecnologias mais recentes proporciona diversas melhorias e diminuição dos riscos.

Dentre os motivos para realizar esta migração, estão a falta de documentação, evolução complicada e processos defasados envolvidos com o sistema. Na maioria das vezes, o processos associado a utilização de um sistema legado é demorado, burocrático e manual. Isso se dá ao fato destes sistemas terem sido desenvolvidos a muitos anos, onde o foco, necessidade do negócio e tecnologias impediam a automação.

Entretanto, desenvolver novos *softwares* não é um procedimento simples. Na década de 90, a grande maioria dos projetos utilizava modelo de desenvolvimento em cascata, onde mudanças nos requisitos e adaptações eram difíceis de serem realizadas em fase posteriores ao planejamento, devido a exigirem um retorno a fase de concepção do projeto. Em oposição a

este modelo, a partir dos anos 2000, novos modelos surgiram, conhecidos como metodologias ágeis. Este tipo de desenvolvimento prioriza o trabalho iterativo com entregas em tempo curto para obter *feedback* constante.

O estudo de caso deste trabalho envolve uma empresa de aluguel de carros que atualmente é líder de mercado na América Latina. A plataforma de negócios envolve aluguel de carros para pessoa física, jurídica, venda de carros usados e gestão de frotas. O trabalho terá foco na gestão de frotas.

No final de 2019, a gestão de frotas alugava cerca de 60 mil veículo para as mais diversas empresas. Esta área é responsável pela parte estratégica e operacional do gerenciamento dos meios de transporte e seus respectivos condutores, garantindo assim que o cliente tenha eficiência e tranquilidade com a frota. A terceirização da frota pode trazer diversos benefícios, como permitir o foco da empresa em seu negócio, administração da troca e manutenção dos veículos, assistência 24 horas, carros novos, gerenciamento da documentação dos carros, faturamento simplificado, além da gestão de avarias e multas dos condutores. Existe grande personalização em como serviços são oferecidos aos clientes e quais deles estarão disponíveis. Por exemplo, algumas empresas optam para que o pagamento dos alugueis dos automóveis, multas e avarias ocasionadas pelos condutores seja realizado pela própria empresa. Porém, outras optam para que o usuário de cada veículo seja responsável por pagar suas multas, avarias e até mesmo parte do aluguel. Para que o cliente não seja responsável por gerenciar esta cobrança, a gestão de frotas se encarrega de administrar a cobrança de cada usuário do cliente.

O percentual que a empresa e os usuários pagam é determinado pelo cliente. Para que o condutor realize o pagamento destas multas, é necessário que o mesmo forneça seus dados de cartão de crédito para a gestão de frotas, e assim a cobrança é realizada por meio dele.

Este tipo de cobrança é crítica para a empresa, pois é realizada no cartão de crédito de funcionários de outra empresa, o que pode gerar diversos problemas, como vazamento de informações de cartões ou cobranças indevidas. Atualmente, o processo é realizado por meio de um sistema chamado Débito no Cartão de Crédito, da gestão de frotas.

O sistema de Débito no Cartão de Crédito é legado e exige diversas intervenções do usuário, além de possuir alguns problemas. O processo envolvido é moroso, pois faz com que o colaborador realize a cobrança de cada extrato por vez, além do sistema possuir experiência ruim.

Apesar do processo de cobrança ser demorado e burocrático, é possível utilizar tecnologias como o desenvolvimento de software para automatizar partes do processo e otimizar o fluxo de trabalho. Investir em tecnologia permite que muitas rotinas operacionais sejam concluídas de forma automática e diminui a quantidade de tarefas repetitivas e burocráticas sob a responsabilidade dos colaboradores. Assim, consegue-se alcançar economia de tempo, fazendo com que as equipes concluam um volume maior de demandas em menos tempo.

Portanto, para se corrigir todos estes problemas foi proposta a construção de um novo sistema, em uma tecnologia mais recente, que assim possibilitasse melhorias nos pontos do processo.

1.1 Objetivos

Este trabalho possui o objetivo de migrar um sistema Web legado de cobrança no cartão de créditos dos clientes para uma empresa de aluguel de frotas de carros. Para isso, tais elementos foram considerados:

- Levantar os problemas do processo atual, por meio de entrevistas com colaboradores relacionados com o sistema
- Estudar e levantar regras do sistema legado, assim identificando problemas e pontos de melhorias na experiência do usuário e agilidade do negócio.
- Realizar a especificação, juntamente com a área de negócio, das funcionalidades do novo sistema, para que assim seja possível automatizar etapas do processo para os usuários
- Desenvolver o novo sistema com tecnologia mais recente visando automatização e melhoria do processo, utilizando de metodologias ágeis para que seja possível realizar entrega mais rápida.
- Mostrar os benefícios do novo sistema em comparação como o legado, enfatizando a experiência do usuário e melhoria de processo.

1.2 Contribuições

Foi desenvolvido um novo sistema de cobrança no cartão de crédito dos clientes utilizando metodologia ágil para substituir a utilização de um legado. Tal migração sistêmica

permitiu a realização de correções de grandes problemas no processo, além de tornar a realização de uma tarefa crítica, como cobrança no cartão de crédito, mais segura. A utilização de metodologia ágil permitiu que o escopo do projeto fosse alterado durante desenvolvimento de acordo com demandas do negócio, aumentando o valor da entrega do software. Com isso, os estudos realizados ao longo deste projeto podem fornecer um incentivo para migração de sistemas legados em empresas, para que assim a área de negócio seja favorecida com melhores processos e o cliente seja melhor atendido.

1.3 Estrutura do trabalho

O trabalho será composto por quatro capítulos, da forma: O Capítulo 1 introduz o trabalho, suas motivações, assim como objetivos esperados e contribuições. No Capítulo 2, é realizado o referencial teórico a fim de contextualizar o leitor sobre temas discutidos nos próximos capítulos. O Capítulo 3 apresenta as metodologias e ferramentas utilizadas durante o trabalho. O 4º Capítulo expõe os resultados do projeto. Finalmente, o Capítulo 5 traz as considerações finais e trabalhos futuros a respeito dos objetivos pretendidos e alcançados.

2 REFERENCIAL TEÓRICO

O objetivo deste capítulo é fornecer ao leitor uma base de conhecimento, para que assim a compreensão dos capítulos posteriores seja fluída.

2.1 Sistemas legados

Um sistema que continua a ser utilizado por razão do custo de substitui-lo e redesenha-lo mesmo com deficiências na competitividade, performance e usabilidade é considerado um sistema legado. Muitas vezes, estes sistemas são grandes, monolíticos e difíceis de serem alterados (FOLDOC, 1998). A TI evolui de forma rápida, porém sistemas legados não acompanham esta evolução devido a seu tamanho, complexidade e criticidade para o negócio.

2.1.1 Por que migrar sistema legados

A seguir são apresentados alguns motivos para se migrar sistemas legados para novas tecnologias segundo (MATTOS, 2015):

- Manutenção de alto custo devido a dificuldade de rastrear problemas, além de falta de documentação e pouco entendimento da implementação usada;
- Integração com outros sistemas é problemática devido a falta de interfaces e compatibilidade;
- Geralmente estes sistemas rodam em hardware e plataformas obsoletas, o que os deixa lentos;
- Evolução do sistema é praticamente impossível, o que pode impossibilitar implementações de novos modelos de negócio de forma rápida;
- Dificuldade ou impossibilidade de separação das regras de negócio das camadas de dados e apresentação;
- Sistemas sem testes automatizados ou unitários, o que impede aprimorar funcionalidades existentes.

2.1.2 Estratégia para migração

Mattos (2015) exemplifica que o processo para migração de um sistema legado deve seguir os seguintes passos:

1. Justificativa e motivação para migração do sistema legado.
2. Entendimento do sistema ou aplicação legada.
3. Desenvolvimento ou refatoração visando a nova plataforma ou versão.
4. Testabilidade do novo sistema.
5. Execução da migração e assim substituindo de fato o sistema legado.

2.2 *Softwares* de automação de processos

Segundo Pluga.co (2016), *softwares* de automação existem para melhorar um processo (de produção, prestação de serviço, vendas e quaisquer outros) ao substituir tarefas repetitivas e burocráticas operadas por pessoas por uma forma de integração com um sistema, tornando-a automática.

Sendo assim, *softwares* de automação têm como objetivo trazer maior produtividade e eficiência para operação do negócio das empresas. Os principais benefícios de automatizar processos são: diminuição de erros humanos e desperdícios, aumento da velocidade do processo, melhora na comunicação, base de dados confiável, geração de relatórios, melhor controle e rastreabilidade.

A automação de processos por meio de implementações de *softwares* deve começar por mudanças básicas e deve ser realizado após o mapeamento do processo. Realizar este mapeamento ajuda a identificar pontos de melhoria e automação (SML, 2018).

2.3 Arquitetura de *software*

De acordo com a definição clássica de Shaw (1996), arquitetura de *software* define os componentes de um sistema e o relacionamento entre eles. Adicionalmente descreve a estrutura técnica, limitações e características dos componentes, bem como as interfaces entre eles. O esqueleto do sistema é definido por sua arquitetura e desta forma, ao se construir um sistema, a determinação da arquitetura é o mais alto nível de abstração (KRAFZIG; BANKE, 2004).

A arquitetura funciona como facilitador na construção de sistemas, padronizando interfaces, tornando sistemas menos acoplados, mais refatoráveis e simples de modificar. A divisão do projeto de software em camadas é uma prática que vem sendo utilizada há vários anos e, em boa parte dos casos, atende às necessidades iniciais dos sistemas.

Se tratando de ambientes corporativos, a arquitetura de *software* das corporações deve ser simples (para que todos os programadores possam entendê-la e utilizá-la); flexível (para suportar em tempo as alterações requeridas pela área de negócios); reutilizável (sobretudo dos blocos de softwares); e ser capaz de desvincular funcionalidades do negócio das tecnologias utilizadas para sua execução.

2.3.1 Arquitetura em camadas UI, BLL e DAL

As camadas em uma arquitetura de *software* são usadas para separar responsabilidades e tornar o código mais fácil de manter e ser reutilizado. Uma arquitetura muito comum é a separação em três camadas de DAL, BLL e UI. Nesta arquitetura é possível criar um nível de abstração entre as camadas. Tal prática torna possível alterar uma camada sem interferir em outra, auxiliando na manutenção do código. Segundo (MICROSOFT, 2010), as camadas podem ser definidas da seguinte forma:

- **DAL:** Do inglês *Data Access Layer* ou Camada de Acesso a Dados. O desenvolvedor deve trabalhar constantemente com dados e uma forma muito comum de gerenciamento dos dados é por meio de bancos de dados. Para manipular os dados é necessário desenvolver códigos responsáveis por tal. A camada é encarregada por acessar e persistir os dados da aplicação.

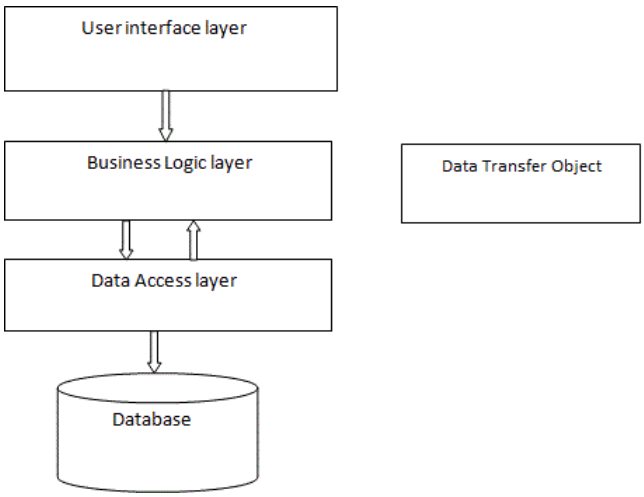
- **BLL:** *Business Logic Layer* ou Camada de Negócio é responsável por intermediar a relação entre a camada de interface com o usuário e a de DAL. É esta camada que contém todas as regras de negócio da aplicação.

- **UI:** *User Interface* ou Camada de Interface é encarregada de fornecer a interação e apresentação dos dados ao usuário.

Para facilitar a transição dos dados entre as camadas, uma quarta camada pode ser acrescentada a arquitetura. Conhecida como DTO (*Data Transfer Object*, ela é incumbida pela transferência de dados entre as camadas, ficando visível e acessível por todos os demais níveis (MACORATTI, 2010).

Cada camada deve utilizar os recursos disponibilizados pelas camadas inferiores hierarquicamente e nunca devem ter conhecimento das implementações das camadas superiores (SANTOS, 2005). Seguindo este princípio, DAL não deve ter acesso a UI ou BLL, e BLL não pode ter acesso a UI. Assim como UI não deve acessar diretamente a camada de DAL. Sendo assim, é possível montar o esquemático presente na Figura 2.1.

Figura 2.1 – Representação da Arquitetura de camadas DAL, BLL e UI



Fonte: do autor

2.3.2 Arquitetura MVC

Em 1988, a arquitetura MVC foi descrita em detalhes por Krasner e Pope e é o acrônimo de *Model-View-Controller*. Eles explicam que existem muitos benefícios ao se construir uma aplicação pensando em sua modularidade, onde ao isolar as unidades uma das outras o máximo possível, torna mais fácil ao desenvolvedor modificar determinada unidade sem conhecer detalhes da implementação da outra (KRASNER; POPE, 1988).

O padrão MVC envolve três camadas, chamadas de *Model*, *View* e *Controller*. Portanto, o objetivo do MVC é desacoplar a interface do usuário (*View*), os dados (*Model*) e lógica da aplicação (*Controller*).

2.3.2.1 *Model*

O *Model* ou modelo é a parte do sistema que gerencia todas as tarefas relacionadas aos dados, como validação, estado e controle. A utilização do *Model* diminui consideravelmente a complexidade do código desenvolvido (GILMORE, 2009).

A camada de modelo é responsável pelas regras de negócio da aplicação e encapsula métodos para o acesso dos dados. Além disso, o modelo é formado de classes que definem o domínio de interesse. Estes objetos que pertencem ao domínio muitas vezes encapsulam dados que são guardados no banco de dados, mas também incluem código para manipular estes dados e realizar regras de negócio (HAACK; GALLOWAY, 2011).

2.3.2.2 *View*

A camada *View* é responsável pelo gerenciamento da interface gráfica do usuário. Isso significa que todos os formulários, botões, gráficos e elementos HTML estarão nesta camada. Ao separar o *design* da aplicação de sua lógica, é possível reduzir significativamente o risco de erros ao alterar alguma das camadas (HAACK; GALLOWAY, 2011)

A *View* controla a forma com que os dados são exibidos e como o usuário interage com eles. Também fornece formas de obter os dados do usuário para processá-los. Como regra geral, uma *View* nunca deve conter elementos que pertencem a lógica da aplicação de forma a propiciar facilidade ao seu desenvolvedor (GILMORE, 2009).

2.3.2.3 *Controller*

O *Controller* é responsável por lidar com eventos, podendo eles ser de origem do usuário ou por processo do sistema. Ele é recebe uma requisição e prepara o dado para a resposta e seu respectivo formato. O *Controller* interage com o *Model* com o objetivo de obter e manipular os dados e gerar as *Views* (HAACK; GALLOWAY, 2011). O *Controller* reúne a lógica da aplicação com a visualização da *View* e as funcionalidades do *Model*. A Figura 2.2 representa este fluxo.

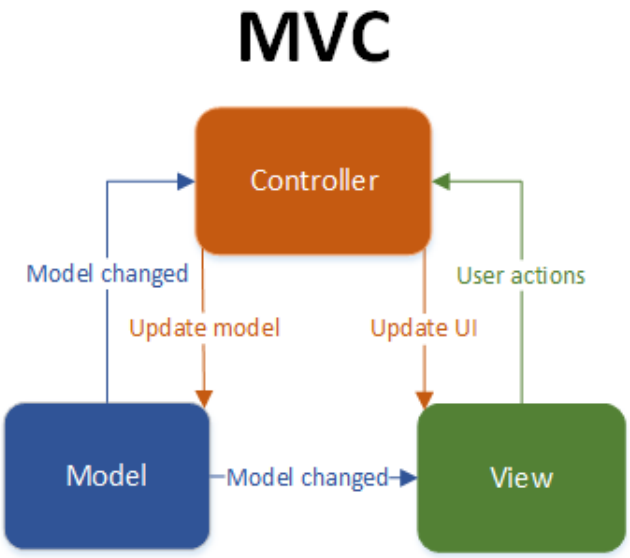
2.4 Tecnologias

Nesta seção será descrito algumas tecnologias utilizadas para a análise e desenvolvimento de sistemas deste trabalho, assim como seus respectivos problemas e alternativas.

2.4.1 .NET Framework 4.0

O .NET Framework 4.0, lançado em 12 de Abril de 2010 (PROTALINSKI, 2010), é uma evolução da iniciativa da Microsoft que têm como objetivo criar uma plataforma única para

Figura 2.2 – Representação da arquitetura MVC



Fonte: (MEDIUM, 2017)

desenvolvimento e execução de sistemas e aplicações. Consiste de dois componentes principais, sendo executada em uma CLR (*Common Language Runtime*) que é um ambiente de execução que independe da linguagem e uma FCL (*Framework Class Library*) - Conjunto de Bibliotecas Unificadas (MICROSOFT, 2017).

A CLR provê gerenciamento de memória, processamento concorrente e paralelo, segurança, compilação, interoperabilidade entre outros. Já o FCL fornece diversas APIs para UI, acesso a dados, conectividade web, bancos de dados, redes, serviços do sistema operacional e algorítimos auxiliares.

Para a versão 4.0 as principais melhorias são: computação paralela melhorada, linguagens melhoradas, novos tipos de dados e CLR 4.0 (PROTALINSKI, 2010). Porém utilizar o .NET Framework 4.0 lançado em 2010, deixa o sistema atrasado em melhorias de performances e funcionalidades oferecidas em novas versões do .NET Framework.

2.4.2 *Web Forms* do ASP.NET

O *Web Forms* do ASP.NET é modelo de programação para criação de aplicativos Web ASP.NET. O *Web Forms*, ou formulários da web, consiste em processar requisições de páginas feitas pelo usuário no servidor, compilando e executando o código e retornando o *HTML* para o navegador. Utilizando da IDE *Visual Studio*, da Microsoft, é possível desenvolver sites com o arrastar de componentes, tendo um desenvolvimento fácil e rápido (MICROSOFT, 2014). O

ASP.NET *Web Forms* apesar de ser uma rápida ferramenta para desenvolvimento, a utilização do *Web Forms* causa alguns débitos técnicos como:

- Utilização de formulários construídos em servidor impossibilita o controle do desenvolvedor sobre o comportamento do aplicativo;
- Difícil organização entre camadas de negócio e acesso a dados e apresentação;
- Impossibilidade de realizar um desenvolvimento orientado a testes. As aplicações nascem sem testes unitários e automatizados, o que as tornam sensíveis a modificação;
- O*HTML* gerado pelo *Web Forms* é mais pesado do que um código puro. Isso é causado devido a grande quantidade de informações geradas automaticamente pelo *framework*, o que deixa a transferência de dados mais lenta entre o servidor e cliente;
- Dificulta a modularização do código-fonte, o que ocasiona falta de organização.

2.4.3 ASP.NET MVC

Em contrapartida ao ASP.NET *Web Forms*, em 2009 surgiu o *framework* ASP.NET MVC (ESPOSITO, 2010). No centro desta mudança, está a separação do comportamento da geração da resposta através da implementação da arquitetura MVC.

Nesta ferramenta, cada requisição resulta na execução de uma ação. Diferentemente do *Web Forms*, ASP.NET MVC é formado por várias camadas de código que não são entrelaçadas, não formando um grande bloco de código monolítico. Desta forma é possível alterar qualquer destas camadas por outras personalizadas para melhorar a testabilidade e manutenção da solução (ESPOSITO, 2010).

Um benefício deste *framework* é a criação de *Web* API de forma integrada (ANDERSON; WASSON, 2019).

2.5 Modelos de desenvolvimento de *software*

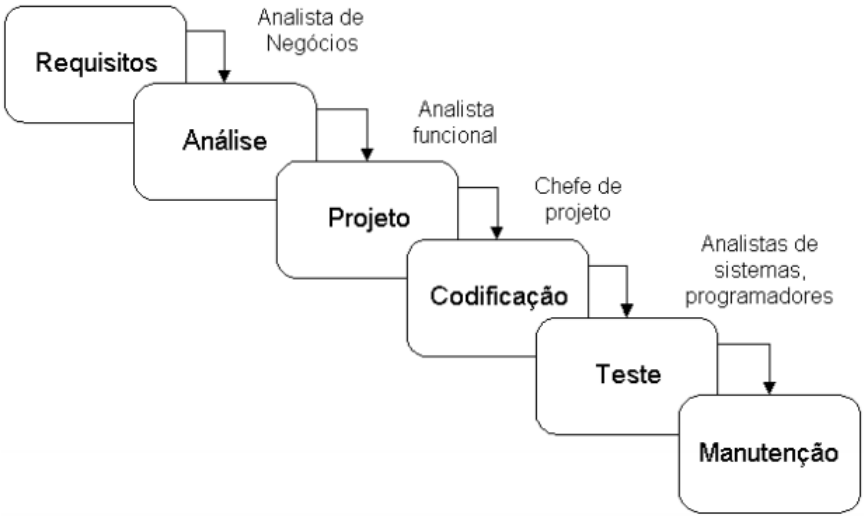
Será abordado, nesta seção, algumas metodologias seguidas pelas empresas para o desenvolvimento de *softwares*.

2.5.1 Modelo de desenvolvimento em cascata

O modelo cascata, ou do inglês *waterfall*, é composto por atividades sequenciais seguindo por levantamento de requisitos, análise, projeto, implementação, teste, implantação e manutenção (SOARES, 2004). Este tipo de metodologia foi herdada de engenharias tradicionais como civil, mecânica e elétrica.

Cada etapa só é iniciada após o término da outra, como o fluir constante para frente, conforme Figura 2.3

Figura 2.3 – Esquema do modelo cascata.



Fonte: (LESSA; JUNIOR, 2010)

Sommerville (2011) descreve o fluxo do desenvolvimento em cascata com cada etapa da seguinte forma:

1. **Requisitos:** Levantamento de requisitos do *software*. Nesta fase são definidos os requisitos de serviços, restrições e metas por meio de consulta a usuários.

2. **Análise:** Análise dos requisitos definidos na etapa anterior, para que sejam usados como especificações.

3. **Projeto:** O processo de projeto de sistemas aloca os requisitos para o *hardware* e *software*, por meio da definição de uma arquitetura geral do sistema. O projeto de software envolve identificação e descrição das abstrações fundamentais do sistema de software e seus relacionamentos.

4. **Codificação:** Durante esta etapa, os desenvolvedores realizam a construção do programa ou conjunto dos mesmos, realizando testes de unidade para que atendam a especificação.

5. **Teste:** As partes individuais do programa são unidas para que se possam realizar testes de integração e garantir que os requisitos do *software* tenham sido atingidos.

6. **Manutenção:** Após o programa ser colocado em produção, devem-se corrigir defeitos originados no desenvolvimento que não estiverem de acordo com as especificações.

Este modelo impõe que o processo de desenvolvimento de *software* seja linear, com uma etapa só iniciando após aprovação da etapa inicial. Até a década de 90, o modelo cascata dominava a forma de desenvolvimento de *software*, porém alguns problemas foram diagnosticados ao adotar a visão sequencial das tarefas.

Por exemplo, as mudanças de requisitos são de difícil ou impossível realização, devido demandar que o processo seja retornado a etapa inicial de levantamento de requisitos (TOMAS, 2009) para que a mudança possa ser implementada. Complementando, Brooks (1987) descreve que a ideia de especificar totalmente um *software* antes do início de sua implementação é impossível.

De acordo com dados da International (1995), tomando como base 8380 projetos de TI, apenas 16,2% foram entregues dentro do prazo, com custo e funcionalidades previstas. Houveram 31% cancelados e 52,7% entregues com prazos maiores, custos maiores ou fora do escopo previsto.

Esta metodologia obtêm sucesso onde os requisitos de *software* são estáveis e requisitos futuros previsíveis. Porém, este tipo de cenário é raro e por isso outros modelos foram desenvolvidos para que seja possível atender a necessidade das empresas, chamadas de metodologias ágeis.

2.5.2 Metodologias ágeis

Em resposta aos modelos de desenvolvimento pesados, como o cascata, na década de 1990 começaram a surgir metodologias alternativas, para que assim, fosse possível acabar com os problemas que o excesso de regras, lentidão, burocracia e imutabilidade ocasionavam nas antigas metodologias.

Em 2001, dezessete especialistas em processos de desenvolvimento de *software*, representando os métodos *Extreme Programming* (XP), *Scrum*, *Crystal* e outros, estabeleceram princípios comuns a estes métodos. Esta reunião tornou popular o termo "metodologias ágeis". O resultado foi a criação da Aliança Ágil e assim o "Manifesto Ágil"(BECK, 2001). São princípios do manifesto ágil:

- Indivíduos e interações ao invés de processos e ferramentas.
- Software executável ao invés de documentação.
- Colaboração do cliente ao invés de negociação de contratos.
- Respostas rápidas a mudanças ao invés de seguir planos.

O manifesto ágil não descarta processos, ferramentas, negociação de contrato ou planejamento, mas mostra que os mesmos tem importância secundária quando comparado com os pontos que os contrastam. Nas próximas seções, serão apresentadas algumas metodologias ágeis.

2.5.2.1 *Extreme programming*

Conhecido popularmente por *XP*, o *Extreme programming* é uma metodologia possui o objetivo de melhorar a qualidade do *software* e reagir melhor a mudanças de requisitos dos clientes. Ela se baseia nos princípios de simplicidade, comunicação e *feedback* (BECK, 2004). Desta forma, provendo que o projeto seja executado dentro do prazo e com satisfação para o cliente.

Esta metodologia ganhou seu nome pois traz diversos conceitos de programação considerados bons e os leva ao extremo, como testes, refatoração de código, revisão de código, clareza e simplicidade. Segundo Beck (2004), do *XP* define quatro atividades que devem ser realizadas durante o processo de desenvolvimento do *software*:

- **Codificação:** O único real produto do sistema é seu código. Sem ele não existe real produto. Código pode ser utilizado para se definir a melhor solução. Por exemplo, um programador a frente de um difícil problema, pode codificar uma solução para que assim outros pares possam revisá-lo ou codificar outras soluções. Código é simples e conciso e

só têm um significado, impossibilitando falhas de comunicação. É indicado que a codificação seja realizada em dupla, com uma pessoa controlando o teclado e mouse e a outra observando e sugerindo melhorias, correções e refatorações.

- **Testes:** Testes são fundamentais para o *XP*, tendo o pensamento que se alguns testes podem eliminar algumas falhas, muitos testes podem remover muitas falhas.

  Os testes de unidade verificam se o código funciona como deveria. Eles devem tentar cobrir todo o código e buscar todos os tipos de falhas possíveis. Devem ser escritos antes do código, e assim, quando o código estiver finalizado, se ele passar nos testes está pronto. Os testes devem ser automatizados.

  Testes de integração entre sistemas são incentivados para que ocorram diariamente, e assim incompatibilidades entre interfaces e outros problemas sejam rapidamente identificados e corrigidos.

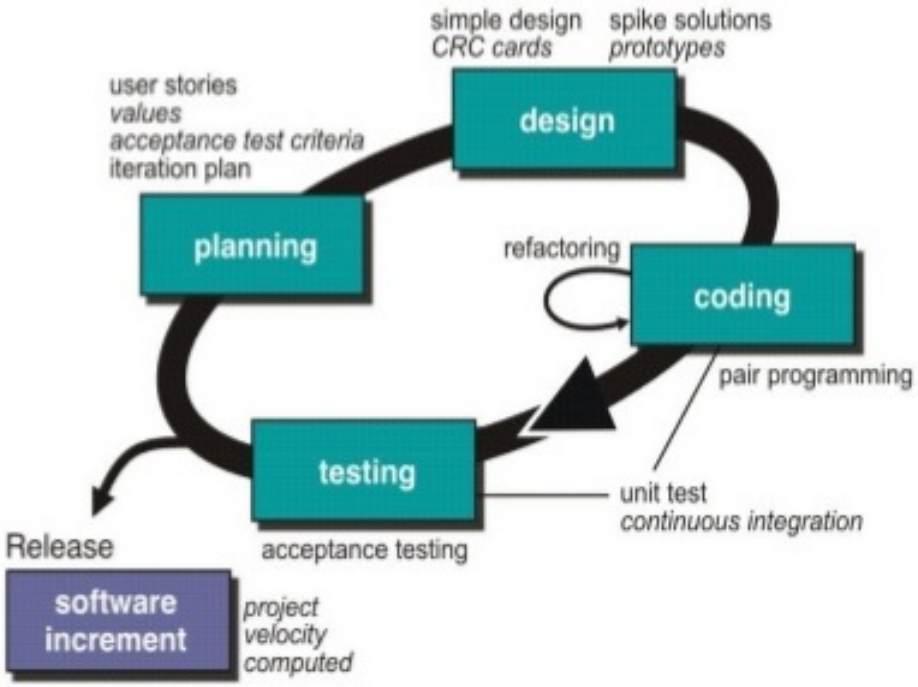
- **Ouvir:** Desenvolvedores devem ouvir constantemente os clientes para saber o que eles realmente precisam e sobre regras de negócio do sistema. Eles devem entender suas necessidades para que assim possam fornecer *feedbacks* constantes aos clientes sobre soluções de problemas ou impossibilidade de realizá-las.

- **Projetar:** Projetar o design de um sistema é importante para que a dependências entre os sistemas seja evitada, e assim, quando uma parte necessitar ser alterada, isto não impacte nas outras.

No *Extreme programming*, o desenvolvimento é realizado de forma incremental, para que assim pequenos lançamentos de versões sejam realizadas para que se consiga maior *feedback*. Os requisitos são baseados em cenários ou as chamadas estórias de usuário, que são usadas como base para decidir quais funcionalidades serão incluídas em um incremento do sistema (SOMMERVILLE, 2011). Para diminuir erros na especificação, é necessário que haja disponibilidade do cliente para sanar dúvidas, realizar alterações de escopo e priorizações.

Resumindo, o ciclo do *XP* revela-se de acordo com a Figura 2.4. O cliente realiza e seleciona as estórias de usuário que entrarão na iteração. As estórias são dividas em tarefas mais específicas com testes de aceitação e com isso o *release* é planejado. O *software* é desenvolvido segundo as boas práticas e atividades da metodologia e assim o mesmo é liberado para o cliente. As correções, melhorias e alterações que surgirem na avaliação do sistema, entram para próximas iterações.

Figura 2.4 – Ciclo de uma iteração no *XP*.



Fonte: (NARULA, 2017)

2.5.2.2 *Scrum*

A metodologia *Scrum* existe desde a década de 1990. Ele estrutura o gerenciamento para o desenvolvimento incremental do produto lidando com uma ou mais equipes multifuncionais e auto-organizadas (SABBAGH, 2013). É uma metodologia para gerenciamento, aprimoramento e manutenção do desenvolvimento e progressão de sistemas. O *Scrum* assume que o processo de desenvolvimento de sistemas é imprevisível e complexo (SCHWABER, 1997).

A estrutura é organizada de forma a enfatizar o *feedback* empírico, equipes auto gerenciáveis e a construção de incrementos do produto em iterações rápidas (SCHWABER, 2004). Seus fundamentos são divididos por papéis, eventos e artefatos. A seguir serão detalhados estes fundamentos.

2.5.2.2.1 Papéis do *Scrum*

Através de três papéis, o *Scrum* implementa o seu ciclo iterativo, sendo eles o Dono de Produto (*Product Owner ou PO*, time de desenvolvimento e Mestre do Scrum (*Scrum Master*

ou SM). Todas as responsabilidades do projeto são divididas entre estes papéis (SCRUM ORG, 2019). Os papéis são detalhados da seguinte maneira:

- **Dono de Produto:** É responsável por representar todos os interessados no produto. Cria os requisitos iniciais do projeto e sua continuação, expectativas de retorno e planos para lançamento do produto. A lista de requisitos é chamada de *Product Backlog* e é gerenciada pelo PO, que deve garantir que as funcionalidades mais importantes sejam construídas primeiro. Isto é conquistado com a priorização dos itens do *Product Backlog* e deve gerenciar o mesmo, criando novos itens para melhoria contínua do produto.

- **Time de desenvolvimento:** O time é responsável por todo o desenvolvimento do produto. Este time deve ser auto gerenciável, auto organizável e multidisciplinar. É encarregado de transformar os itens do *backlog* do produto em incrementos de funcionalidades durante a iteração e gerenciar a execução do processo.

- **Mestre do Scrum:** Responsável pelo processo do *Scrum*, deve ensinar a todos envolvidos no projeto a implementá-lo e garantir que todos sigam suas regras e práticas. Além disso, serve como facilitador da relação entre os papéis e remove impedimentos do time de desenvolvimento.

2.5.2.2.2 Artefatos do *Scrum*

Volaroint (2014) define três artefatos para o *Scrum*, sendo eles:

- **Backlog do Produto:** Conhecido como *Product Backlog*, representa os requisitos do produto. O PO é responsável por seu conteúdo, disponibilidade e priorização. O *backlog* do produto nunca é finalizado enquanto o produto existir e evolui constantemente com o mesmo e seu ambiente imerso. A medida que surge a necessidade de competitividade no mercado ou usabilidade, requisitos são adicionados ou alterados.

- **Backlog da Sprint:** Representa o conjunto de tarefas que o time de desenvolvimento definiu para transformar os itens do *backlog* do Produto selecionados para a *Sprint* em incrementos prontos do produto. Apenas o time de desenvolvimento pode alterar o *backlog* da *Sprint*, ele e deve ser visível a todos para expor o progresso atual das tarefas.

- **Incremento:** Representa o conjunto de itens do *backlog* do Produto que foram finalizados na *Sprint*.

2.5.2.2.3 Eventos do *Scrum*

Os eventos ou cerimônias do *Scrum* têm por finalidade minimizar reuniões não planejadas e fora do escopo do *Scrum*, assim evitando desperdícios. São eventos criados para possibilitar a inspeção contínua do projeto, promovendo a transparência para todos. Todos os eventos tem tempo limite estabelecido (*time-box*), e devem ser obrigatoriamente seguidos. Segundo (SCRUM ORG, 2019), os eventos definidos no *Scrum* são os seguintes:

- ***Sprint:*** São iterações de tempo fixo, que variam entre uma a quatro semanas e dentro dela todos os eventos do *Scrum* ocorrem. É neste evento que os membros do time de desenvolvimento transformam os itens de *backlog* do produto em incremento, gerando valor para o cliente. Uma *Sprint* é iniciada logo ao final da anterior e é o cerne do Scrum.

- **Planejamento da *Sprint*:** Reunião com *time-box* variando de 2 a 8 horas de acordo com o tempo da *Sprint*. Nela, o time Scrum monta o *backlog* da *Sprint*, considerando assim a capacidade do time de desenvolvimento e prioridade dos itens do *backlog* do produto. É neste evento que os itens do *backlog* do produto são refinados para que os mesmos tenham um nível de detalhe suficiente para serem desenvolvidos.

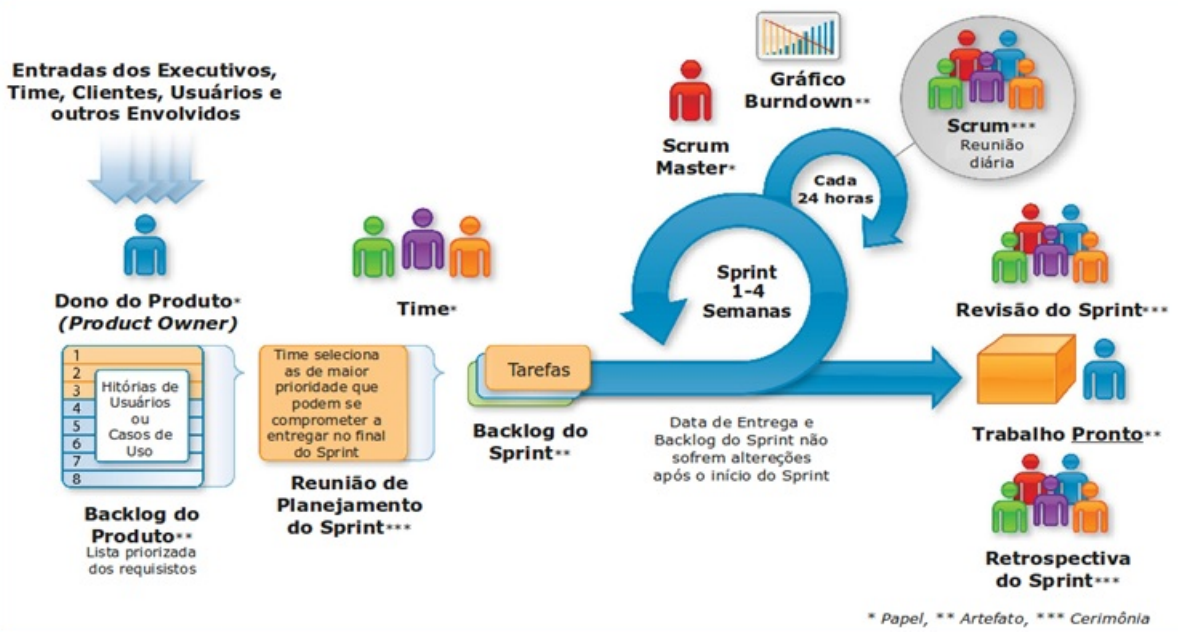
- **Reunião diária:** Reunião com time-box de 15 minutos, sendo realizada diariamente pelo time de desenvolvimento para avaliar a evolução da construção dos itens do *backlog* da *Sprint*, resolver pendências, relatar problemas e identificar futuras tarefas.

- **Revisão da *backlog*:** Reunião realizada após o término de uma *Sprint* e é aonde o time de desenvolvimento demonstra para PO e interessados no projeto o que foi finalizada na *Sprint*.

- **Retrospectiva da *Sprint*:** O time de desenvolvimento, com auxílio do SM, analisa os pontos positivos e negativos da última *Sprint*, avaliando melhorias para a próxima *Sprint*.

A Figura 2.5 exemplifica o fluxo completo do *Scrum*. Primeiramente o PO recebe as necessidades dos interessados no produto, organiza o *backlog* do Produto, ocorre a reunião de planejamento da *Sprint* e é formado o *backlog* da *Sprint*, que após o término da mesma é entregue o produto ou parte dele pronto. Ocorre a revisão da *Sprint* e sua retrospectiva e finalmente o ciclo é reiniciado.

Figura 2.5 – Ciclo do *Scrum*.



Fonte: (ALPHA, 2018)

3 METODOLOGIA

Este capítulo possui o objetivo de expor as etapas necessárias para a concepção e execução do projeto. Serão apresentados os pontos de melhoria no processo de cobrança no cartão de crédito além de levantamento de requisitos do novo sistema e metodologias utilizadas.

3.1 Materiais e Métodos

Para a concepção e desenvolvimento do novo sistema de cobrança no cartão de crédito, foram utilizadas as seguintes ferramentas.

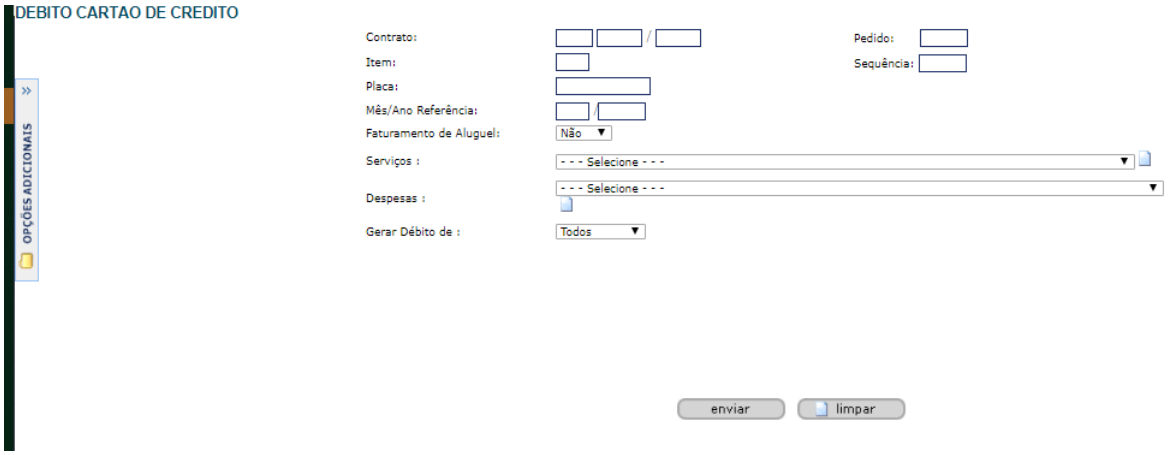
- Metodologia *Extreme Programming*.
- Metodologia *Scrum*.
- Linguagens C#, JavaScript, Razor e CSS.
- *Framework* para aplicações Web ASP.NET MVC.
- Gerenciador de banco de dados Sybase.
- IDE Microsoft Visual Studio.
- IDE para banco de dados SQL SqlDbx.
- Azure DevOps para recursos de gerenciamento de código fonte, relatórios, gerenciamento de requisitos, gerenciamento de projetos, *builds* automatizados, gerenciamento de laboratório, gerenciamento de testes e *release*.
- *SonarQube* para inspeção contínua da qualidade do código para executar revisões automáticas com análise estática do código para detectar *bugs*, odores de código e vulnerabilidades de segurança.
- Sistema Operacional Windows 10.

3.2 Contextualização do problema

O sistema antigo de cobrança no cartão de crédito dos usuário se encontra na intranet da gestão de frotas. Intranet é composta de *softwares* internos, formando assim uma Internet interna, da organização, utilizada para facilitar o funcionamento da mesma. Todos os subsistemas

da intranet da área possuem interface gráfica similar por terem sidos desenvolvidos utilizando ASP.NET *Web Forms* e estarem no mesmo repositório de código-fonte. A interface gráfica do sistema de Débito no Cartão de Crédito se encontra na Figura 3.1.

Figura 3.1 – Tela antiga de cobrança no cartão de crédito



Fonte: Do autor

O sistema fornece opções para cobrança de serviços, despesas e aluguel. O colaborador pode optar por realizar a cobrança de todos os extratos de uma placa ou por um contrato, pedido, item ou sequência, onde eles se definem da forma:

- **Contrato:** É a firmação do acordo jurídico entre a empresa de locação e o cliente. Nele são indicados diversos termos orçamentários, formas de faturamento dentre outras coisas. Na maioria dos casos, uma empresa possui somente um contrato aberto;
- **Pedido:** Após a abertura do contrato, é necessário realizar os pedidos dos carros. Então, para cada nova leva de carros que se deseja alugar, é necessário criar um pedido. Portanto, podem existir *N* pedidos por contrato;
- **Item:** Dentro de um pedido de veículos, para se separar por tipos de veículos ou se existirem muitos carros no pedido, são criados os itens. Por exemplo, pode-se realizar um pedido com carros modelo sedã e *hatch*, sendo assim, criam-se dois itens dentro do pedido. Assim, podendo existir *N* itens dentro de um pedido;
- **Sequência:** Finalmente, o cliente determina quantos carros irá necessitar em sua frota. Cada necessidade de carro é representado por uma sequência, ou seja, um item possui várias sequências. A sequência fornece o vínculo com a placa do automóvel.

Porém, a opção de cobrança por contrato, pedido, item ou sequência não funciona de forma apropriada e por isso os usuários do sistema somente realizam o débito por placa. Isso impede com que o colaborador solicite a cobrança de um contrato inteiro de uma vez. Isso gera a necessidade de enviar a solicitação de cobrança para cada placa de todos os clientes que optam por esta opção de pagamento. Porém, os campos de contrato, pedido, item e sequências ainda continuam obrigatórios pelo sistema e isso exige que o colaborador também os informe, gastando tempo em seu preenchimento. Atualmente, são cerca de 60 mil carros alugados e muitos clientes utilizam este tipo de serviço.

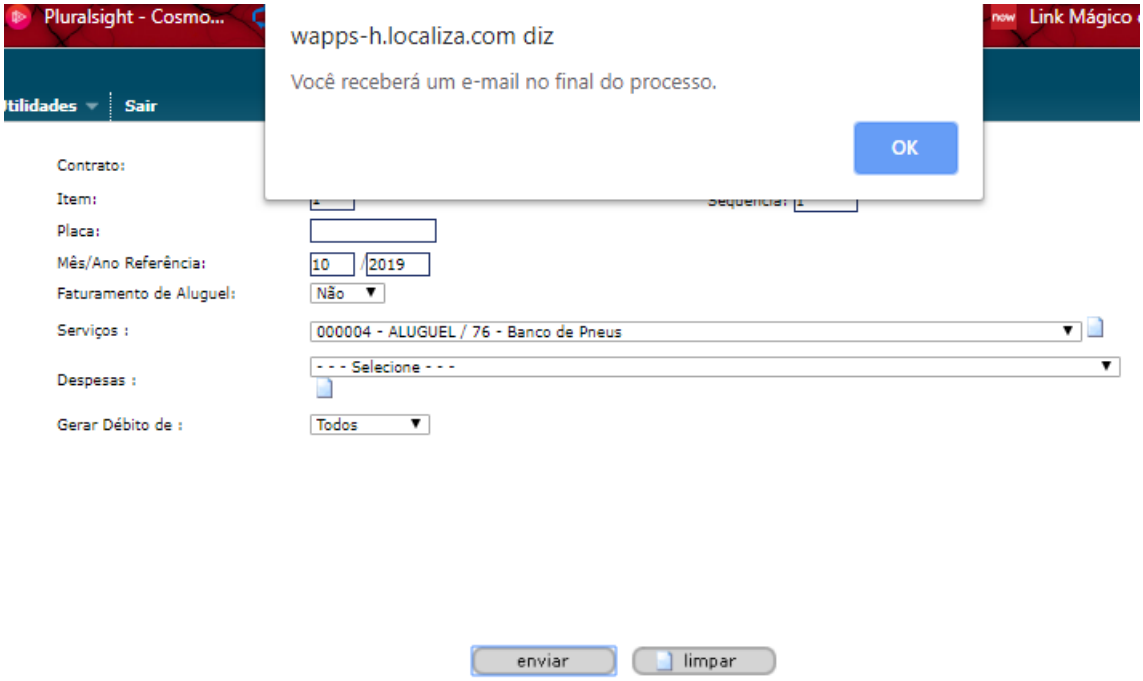
O sistema não mostra quais extratos serão cobrados, tampouco o valor que está sendo cobrado. Após o usuário solicitar a cobrança, o sistema dispara um *e-mail* informando o resultado. Caso ocorra um erro, não é indicado o motivo do erro e cabe ao usuário validar. Muitas vezes o sistema não realiza a tentativa de cobrança, visto que por alguma regra de negócio errada, o extrato não foi listado para tentativa de débito pelo sistema, e novamente o usuário deve tentar descobrir a razão de forma operária.

Portanto, o fluxo de trabalho do colaborador pode ser resumido por: verificar todos os contratos que contam com esta forma de cobrança em outro sistema; organizar a listagem e buscar todos os extratos que necessitam realizar esta cobrança em outro sistema; e para cada placa, ir até a tela de Débito no Cartão de Crédito, informar a placa, contrato, pedido, item, sequência, o tipo de cobrança (se é um serviço, despesa ou aluguel mensal), mês de referência, pressionar no botão *"enviar"* (Figura 3.2) e aguardar o *e-mail* de retorno.

Como o *e-mail* com resultado é enviado para cada cobrança, isso exige que o colaborador utilize de ferramentas externas ao sistema, como planilhas, para se organizar no que já foi cobrado e o que ainda está pendente, tipo de prática que pode levar a erros humanos. Além disso, o sistema sofre de lentidão para realizar os processos na tela. Se por erro o colaborador indicar que a cobrança seria do faturamento de aluguel e depois desejar corrigir para algum serviço ou despesa, a tela é recarregada, limpando todas as informações preenchidas, prejudicando a experiência do usuário. Além disso, caso o usuário digite contrato, pedido, item, sequência ou placa incorreta, ele não saberá até receber *e-mail* de retorno informando que nenhum débito foi realizado.

Sendo assim, após conversa com colaboradores que utilizam o sistema e analistas de sistemas, foram sintetizados os principais problemas do mesmo:

Figura 3.2 – Débito solicitado através da tela antiga de cobrança no cartão de crédito



Fonte: Do autor

- **Experiência ruim de usuário:** Lentidão ao alterar parâmetros da tela além da falta de validações e mensagens informativas para auxiliar o usuário;
- **Falta de transparência:** Usuário não sabe o que será cobrado até receber o *e-mail* de retorno;
- **Processo moroso:** Usuário necessita buscar em outros sistemas quais cobranças deve solicitar e executá-las uma por vez; impossibilidade de processar a cobrança para todo contrato em uma única solicitação; além disso, recebe um *e-mail* para cada cobrança realizada, tornando a conferência complexa;
- **Falta de tratativas para inconsistências:** Sistema não fornece nenhum tipo de validação sobre extratos pendentes, o que acarreta em grande dificuldade ao colaborador para corrigir o erro que impossibilitou a cobrança;
- **Sistema legado:** Devido a construção ter sido realizada utilizando .NET Framework 4, ASP.NET Web Forms e linguagem VB.NET, o sistema já é considerado legado. Além disso, o código-fonte se encontra em um monólito (intranet), sem testes unitários e automatizados, utilizando arquitetura DAL, BLL e UI, o que dificulta que melhorias e correções sejam realizadas.

3.3 Detalhamento do processo antigo

Para que fosse possível definir melhorias no processo e sistema envolvidos, primeiramente foi realizado o detalhamento das regras sistêmicas e de negócio envolvidas no processo antigo. Para isto, o código-fonte da aplicação foi avaliado, além de conversas com usuários do sistema, e testes em ambientes sistêmicos próprios para este fim.

O processo pode ser definido como um agregado de responsabilidades inerentes ao usuário e ao sistema. Este fluxo é realizado uma vez por mês, onde a área do faturamento gera todos os extratos e em seguida é realizado a cobrança dos mesmos. Sendo assim, após análises, foi averiguado que o processo é definido a partir das seguintes etapas:

1. Usuário verifica em outro sistema quais são os contratos que apresentam a opção de cobrança no cartão de crédito dos usuários para serviços e despesas.

2. Em outro sistema, o colaborador obtém a lista com todos os extratos pendentes de cobrança para cada contrato obtido na etapa anterior.

3. No sistema de Débito no Cartão de Crédito, para cada extrato pesquisado na etapa anterior, o colaborador informa os respectivos parâmetros para realizar a cobrança e pressiona o botão *"enviar"* para iniciar o débito. Após pressionado, o sistema valida se todos os parâmetros necessários estão preenchidos e envia a requisição. Assim, o usuário fica livre para enviar outra solicitação.

4. O sistema recebe a solicitação de débito com os parâmetros e realiza uma busca para listar os respectivos extratos referentes aos parâmetros. Caso algum erro ocorra durante a busca, o sistema envia *e-mail* ao colaborador informando que ocorreu um erro e finaliza o processo. Caso nenhum extrato tenha sido listado pelo sistema, também é enviado *e-mail* ao usuário informando que não foi possível encontrar extratos para os respectivos parâmetros e finaliza o processo. Muitas vezes, um extrato pendente de cobrança não é listado pelo sistema neste momento, isto é causado por regras de negócio não cumpridas, como CPF, informações de cartão de crédito e outros cadastros inválidos do cliente. O sistema não retorna quais são os problemas cadastrais, cabendo ao colaborador validar qual inconsistência impediu o extrato de ser processado e assim corrigi-la.

5. No sistema, para cada extrato listado, é validado se já ocorreu algum débito no cartão do usuário no dia com o mesmo valor. Este processo é importante pois a operadora de

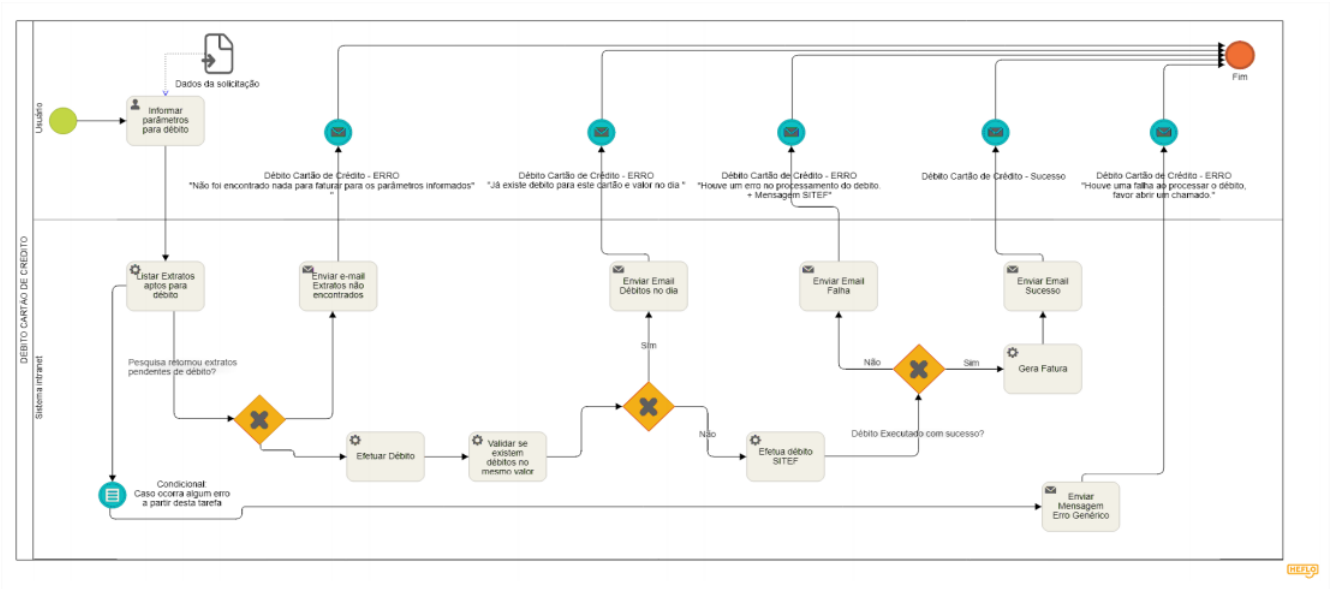
cartão impede a cobrança de mesmos valores no dia. Caso exista, um *e-mail* é enviado ao usuário informando a inconsistência.

6. É realizado a tentativa de cobrança no cartão do usuário através de um serviço externo. Caso a cobrança tenha sucesso, a fatura do cliente é gerada e um *e-mail* informando sucesso para o usuário é disparado. Se algum erro ocorrer nesta etapa, é enviado um *e-mail* informando que ocorreu um erro no serviço de cobrança no cartão e o processo é finalizado.

7. A partir dos *e-mails* enviados pelo sistema, o colaborador constrói uma planilha com a relação de todas as cobranças e seus respectivos *status*. Assim realiza as tentativas de cobrança para os pendentes durante os dias de trabalho, até obter sucesso para todos.

Pode-se observar que o processo exige uma dependência do colaborador envolvido. Ele deve levantar quais são os extratos pendentes de débito, validar inconsistências e erros, organizar quais extratos já foram cobrados e realizar correções e tentativas para realizar a cobrança dos restantes. A Figura 3.3 mostra o fluxo do processo de cobrança, separando em responsabilidades do usuário e do sistema.

Figura 3.3 – Diagrama do processo sistêmico de cobrança no cartão de crédito dos clientes na Gestão de Frotas



Fonte: Do autor

3.4 Avaliação de melhorias no processo

Conforme observado na seção 3.2 e 3.3, é possível notar que o processo antigo de cobrança no cartão de crédito possui diversas falhas que podem ser resolvidas e melhorias que podem ser realizadas, a fim de trazer benefícios a empresa.

A experiência do usuário ao utilizar o sistema é ruim, o que prejudica sua produtividade, onde as mensagens exibidas não são claras e o sistema prejudica a produtividade quando se deseja alterar parâmetros já inseridos na tela. Além disso, o fato do usuário ter que digitar todos os dados de forma manual pode induzir a erros humanos. A interface gráfica também está ultrapassada. Portanto, é evidente que deve-se construir uma nova interface do usuário para o sistema que solucione estes problemas.

Um ponto que torna o processo trabalhoso é o fato do colaborador ter que realizar a pesquisa de quais contratos contam com a opção de cobrança de serviços e despesas no cartão de crédito de seus usuários. Torna-se importante então fazer com que este ponto seja melhorado no processo, para que não seja mais necessário o usuário utilizar outro sistema para este tipo de processo.

Além de ter que pesquisar quais contratos possuem a opção de cobrança no cartão de crédito, o usuário deve, para cada um destes contratos, buscar os extratos pendentes de débito em outro sistema, assim organizando as informações em planilhas. O novo sistema então, deve ser capaz de listar todos os extratos pendentes de cobrança dentro de um contrato e exibi-los ao usuário.

Após a pesquisa dos extratos pendentes, no sistema de Débito no Cartão de Crédito, o usuário insere os parâmetros de cada extrato e solicita a cobrança. Este é o ponto que mais torna o processo moroso, sendo a automação da tarefa um requisito para melhorar o processo, que é demasiadamente repetitivo. Deve-se então, encontrar uma forma para que seja necessário uma menor quantidade de iterações para cobrar todos os extratos.

O colaborador têm conhecimento do que será debitado e seu valor antes de enviar a requisição de cobrança, só tomando conhecimento ao receber *e-mail* de retorno do sistema. Isso torna o comportamento do sistema opaco, o que gera falta de confiança, se tratando de um processo tão crítico para a empresa. Portando, o usuário deve ter conhecimento do que será cobrado antes de enviar a solicitação.

Para automatizar e otimizar o trabalho do usuário, uma melhoria a ser realizada seria ao invés do sistema enviar em *e-mail* para cada tentativa de cobrança informando o resultado, mon-

tar uma planilha com a relação da solicitação inteira e encaminhá-la para o usuário. Isso economizaria o esforço do usuário de organizar cada *e-mail* e montar planilhas para auto-organização.

Um desperdício do processo ocorre na situação onde o usuário solicita alguma cobrança, mas por alguma razão cadastral o mesmo não for processado. Isso faz com que o colaborador tenha que revisitar o cadastro relacionado ao extrato em busca de inconsistências. Sendo assim, uma melhoria seria exibir ao usuário os extratos com problemas cadastrais antes de demandar o débito.

Como o processo atual é totalmente dependente dos colaboradores para ser executado, um ponto seria automatizar todo o processamento, tornando independente de processos humanos. A intervenção só seria necessária para correção de erros pontuais.

3.5 *Backlog* do Produto

A metodologia determinada para o desenvolvimento do novo sistema de cobrança no cartão de crédito foi o *Scrum*, portanto a primeira etapa a se realizar seria a determinação do *Backlog* do Produto. Esta etapa foi realizada em colaboração com a analista de processos e negócio da área do faturamento, que é a pessoa com maior experiência no sistema e realiza interface com os usuários do sistema. Esta pessoa atuou como PO do projeto. Com base nas análises realizadas na Seção 3.4, a Tabela 3.1 foi construída, de forma a resolver problemas e melhorar o processo de forma geral. A tabela contém o número de identificação do item do *Backlog*, sua descrição e prioridade definida, realizada em uma escala de 1 a 10, com o 1 representando prioridade máxima.

O item 1 foi pensado para otimizar o tempo gasto pelo colaborador para processar todas as cobranças, pois atualmente é realizado uma cobrança por solicitação. Ao permitir a cobrança de todos os extratos pendentes dentro de um contrato, economizaria um grande esforço, melhorando o processo.

Porém, antes de solicitar o débito, deve-se ter o conhecimento do que exatamente está sendo cobrado, portando o item 2 foi incluído para tratar disso, permitindo que o colaborador confira todos os extratos que serão processados previamente.

Devido a restrições do modelo da companhia, as regras de negócio do sistema não podem ser modificadas, sendo assim o item 3 foi colocado no *backlog*, para que as especificações fossem replicadas do sistema antigo para o novo.

Tabela 3.1 – *Backlog* do produto

Item	Descrição	Prioridade
1	Permitir que o usuário realize o débito de todos os extratos pendentes dentro de um contrato de uma só vez	1
2	Exibir os extratos pendentes de cobrança de acordo com os parâmetros antes que o débito seja solicitado	2
4	O novo sistema deve ter opções de filtros compatíveis com o sistema antigo, permitindo a busca por tipos de serviços, despesas ou aluguel	3
5	Após o processamento de uma solicitação de cobrança, o sistema deve enviar um relatório ao usuário, em formato de planilha, com o resultado das cobranças realizadas	4
6	Ao exibir os extratos pendentes antes da cobrança, validar os extratos que possuem algum erro cadastral associado que impedirá a cobrança	7
7	Permitir que o colaborador selecione quais extratos deseja cobrar de forma individual	8
8	Restringir as buscas por contratos que possuírem opção de cobrança no cartão de crédito	9
9	Tornar o processo automático	10

Fonte: Do autor

O item 5 descreve que após as cobranças das solicitações, o sistema deve enviar um *e-mail* ao usuário com a relação de todas processadas. Isto economizaria tempo na revisão das cobranças, visto que atualmente é enviado um *e-mail* por cobrança e o usuário deve organizá-los para conferência.

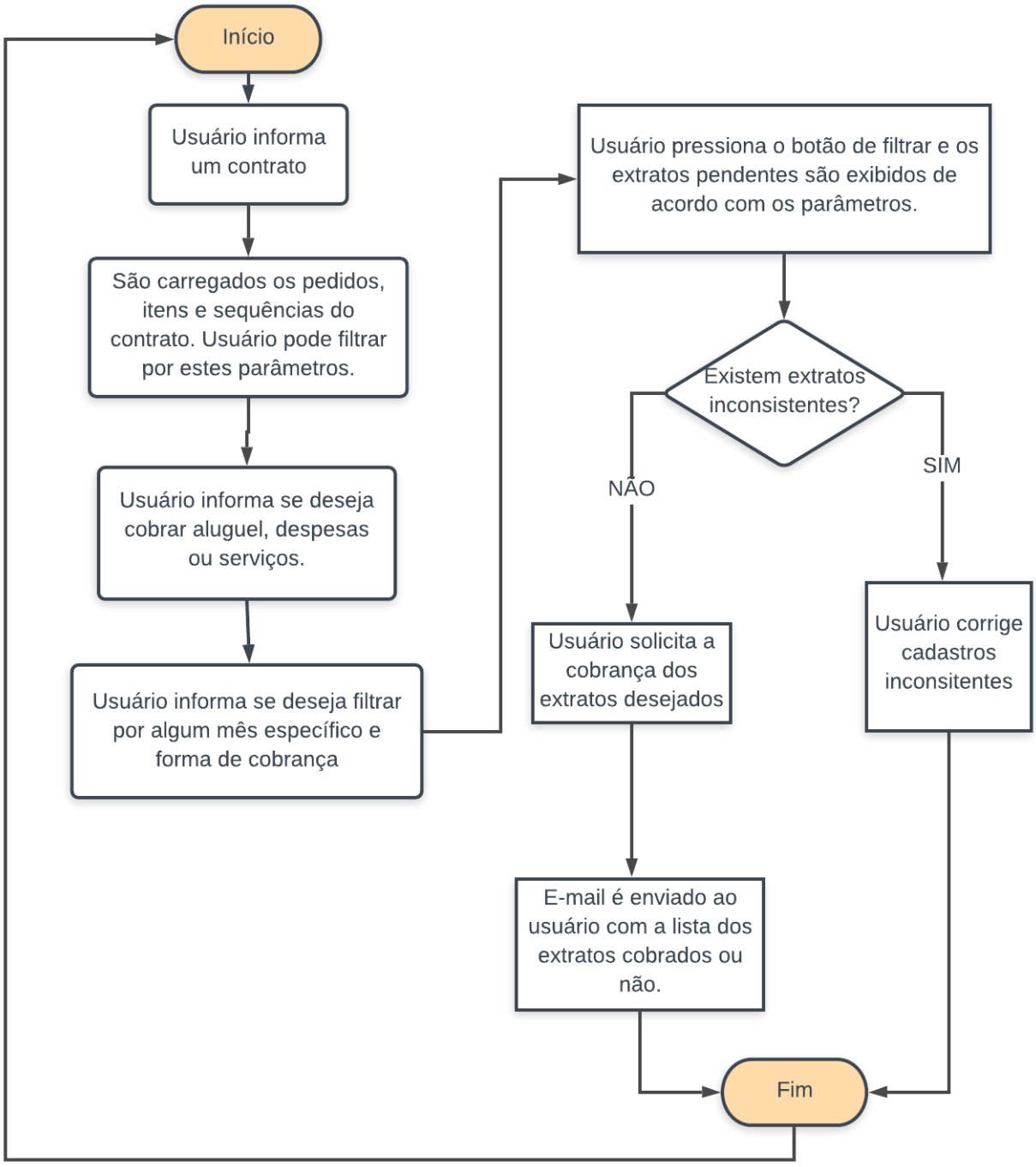
3.6 Planejamento da *Sprint*

Seguindo o *backlog* do Produto, foram selecionados os itens de maior prioridade para entrar no *backlog* da *Sprint*. Desta forma, os itens de 1 a 7 foram selecionados para detalhamento técnico para que possam ser implementados na *sprint*. O item 8 não entrou para o *backlog* da *Sprint* devido a prioridade. O item 9 não foi selecionado para a primeira *Sprint* pois foi pensado que o novo sistema deveria estar estável antes de automatizá-lo, porém, todos os outros requisitos foram pensados da forma que fosse possível reaproveitá-los ao iniciar a etapa de automatização do processo.

Analisando as especificações, foi determinado que uma tela fosse construída, com um fluxo de interação. A tela seria responsável por permitir a pesquisa dos extratos pendentes a partir de filtros de contrato, pedido, item, sequência, tipo de cobrança, mês de referência, tipos de serviço, despesa ou cobrança de aluguel. Estes extratos seriam exibidos na tela e também seriam mostrados quais estão com problemas cadastrais que impediriam a cobrança.

Para organização da tela, foi planejado que a mesma fosse dividida em três blocos. O primeiro bloco seria o dos filtros para pesquisa dos extratos, contendo assim todos os campos para o mesmo, inclusive o botão para pesquisar. O bloco que exibiria os extratos pendentes seria o segundo bloco e o terceiro seria o que mostraria os extratos com problemas cadastrais. Assim, o fluxo do novo processo é representado a partir da Figura 3.4.

Figura 3.4 – Fluxograma do novo processo de cobrança de crédito



Fonte: Do autor

3.7 Implementação do novo sistema

Esta seção irá tratar dos detalhes da implementação da solução proposta no planejamento da *Sprint* que seria uma tela central ao processo e seus sub-blocos.

3.7.1 Bloco de filtro de extratos

Este bloco foi construído para centralizar todos os tipos de filtros de pesquisa para realizar a cobrança de extratos pendentes. O bloco está representado na Figura 3.5 e possui os seguintes componentes:

- **Campo Contrato:** Este campo se refere ao contrato no qual se deseja realizar a cobrança. O usuário pode buscar o contrato por seu código, cliente ou CNPJ. Este campo se completa automaticamente de acordo com resultados de busca na base de dados Sybase (Figura 3.6). Isso impede erros humanos de digitação. O campo é obrigatório e sem o preenchimento do mesmo não é possível realizar nenhuma outra ação na tela.
- **Campo Cliente:** Indica o cliente do contrato informado no campo *Contrato* e seu CNPJ.
- **Campo Pedido:** Lista os pedidos do contrato selecionado. É um campo de seleção múltipla, permitindo que o colaborador pesquise extratos de apenas pedidos desejados dentro de um contrato.
- **Campo Item:** Campo de seleção múltipla, permite a pesquisa de apenas extratos de itens de pedidos pretendidos dentro de um contrato.
- **Campo Sequência:** Ideia semelhante aos campos *Pedido* e *Item*, permitindo o filtro por sequências específicas de um contrato.
- **Indicador de Faturamento de Aluguel:** Indica se será pesquisado os extratos referentes a aluguel do veículo.
- **Campo Serviço:** Permite que os extratos sejam filtrados pelo tipo de serviço do mesmo. O campo é selecionável.
- **Campos Despesa:** Ideia semelhante ao campo *Serviço*, porém para despesas.
- **Campo Mês/Ano Referência:** Permite que o usuário pesquise por extratos pendentes de algum mês específico. Caso não informado, a data será desconsiderada na pesquisa.

- **Indicador Gerar Débito De:** Na gestão de frotas, é possível que a cobrança seja realizada por meio do cartão do usuário do veículo ou por algum cartão corporativo. O campo têm a função de indicar de qual cartão cobrar.

- **Botão Filtrar:** Botão que realiza a pesquisa de acordo com os parâmetros informados pelo usuário. A pesquisa só ocorre se ao menos o contrato for informado corretamente e se algum tipo de serviço ou despesa for selecionado. Os outros campos são opcionais.

- **Botão Limpar:** Reinicia o estado da tela.

Os componentes da implementação deste bloco surgiram de forma a replicar o filtro do sistema legado, porém com melhorias na experiência do usuário. A navegação pelos componentes é rápida, a possibilidade do usuário informar algum dado errado foi removida, visto todos os campos têm validação da informação e o usuário não necessita digitar as informações, apensas selecioná-las, removendo a possibilidade de erros humanos. Outra melhoria é a possibilidade de informar mais de um pedido, item ou sequência para a cobrança, além da tela não recarregar caso alguma informação seja informada indevidamente e o usuário deseje corrigi-la.

3.7.2 Bloco de listagem de extratos pendentes

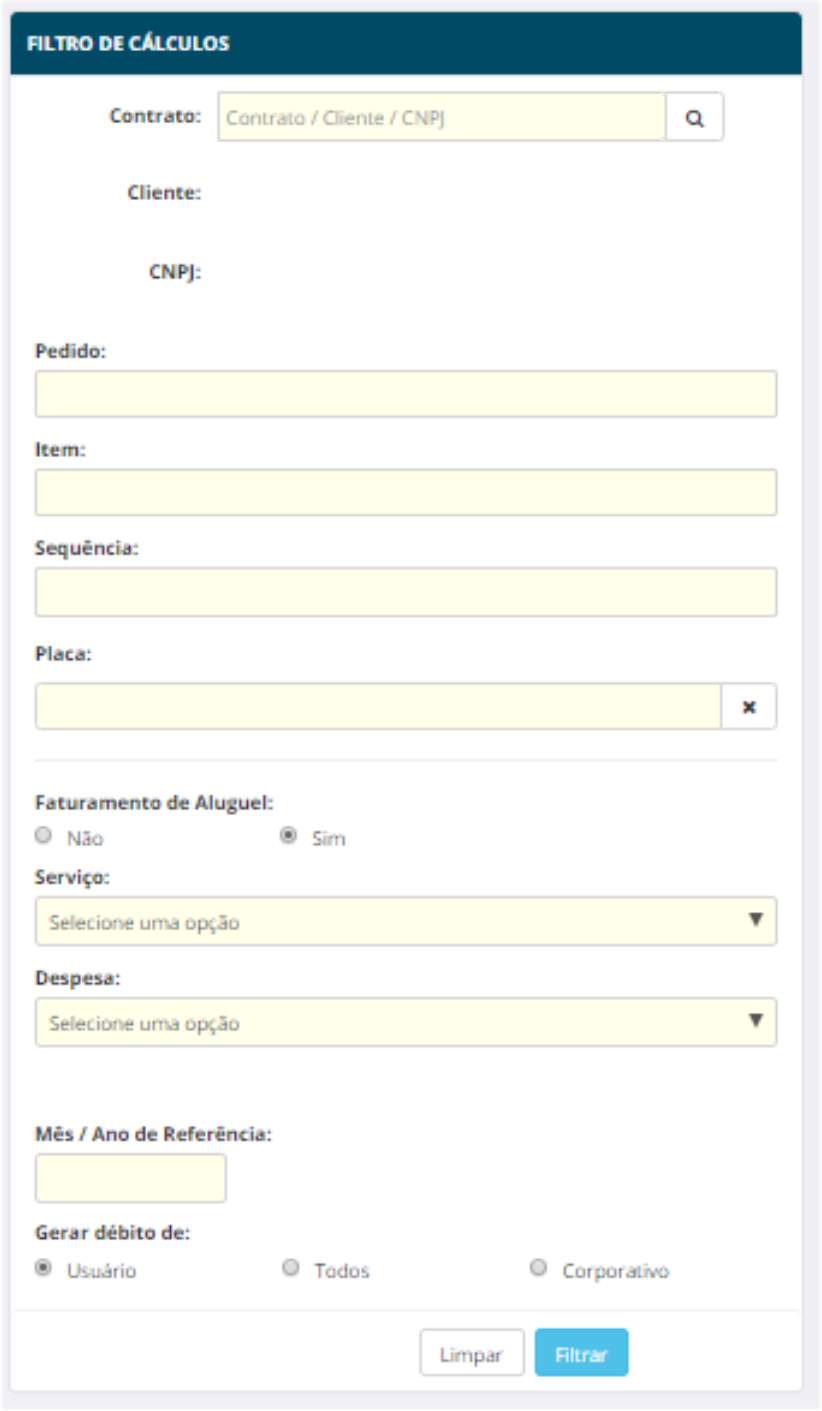
Com a necessidade do colaborador visualizar as informações do extrato antes de solicitar a cobrança, foi construído o bloco de listagem de extratos pendentes (Figura 3.7). Além de informar quais são os extratos pendentes de cobrança, é por este bloco que o usuário informa quais extratos deseja cobrar e efetivamente solicita o débito.

Os extratos são carregados de acordo com as informações previamente informadas, portanto é realizado uma Query, de acordo com os parâmetros informados pelo usuário, na base de dados *Sybase*, procurando por extratos pendentes de fatura, ou seja, que ainda não foram cobrados. Caso nenhum extrato seja encontrado para as respectivas condições, é exibida uma mensagem informativa na tela.

Juntamente com os colaboradores que utilizam o sistema, foi definido que para cada extrato, seriam exibidos as informações da placa, número do pedido, número do item, número da sequência, número do extrato, descrição do serviço ou despesa, valor e nome do usuário do veículo respectivo.

Além de listar extratos pendentes, os extratos que possuírem algum problema cadastral associado são marcados em vermelho e desabilitados para cobrança, conforme demonstrado

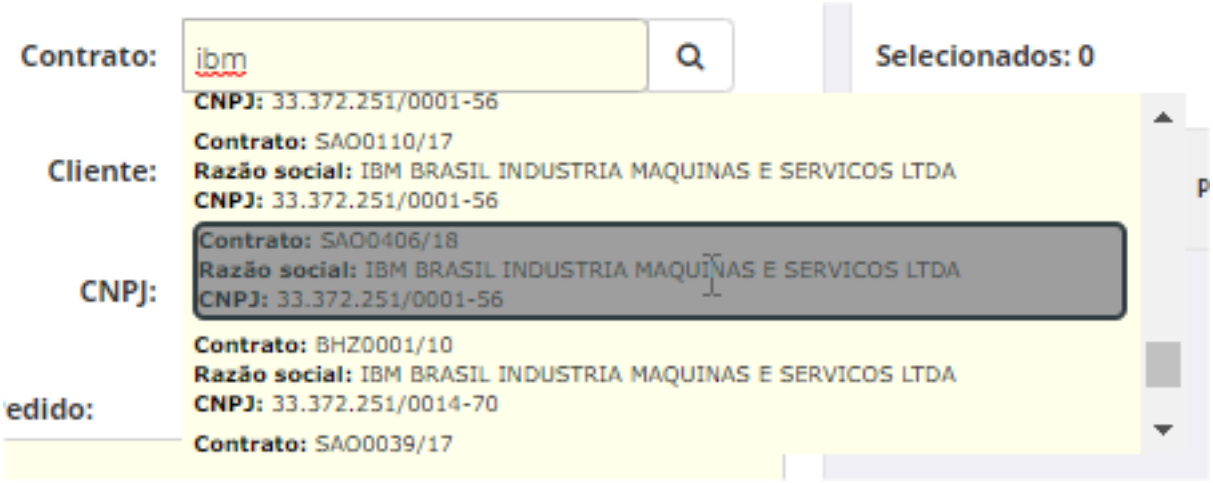
Figura 3.5 – Bloco de filtro de extratos pendentes do novo sistema de cobrança no cartão de crédito



Fonte: Do autor

na Figura 3.7. Esta validação é realizada no momento da busca. Primeiramente no *back-end* são listados todos os extratos pendentes de cobrança de acordo com os parâmetros, depois são listados os extratos pendentes com todas as validações que impediriam a cobrança, da forma que o que foi obtido na primeira lista mas não na segunda, está com algum problema cadastral. Este passo auxilia o colaborador a corrigir cadastros incorretos, e após correção, a cobrança pode ser realizada.

Figura 3.6 – Campo Contrato do bloco de filtro



Fonte: Do autor

Figura 3.7 – Bloco de listagem dos extratos pendentes do novo sistema

LISTA DE CÁLCULOS A DEBITAR

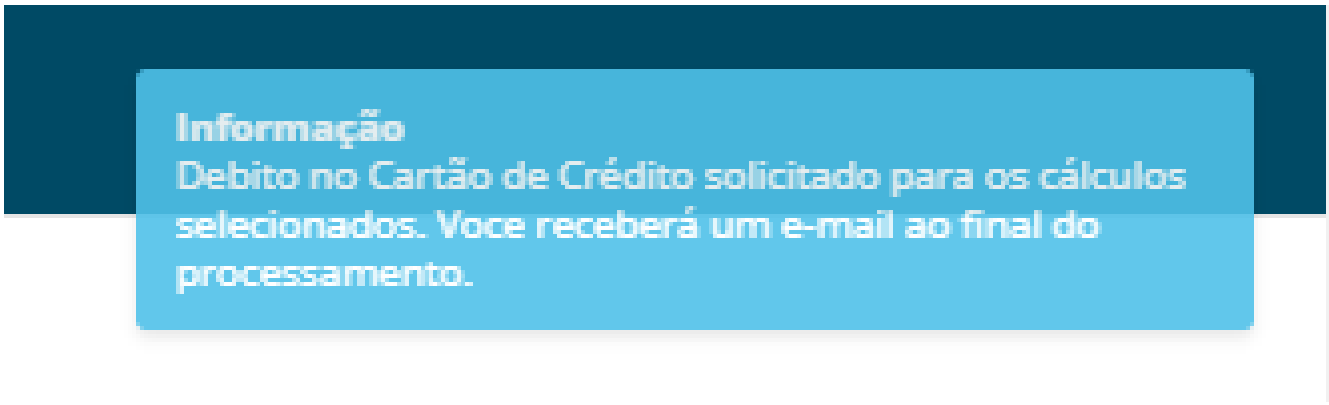
Selecionados: 0 **Total: 16** Debitar

PLACA	PED-ITEM-SEQ	Nº EXTRATO	SERVIÇO / DESPESA	VALOR	USUÁRIO
PZS2758	1-1-13	72	Multas de trânsito	R$ 114,54	Mauro Cesar Galindo Madeira
PZT7591	1-1-150	183	Multas de trânsito	R$ 114,54	Pedro De Alcantara Da Silva Pedrosa
PZS2336	1-1-156	137	Multas de trânsito	R$ 114,54	Marcus Tulio Buzollo De Aquino
PZS2470	1-1-193	67	Multas de trânsito	R$ 114,54	Alessandro Amaro Quesada
PZS2752	1-1-211	91	Multas de trânsito	R$ 171,81	Claudia Cortes Romanelli
PZU8753	1-1-291	111	Multas de trânsito	R$ 114,54	Guilherme Messora Homen Del Rei Silva
PZU8753	1-1-291	126	Multas de trânsito	R$ 114,54	Guilherme Messora Homen Del Rei Silva
PZU8753	1-1-291	118	Multas de trânsito	R$ 114,54	Guilherme Messora Homen Del Rei Silva
PZU8753	1-1-291	127	Multas de trânsito	R$ 114,54	Guilherme Messora Homen Del Rei Silva
PZS2484	1-2-8	153	Multas de trânsito	R$ 114,54	Maria Fernanda Meyer Drescher
PZS2415	1-3-17	145	Item Reembolsavel	R$ 50,00	Alessandra Libonatti

Fonte: Do autor

O bloco também possui o botão para solicitação do débito, que após pressionado envia uma solicitação a aplicação para a cobrança dos extratos selecionados. Após confirmação de recebimento, o sistema exibe mensagem informativa na tela confirmando que a solicitação foi recebida com sucesso (Figura 3.8) e o usuário pode recomeçar o fluxo para outros parâmetros. Enquanto isso, no servidor, é iniciado o processamento das cobranças no cartão de crédito do usuário. Após o processamento, é enviado um *e-mail* ao usuário com a relação de todas as solicitações e seus respectivos resultados.

Figura 3.8 – Mensagem informativa ao solicitar débitos no novo sistema



Fonte: Do autor

3.7.3 Bloco de inconsistências dos extratos

Este bloco é responsável por exibir a descrição das inconsistências que impediriam a cobrança e têm como objetivo auxiliar o colaborador na correção do cenário. O bloco exibe os resultados em forma de tabela, relacionando a placa do extrato com sua respectiva inconsistência. Um possível cenário é demonstrado na Figura 3.9. Atualmente são validados dados cadastrais, como CPF, cartão de crédito e se a data da cobrança condiz com a data que o usuário foi cadastrado. Além disso, é validado se já ocorreu uma cobrança no cartão de crédito do usuário com o mesmo valor do extrato naquele dia da gestão de frotas, o que ocasionaria erro na operadora de cartão. Caso um extrato possua mais de uma inconsistência, é gerada uma nova linha no bloco relacionando a placa com o respectivo erro.

3.7.4 Criação de serviço na Intranet

Para a realização da cobrança no cartão de crédito do usuário são necessárias diversas etapas sistêmicas após a solicitação para completar o fluxo. Várias regras de negócio são efetua-

Figura 3.9 – Bloco de inconsistências do novo sistema de débito no cartão de crédito

Fonte: Do autor

das para validação, além disso, existe interface com o serviço da operadora de cartão de crédito. Após a cobrança, é gerada a fatura do usuário para que o extrato não seja mais considerado pendente. Todo este fluxo já se encontra na intranet e portanto, foi optado para esta *Sprint* que a estrutura fosse reaproveitada.

Foi definido que todo este fluxo seria exposto por meio de uma API na intranet, onde o novo sistema realizaria uma requisição para solicitar a cobrança, passando apenas os parâmetros de cada extrato. Portanto, para n cobranças realizadas, será realizada n chamadas a API do sistema da intranet.

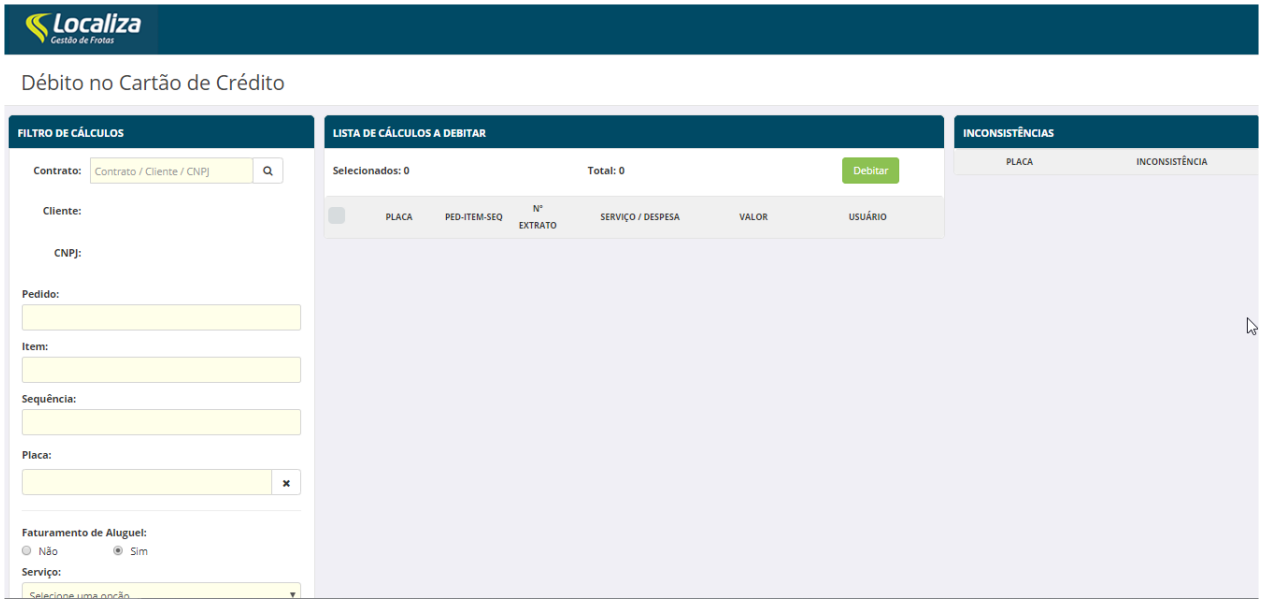
3.7.5 Tela completa

Após a construção de cada bloco individualmente, foi possível montar a tela completa com a união de todos. A Figura 3.10 e 3.11 mostram respectivamente a tela sem informações preenchidas e após uma pesquisa.

3.7.6 Implementação futura para completa automatização do processo

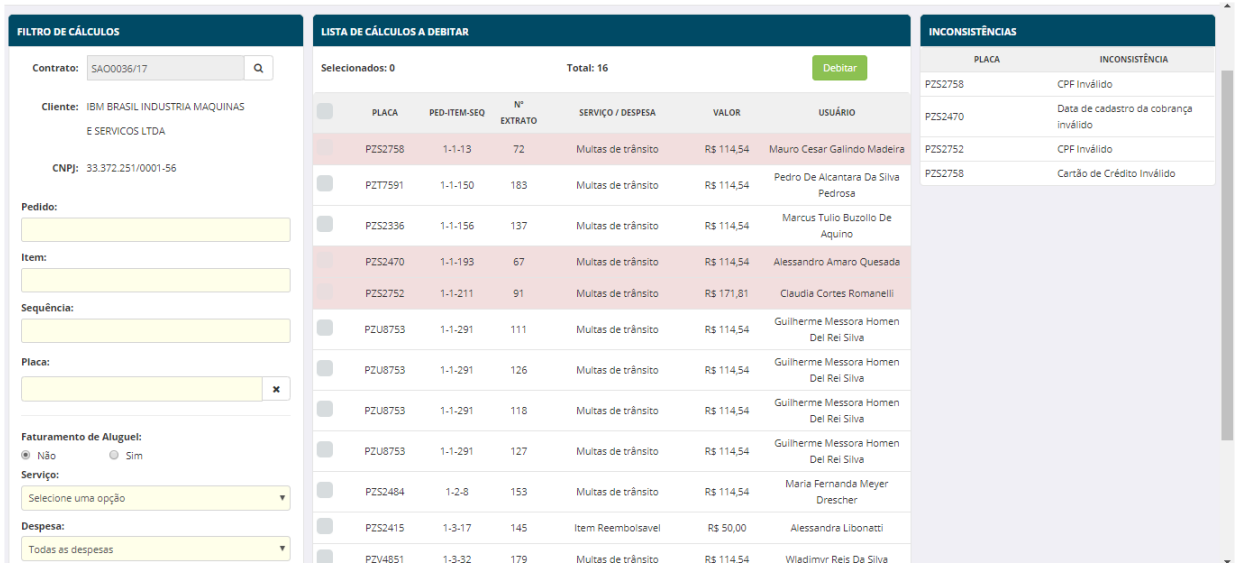
Como o novo sistema foi construído utilizando ASP.NET MVC, todos os métodos no novo sistema funcionam como uma API. Portanto, para automatizar o processo completamente, bastaria desenvolver uma rotina sistêmica que buscaria todos os contratos com a opção de cobrança no cartão de crédito e para cada contrato realizar a solicitação de cobrança através da API da tela do novo sistema, passando os parâmetros adequados. Esta rotina poderia ser configurada para ser executada uma vez no mês e a tela seria utilizada apenas para contingências.

Figura 3.10 – Tela nova de cobrança no cartão de crédito



Fonte: Do autor

Figura 3.11 – Tela nova de cobrança no cartão de crédito com pesquisa



Fonte: Do autor

4 RESULTADOS E DISCUSSÃO

Neste capítulo, será demonstrado e discutido os resultados da entrega do novo sistema de cobrança no cartão de crédito desenvolvido neste trabalho. Será mostrada a análise de produtividade comparativa com o sistema antigo e ganhos econômicos com o mesmo. O sistema foi liberado para utilização do usuário em Outubro de 2019 e o comparativo será realizado baseado neste mês.

4.1 Análise de produtividade

Para avaliar o valor que o novo *software* agregou ao negócio, foi levado em consideração o tempo que o colaborador gastava para realizar todas as cobranças antes e depois da implantação. Atualmente, na gestão de frotas, apenas um colaborador é responsável por todo o processo e o tempo gasto por mês foi informado pelo mesmo. A Tabela 4.1 foi montada a partir do mês de Agosto até Novembro de 2019, que foi o período onde foi possível acompanhar o colaborador em suas tarefas.

Tabela 4.1 – Tempo gasto pelo colaborador para realizar todo o processo de cobrança no cartão de crédito por mês

	agosto	setembro	outubro	novembro
Tempo gasto pelo colaborador	67h	76h	25h	21h

Fonte: Do autor

Conforme análise da Tabela 4.1, a partir do mês de Outubro, que foi onde o novo sistema começou a ser utilizado, é notável a diferença na produtividade do colaborador. Utilizando o sistema antigo de cobrança, era gasto um tempo médio de 71,5 horas e após implantação do novo sistema, o tempo médio caiu para 23 horas. Assim, pode-se calcular uma redução de 67% do tempo gasto para realizar todo o processo de cobrança no cartão de crédito da gestão de frotas, evidenciando assim um grande ganho de produtividade no processo.

4.2 Análise econômica

Para realizar análise econômica foi necessário simular o preço do software com o retorno deste para área. A simulação foi necessária devido ao sistema ter sido construído sem custos por fim do trabalho. Assim foi considerado que a hora de cada desenvolvedor custa em média

$R\$125,00$, e para o projeto, foi gasto cerca de 80 horas. Logo, a primeira versão do software teria o valor de $R\$10.000,00$.

Foi considerado que o colaborador que realiza o processo de cobrança trabalha 8 horas e 48 minutos por dia e 20 dias no mês, com um salário bruto de $R\$3.000,00$. Sendo assim, antes da implantação do novo sistema, ele gastava cerca de 42% do seu mês de trabalho com o processo de cobrança e após implantação do novo sistema, o valor diminuiu para aproximadamente 14%. Sendo assim, é possível calcular através da Equação 4.1 em quantos meses o *software* teria seu valor recuperado.

$$E(N,S) = \frac{N}{(0,42-0,14)\times S} \tag{4.1}$$

Sendo N o valor gasto para o desenvolvimento do *software* ($R\$10.000,00$) e S o salário bruto do colaborador. Calculou-se então aproximadamente 12 meses para que o investimento fosse totalmente recuperado com a economia na jornada de trabalho do colaborador.

5 CONSIDERAÇÕES FINAIS

Este trabalho propôs o desenvolvimento de um *software* para substituição de um sistema legado responsável pela cobrança no cartão de crédito de clientes da gestão de frotas em uma empresa de aluguel de carros. Foi realizado um estudo sobre os problemas e restrições do processo do utilizando o sistema antigo, assim como formas de automatizar o processo. A metodologia ágil aplicada mostrou-se bastante eficaz em dividir o software em vários pacotes entregáveis e priorizar os que mais agregariam valor.

A análise realizada no capítulo de discussões demonstrou que este *software* aumentou significativamente a produtividade do colaborador que trabalha no setor. Sendo assim, o objetivo de demonstrar a influência de migração de sistemas legados na produtividade do processo foi alcançado. Além disso, foi realizada a análise econômica do projeto e constatado que a diminuição do tempo de processamento de todas as cobranças faria com que o *software* fosse pago nos primeiros 12 meses caso o projeto tivesse sido construído de forma a cobrar o valor da hora de um desenvolvedor de sistemas.

Para trabalhos futuros será considerada, a retroalimentação do *Backlog* do produto com novos requisitos como a criação de uma rotina para automatizar o processo completamente, que seria implementado nas próximas *Sprints*. Além disso, migrar o serviço que está na intranet para efetuar a cobrança para o novo sistema, diminuindo o tempo de processamento e finalizando a migração completamente.

REFERÊNCIAS

ALPHA. **Seu desafio é economizar tempo e dinheiro? Conheça o Scrum**. 2018. Disponível em: <http://alphacontabil.com.br/seu-desafio-e-economizar-tempo-e-dinheiro-conheca-o-scrum/>. Acesso em: 02 nov. 2019.

ANDERSON, R.; WASSON, M. **Tutorial: criar uma API Web com ASP.NET Core**. [S.l.], 2019. Disponível em: <https://docs.microsoft.com/pt-br/aspnet/core/tutorials/first-web-api?view=aspnetcore-3.0&tabs=visual-studio>.

BECK, K. **Manifesto for Agile Software Development**. 2001. Disponível em: <http://agilemanifesto.org/>. Acesso em: 01 out. 2019.

BECK, K. **Extreme programming explained: embrace change**. 12. ed. [S.l.]: Addison-Wesley, 2004.

BROOKS, F. P. No silver bullet: Essence and accidents of software engineering. **IFIP Tenth World Computing Conference**, p. 1–19, abr. 1987. Disponível em: <http://worrydream.com/refs/Brooks-NoSilverBullet.pdf>.

CASTELLS, M. **A sociedade em rede**. 29. ed. [S.l.]: Paz Terra, 2009.

ESPOSITO, D. **Programming Microsoft ASP.NET MVC**. 2. ed. [S.l.]: Microsoft, 2010.

FOLDOC. **Legacy System**. 1998. Disponível em: <http://foldoc.org/legacy>. Acesso em: 28 set. 2019.

GILMORE, W. J. **Easy PHP Websites**. 1. ed. [S.l.]: W.J. Gilmore, 2009.

GRUBB P.; TAKANG, A. A. **Software Maintenance: Concepts and Practice**. 2. ed. [S.l.]: World Scientific, 2003.

HAACK, P.; GALLOWAY, J. **Professional ASP.NET MVC 3**. 1. ed. [S.l.]: John Wiley Sons, Inc., 2011.

INTERNATIONAL, S. G. Chaos report. jan. 1995. Disponível em: <https://www.csus.edu/indiv/r/rengstorffj/obe152-spring02/articles/standishchaos.pdf>.

KRAFZIG, D.; BANKE, K. **Enterprise SOA: Service-Oriented Architecture Best Practices**. 1. ed. [S.l.]: Prentice Hall, 2004.

KRASNER, G.; POPE, S. A cookbook for using the model-view controller user interface paradigm in smalltalk-80. jan. 1988. Disponível em: <JournalofObject-OrientedProgramming>.

LESSA, R. O.; JUNIOR, E. O. lessa. Modelos de processos de engenharia de software. v. 1, n. 1, 2010. Disponível em: <https://ead.uepg.br/apl/sigma/assets/editais/PS0059E0080.pdf>.

MACORATTI. **ASP .NET - Criando um projeto em 3 Camadas - UI, BLL e DAL**. 2010. Disponível em: <http://www.macoratti.net/14/05/aspn_3cam.htm>. Acesso em: 02 nov. 2019.

MATTOS, C. **Desafios e Soluções para Migração de Sistemas Legado**. 2015. Disponível em: <https://blog.gft.com/br/2015/06/09/desafios-e-solucoes-para-migracao-de-sistemas-legado/>. Acesso em: 25 abr. 2019.

MEDIUM. **Android MVC x MVP x MVVM qual Pattern utilizar — Parte 1**. 2017. Disponível em: <https://medium.com/@FilipeFNunes/android-mvc-x-mvp-x-mvvm-qual-pattern-utilizar-parte-1-3defc5c89afd>. Acesso em: 02 nov. 2019.

MICROSOFT. **Creating a Data Access Layer**. 2010. Disponível em: <https://docs.microsoft.com/en-us/aspnet/web-forms/overview/data-access/introduction/creating-a-data-access-layer-vb>. Acesso em: 06 nov. 2019.

MICROSOFT. **O que é o Web Forms**. 2014. Disponível em: <https://docs.microsoft.com/pt-br/aspnet/web-forms/what-is-web-forms>. Acesso em: 28 set. 2019.

MICROSOFT. **Overview of the .NET Framework**. 2017. Disponível em: <https://docs.microsoft.com/en-us/dotnet/framework/get-started/overview>. Acesso em: 28 set. 2019.

NARULA, A. **Agile Methodology : Extreme Programming(XP)**. 2017. Disponível em: <https://atesterthing.wordpress.com/2017/08/17/agile-methodology-extreme-programmingxp/>. Acesso em: 01 out. 2019.

PLUGA.CO. **Revolucione sua operação com um software de automação**. 2016. Disponível em: <https://pluga.co/blog/gestao-empresarial/software-de-automacao/>. Acesso em: 02 nov. 2019.

PROTALINSKI, E. **Visual Studio 2010 and .NET Framework 4.0 arrive**. 2010. Disponível em: <https://arstechnica.com/information-technology/2010/04/visual-studio-2010-and-net-framework-40-arrive/>. Acesso em: 28 set. 2019.

SABBAGH, R. **Scrum: Gestão Ágil para Projetos de Sucesso**. 1. ed. [S.l.]: Casa do Código, 2013.

SANTOS, C. dos. Padrões de arquitetura em projetos de sistemas multicamadas. 2005. Disponível em: <http://linux.ime.usp.br/~cef/mac499-05/monografias/cleiton/monografia.pdf>.

SCHWABER, K. Scrum development process. jan. 1997. Disponível em: <https://link.springer.com/chapter/10.1007/978-1-4471-0947-1_11>.

SCHWABER, K. **Agile Project Management with Scrum**. 1. ed. [S.l.]: Microsoft Press, 2004. 3-15 p.

SCRUM ORG. **Whats is Scrum?** 2019. Disponível em: <https://www.scrum.org/resources/what-is-scrum>. Acesso em: 22 abr. 2019.

SHAW, M. . G. **Software Architecture. Perspectives on an Emerging Discipline.** 1. ed. [S.l.]: Prentice Hall., 1996.

SML. **Automatização de processos ou automação de processos?** 2018. Disponível em: <https://blog.smlbrasil.com.br/automatizacao-de-processos/>. Acesso em: 02 nov. 2019.

SOARES, M. Metodologias Ágeis extreme programming e scrum para o desenvolvimento de software. **Revista Eletrônica de Sistemas de Informação**, v. 3, n. 1, 2004. Disponível em: <http://periodicosibepes.org.br/index.php/reinfo/article/view/146>.

SOMMERVILLE, I. **Engenharia De Software**. 9. ed. [S.l.]: Pearson Universidades, 2011.

TOMAS, M. Métodos ágeis: características, pontos fortes e fracos e possibilidades de aplicação. **IET Working Papers Series**, p. 1–19, jul. 2009. Disponível em: <https://run.unl.pt/bitstream/10362/2003/1/WPSeries_09_2009Tomas.pdf>.

VOLAROINT. **What is Scrum**. 2014. Disponível em: <http://www.volaroint.com/wp-content/uploads/dlm_uploads/2014/03/DC-VOLARO-Training-Scrum-What_Is_Scrum.pdf>. Acesso em: 02 nov. 2019.