SÁLUA JAWHARI DUARTE

PROPOSTA DE DOCUMENTO DE ESPECIFICAÇÃO DE REQUISITOS SEGUNDO O RUP PARA GESTÃO DE CLIENTES DO RESTAURANTE MONALISA

FACULDADES UNIFICADAS DE TEÓFILO OTONI

TEÓFILO OTONI – MG

2017

SÁLUA JAWHARI DUARTE

PROPOSTA DE DOCUMENTO DE ESPECIFICAÇÃO DE REQUISITOS SEGUNDO O RUP PARA GESTÃO DE CLIENTES DO RESTAURANTE MONALISA

Monografia apresentada ao Curso de Sistemas de Informação das Faculdades Unificadas de Teófilo Otoni, como requisito parcial à obtenção do título bacharel em Sistemas de Informação.
Área de Concentração: Engenharia de Requisitos.
Orientador: Prof. Yvssa Carneiro Desmots Eliote.

FACULDADES UNIFICADAS DE TEÓFILO OTONI

TEÓFILO OTONI – MG

2017

rede de ensino DOCTUM

FACULDADES UNIFICADAS DE TEÓFILO OTONI
NÚCLEO DE TCC / SISTEMAS DE INFORMAÇÃO
Autorizado pela Portaria 4.012 de 06/123/2004 – MEC

FOLHA DE APROVAÇÃO

A monografia intitulada: *Proposta de Documentação de Especificação de Requisitos segundo o Rup para gestão de clientes do Restaurante Monalisa,*

elaborada pela aluna Salua Jawhari Duarte,

foi aprovada por todos os membros da Banca Examinadora e aceita pelo curso de Sistemas de Informação das Faculdades Unificadas de Teófilo Otoni, como requisito parcial da obtenção do título de

BACHAREL EM SISTEMAS DE INFORMAÇÃO.

Teófilo Otoni, 21 de novembro de 2017

Professora Orientadora: Yvssa Carneiro Desmots Eliote

Professor Examinador: Amaury Gonçalves Costa

Professor Examinador: Marinho Soares

LISTA DE SIGLAS E ABREVIATURAS

DCU – Diagrama de casos de uso

E-R – Entidade/Relacionamento

IBM – *International Business Machines* (Comércio Internacional de Máquinas)

OMG – *Object Management Group*

RUP – *Rational Unified Process* (Processo Unificado Racional)

SGBD – Sistema de Gerenciamento de Banco de Dados

SRS ou ERS – *Software Requirements Specification* (Especificação de Requisitos de *Software*)

UML – *Unified Modeling Language* (Linguagem Unificada de Modelagem)

LISTA DE ILUSTRAÇÕES

SUMÁRIO

RESUMO

A ambição do trabalho acadêmico em questão incide na elaboração de um documento de Especificação de Requisitos de *Software*, empregando os padrões da metodologia RUP, como suporte ao desenvolvimento de *software*. O Restaurante Monalisa foi a empresa alvo da pesquisa, com o objetivo de criar o documento ERS para a gestão de clientes mensalistas. Com um grande fluxo de clientes e na ausência de um sistema capaz de auxiliar na organização e gestão dos seus serviços, o restaurante enfrentava dificuldades na tomada de decisão, afetando não só os negócios como o relacionamento com o cliente. Para essa análise foi utilizado o método indutivo, que constatou a informatização como sendo a solução mais viável. E para que essa proposta tornasse realidade, houve uma preocupação em realizar um bom planejamento antes da construção do *software*. A partir disso foi necessário ratificar as vantagens do uso da Engenharia de *Software* que influencia diretamente no sucesso do projeto e contribui na qualidade do futuro *software*. Aprofundando a Engenharia de *Software*, o estudo da Engenharia de Requisitos suscita a evolução da pesquisa, que orientou desde o processo de coleta requisitos até a gerência de qualidade. Com o auxílio do RUP, a metodologia se mostrou eficiente em instruir todo o processo, através do fluxo Requisitos, na elaboração do artefato. As noções de UML e Banco de Dados auxiliaram na criação dos diagramas principais que envolve a ERS, afim de detalhar o funcionamento do futuro *software*. Como desfecho, apresenta-se os prováveis resultados das hipóteses que permeiam a pesquisa, além do documento concluído, validado pelo cliente e disponível como apêndice do trabalho.

Palavras-Chave: Engenharia de *Software*; Engenharia de Requisitos; RUP; qualidade de *software*; Especificação de Requisitos.

INTRODUÇÃO

O presente estudo propõe fazer um documento de Especificação de Requisitos de *Software*, baseado nos princípios da Engenharia de Requisitos, seguindo os padrões da metodologia RUP para arquitetar um *software* capaz de realizar o controle de clientes mensalistas do Restaurante Monalisa, uma empresa de pequeno porte da cidade de Teófilo Otoni.

A maioria dos restaurantes ainda ficam à mercê de modos rudimentares de atendimento, fazendo com que haja perca de informações, dados incompletos e/ou desatualizados e sem relatórios detalhados. Com isso encontra-se dificuldades na tomada de decisão, afetando não só os negócios como o relacionamento com o cliente. Durante a análise descobriu-se que a realidade do Restaurante Monalisa não se distancia desse cenário, a informatização foi a solução encontrada para driblar a insuficiência do processo de gestão manual.

Com o objetivo de solucionar o problema, o documento foi construído voltado para o Restaurante Monalisa, que dispôs de todo o planejamento e coleta de requisitos necessários. O documento será um artefato crucial para o programador compreender a ideia geral do *software*, facilitando-o desse trabalho, evitando perda de tempo e ganhando maior agilidade na codificação. Da mesma forma, o cliente poderá compreender o futuro *software*, esclarecer possíveis dúvidas, analisar os requisitos levantados e realizar correções antes mesmo do desenvolvimento.

Um planejamento bem elaborado de requisitos, no qual passará pelos critérios de qualidade, resulta posteriormente em um produto final de alta qualidade. A engenharia de requisitos constrói uma ponte para o projeto e a construção. Os *softwares* de baixa qualidade muitas vezes tem essa característica devido a um pobre levantamento e análise de requisitos (SPÍNOLA, 2007, p.46-50).

Diante das situações relatadas, tem-se o seguinte problema: Como a criação de um Documento de Especificação de Requisitos de *Software*, segundo o RUP,

poderia melhorar a qualidade do *software* a ser desenvolvido para o Restaurante Monalisa?

Para a pergunta problema surgiram hipóteses que possivelmente respondam a questão:

H0 – Seria inviável a criação de um documento de especificação de requisitos de *Software*, seguindo os padrões da metodologia RUP aplicada ao Restaurante Monalisa, pois não haveria nenhuma melhora no entendimento do *software* a ser construído e consequentemente na sua qualidade.

H1 – A documentação de Requisitos proporcionaria ao cliente e ao desenvolvedor a possibilidade de analisar o futuro *software*, expondo dúvidas, opiniões e possíveis mudanças antes do processo de codificação.

H2 – A configuração de mudanças conforme a Engenharia de Requisitos possibilitaria um controle de versões, afim de evitar mudanças futuras em excesso, reduzindo consequentemente os gastos.

H3 – A proposta de documentação traria uma visão ampla do negócio auxiliando a tomada de decisão relacionada ao desenvolvimento, melhorando a produtividade, evitando o retrabalho e a perda de tempo no processo de desenvolvimento do *software*.

H4 – A documentação de requisitos facilitaria para o programador no desenvolvimento do *software* por conter informações que o orienta em todo processo, permitindo aos profissionais controle sobre processo de desenvolvimento, acerca dos custos, dos níveis de qualidade desejado e os prazos estimados.

H5 – A documentação de requisitos serviria como um contrato entre as ambas partes, resguardando o desenvolvedor e o cliente, inibindo futuras reclamações por estabelecer uma espécie de acordo entre os envolvidos.

Contendo como objetivo geral criar um documento de Especificação de Requisitos de *Software*, seguindo os padrões da metodologia RUP para o desenvolvimento de um *software* capaz de realizar a gestão de clientes mensalistas no Restaurante Monalisa, foram pontuados objetivos específicos para conduzir o andamento da pesquisa:

- Aprimorar os conhecimentos acerca da regra de negócio da empresa e das necessidades dos *stakeholders*.
- Avaliar o ambiente da empresa para possível aplicação da proposta.

- Realizar um levantamento de requisitos do *software* para a proposta de documentação do RUP.
- Criar o documento de Especificação de Requisitos.
- Verificar a especificação de forma a assegurar se todos os requisitos foram definidos
- Validar o documento para aprovação junto ao cliente.

O estudo é classificado como pesquisa descritiva, por trabalhar com dados e/ou fatos colhidos da própria realidade.

Quanto aos meios é classificada como pesquisa bibliográfica, por tratar-se do levantamento e seleção de informações publicadas em livros, jornais, revistas, artigos disponíveis no meio virtual, monografias, teses e dissertações. Além da revisão bibliográfica anteriormente tratada, a pesquisa é classificada como estudo de caso, por ser necessário a observação direta do pesquisador na empresa alvo deste estudo.

A presente pesquisa utilizou o método indutivo por através dos dados induzir uma realidade.

Este estudo será relevante afim de utilizar os conhecimentos adquiridos no curso de Sistemas de Informação para visar melhorias à gestão de clientes do Restaurante Monalisa. Trazendo do ponto de vista social ganhos não só para a empresa em questão, como para as demais empresas envolvidas no mesmo segmento do mercado com a utilização do documento de especificação como modelo.

1. REFERENCIAL TEÓRICO

1.1. Software

É muito comum associar *software* a programas de computador, mas esse conceito é muito raso e restritivo. Segundo Sommerville (2007, p.4), *software* não é simplesmente o programa, mas inclui também todos os dados de documentação e configuração associados, necessários para que o programa execute corretamente.

Segundo Pressman (2006, p.1):

> *Software* de computador é o produto que os profissionais de software constroem e, depois mantêm ao longo do tempo. Abrange programas que executam em computadores de qualquer tamanho e arquitetura, conteúdo que é apresentado ao programa a ser executado e documentos de todas as formas de mídia eletrônica.

Todo projeto de *software* começa com um problema que precisa de uma solução, e o *software* tem se tornado um solucionador de problemas organizacionais. O desenvolvedor solitário de antigamente foi substituído por equipes que unidos buscam produzir uma aplicação complexa, que leva tempo, esforço e investimentos altos para construir e em seguida manter o progresso do software. (PRESSMAN, 2006, p.3).

Spínola (2007, p.46) afirma que:

> Desenvolver softwares é uma atividade complexa por natureza. Uma das razões para esta afirmativa é que não existe uma única solução para cada cenário de desenvolvimento. Além disso lidamos o tempo todo com pessoas, o que torna o sucesso do projeto bastante relacionado à competência da equipe à forma como trabalham, e, para dificultar ainda mais, muitas vezes não fazemos uso de um processo bem definido para apoiar as atividades de projeto.

Para Pressman (2006, p.2), na mesma proporção que a importância do *software* cresceu, a comunidade de *software* tem buscado criar tecnologias que torne

mais fácil, mais rápido e menos dispendioso construir e manter *softwares* de alta qualidade.

A preocupação da indústria com o *software*, e a maneira pelo qual é desenvolvido, levou a adoção de métodos e práticas da Engenharia de Software, pensando em qualidade, redução de custos e ganho de tempo (SOMMERVILLE, 2007, p.4).

1.2. Engenharia de Software

Em 1968, o conceito de Engenharia de Software foi proposto em uma conferência, afim de discutir o abalo do que foi chamado “crise de *software*”. Neste cenário apresentavam projetos importantes com anos de atraso, o custo superava as previsões, o *software* não era confiável, era difícil de manter e seu desempenho era insatisfatório. Enquanto os custos de *hardware* estava caindo, os preços do *software* aumentava de maneira alarmante. Para o controle da crise, eram necessários novas técnicas e métodos que tornaram-se parte da Engenharia de Software. No entanto, assim como aumentou a habilidade de produzir *software*, cresceu também a necessidade por sistemas de *software* complexos (SOMMERVILLE, 2007, p.3-4).

Sommerville (2007, p.3), mostra que “A Engenharia de Software é um ramo da engenharia cujo foco é o desenvolvimento dentro de custos adequados de sistemas de *software* de alta qualidade”.

> A Engenharia de Software é composta de diversos conceitos de fundamental importância na área e abrange um processo, um conjunto de métodos ou práticas e diversas ferramentas que possibilitam aos profissionais desenvolverem software de alta qualidade. [1]

Os engenheiros de *software*, em geral, adotam a abordagem sistemática e organizada em seu trabalho por ser, na maioria das vezes, a maneira mais eficaz de produzir software de alta qualidade (SOMMERVILLE, 2007, p.5).

Segundo Spínola (2007, p.47):

> Sistemática por que parte do princípio de que existe um processo de desenvolvimento definindo as atividades que deverão ser executadas. Disciplinada por que parte do princípio de que os processos definidos serão seguidos. Quantificável por que se deve definir um conjunto de medidas a serem extraídas do processo durante o desenvolvimento de forma que as tomadas de decisão relacionadas ao desenvolvimento do software (por exemplo, melhoria de processo) sejam embasadas em dados reais e não em “achismos”.

[1] Disponível em <http://www.devmedia.com.br/principios-da-engenharia-de-software/29630>

O autor acrescenta que os principais objetivos da Engenharia de Software são a qualidade de software, a produtividade no desenvolvimento, operação e manutenção do software, e a permissão para que profissionais tenham controle sobre o desenvolvimento de software dentro de custos, prazos e níveis de qualidade desejados.

Os resultados produzidos pelo estudo da engenharia de software são denominados artefatos, que para um engenheiro de *software* consiste em um conjunto de programas, já no ponto de vista do usuário consiste em informações resultantes que tornam melhor a vida dele. O documento de especificação de requisitos encaixaria como um exemplo de artefato, como sendo o resultado de uma análise de registro dos requisitos que ilustra o sistema. [2]

1.2.1. Camadas da Engenharia de Software

A engenharia de *software* é uma tecnologia em camadas que inclui processo, métodos e ferramentas, sendo que a base se apoia no compromisso com a qualidade, como retrata a Figura 1.

Figura 1 - As quatro camadas base da Engenharia de Software

Fonte: PRESSMAN, 2006, p.17.

A camada de processo é o alicerce da engenharia de *software*, é o adesivo que une as camadas de tecnologia e permite o desenvolvimento racional e oportuno de *softwares*. (PRESSMAN, 2006, p.17-18). O mesmo ainda acrescenta que:

[2] Disponível em <http://www.devmedia.com.br/principios-da-engenharia-de-software/29630>

> Os processos de *software* formam a base para o controle gerencial de projetos do *software* e estabelecem o contexto no qual os métodos técnicos são aplicados, os produtos de trabalho (modelos, documentos, dados, relatórios, formulários etc.) são produzidos, os marcos são estabelecidos, a qualidade é assegurada e as modificações são adequadamente geridas.

O autor completa que os métodos de engenharia de *software* repousam num conjunto de princípios básicos que regem cada área da tecnologia e incluem atividades de modelagem e outras técnicas descritivas além de fornecerem a técnica de “como fazer” para construir *softwares*. Eles abrangem tarefas de comunicação, análise de requisitos, modelagem de projeto, construção de programas, testes e manutenção.

“As ferramentas de engenharia de *software* fornecem apoio automatizado ou semiautomatizado para o processos e para os métodos” (PRESSMAN, 2006, p.18).

1.2.2. Processo de Software

Um processo de *software* é um conjunto de atividades com os respectivos responsáveis por execução, ferramentas de apoio e artefatos produzidos que somados gera um produto de *software*. O uso de um processo bem definido facilita e apoia as atividades do projeto, isto é, definir como a equipe deverá trabalhar para alcançar o objetivo de desenvolver um *software* com qualidade dentro dos prazos, custos e requisitos definidos (SPÍNOLA, 2007, p.46).

Já para Pressman (2006, p.16) o processo de *software* é um roteiro que segundo ele quando você elabora um produto ou sistema é importante percorrer uma série de passos previsíveis – um roteiro que o ajuda a criar a tempo um resultado de alta qualidade. Todo o roteiro tem importância porque fornece estabilidade, controle e organização para uma atividade que pode, se deixada sem controle, tornar-se bastante caótica.

De acordo com Sommerville (2007, p.6), existem quatro atividades fundamentais que são comuns a todos os processos de *softwares*:

- Especificação de *software* ou engenharia de requisitos*:* clientes e engenheiro definem o *software* e as restrições para essa operação. Esse processo produz um documento de requisitos que é a especificação do sistema.
- Desenvolvimento de *software*: projeto e codificação do *software*.
- Validação de *software*: verificação para garantir que o *software* supre com as necessidades do cliente.
- Evolução de *software*: modificação para se adaptar as mudanças vindouras dos requisitos.

1.3. Requisitos

Por um lado Spínola (2007, p.48) afirma que “Podemos entender requisitos como sendo o conjunto de necessidades explicitadas pelo cliente que deverão ser atendidas para solucionar um determinado problema do negócio no qual o cliente faz parte”. Por outro lado Sommerville (2007, p.79) defende que “os requisitos de um sistema são as descrições dos serviços fornecidos pelo sistema e as suas restrições operacionais”.

Os tipos de requisitos são divididos em três segmentos: funcionais, não funcionais e de domínio, entretanto, olhando sobre outro prisma, Sommerville (2007, p.80), defende que:

> Alguns problemas que surgem durante o processo de engenharia de requisitos são resultantes da falta de uma clara separação entre esses diferentes níveis de descrição. Faço essa distinção entre eles usando o termo requisitos de usuário para requisitos abstratos de alto nível e requisitos de sistema para a descrição detalhada do que o sistema deve fazer.

1.3.1. Requisitos Funcionais

São requisitos diretamente ligados a funcionalidade do *software*, descrevem as funções que o *software* deve executar e a interação entre o ambiente e o sistema, o seu comportamento (SPÍNOLA, 2007, p.48).

Sommerville (2007, p. 80-81), expõe que os requisitos funcionais são declarações de serviços que descrevem a função do sistema detalhadamente, suas entradas, saídas, exceções etc. Ou seja, fornecem informações de como o sistema deve se comportar em determinadas situações.

1.3.2. Requisitos Não-Funcionais

Spínola (2007, p.48) diz que “São requisitos que expressam condições que o *software* deve atender ou qualidades específicas que o *software* deve ter.”

Eles podem estar relacionados às propriedades emergentes do sistema, como confiabilidade, tempo de resposta e espaço de armazenamento. Os requisitos não funcionais aplicam-se no sistema como todo e colocam restrições sobre os serviços ou as funções oferecidos pelo sistema (SOMMERVILLE, 2007, p.80-82).

1.3.3. Requisitos de Domínio

“São requisitos derivados do domínio da aplicação e descrevem características do sistema e qualidades que refletem o domínio. Podem ser requisitos funcionais novos, restrições sobre requisitos existentes ou computações específicas” (SPÍNOLA, 2007, p.48).

De acordo com Sommerville (2007, p.81) “são requisitos provenientes do domínio da aplicação do sistema e refletem as características e as restrições desse domínio”.

1.4. Engenharia de Requisitos

Uma das disciplinas mais evidenciadas pelos seus benefícios, a Engenharia de Requisitos é fundamental para o sucesso do desenvolvimento de *software*, portanto a análise criteriosa dos dados deve ser executada com cuidado, para que não haja consequências prejudiciais ao projeto. Sommerville (2007, p.77) afirma:

> [...] o maior problema que enfrentamos no desenvolvimento de sistemas de *software* grandes e complexos seja o da engenharia de requisitos. Ela está relacionada com a definição do que o sistema deve fazer, suas propriedades emergentes desejáveis e essenciais e as restrições quanto à operação do sistema e quanto aos processos de desenvolvimento de *software*. Você pode, portanto pensar na engenharia de requisitos como o processo de comunicação entre os clientes e os usuários de *software* e os desenvolvedores de *software.*

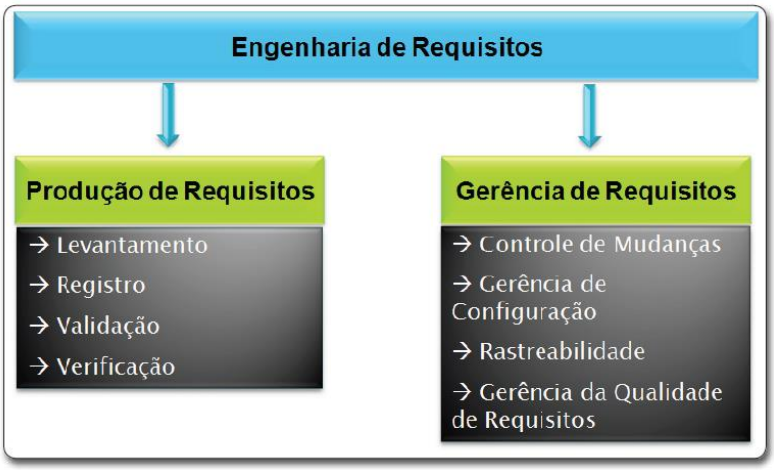
Já Filho (2003, p.5) lembra que "O conjunto das técnicas de levantamento, a documentação e análise forma a Engenharia de Requisitos, que é uma das disciplinas da Engenharia de Software."

Podem-se citar alguns objetivos da Engenharia de Requisitos, segundo Spínola (2007, p.49)

> - Estabelecer uma visão comum entre o cliente e a equipe de projeto em relação aos requisitos que serão atendidos pelo projeto de *software*;
> - Registrar e acompanhar requisitos ao longo de todo o processo de desenvolvimento;
> - Documentar e controlar os requisitos alocados para estabelecer uma *baseline* para uso gerencial e da engenharia de *software*;
> - Manter planos, artefatos e atividades de *software* consistentes com os requisitos alocados.

Para melhor compreensão a Engenharia de Requisitos serão apresentadas em duas partes: produção de requisitos e gerência de requisitos, como mostra a Figura 2 a seguir:

Figura 2 – Engenharia de Requisitos



Fonte: SPÍNOLA, 2007, p.48.

1.4.1. Produção de Requisitos

Durante a produção de requisitos, devem possuir além das atividades essenciais de levantamento e especificação, atividades relacionadas a garantia da qualidade. Um dos artefatos produzidos logo no início do processo de desenvolvimento do *software* é a sua especificação de requisitos, que quando bem elaborada é um pré-requisito para um *software* de qualidade, embora não seja a garantia disso (SPÍNOLA, 2007, p.50).

1.4.1.1. Levantamento

Diante da preocupação em detalhar, analisar e observar os requisitos do *software* a ser desenvolvido, a especificação de requisitos possui um grande papel de importância para o processo de codificação. Infelizmente muitos clientes e até mesmo desenvolvedores não entendem da necessidade dessa etapa fundamental. Menosprezar a especificação de requisitos é correr um risco de chegar no processo final e não se deparar com o produto final esperado.

É papel do engenheiro de *software* mostrar para o cliente e usuários que uma boa especificação de requisitos é indispensável para que correspondam com as

necessidades do cliente, é um investimento essencial e não supérfluo. Em uma análise comparativa, se uma boa especificação gasta tempo e dinheiro, a ausência da mesma gasta mais tempo e dinheiro (FILHO, 2003, p.5). O autor ainda acrescenta que:

> Um dos problemas básicos da engenharia de *software* é o levantamento e a documentação dos requisitos dos produtos de *software*. Quando esse levantamento é bem-feito, os requisitos implícitos são minimizados. Quando a documentação é bem-feita, os requisitos documentados tem maiores chances de serem corretamente entendidos pelos desenvolvedores. Algumas técnicas de análise dos requisitos ajudam a produzir especificações mais precisas e inteligíveis (FILHO, 2003, p.5).

1.4.1.2. Registro

"Os requisitos devem ser documentados para que possam servir de base para o restante do processo de desenvolvimento" (SPÍNOLA, 2007, p.50). Em geral, é produzido um documento de especificação de requisitos, documentado em um nível apropriado de detalhes, de forma que as partes interessadas compreendam com clareza os requisitos, apesar da perspectiva do documento SRS ser voltado, em especial, para os desenvolvedores. Sommerville (2007, p.91) acrescenta:

> O documento de requisitos de *software* (algumas vezes chamado de especificação de requisitos de *software* ou SRS – *Software Requirements Specification*) é a declaração oficial do que os desenvolvedores de sistema devem implementar. Deve incluir os requisitos de usuário de um sistema e uma especificação detalhada dos requisitos de sistema.

Acompanhando o pensamento de Spínola (2007, p.10), "O registro dos requisitos num documento próprio facilita o controle de alterações de todos os envolvidos na manutenção dos requisitos, bem como a geração de versões do documento e a facilidade de acesso por todos os envolvidos". O texto abaixo acrescenta:

> Este documento tem por objetivo orientar o processo de coleta e elicitação dos requisitos necessários para o desenvolvimento de um *software.* Nele deverão estar contidas as informações necessárias que permita ao cliente (e outras partes interessadas) descreverem as funções desejadas do produto, o seu desempenho, aparência, as relações entre as partes interessadas, expectativas, e as outras características que se julgar necessária.[3]

[3] Disponível em <https://www.cti.ufu.br/sites/cti.ufu.br/files/cti-especificacao-requisito-software-em%20conformidade-com-cmmi.pdf>

1.4.1.3. Verificação

Spínola (2007, p.51) entende a verificação como uma atividade que examina a especificação do *software*, de forma a assegurar que todos os requisitos foram definidos, detectando e corrigindo, sem ambiguidades, incoerências ou omissões, durante a fase de definição dos requisitos. Textualmente: “Um tipo particular de revisão de *software* são as inspeções. Inspeções possuem um processo de detecção de defeitos rigoroso e bem definido” (SPÍNOLA, 2007, p.51).

Já para Sommerville (2007, p.341), verificar um sistema é avaliar se ele atende a sua especificação, verificar se atende aos requisitos funcionais e não funcionais especificados.

1.4.1.4. Validação

Sommerville (2007, p.105-106) entende que a validação de requisitos dedica-se a mostrar se os requisitos realmente definem o sistema que o usuário deseja; sobrepõe à análise; está relacionada à descoberta de problemas com os requisitos. Raramente é encontrado todos os problemas de requisitos durante o processo de validação, mesmo assim não se deve subestimar o etapa de validação. Por outro lado Spínola (2007, p.51) expõe que “A validação representa a atividade em que obtemos o aceite do cliente sob determinado artefato”. Veja-se:

> A validação de requisitos é importante porque os erros em um documento de requisitos podem levar a custos excessivos de retrabalho quando são descobertos durante o desenvolvimento ou depois que o sistema está em operação. O custo de correção de um problema de requisitos, fazendo uma mudança de sistema, é muito maior do que a correção de erros de projeto e de codificação (SOMMERVILLE, 2007, p.105).

Observe-se a Tabela 1 a seguir:

Tabela 1 - Cenário atual de desenvolvimento

	% do Custo de Desenvolvimento	% dos erros introduzidos	% dos erros encontrados	Custo relativo de correção
Análise de Requisitos	5	55	18	1
Projeto	25	30	10	1 – 1.5
Códificação e teste de unidade	50			
Teste	10	10	50	1 – 5
Validação e Documentação	10			
Manutenção		5	22	10 – 100

Fonte: SPÍNOLA, 2007, p.47

De acordo com a Tabela 1, no início do projeto, o custo das atividades relacionadas à análise de requisitos é baixo, entretanto, nessa fase o percentual de erros introduzidos são maiores e a correção destes problemas ainda nesta fase é baixo. Por outro lado, deixar passar um erro para tratá-lo posteriormente pode aumentar bastante o custo do projeto, quanto mais tarde os erros forem encontrados mais custosa será a correção.

"Embora simples, esta análise nos permite concluir que é importante que seja dada maior importância às atividades relacionadas à especificação dos requisitos do *software*" (SPÍNOLA, 2007, p.47).

1.4.2. Gerência de Requisitos

Os requisitos são voláteis e instáveis, mudanças externas no ambiente, erros incorridos no processo de requisitos, todos esses fatores levam a alteração de requisitos. A atividade de gerência de requisitos por sua vez compreende em administrar os requisitos ao longo do tempo (SPÍNOLA, 2007, p.51).

Observando, segundo Cintra (2006, p.74), "A gerência de requisitos monitora o desenvolvimento e implementação dos requisitos, registrando seus atributos, status e dependências, com o objetivo de controlar o andamento e as mudanças realizadas."

Mas também Sommerville (2007, p.107) apresenta que a gerência de requisitos tem o objetivo de compreender e controlar as mudanças dos requisitos de sistema. Este processo se inicia assim que uma versão inicial do documento de requisitos esteja disponível.

1.4.2.1. Controle de mudanças

Para conduzir as mudanças ao longo do tempo, Spínola (2007, p.51), recomenda preparo e planejamento, uma das maneiras mais utilizadas é a *baseline*[4] de requisitos que permite diferenciar o que era requisito original, o que foi introduzido e o que foi descartado.

> É verdade que os requisitos de *software* mudam, mas o impacto de mudança varia com a época em que é introduzida. Quando mudanças são solicitadas antecipadamente (antes que o projeto e a codificação tenham começado), o impacto de custo é relativamente baixo. No entanto, à medida que o tempo passa, o impacto de custo cresce rapidamente – recursos foram comprometidos, a estrutura do projeto foi estabelecida e a mudança pode causar consequências que exijam recursos adicionais e grandes modificações no projeto (SOMMERVILLE, 2007, p.11).

Uma análise deve ser feita para um processo de controle de mudanças. Segundo Spínola (2007, p.52), os seguintes passos devem ser seguidos:

- Checar validade da solicitação de mudança;
- Identificar os requisitos diretamente afetados com a mudança;
- Identificar dependências entre requisitos para buscar os requisitos afetados indiretamente;
- Assegurar com solicitante a mudança a ser realizada;
- Estimar custo da mudança;
- Obter acordo com usuário sobre o custo da mudança.

1.4.2.2. Gerência de configuração

Alguns fatores são responsáveis pelas mudanças, pode-se citar as mudanças no negócio, o aumento da complexidade dos requisitos, a alteração estrutural do requisito, evolução da percepção do sistema pelos *stakeholders*[5] e descoberta de falhas que exigem correção (CINTRA, 2006, p.74).

Ressalta-se a concepção de Spínola (2007, p.52):

---

[4] *Baseline* (linha de base), é um modelo que serve para acompanhamento do projeto, é uma 'imagem' de versão do projeto. Além de acompanhar, a *baseline* é utilizada para reproduzir e rastrear todas as etapas e acontecimentos realizados durante todo o projeto. (SPÍNOLA, 2007, p.52)

[5] *Stakeholder*: as partes interessadas do projeto, na maioria das vezes direcionada para o cliente.

> A gerência de configuração de *software* existe no intuito de definir os critérios que permitam realizar tais modificações mantendo-se a consistência e a integridade do *software* com as especificações, minimizando problemas decorrentes ao processo de desenvolvimento, através de um controle sistemático sobre as modificações.

O mesmo autor explica que a gerência de configuração por sua vez consiste em estabelecer normas, ferramentas e *templates* que permitam gerenciar de maneira satisfatória os itens de configuração de um sistema. As *baselines* são estabelecidas para que os itens de configuração sejam controlados pelos seus desenvolvedores, tornando mais fácil realizar as modificações nos artefatos e avaliar o seu grau de evolução.

1.4.2.3. Rastreabilidade

A rastreabilidade pode ser definida como a habilidade de se acompanhar a vida de um requisito em ambas as direções do processo de *software* e durante todo o seu ciclo de vida. Para implementá-la usualmente é confeccionado em paralelo a especificação de requisitos, um artefato chamado matriz de rastreabilidade com o objetivo de mapear os rastros dos requisitos descritos na especificação (SPÍNOLA, 2007, p.52).

Segundo Spínola, os rastros dos requisitos podem ser de duas formas: pré-rastreabilidade, que fundamentam a criação do requisito, ou pós-rastreabilidade, que se formam a partir do requisito criado. A dificuldade com a rastreabilidade está no grande volume de dados, no entanto para implementá-la é confeccionado ao mesmo tempo que a especificação de requisitos a matriz de rastreabilidade, um artefato com o objetivo de mapear os rastros dos requisitos da especificação.

1.4.2.4. Gerência de Qualidade de Requisitos

Neste cenário a gerência “é responsável por manter uma infraestrutura necessária para atividades de verificação que tornem possível investigarmos a qualidade dos requisitos que estamos definindo” (SPÍNOLA, 2007, p.52).

Então ele mostra que existem critérios de qualidade quando se trabalha com requisitos que são:

- Correção: os requisitos reais devem coincidir com os requisitos identificados.
- Não ambiguidade: quando consegue ser interpretado por todos os envolvidos.
- Completude: quando descreve todas as demandas de interesse dos usuários.
- Consistência: quando nenhum subconjunto do sistema entra em conflito com os demais requisitos.
- Verificabilidade: quando existe uma forma efetiva para indicar se o sistema cumpre o requisito.
- Modificabilidade: quando as alterações podem ser realizadas de maneira simples e consistente.

1.5. Processo RUP

"O *Rational Unified Process* (RUP) é um exemplo de modelo de processo moderno que foi derivado do trabalho sobre a UML[6] e do Processo Unificado[7] de desenvolvimento de *Software* associado" (SOMMERVILLE, 2007, p.54).

O RUP, segundo Cintra (2006, p.32), descreve:

> • Papéis: ou perfis de trabalho que definem quem é responsável por cada tarefa;
> • Artefatos: que representam os produtos do processo, o que será gerado durante o processo de desenvolvimento;
> • Atividades: executadas durante o processo de desenvolvimentos, que descrevem como os representantes de cada papel trabalham para atingir os objetivos do projeto e construir o sistema de *software*;
> • Fluxos de atividades: que orientam a execução das atividades, definindo quando esta deve ocorrer.

Os elementos descritos acima são organizados em um *framework*[8] de processo de desenvolvimento de *software* representado na Figura 3.

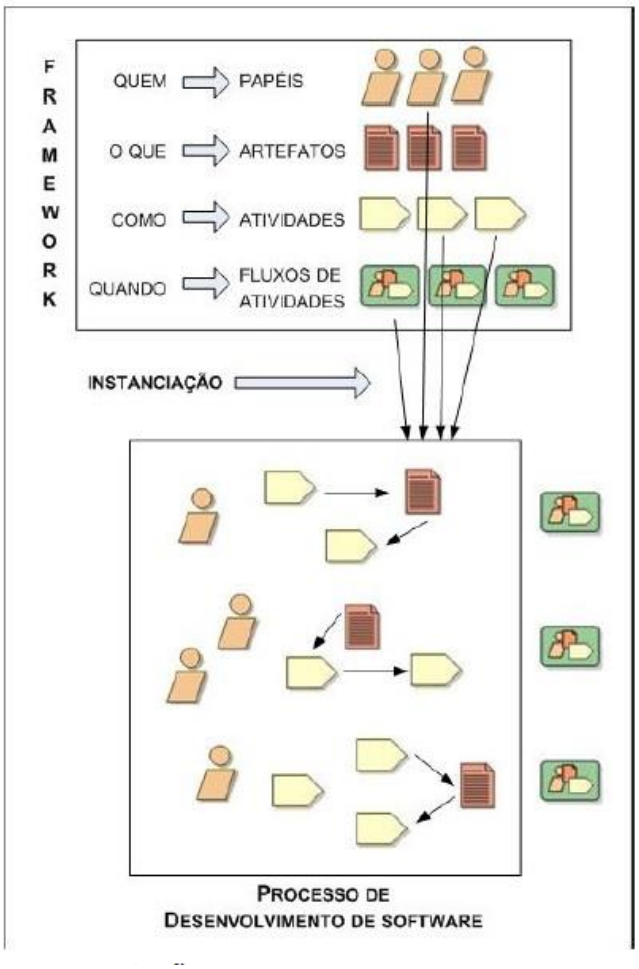
---

[6] UML - "[...] notação mais amplamente usada para modelagem de análise e projeto [...]" (PRESSMAN,2006, p.51).

[7] "O Processo Unificado é algumas vezes chamado de Processo Unificado Racional (RUP), por causa da *Rational Corporation*, uma contribuinte pioneira para o desenvolvimento e refinamento do processo e uma construtora de ambientes completos (ferramentas e tecnologias) que apoiam o processo" (PRESSMAN,2006, p.52).

[8] "Estrutura genérica em um domínio específico que pode formar a base de uma família de aplicações [...] são geralmente implementados como um conjunto de classes concretas e abstratas, especializadas e instanciadas para criar uma aplicação" (SOMMERVILLE, 2007, p.527)

Figura 3 - Representação do framework de Processo e uma Instância



Fonte: CINTRA, 2006, p.33

Citando Kruchten (2001), Cintra (2006, p.33) afirma que o:

> RUP foi intencionalmente desenvolvido para uma ampla aplicabilidade, ajustando-se a variações de tamanho do projeto; domínio da aplicação (sistemas de negócio, sistemas técnicos); tecnologia utilizada (linguagens, plataformas); e contexto de negócio (desenvolvimento interno, desenvolvimento de produto, contratos de desenvolvimento).

1.5.1. Práticas do RUP

As práticas do RUP foram criadas para a solução de diversos problemas de forma a assegurar maior possiblidade de sucesso em um projeto de *software*. Dentre os problemas foram levantados os principais sintomas, conforme o quadro 1 a seguir:

Quadro 1 - Fatores críticos do sucesso

Fatores Críticos	%
1. Requisitos Incompletos	13.1%
2. Falta de Envolvimento do Usuário	12.4%
3. Falta de Recursos	10,6%
4. Expectativas Irreais	9,9%
5. Falta de Apoio Executivo	9,3%
6. Mudança de Requisitos e Especificações	8,7%
7. Falta de Planejamento	8,1%
8. Sistema não mais necessário	7,5%

Fonte: SPÍNOLA, 2007, p.47

Com base nos sintomas mostrados na Quadro 1, foram encontradas as causas desses problemas, chegando a seguinte conclusão, segundo Cintra (2006, p.34):

- Falta de definição formal de um processo de gerenciamento de mudanças;
- Propagação de mudanças descontroladas;
- Comunicação ambígua e imprecisa;
- Arquiteturas não robustas;
- Alta complexidade;
- Inconsistências não detectadas em requisitos, no projeto (design) e na implementação;
- Testes insuficientes do sistema;
- Controle subjetivo do status do projeto;
- Falha ao atacar riscos do projeto;
- Automação insuficiente

“As práticas recomendadas por RUP foram identificadas em um grande número de projetos de sucesso e a sua falta foi verificada em um grande número de projetos fracassados” (CINTRA, 2006, p.34). Essas práticas estão ilustradas na Figura 4:

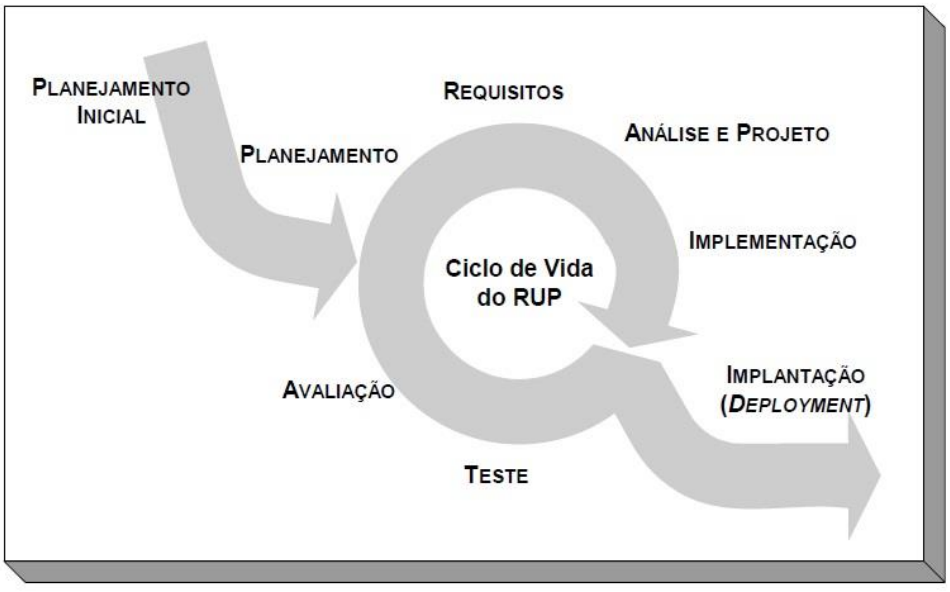
Figura 4- Práticas do RUP

Fonte: CINTRA, 2006, p.35

1.5.1.1. Desenvolver software iterativamente

O ciclo de vida proposto pelo RUP é baseado no modelo em espiral, que divide a construção do *software* em iterações resultando em uma liberação de versão executável do sistema (CINTRA, 2006, p.35). O ciclo de vida iterativo do RUP é ilustrado na Figura 5, da seguinte forma:

Figura 5 - Ciclo de vida do RUP



Fonte: KRUCHTEN, 2011 apud CINTRA, 2006, p.35

O autor ainda explica:

> O desenvolvimento de *software* de forma iterativa soluciona ou minimiza vários dos problemas de desenvolvimento. Os usuários recebem as liberações de versão de cada iteração, o que facilita seu retorno sobre o *software* produzido, apontando o que foi mal entendido e esclarecendo os demais requisitos. Além disso, os usuários *stakeholders* têm evidências concretas do andamento do sistema. As porções do sistema que oferecem maior risco são desenvolvidas primeiro, e possíveis problemas já podem começar a ser solucionados. O status do projeto é determinado objetivamente, analisando os artefatos que foram concluídos até uma determinada data. A carga de trabalho da equipe é distribuída de maneira uniforme ao longo do projeto. Por fim, a equipe pode utilizar as lições aprendidas em cada iteração para melhorar o processo de desenvolvimento continuamente (CINTRA, 2006, p.35).

1.5.1.2. Gerenciar os requisitos

Os requisitos de um sistema são bem dinâmicos o que contribui para uma constante alteração no processo de desenvolvimento, devido a fatores internos e externos ao projeto, isso faz da gerência de requisitos uma atividade desafiadora. O RUP utiliza essa prática em uma abordagem disciplinada, comunicando os requisitos claramente entre os envolvidos e oferecendo formas de priorização seleção e rastreamento de requisitos (CINTRA, 2006, p.36), o autor ainda aborda que:

> Em RUP, a gerência de requisitos consiste em elicitar, organizar e documentar a funcionalidade e as restrições requeridas pelo sistema; avaliar mudanças nesses requisitos e estimar seu impacto; e registrar e documentar decisões.

1.5.1.3. *Usar arquiteturas baseadas em componentes*

Cintra (2006, p.36) aponta que a arquitetura é um dos produtos de trabalho mais importantes de um sistema, onde descreve toda a organização do *software*, os elementos estruturais que o compõe, o comportamento e a composição desses elementos. Uma possível abordagem para a composição da arquitetura é o desenvolvimento baseado em componentes, onde é organizado em módulos coesos e fracamente acoplados uns aos outros, para a possibilidade de aquisição de cada um desses módulos. Ainda acrescenta que:

> Os objetivos do desenvolvimento de uma arquitetura baseada em componentes são: promover a criação de arquiteturas estáveis e robustas, através de utilização de componentes cuja qualidade é comprovada por sua utilização em outros sistemas; dividir o sistema em módulos de forma que sua evolução possa ser feita de forma isolada (alterações em um módulo não precisarão envolver todo o sistema); facilitar o reuso; facilitar a gerência de configuração (CINTRA,2006, p.36).

1.5.1.4. Modelar software visualmente

Cintra (2006, p.36), conceitua da seguinte forma:

> Um modelo é uma simplificação da realidade que descreve um sistema de acordo com uma perspectiva particular. Modelos são usados para compreender melhor um problema ou uma solução, para comunicar uma ideia, para descrever uma abordagem ou alternativas de resolução de questões relacionadas ao sistema; para documentar essas alternativas ou a decisão tomada em prol de uma das alternativas. Todas essas abstrações do sistema permitem que a equipe de desenvolvimento construa o sistema por partes, mantendo o foco na perspectiva que está sendo analisada. Assim, a modelagem melhora a habilidade da equipe de gerenciar a complexidade do *software*.

Através da prática da visualização de modelos, os comportamentos do sistema podem ser descritos sem ambiguidade, detalhes podem ser abstraídos e inconsistência e problemas são facilmente encontrados (CINTRA, 2006, p.37).

1.5.1.5. Verificar a qualidade do software continuamente

Com a iteratividade, a verificação é constante e esta se dá por meio de testes em cenário de utilização. É um processo de contínuo de avaliação quantitativa e qualitativa. A verificação contínua de qualidade faz com que o acompanhamento do projeto seja mais objetivo, sendo evidenciadas as inconsistências em requisitos, projeto e implementações, logo os riscos são mitigados e os defeitos encontrados cedo (CINTRA, 2006, p.37).

1.5.1.6. Controlar as mudanças no software

A crença em manter a rastreabilidade entre os elementos de cada versão liberada e entre os elementos ao longo de liberações múltiplas e paralelas é fundamental para avaliar e gerenciar ativamente o impacto de mudanças, essa é a principal motivação para essa prática (CINTRA,2006, p.37), o autor ainda agrega que:

> O controle de mudanças em RUP fornece uma sequência de atividades claramente descritas e padronizadas para execuções repetidas. São definidas as Requisições de Mudança (*Change Requests*), documentos descrevendo criteriosamente as mudanças. A propagação de mudanças é controlada pelo processo e é passível de verificação.

1.5.2. Características do RUP

Para Cintra (2006, p.37), o RUP tem três características que envolve a sua utilização:

- É um processo de desenvolvimento orientado a casos de uso
- Define um *framework* de processo que pode ser adaptado e estendido pela organização que o adota.
- Utiliza largamente o suporte de ferramentas automatizadas

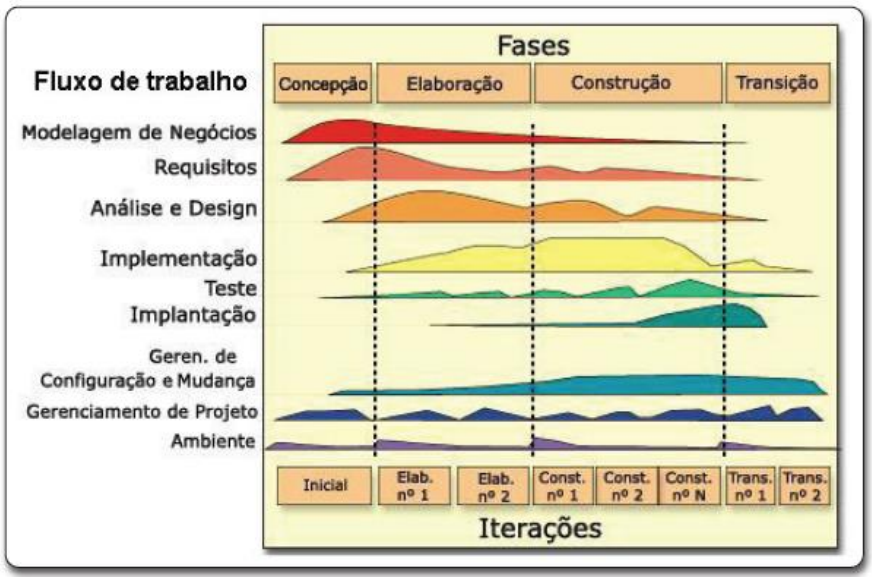
Os casos de uso são utilizados pelo RUP para organizar o desenvolvimento do sistema, estes por sua vez são cenários de utilização do sistema pelo usuários e sua principal meta é caracterizar um fluxo de execução do sistema. Por ser um *framework* adaptável, ele pode ser utilizado de diversas formas diferentes de acordo com o tamanho do projeto e a necessidade da organização (CINTRA, 2006, p.37-38), como abaixo:

> O suporte a ferramentas automatizadas a RUP é muito extenso, o que é esperado, já que esta é uma metodologia criada comercialmente, por uma empresa interessada em difundir suas ferramentas de suporte ao desenvolvimento de projetos de *software*. O próprio *framework* RUP é documentado como um produto comercial, representado por uma base de conhecimento disponível via navegador *web*. As ferramentas tem o objetivo de aumentar a produtividade em cada atividade do processo e de facilitar as práticas de RUP (CINTRA, 2006, p.38).

1.5.3. Estrutura do RUP

O modelo RUP é constituído por quatro fases discretas no processo, como mostra Gráfico 1 a seguir.

Gráfico 1 – Fases do processo RUP



Fonte: SPÍNOLA,2007, p. 29

Sommerville (2007, p.54) diz que Concepção é onde estabelece uma *business case*[9] para o sistema e identifica todas as entidades externas que irão interagir com o sistema com o negócio. Spínola (2007, p.29) completa dizendo que “Essa fase tem como objetivo verificar a viabilidade do projeto, bem como os riscos e um dos fatores não menos importantes: definir os casos de uso mais críticos obtendo as funções chave do sistema”.

Elaboração desenvolve um entendimento do domínio do problema, resultando em um modelo de requisitos para o sistema, uma descrição de arquitetura e um plano de desenvolvimento para o *software* (SOMMERVILLE, 2007, p.55). Spínola (2007, p.29) completa que:

> Durante essa fase, a maioria dos casos de uso são especificados e detalhados. A arquitetura do sistema é projetada utilizando artefatos que podem ser estáticos ou dinâmicos. Neste instante são apresentados, o *baseline* completo do projeto, os componentes que formarão a equipe de desenvolvimento, etc. No final dessa fase os envolvidos devem estar aptos a planejar a fase de construção em detalhes.

[9] Em português, caso de negócio, conhecido também como Plano de Negócio, “é um guia para orientar a criação de um novo negócio ou empreendimento.” (Disponível em <https://www.projectbuilder.com.br/blog-pb/entry/projetos/como-fazer-um-business-case-eficiente-do-seu-projeto>)

A fase da construção pela visão de Sommerville (2007, p.55) está relacionada a programação e teste de sistema, ao concluir esta fase o sistema de *software* deverá estar em funcionamento e a documentação associada pronta para ser liberada para os usuários.

> A fusão de vários artefatos de *software* ocorre neste momento, possibilitando que o sistema seja implementado quase que completamente. Tem-se uma visão geral de como o *baseline* do projeto está sendo seguido. No final dessa fase, o sistema deve estar totalmente preparado para a transição ao usuário (SPÍNOLA, 2007, p.29).

E Transição está relacionada à transição da comunidade de desenvolvedores para a comunidade de usuários para funcionamento no ambiente real (SOMMERVILLE, 2007, p.54-55). Spínola (2007, p.29) ainda acrescenta que:

> O objetivo dessa fase é garantir que todos os requisitos do projeto foram atendidos e implementados corretamente. O produto final pode ser liberado em uma versão beta. Existem ainda outras atividades que, de acordo com o projeto, podem ocorrer de maneira paralela, por exemplo, a preparação do ambiente, a conclusão do manual do usuário, identificação e correção de defeitos. No final dessa fase deve-se tirar uma conclusão geral do projeto, obtendo os pontos positivos e negativos os quais devem ser utilizados durante a concepção de projetos futuros.

Os *workflows* também chamados de fluxos de trabalho, como apresentado no gráfico 1, são fluxos que são distribuídos ao longo de todas as fases do RUP. De acordo com Pressman (2006, p.53), um fluxo de trabalho identifica as tarefas exigidas para realizar os produtos que são produzidos na conclusão bem sucedida dessas tarefas, pode-se dizer que um fluxo de trabalho é análogo a um conjunto de tarefas. Na Tabela 2 a seguir, estão descritos os principais *workflows* do RUP.

Tabela 2 – Workflows estáticos do RUP

Workflow	Descrição
Modelagem de negócios	Os processos de negócios são modelados usando casos de uso de negócios.
Requisitos	Os agentes que interagem com o sistema são identificados e os casos de uso são desenvolvidos para modelar os requisitos de sistema.
Análise e projeto	Um modelo de projeto é criado e documentado usando modelos de arquitetura, modelos de componente, modelos de objeto e modelos de seqüência.
Implementação	Os componentes de sistema são implementados e estruturados em subsistemas de implementação. A geração automática de código com base nos modelos de projeto ajuda a acelerar esse processo.
Teste	O teste é um processo iterativo realizado em conjunto com a implementação. O teste de sistema segue o término da implementação.
Implantação	Uma versão do produto é criada, distribuída aos usuários e instalada no local de trabalho.
Gerenciamento de configuração e mudanças	Este workflow de apoio gerencia as mudanças do sistema
Gerenciamento de projetos	Este workflow de apoio gerencia o desenvolvimento do sistema
Ambiente	Este workflow está relacionado à disponibilização de ferramentas apropriadas de software para a equipe de desenvolvimento.

Fonte: SOMMERVILLE, 2007, p.55

Sommerville (2007, p.56) explica que, as fases do RUP são dinâmicas e objetivas, já os *workflows* são estáticos e contém atividades técnicas que não se limitam a uma única fase, mas podem ser utilizados ao longo do desenvolvimento para atingir os objetivos de cada fase. Conclui-se que o RUP não é um processo adequado a todos os tipos de processos de desenvolvimentos, mas representa uma nova geração de processos genéricos.

1.6. Banco de Dados

Imersos em uma grande quantidade de dados que são gerados a cada instante, há uma grande preocupação em organizá-los e armazená-los afim de evitar perdas e transtornos. O banco de dados se encarrega em manter esses dados gerados, de forma organizada e gerenciada para os usuários poderem pesquisar, recuperar e atualizar os dados quando necessário.

“Um banco de dados é uma coleção de dados relacionados. Os dados são fatos que podem ser gravados e que possuem um significado implícito” (ELMASRI; NAVATHE, 2005, p.4).

Segundo Siberschatz, Korth e Sudarshan (2006, p.1), pode-se chamar banco de dados de uma coleção de dados, dados esses que contém informações relevantes a uma empresa. Os SGBD’s (Sistema de Gerenciamento de Banco de Dados), é uma coleção de dados inter-relacionados e um conjunto de programas para acessar esses dados, o objetivo principal é fornecer uma maneira de recuperar informações de banco de dados que seja tanto conveniente quanto eficiente. Já os sistemas de banco de dados, são projetados para gerenciar grandes blocos de informações.

1.6.1. Modelo de Dados

Afim de apoiar a estrutura de um banco de dados, o modelo de dados pode ser definido como sendo uma coleção de ferramentas conceituais para descrever dados, relações de dados, semântica de dados e restrições de consistência. Existem vários modelos diferentes de modelos de dados, com a finalidades de oferece uma maneira de descrever o projeto de um banco de dados no nível físico (baixo nível, fornece uma visão mais detalhada), lógico (projeção mais próximo da visão do usuário, alto nível) e *view* (definição das visões no banco) (SIBERSCHATZ; KORTH; SUDARSHAN, 2006, p.5).

1.6.1.1. Modelo Entidade / Relacionamento

O modelo de dados de entidade/relacionamento (E-R) é baseado numa percepção de um mundo real que contém uma coleção de objetos básicos, chamados entidade e relações entre esses objetos. As entidades são descritas em um banco de dados por um conjuntos de atributos (SIBERSCHATZ; KORTH; SUDARSHAN, 2006, p.11).

O modelo E-R também pode ser conceituado de acordo Elmasri e Navathe (2005, p.5) como:

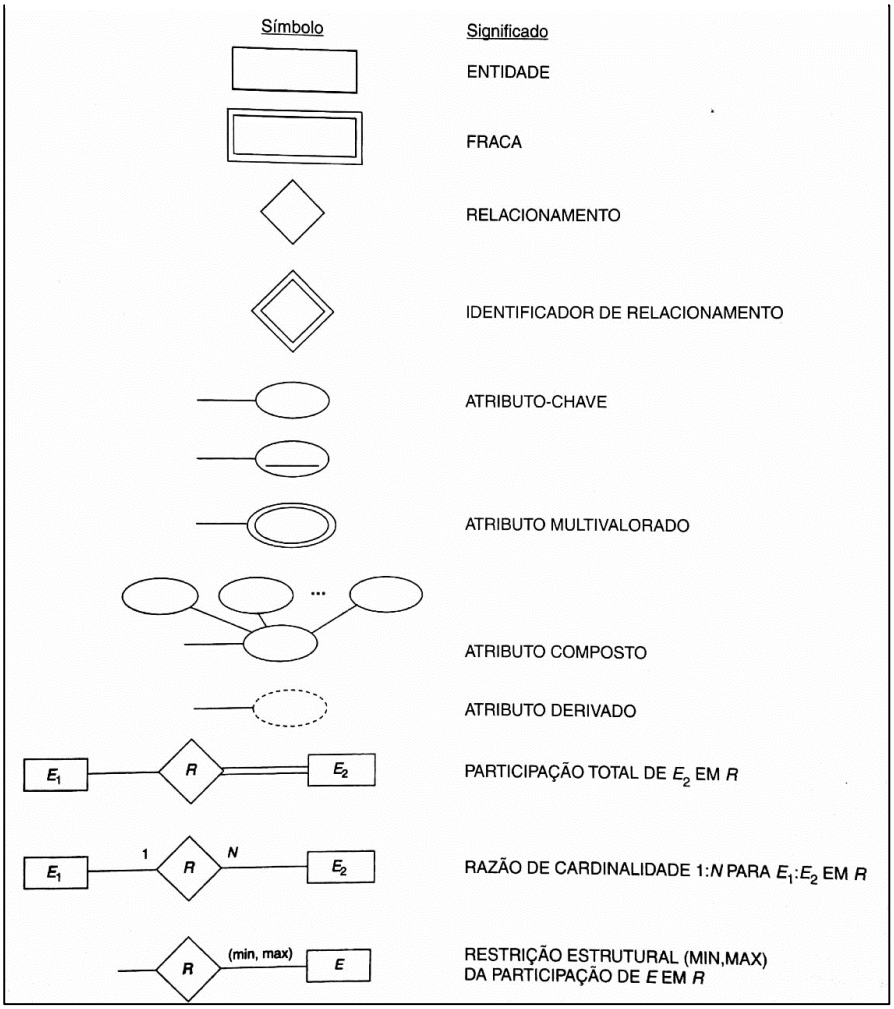
> [...] um modelo de dados conceitual de alto nível, além de muito popular. Esse modelo e suas variações são normalmente empregados para o projeto conceitual de aplicações de um banco de dados, e muitas ferramentas de projeto de um banco de dados aplicam seus conceitos.

Esse modelo foi desenvolvido para facilitar o projeto de banco de dados, permitindo a especificação de um esquema de empresa que representa a estrutura lógica geral de um banco de dados. É um modelo muito útil no mapeamento dos significados e interações de empresas reais para um esquema conceitual. Emprega-se três noções básicas: conjuntos de entidades, conjuntos de relacionamento e atributos (SIBERSCHATZ; KORTH; SUDARSHAN, 2006, p.135). Todavia:

> Além das entidades e relações, o modelo E-R representa certas restrições as quais o conteúdo de um banco de dados precisa se conformar. Uma restrição importante é a cardinalidade de mapeamento, que expressa o número de entidades ao qual outra entidade pode estar associada por meio de um conjunto de relação (SIBERSCHATZ; KORTH; SUDARSHAN, 2006 p.11).

Os autores explicam que um diagrama E-R pode expressar graficamente a estrutura lógica geral de um banco de dados. Eles são simples e claros, qualidades que podem ter motivado o amplo uso do modelo E-R (SIBERSCHATZ; KORTH; SUDARSHAN, 2006, p.142). As notações do diagrama E-R são descritas na Figura 6 a seguir.

Figura 6 - Resumo da notação de diagramas E-R



Fonte: ELMASRI; NAVATHE, 2005, p.51

1.6.2. Arquitetura de Banco de Dados

Conforme Siberschatz, Korth e Sudarshan (2006, p.16) “A arquitetura de um sistema de banco de dados é bastante influenciada pelo sistema de computador subjacente em que o sistema de banco de dados é executado”.

Ainda ressalta que a arquitetura normalmente é particionada em duas ou três partes. Em duas camadas a aplicação é particionada em um componente que reside na máquina cliente, que chama a funcionalidade do sistema de banco de dados na máquina servidora por meio de instruções da linguagem. Já em uma arquitetura de três camadas, a máquina cliente age meramente como um *front-end* e não contém quaisquer chamadas de banco de dados diretas.

1.7. UML e a Engenharia de Requisitos

"A UML é uma linguagem visual para modelar sistemas orientados a objetos. Isso quer dizer que é uma linguagem que define elementos gráficos (visuais) que podem ser utilizados na modelagem de sistemas" (BEZERRA, 2007, p.15).

Segundo Pfleeger (2004, p.220):

> A UML *(Unified Modeling Language)* é uma abordagem de notação, muito utilizada para descrever soluções orientadas a objetos. Ela pode ser adaptada para se adequar a diferentes situações de desenvolvimento e ciclos de vida de *software*.

Desde a sua aprovação em 1997 pela OMG[10], a UML tem tido grande aceitação pela comunidade de desenvolvedores de sistemas. A sua definição conta com diversos colaboradores da área comercial como Digital, HP, IBM, Oracle, Microsoft, Unisys, IntelliCorp, i-Logix e Rational. Já houveram várias atualizações, desde o surgimento, afim de torná-la mais clara e útil (BEZERRA, 2007, p.15).

Segundo Bezerra (2007, p.16) "A UML é independente tanto de linguagens de programação quanto de processos de desenvolvimento. Isso quer dizer que a UML pode ser utilizada para a modelagem de sistemas [...]", ainda acrescenta que:

> Cada elemento gráfico da UML possui uma sintaxe e uma semântica. A sintaxe de um elemento corresponde à forma predeterminada de desenhar o elemento. A semântica define o que significa o elemento e com que objetivo ele deve ser utilizado.

Pfleeger (2004, p.220), acresce que ela pode ser utilizada para visualizar, especificar ou documentar um problema, a UML é útil principalmente para descrever diferentes projetos alternativos e, eventualmente documentar artefatos do projeto.

[10] "O OMG é um consórcio internacional de empresa que define e ratifica padrões na área da orientação a objetos" (BEZERRA, 2007, p.15).

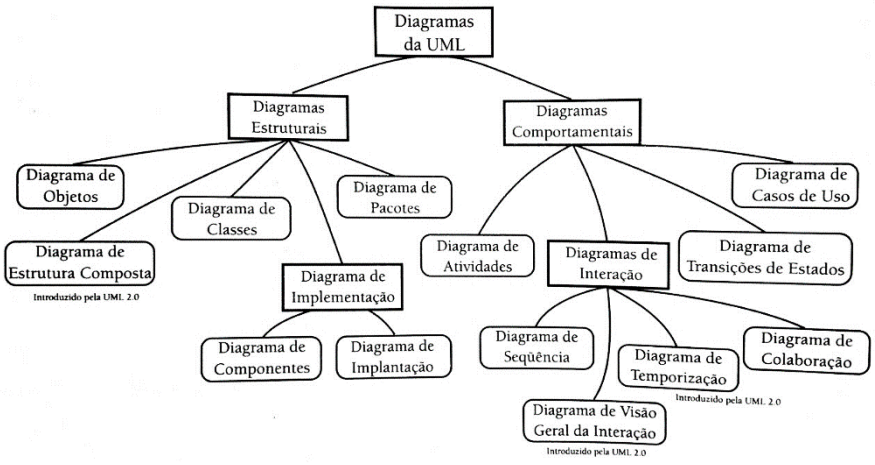
1.7.1. Diagramas UML

"Um processo de desenvolvimento que utilize a UML como linguagem de suporte à modelagem envolve à criação de diversos documentos. Esses documentos podem ser textuais ou gráficos" (BEZERRA, 2007, p.17).

Segundo Pfleeger (2004, p.220), os diagramas em UML incluem a visão dinâmica, representada com os casos de uso, listas de atividades, diagramas de interação e as máquinas de estado. E a visão estática do sistema, retratadas pelos diagramas de classes (onde mostram as relações de associação, generalização, dependência e realização) e a extensibilidade (restrições e estereótipos). Os diagramas em UML incluem também restrições e formalização.

Em geral, esses diagramas propostos pela UML, conforme ilustrado na Figura 7, descrevem a estrutura, o limite e o comportamento do sistema e os objetos contidos nele. Bezerra (2007, p.16,18) afirma que a modelagem de sistemas fornece uma perspectiva parcial do sistema, permitindo que os desenvolvedores tenham a possibilidade de entender e estudar o *software* a partir de várias perspectivas.

Figura 7 - Diagramas definidos pela UML



Fonte: BEZERRA, 2007, p.18

1.7.1.1. Diagrama de Caso de Uso

O diagrama de Caso de Uso (DCU) corresponde a uma visão externa de alto nível do sistema, com o objetivo de ilustrara as funcionalidades do sistema, ele

representa graficamente os atores, os casos de uso e relacionamentos entre esses elementos (BEZERRA, 2007, p.70).

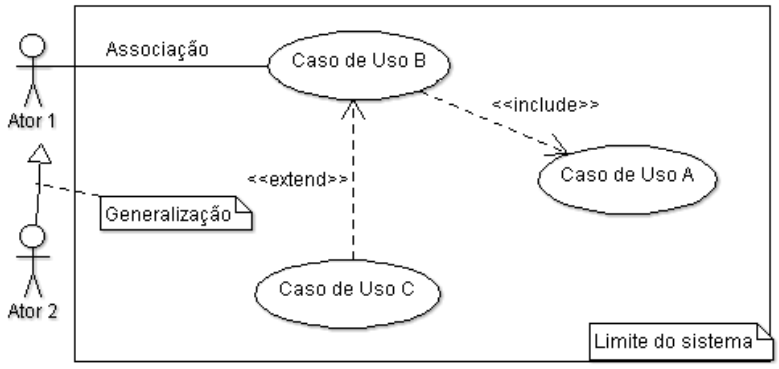
Para Pfleeger (2004, p.216) :

> Um caso de uso descreve a funcionalidade específica que um sistema, supostamente, deve desempenhar ou exibir, por meio da modelagem do diálogo que um usuário, um sistema externo ou outra entidade terá com o sistema a ser desenvolvido.

Segundo Siberschatz, Korth e Sudarshan (2006, p.168), “os diagramas de caso de uso mostram a interação entre os usuários e o sistema, em especial, as etapas das tarefas que os usuários realizam [...]”.

A notação utilizada para os atores no DCU é a figura de um boneco, com o nome do ator logo abaixo. Cada caso de uso é representado por uma elipse, com o nome no centro. Um relacionamento de comunicação é representado por um segmento de reta ligando ator e caso de uso (BEZERRA, 2007, p.70). A Figura 8, ilustra com clareza a notação do DCU.

Figura 8 - Notação do DCU



Fonte: Do próprio autor.

1.7.1.2. Diagrama de Atividades

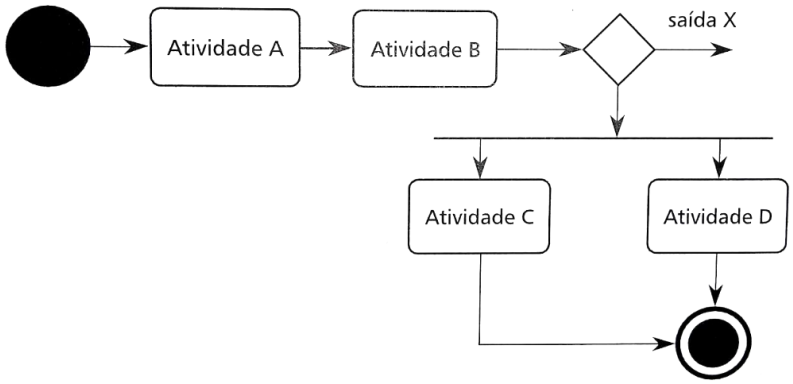
“O diagrama de Atividades mostra todas as atividades que podem ocorrer no sistema à medida que os valores de um objeto[11] são modificados” (PFLEEGER, 2004, p.221).

[11] “Um objeto é uma entidade que possui um estado e um conjunto definido de operações definidas para funcionar nesse estado. O estado é representado como um conjunto de atributos de objeto.” (SOMMERVILLE, 2007, p.210)

De acordo com Bezerra (2007, p.307), “um diagrama de atividade é um tipo especial de diagrama de estados[12], em que são representados os estados de uma atividade, em vez dos estados de um objeto.”

A UML utiliza esse diagrama para modelar o fluxo de procedimento ou atividades em uma classe. Quando as condições são utilizadas para decidir quais atividades solicitar, o diagrama de atividades utiliza um nó de decisão para representar as opções. A notação, conforme ilustrada na Figura 9, define um nó inicial representado com um ponto preto, e o nó final com um ponto preto menor dentro de um ponto branco. Um retângulo representa um estado, com uma seta mostrando as transições de um estado para o outro. A longa barra horizontal indica que uma mensagem de uma atividade pode ser transmitida para outras atividades (PFLEEGER, 2004, p.252).

Figura 9- Notação do diagrama de atividade



Fonte: PFLEEGER, 2004, p.232

1.7.1.3. Diagrama de Sequência

“Um diagrama de sequência mostra a sequência em que as atividades ou os comportamentos ocorrem” (PFLEEGER, 2004, p.229).

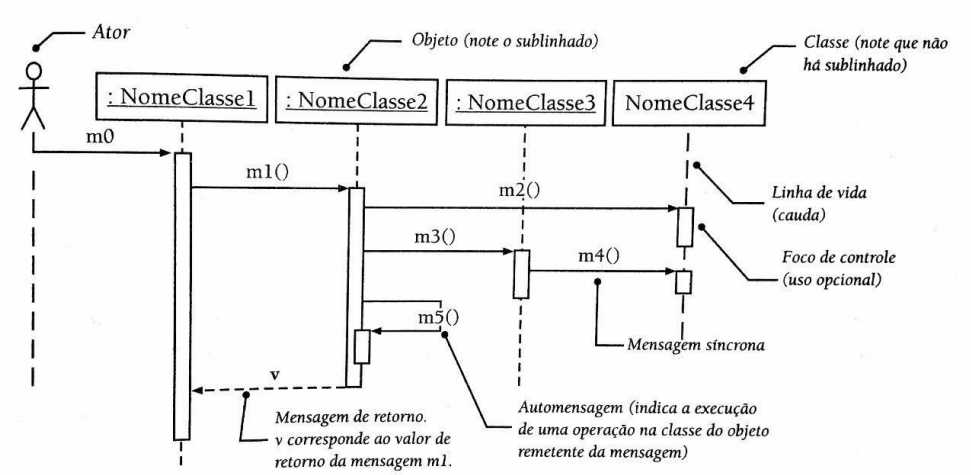
Ainda segundo Pfleeger, o objetivo é apresentar as interações entre os objetos na ordem temporal em que acontece, através de elementos gráficos. “Um

[12] Diagrama de estados: “representa os estados ativos de cada classe e os eventos (disparos) que causam mudanças entre esses estados ativos.” (PRESSMAN, 2006, p.179)

objeto é retratado como uma caixa no topo de uma linha vertical, conhecida como linha de vida". Bezerra (2004, p.221) diz que uma linha de vida é composta de duas partes: a cabeça e a cauda que corresponde a uma linha vertical tracejada.

O diagrama de sequência indica como eventos provocam transições de objeto para objeto, em resumo essa é uma versão abreviada do caso de uso, ele representa classes-chave e os eventos que fazem o comportamento fluir de classe para classe (PRESSMAN, 2006, p.179).

Figura 10- Notação do diagrama de sequência



Fonte: BEZERRA, 2004, p.183

Pfleeger (2004, p.229), descreve a notação do diagrama de sequência, ilustrado na Figura 10, da seguinte forma:

> Uma caixa estreita na linha de vida indica o início ou fim da mensagem. Uma seta entre duas linhas de vida representa uma mensagem entre os dois objetos, e é rotulada com o nome da mensagem e, algumas vezes, com a condição que deve ser satisfeita para que a mensagem seja enviada. Um asterisco na seta indica que a mensagem é enviada várias vezes, para diferentes objetos receptores. Quando a seta da mensagem volta para a caixa inicial no mesmo objeto, isso significa que o objeto está enviando uma mensagem para si mesmo. Esse tipo de mensagem é chamada de autodelegação.

1.8. Ferramentas de Modelagem

As ferramentas de modelagens, de acordo com Oliveira, Souza e Figueiredo (p.1), existem para facilitar a construção de diagramas, projetos, interfaces e toda e qualquer modelagem visual, afim de orientar e disciplinar o processo de desenvolvimento de software durante a fase de projeto.

Algumas ferramentas são de *software* livre, gratuitas para download como Argo UML e o *My*SQL *Workbench,* diferentemente de outras como o *Visio* e o *Balsamiq Mockup.*

1.8.1. Argo UML

Uma das principais ferramentas de modelagem UML de *open source* (código aberto). É executado em qualquer plataforma Java e está disponível em dez idiomas. Capaz de criar nove diagramas: classe, estado, atividade, caso de uso, interação, distribuição e sequências. Inclui suporte para todos os diagramas a partir da versão UML 1.4 padrão.[13]

1.8.2. Visio

A ferramenta *Visio* comercializada pela *Microsoft,* sendo incorporada ao pacote *Office*. Experiente em modelagem de diagramas, modelagem de processos e visualização de dados. Na modelagem de diagramas, o *Visio* auxilia usando formas e modelos modernos na criação de fluxogramas, diagramas de rede, organogramas, plantas baixas, projetos de engenharia e muitos outros, sendo usando a notação UML ou qualquer outra. Sua infinidade de recursos e simplicidade de uso, mostra o que o Visio tanto em diagramação pessoal como avançada, é uma excelente escolha para

[13] Disponível em < http://argouml.tigris.org/>

modelagens de qualidade. O conjunto de modelos disponíveis facilitam a modelagem e simplificam esforços.[14]

O *Visio Online* também tem sido um ótima opção para usuários que queriam ampliar os recursos do *Visio* podendo compartilhar e acessar diagramas de qualquer lugar. [15]

1.8.3. MySQL Workbench

O *My*SQL *Workbench* é uma ferramenta com uma gama de recursos desde modelagem de dados, desenvolvimento SQL até administração de usuários, *backup*, painel de desempenho visual, migração de banco de dados e muito mais. Ela está disponível para o *Windows*, Linux e Mac OS.[16]

Como modelagem de banco de dados, o *My*SQL *Workbench* permite que o usuário, projete, modele, gere e gerencie banco de dados. Permite o *design* de banco de dados orientado por modelos, fornece recurso para engenharia reversa, gerenciamento de mudanças de banco do dados e a documentação do projeto de banco de dados.[17]

1.8.4. Balsamiq Mockups

O *Balsamiq* é uma ferramenta rápida e inteligente de design de interface, também conhecida como *wireframes*. Disponível nas plataformas *Windows*, Linux e Mac. Um diferencial da ferramenta são os designes do estilo esboço que foca no conteúdo e na interação, e não nos minunciosos detalhes. Ela dispõe de uma biblioteca de *interface* de usuário com uma grande variedade de elementos para a criação. A ferramenta se tornou popularmente conhecida pela prototipagem rápida e intuitiva e foi criada para que qualquer pessoa consiga projetar *interfaces*, programas e *sites* com qualidade. Agora a nova solução do aplicativo Web *Balsamiq Cloud*, vem tomando espaço através dos seus recursos novos de controle de versões, usuários

[14] Disponível em < https://products.office.com/pt-br/visio/online-diagram-business-solutions?&tab=tabs-1>
[15] Disponível em <https://products.office.com/pt-br/visio/visio-online>
[16] Disponível em <https://www.mysql.com/products/workbench/>
[17] Disponível em <https://www.mysql.com/products/workbench/design/>

colaborativos, compartilhamento de projetos, chat entre membros do projeto e muito mais. [18]

[18] Disponível em <https://balsamiq.com/>

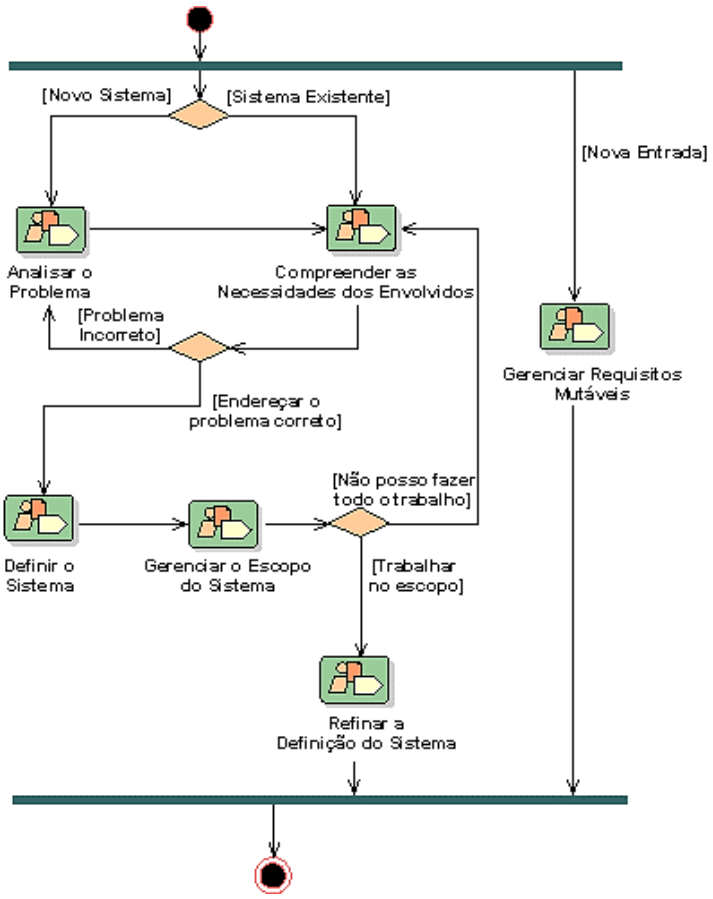
1.9. RUP e a Engenharia de Requisitos

A Engenharia de Requisitos é tratada em RUP através da disciplina Requisitos, que explica como transformar as necessidades das partes interessadas em requisitos que serão usados para criar o sistema. Os objetivos dessa disciplina são:

> - Estabelecer e manter um acordo com os clientes e outros *stakeholders* sobre o que o sistema deve fazer.
> - Fornecer aos desenvolvedores do sistema, um melhor entendimento dos requisitos do sistema.
> - Definir os limites do sistema: o que faz parte do sistema e o que não faz.
> - Fornecer uma base para o planejamento do conteúdo técnico das iterações do processo de desenvolvimento.
> - Fornecer uma base para a estimativa de custo e de tempo para o desenvolvimento do sistema.
> - Definir uma *interface* com o usuário do sistema, com o foco nas necessidades e objetivos dos usuários
>
> (IBM *RATIONAL CORPORATION*, 2003, apud CINTRA, 2006, p.81).

A disciplina de Requisitos do RUP segue o fluxo de trabalho ilustrado na Figura 11:

Figura 11 – Fluxo de Trabalho de Requisitos



Fonte: IBM *RATIONAL CORPORATION*, 2003, apud CINTRA, 2006, p.82.

O fluxo de trabalho como demonstrado no diagrama possui os seguintes subfluxos:

- Analisar o problema;
- Compreender as necessidades dos *stakeholders*;
- Definir o sistema;
- Gerenciar o escopo do sistema;
- Refinar a definição do sistema;
- Gerenciar mudanças nos requisitos.

Cada um dos subfluxos tem objetivos específicos e para que possam alcançar esses objetivos os subfluxos contém atividades para ser executadas, podendo uma atividade ter diferentes objetivos, dependendo do status do projeto (CINTRA, 2006, p.82).

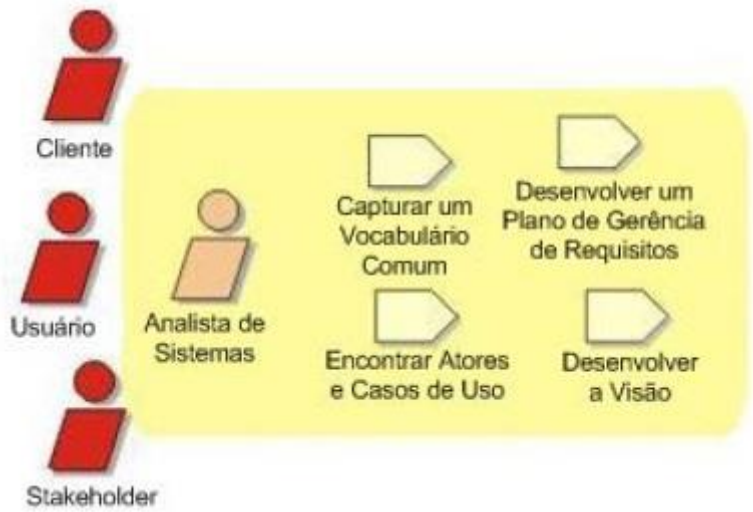
1.9.1. Analisar o Problema

O objetivo de forma geral do subfluxo analisar o problema, segundo Didier (2003, p.67), é ter a certeza que todos os envolvidos compreenderam e concordaram com o problema que está se tentando solucionar com o novo sistema, devendo ser identificados os *stakeholders*, definidos os limites do sistema e identificadas as restrições a ele impostas.

Contudo, Cintra (2006, p.83) acrescenta minuciosamente que tais objetivos são:

> • **Obter um acordo quanto ao problema que está sendo resolvido e suas causas**: Ou seja, certificar-se de que todas as partes interessadas concordam quanto ao problema a ser resolvido. Este objetivo também se preocupa em desenvolver uma terminologia comum para utilização ao longo da documentação do sistema.
> • **Identificar os *stakeholders***: Conforme descrito nas seções de definição deste trabalho, identificar os *stakeholders* é uma tarefa essencial no desenvolvimento de uma solução efetiva.
> • **Definir os limites do sistema**: Os limites do sistema são a fronteira entre o que faz parte da solução e o que está ao redor da mesma, ou seja, entidades externas que interagem com o sistema.
> • **Identificar restrições impostas ao sistema:** Conforme mencionado anteriormente neste trabalho, restrições limitam o grau de liberdade que se tem no desenvolvimento de uma solução. As restrições podem ser de ordem econômica, política, técnica ou ambiental e podem estar relacionadas a prazos, recursos ou orçamento.

Figura 12- Analisar Problema



Fonte: IBM *RATIONAL CORPORATION*, 2003, apud CINTRA, 2006, p.83

As atividades envolvidas para alcançar os objetivos do subfluxo, como ilustradas na Figura 12, são:

> • **Capturar um Vocabulário Comum:** definir um vocabulário comum que possa ser utilizado em todas as descrições textuais do sistema, especialmente na descrição de casos de uso.
> • **Desenvolver o Plano de Gerência de Requisitos:** desenvolver um plano para a documentação de requisitos, seus atributos e orientações para rastreabilidade e gerência de requisitos do produto. Este plano especifica mecanismos de informação e controle que serão coletados e utilizados para medir, relatar e controlar mudanças nos requisitos do produto.
> • **Encontrar Atores e Casos de Uso:** delinear a funcionalidade do sistema, utilizando a modelagem de casos de uso; definir o que será tratado pelo sistema e o que será tratado fora do mesmo; definir quem (pessoas) e o que (sistemas, dispositivos) estará interagindo com o sistema; dividir o modelo em pacotes com atores e casos de uso; criar diagramas de modelos de casos de uso; desenvolver a descrição inicial dos casos de uso do modelo.
> • **Desenvolver a Visão:** obter um acordo sobre que problemas precisam ser resolvidos; identificar os *stakeholders* do sistema; definir os limites do sistema; descrever as principais características do sistema
> (CINTRA, 2006, p.83-84).

1.9.2. Compreender as Necessidades dos Stakeholders

Obter as informações de todos os *stakeholders* do projeto é o principal objetivo desse subfluxo. Compreender as necessidades dos *stakeholders* é necessário para definir as características do sistema, e essas, por sua vez, são registradas no documento Visão. As necessidades dos *stakeholders* podem ser representadas por regras de negócio, requisitos de melhoria, listas de requisições. O levantamento dessas informações podem ser feitas de formas variadas, desde entrevistas, recebimentos de documentos produzidos pelos *stakeholders*, como também optar por outras técnicas de elicitação de requisitos (CINTRA, 2006, p.84).

Figura 13 - Compreender as Necessidades dos *Stakeholders*

Fonte: IBM *RATIONAL CORPORATION*, 2003, apud CINTRA, 2006, p.84

A Figura 13 mostra as atividades envolvidas para alcançar os objetivos desse subfluxo, que segundo Cintra (2006, p.85) são:

> • **Capturar um Vocabulário Comum:** continuar a definição de um vocabulário comum que possa ser utilizado em todas as descrições textuais do sistema, incluindo conceitos aprendidos ao longo das demais atividades do projeto.
> • **Elicitar os Requisitos dos *Stakeholders*:** compreender quem são os *stakeholders* do projeto e quais são seus pontos de vista; listar os requisitos e necessidades dos *stakeholders* e as consequentes características que o sistema deve apresentar; priorizar estes requisitos.
> • **Desenvolver a Visão:** complementar e refinar as principais características do sistema, definidas no sub-fluxo Analisar o Problema.
> • **Encontrar Atores e Casos de Uso:** continuar a procura por atores e casos de uso; refinar os modelos já criados complementando e melhorando descrições de atores e principalmente a descrição inicial dos fluxos de execução de casos de uso; ajustar interações entre estes elementos, de acordo com a compreensão atual do sistema (que fica mais elaborada ao longo de cada fase do projeto).
> • **Gerenciar Dependências:** gerenciar o escopo do projeto e as mudanças nos requisitos; esta gerência é feita com base em atributos especiais dos requisitos (importância, complexidade, tamanho, etc.) e também leva em conta a rastreabilidade entre os requisitos, ou seja, a influência que cada requisitos tem sobre os demais.

1.9.3. Definir o Sistema

Os principais objetivos desse subfluxo é alinhar o time de desenvolvimento como um todo, no entendimento do problema, realizar uma análise de alto nível sobre os requisitos colhidos (Visão) e seus atributos, refinar a visão do sistema e refinar o modelo de casos de uso, como também a especificação dos casos de uso e por fim, avaliar resultados parciais (DIDIER, 2003, p.60).

Figura 14 - Definir o sistema

Fonte: IBM *RATIONAL CORPORATION*, 2003, apud CINTRA, 2006, p.85

Percebe-se que as atividades ilustradas na Figura 14 são as atividades envolvidas para alcançar os objetivos do subfluxo definir sistema, conforme mostra Cintra (2006, p.86):

> • **Desenvolver a Visão:** complementar e refinar as características do sistema documentadas até a execução deste subfluxo.
> • **Gerenciar Dependências:** gerenciar o escopo do projeto e as mudanças nos requisitos; esta gerência é feita com base em atributos especiais dos requisitos (importância, complexidade, tamanho, etc.) e também leva em conta a rastreabilidade entre os requisitos, ou seja, a influência que cada requisitos tem sobre os demais.
> • **Capturar um Vocabulário Comum:** continuar a definição de um vocabulário comum que possa ser utilizado em todas as descrições textuais do sistema, incluindo conceitos aprendidos ao longo das demais atividades do projeto.
> • **Encontrar Atores e Casos de Uso:** continuar a procura por atores e casos de uso; refinar os modelos já criados complementando e melhorando descrições de atores e principalmente de casos de uso; ajustar interações entre estes elementos, de acordo com a compreensão atual do sistema (que fica mais elaborada ao longo de cada fase do projeto). O foco principal desta atividade dentro do sub-fluxo. Definir o Sistema é descrever os fluxos de execução dos casos de uso.

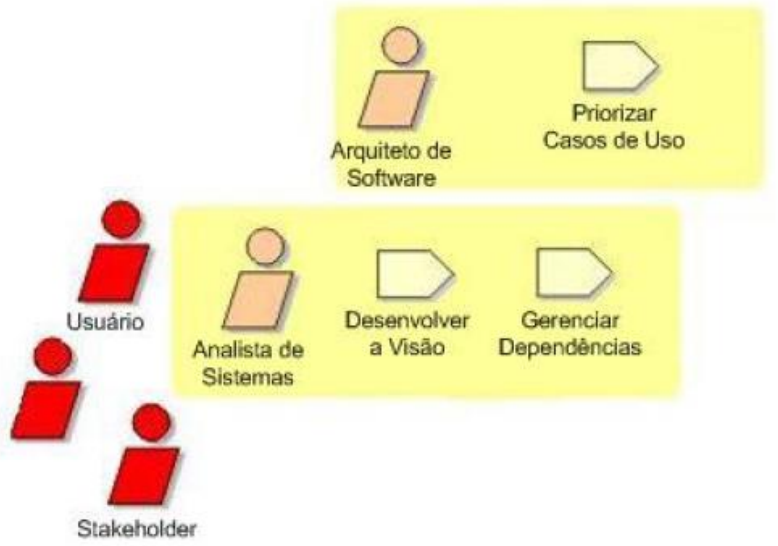
1.9.4. Gerenciar o Escopo do Sistema

O subfluxo gerenciar o escopo do sistema tem como objetivo assegurar se os objetivos do projeto estão sendo alcançados, selecionar os requisitos a serem desenvolvidos em cada iteração, bem como a definição, o refinamento dos atributos de requisitos e a compreensão da rastreabilidade entre os requisitos (CINTRA, 2006, p.86-87).

Para Didier (2003, p.71):

> O escopo de um projeto é definido pelo conjunto de requisitos alocados a ele, bem como o conjunto de recursos disponíveis para o desenvolvimento (tempo, pessoas e finanças). Assim, gerenciar o escopo do projeto é uma atividade contínua que requer o desenvolvimento iterativo e incremental, onde o escopo é dividido em partes menores, tornando gerenciamento mais efetivo.

Figura 15 - Gerenciar o Escopo do Sistema



Fonte: IBM *RATIONAL CORPORATION*, 2003, apud CINTRA, 2006, p.86

As atividades envolvidas para alcançar os objetivos do subfluxo, conforme a Figura 15, são:

> • **Priorizar Casos de Uso:** definir os casos de uso e cenários que serão analisados em cada iteração; definir os principais casos de uso e cenários que são significativos do ponto de vista das funcionalidades do sistema, do ponto de vista da arquitetura e em contextos específicos como performance, adaptabilidade ou outro ponto importante em termos de restrições do sistema.
> • **Desenvolver a Visão:** complementar e refinar as características do sistema documentadas até a execução deste sub-fluxo. É importante destacar que mudanças feitas na documentação de Visão fora da etapa de Iniciação devem ser tratadas como requisições de mudança, ou seja, devem seguir o processo de verificação de impacto, incluindo atividades como Gerenciar Dependências e Gerenciar Requisições de Mudança.
> • **Gerenciar Dependências:** gerenciar o escopo do projeto e as mudanças nos requisitos; esta gerência é feita com base em atributos especiais dos requisitos (importância, complexidade, tamanho, etc.) e também leva em conta a rastreabilidade entre os requisitos, ou seja, a influência que cada requisitos tem sobre os demais (CINTRA, 2006, p.87).

1.9.5. Refinar a Definição do Sistema

Os objetivos desse subfluxo são “[...] detalhar os fluxos de eventos dos casos de uso, detalhar as especificações suplementares, descrever a especificação de requisitos de *software* e modelar o protótipo de interface” (DIDIER, 2003, p. 72). Cintra (2006, p.87) completa explicando que todos os objetivos levam ao refinamento da

solução descrita para o sistema, como também o detalhamento dos requisitos levantado inicialmente para que projeto seja iniciado.

Figura 16 - Refinar a definição do sistema

Fonte: IBM *RATIONAL CORPORATION*, 2003, apud CINTRA, 2006, p.87

Analisando a Figura 16, algumas atividades necessárias para alcançar os objetivos do subfluxo são:

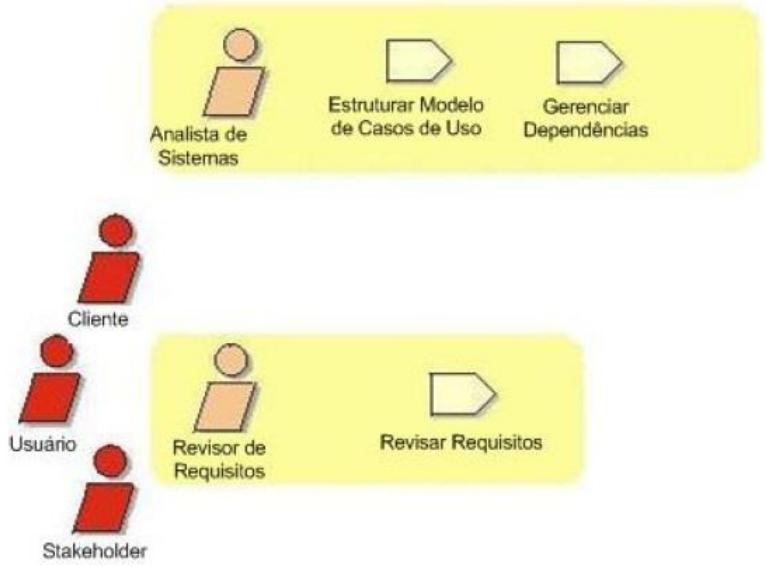
> • **Detalhar um Caso de Uso:** descrever detalhadamente um caso de uso que já tenha sido identificado em atividades anteriores. O caso de uso também já deve ter sido brevemente descrito e seus principais passos identificados. A descrição detalhada inclui uma explicação minuciosa do fluxo de execução (principais passos e iterações), pré e pós-condições, fluxos de erro e fluxos alternativos e demais informações relacionadas ao caso de uso. Esta descrição é utilizada para verificar que a equipe e o cliente compreendem o funcionamento desejado do sistema da mesma forma.
> • **Detalhar os Requisitos de *Software*:** recolher, detalhar e organizar o conjunto de artefatos que descreve, de forma completa, os requisitos de *software* do sistema. Os principais artefatos são: os atributos dos requisitos, as especificações suplementares (restrições que não se aplicam ao contexto específico de um caso de uso) e a especificação de requisitos de *software*, listando todos os requisitos do sistema (geralmente uma lista de casos de uso).
> • **Modelar a Interface com o Usuário:** construir um modelo de *interface* do sistema com o usuário, considerando os atores, os fluxos de eventos dos casos de uso e requisitos de usabilidade; identificar classes de *interface* (*boundary classes*) entre o sistema e seus usuários; descrever a interação entre essas classes e complementar os demais modelos do sistema a partir das novas definições feitas.
> • **Prototipar a Interface com o Usuário:** criar um protótipo de *interface* do sistema com o usuário, partindo do projeto de elementos da *interface* (ex: janelas, páginas web) até sua implementação; avaliar os resultados com os usuários (CINTRA, 2006, p.88).

1.9.6. Gerenciar Mudanças nos Requisitos

Durante o processo as mudanças das necessidades dos *stakeholders* são previstas e estas estão relacionadas a diversos fatores, como novas regras de negócio ou, até mesmo, uma melhor compreensão do sistema com o amadurecimento do projeto. Pode-se dizer que os principais objetivos deste subfluxo, segundo Didier (2003, p.73), são:

> [...] avaliar formalmente as solicitações de mudança e determinar seu impacto no conjunto de requisitos existentes, estruturar o modelo de casos de uso, estabelecer os atributos dos requisitos e definir rastreabilidade, e verificar se os requisitos definidos estão de acordo com a visão que os usuários têm do sistema.

Figura 17- Gerenciar Mudanças nos Requisitos



Fonte: IBM *RATIONAL CORPORATION*, 2003, apud CINTRA, 2006, p.87

Como demonstra na Figura 17, as atividades envolvidas para alcançar os objetivos do subfluxo, segundo Cintra (2006, p.89-90) são:

> • **Estruturar o Modelo de Casos de Uso:** refinar o modelo de casos de uso, extraindo comportamento comum para casos de uso abstratos e definindo relações "*include*" ou "*extend*"; procurar novos atores abstratos que definam papéis compartilhados por vários atores.
> • **Gerenciar Dependências:** gerenciar o escopo do projeto e as mudanças nos requisitos; esta gerência é feita com base em atributos especiais dos requisitos (importância, complexidade, tamanho, etc.) e também leva em conta a rastreabilidade entre os requisitos, ou seja, a influência que cada requisitos tem sobre os demais.
> • **Revisar os Requisitos:** verificar formalmente que os resultados da disciplina de Requisitos estão de acordo com as expectativas do cliente; revisar o estado atual dos requisitos e as requisições de mudança.

2. DESENVOLVIMENTO

2.1. Criação do Documento de Especificação de Requisitos segundo o RUP para o Restaurante Monalisa

A análise de requisitos dá uma sustentação para qualquer processo de desenvolvimento, e é um fator característico para o sucesso do mesmo.

O estudo realizado para a empresa do ramo alimentício foi baseado nos subfluxos apresentados pela metodologia do RUP, estruturado dentro do fluxo requisitos onde se concentra o foco desta pesquisa cientifica. O RUP explicita com clareza sua metodologia, disposta através de gráficos e diagramas facilitando o trabalho do analista de sistemas em entender seus processos.

O Documento de Especificação de Requisitos (ERS) estrutura-se em descrever a proposta e o escopo do sistema, incluindo a descrição das ferramentas e tecnologias utilizadas. Dispõe ainda da descrição geral do sistema, proporcionando uma visão global, para que se tenha uma perspectiva do produto tanto no seu funcionamento externo como interno. Em seguida uma série de diagramas, especifica o comportamento e a interação do sistema, minuciando o funcionamento do mesmo.

O documento completo pode ser encontrado no apêndice 2.

2.1.1. Subfluxo: Analisar o Problema

Nesta fase foi feita uma tarefa criteriosa, de análise da problematização. Descobriu-se que a má gestão dos clientes, era a consequência de um grande fluxo do mesmo com a ausência de um sistema capaz de auxiliar na organização e gestão dos serviços do restaurante. Essa análise reafirmou que a criação de um *software* de qualidade melhoraria a qualidade do serviço prestado e geriria de forma eficaz os clientes da empresa. As principais características do sistema foram descritas, juntamente com suas restrições e limites. Ficou decidido com base na análise, a

criação de um sistema desktop, utilizando os conceitos de usabilidade[19]. Pelo fato de ser uma empresa de pequeno porte um sistema desktop seria suficiente, tendo como vantagem, a velocidade, o fato possuir mais recursos para melhorar a performance e uma melhor usabilidade para sistema.

Foram encontrados os atores para modelo de casos de uso, a princípio o funcionário e o administrador.

2.1.2. Subfluxo: Compreender as Necessidades dos Stakeholders

Fez-se necessário utilizar técnicas capazes de traduzir com clareza para que não houvesse um duplo entendimento. O *stakeholder* foi o proprietário do Restaurante. E para ter as informações necessárias do *stakeholder* foram feitas entrevistas estruturadas em funil[20] para a elicitação de requisitos.

A modelagem de casos de uso se iniciou já com a busca definitiva dos atores. As ferramentas utilizadas para a modelagem dos diagramas foram o Argo UML e o *Visio*.

Observou-se mais claramente não só a definição dos atores como também as funcionalidades principais do sistema, como gerir clientes, serviços e horários, controle de pagamentos através de dinheiro e cartão e emissão de relatórios. A linguagem de programação sugerida foi Java[21].

Todos esses requisitos foram reunidos e registrados no Documento de Especificação de Requisitos com base na metodologia do RUP.

2.1.3. Subfluxo: Definir o Sistema

Entendida a problematização, foram feitas análises dos requisitos colhidos, para apurar a compreensão do sistema. Com uma visão mais detalhada, os casos de uso foram criados, assim como os ajustes dos atores encontrados.

---

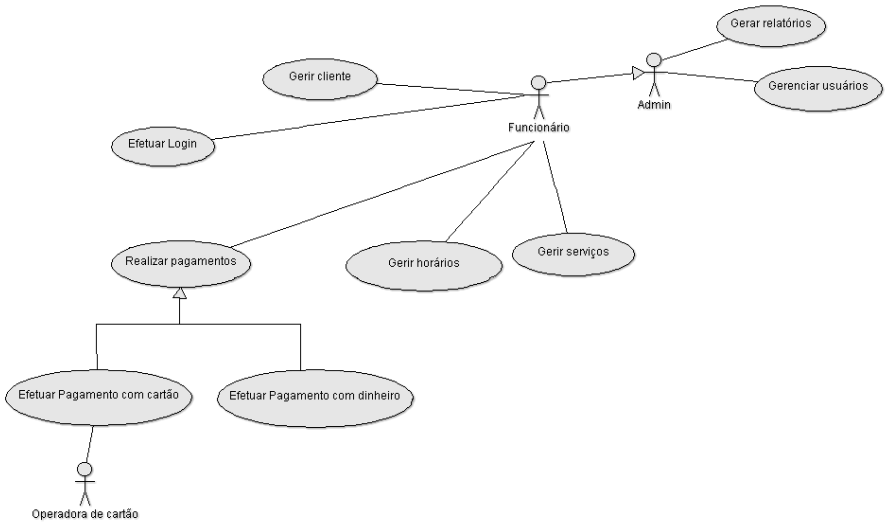
[19] "O software deve ser usável, sem esforço excessivo, pelo tipo de usuário para o qual ele foi projetado. Isso significa que ele deve apresentar uma interface com o usuário e documentação adequadas." (SOMMERVILLE, 2007, p. 9)

[20] Forma de estrutura de entrevista iniciada com perguntas mais genéricas acerca do sistema e são fechadas com perguntas mais específicas. Essa estrutura é geralmente utilizada por usuários que tem uma relação mais afetiva com o assunto (SPÍNOLA, 2007, p.50).

[21] "Linguagem de programação orientada a objetos projetada pela *Sun* para ser independente de plataforma." (SOMMERVILLE, 2006, p. 527)

Os atores são: o Funcionário, a Operadora de Cartão e o Administrador (Admin) que herda todas as funções do Funcionário e apresenta ainda funcionalidades específicas de gerenciar usuários e gerar relatórios. Os casos de uso representados por uma eclipse, são todas as funcionalidades presentes do sistema. A Figura 18 mostra o DCU definido.

Figura 18 – DCU do sistema



Fonte: Da própria autora

Com isso, a descrição textual, de todos os sete casos de uso foram feitos com a incumbência de detalhar as funcionalidades do sistema. O Quadro 2 exemplifica uma das descrições textuais feitas para o caso de uso ‘Gerir Cliente’. As demais descrições podem ser encontradas no apêndice 2.

Quadro 2– Descrição Textual Gerir Cliente

Gerir Cliente (CSU01)
Sumário: O funcionário irá manter clientes, podendo realizar consultas, cadastros e alterações de dados.
Ator primário: Funcionário
Pré-condições: O funcionário deve estar logado no sistema.
Fluxo Principal: 1. O sistema exibe o menu da tela principal com as funcionalidades específicas. 2. O funcionário seleciona a opção desejada [1], [2], [3]. 3. O caso de uso é encerrado.
Fluxo Alternativo (2A): Cadastrar Cliente [1] a. O funcionário seleciona a opção ‘Cadastros’. b. Em ‘Cadastros’ o funcionário seleciona a opção ‘Cadastro de Clientes’. c. O sistema exibe a tela ‘Cadastros de clientes’. d. O funcionário solicita os dados para ser preenchidos.

e. O sistema habilita o botão ‘Salvar’.
f. O funcionário salva os dados.
g. O sistema cadastra o cliente.
h. O caso continua a partir do passo 3

Fluxo Alternativo (2B): Atualizar Cliente [2]
a. O funcionário seleciona a opção ‘Clientes’
b. O sistema exibe a tela ‘Clientes’
c. O funcionário pesquisa pelo o nome, o cliente desejado.
d. Na tela, o sistema, exibe todas as informações do cliente.
e. O funcionário clica no botão ‘Editar’
f. Todos os campos são habilitados para edição, pelo sistema.
g. O funcionário seleciona e altera os dados preenchidos.
h. O sistema habilita a opção ‘Salvar’
i. O funcionário valida os dados.
j. O sistema atualiza os dados
k. O caso continua a partir do passo 3.

Fluxo Alternativo (2C): Inativar Cliente [3]
a. O funcionário seleciona a opção ‘Clientes’
b. O sistema exibe a tela ‘Clientes’
c. O funcionário pesquisa pelo o nome, o cliente desejado.
d. Na tela, o sistema, exibe todas as informações do cliente.
e. O funcionário clica no botão ‘Editar’
f. Todos os campos são habilitados para edição, pelo sistema.
g. O funcionário seleciona o campo ‘Situação’ e desmarca a função Ativo.
h. O sistema habilita a opção ‘Salvar’.
i. O funcionário clica em ‘Salvar’.
j. O sistema inativa o cliente.
k. O caso continua a partir do passo 3

Fluxo de exceção: No fluxo alternativo [1] e [2], os campos nome, celular, serviços e horário em branco
a. O sistema envia uma mensagem de advertência na tela.

Fluxo de exceção: No fluxo alternativo [1] não há horários cadastrados
a. Ir para o caso de uso: Gerir horário

Fluxo de exceção: No fluxo alternativo [1] e [2] limite de vagas por horários excedido
a. O sistema envia uma mensagem de advertência na tela.
b. Incluir outro horário disponível.

Pós-condições: O cliente foi cadastrado/atualizado/inativado com sucesso. O sistema manterá o usuário logado, exibindo a tela inicial do usuário. Os dados estarão atualizados nas tabelas do BD.

Histórico: Criado por Sálua Jawhari Duarte em 21/09/2017

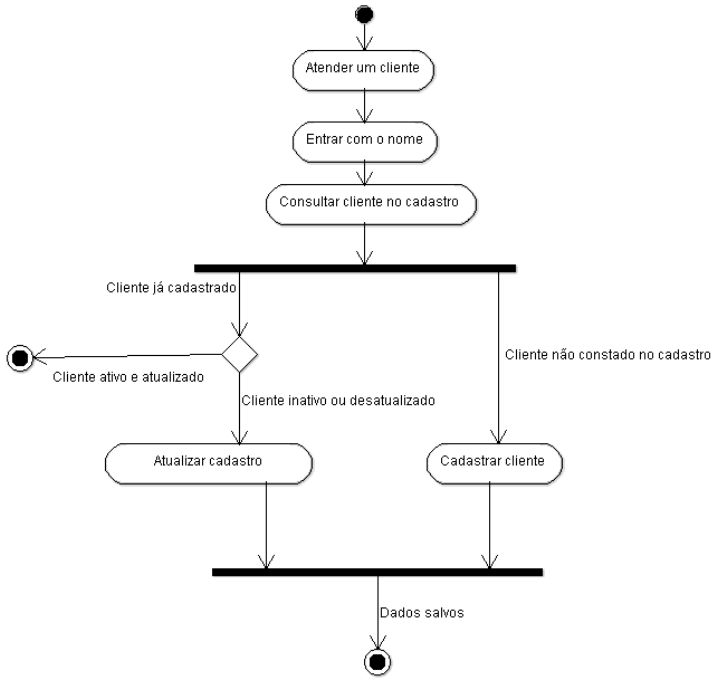
Fonte: Da própria autora

2.1.4. Subfluxo: Gerenciar o Escopo do Sistema

Após a definição da modelagem dos casos de uso, a descrição textual dos mesmos foi melhorada. Com o intuito de se obter o documento ERS segundo a proposta do RUP, outros diagramas, iniciados após a modelagem de casos de uso, fazem parte desse projeto. Sendo sete diagramas de atividades e quinze diagramas de sequência. Todos os diagramas são apresentados no documento ERS encontrado no apêndice 2.

O diagrama de atividades exemplificado na Figura 19 representa o fluxo de atividades do caso de uso “Gerir Cliente”. O fluxo começa no nó inicial em atender cliente. Após entrar com o nome e consultar o cadastro, se o cliente não estiver no cadastro, o ator cadastra o cliente e a atividade termina com os dados salvos. Caso o cliente estiver cadastrado e atualizado, a atividade termina, pelo contrário o cadastro deve ser atualizado e a atividade termina com os dados salvos.

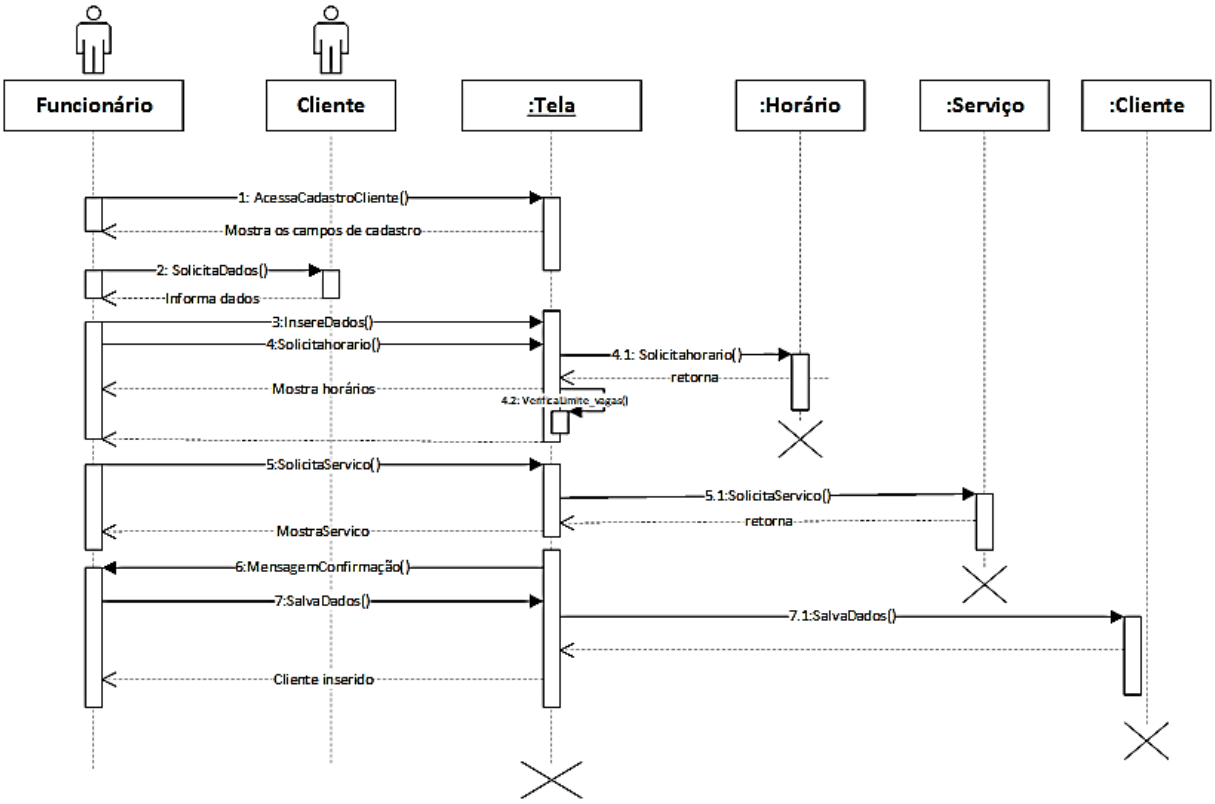
Figura 19 – Diagrama de atividades ‘Gerir Cliente’



Fonte: Da própria autora

No diagrama de sequência, mostrado pela Figura 20, evidencia a comunicação entre o ator funcionário e a tela,desde o pedido de Acesso ao Cadastro até a resposta de cliente inserido. A cada requisição, a tela tem liberdade de chamar as classes: Horário, Serviço e Cliente, para solicitar os dados e mostrar as informações ao usuário.

Figura 20 – Diagrama de sequência ‘Cadastro de Cliente’



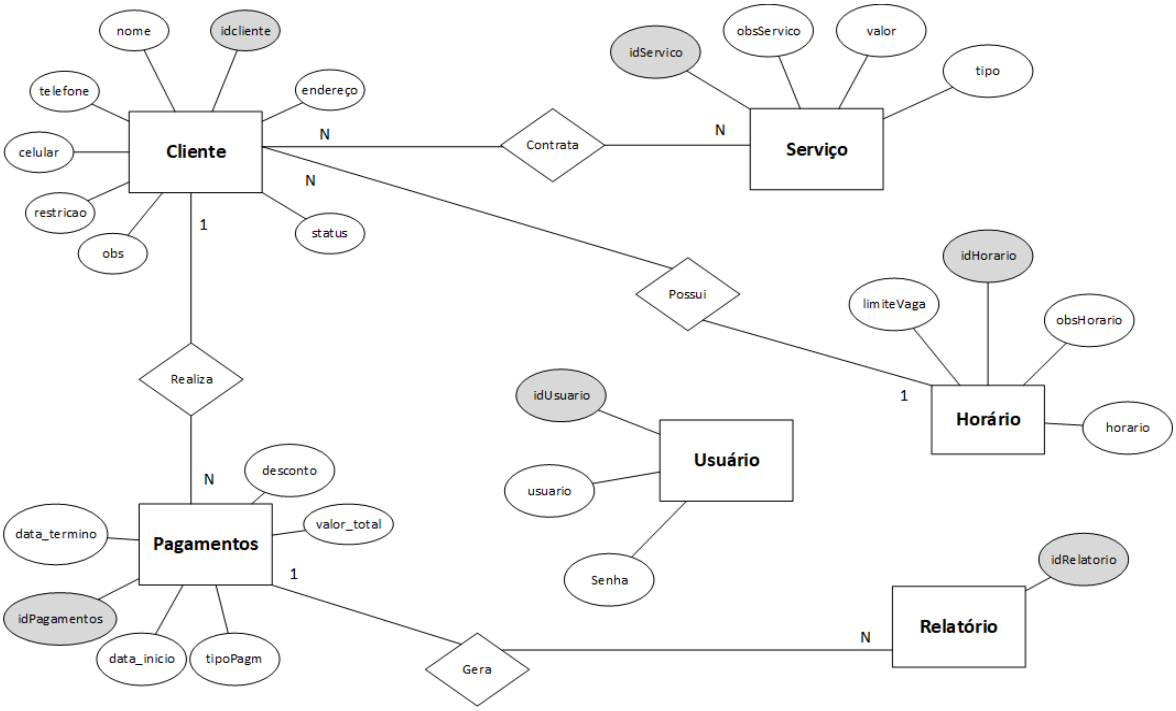
Fonte: Da própria autora

2.1.5. Subfluxo: Refinar a Definição do Sistema

O Especificador de Requisitos, uma das figuras chaves desta fase, foi o responsável por detalhar os Casos de Uso e os Requisitos de Software. Nesse subfluxo mais precisamente, a especificação de Requisitos de *Software* começou a surgir como artefato. Os demais diagramas, assim como a modelagem do banco de dados e o modelo entidade-relacionamento, foram definidos.

O diagrama E-R facilitou o projeto de banco de dados. A sua estrutura define as entidades, seus atributos e o relacionamento entre as entidades, incluindo a carnalidade de cada relação. A Figura 21 demonstra o diagrama E-R.

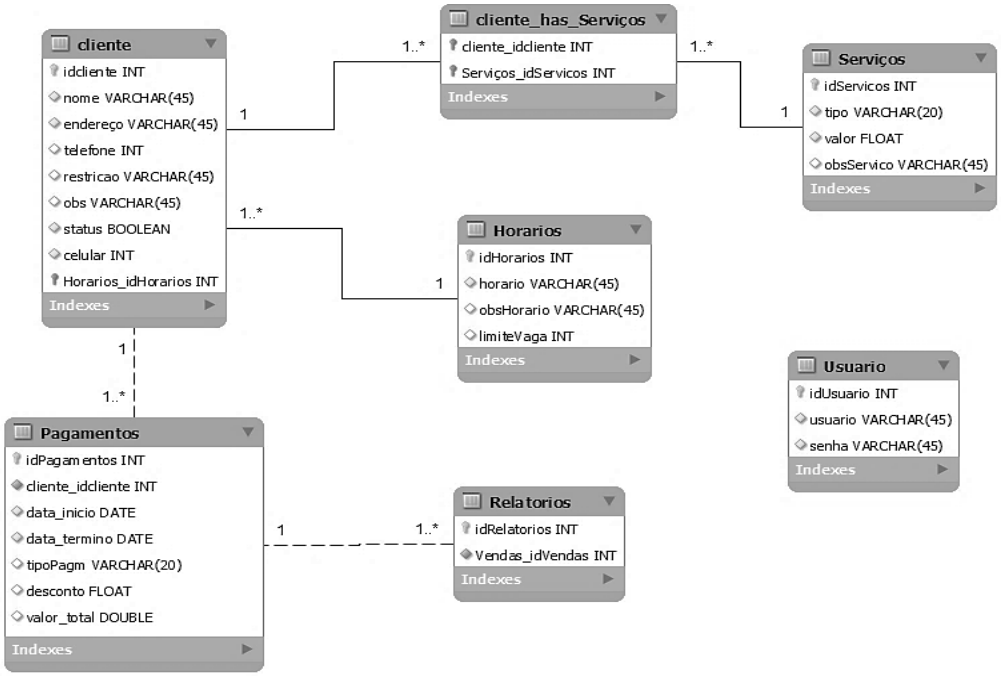
Figura 21- Diagrama E-R do sistema



Fonte: Da própria autora

As tabelas do banco de dados, exemplificada pela Figura 22, demonstra a modelagem de banco de Dados com base no diagrama E-R. A modelagem foi feita utilizando a ferramenta *My*SQL *Workbench.*

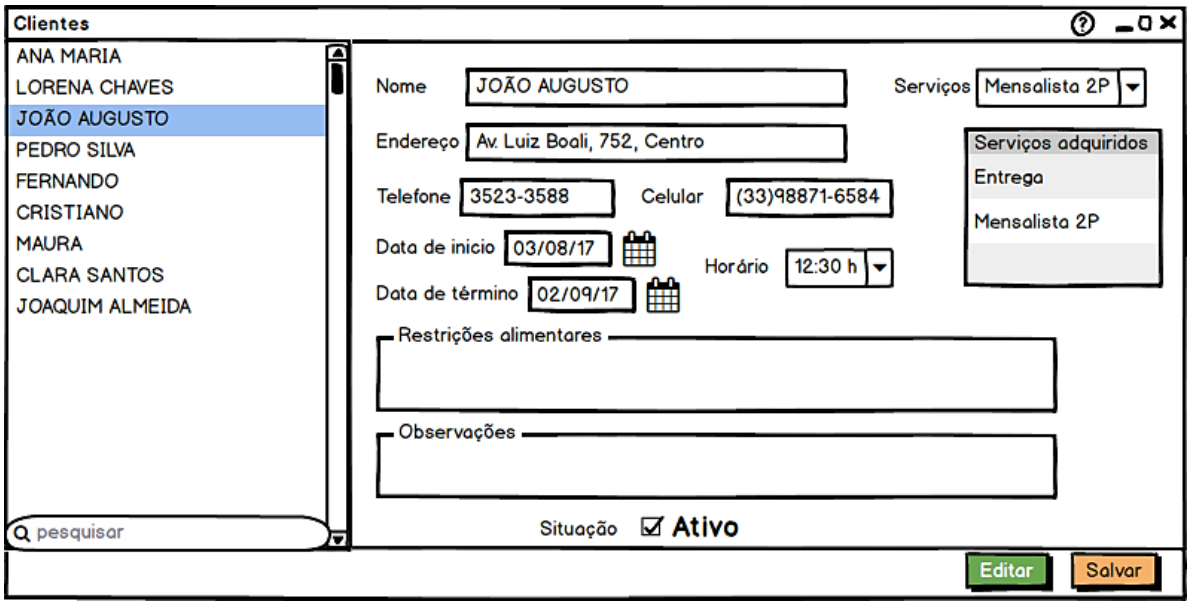
Figura 22 - Banco de dados do sistema



Fonte: Da própria autora

No mesmo instante, os protótipos das janelas do futuro sistema foram criados para ilustrar as necessidades do *stakeholder*, afim de obter um melhor entendimento do projeto proposto. Os protótipos foram criados através da ferramenta *Balsamiq Mockups,* um exemplo de uma interface criada é exemplificada pela Figura 23.

Figura 23 - Protótipo da tela Gerir Clientes



Fonte: Da própria autora

Na interface “Gerir Cliente”, é um exemplo dos onze protótipos de tela, disponíveis no apêndice 1. Na tela concentra-se todas as informações dos clientes auxiliando o usuário a ter uma visão global dos mesmos. Ao selecionar o cliente desejado, organizados na parte lateral esquerda, a tela mostra as informações dos serviços contratados, do horário e outras informações como a situação do cliente.

As telas foram criadas pensando no conceito de usabilidade, para que o usuário se familiarize com o ambiente.

2.1.6. Subfluxo: Gerenciar Requisições de Mudança

As mudanças são previstas e inevitáveis. Com o amadurecimento do projeto, surgem novos requisitos. No projeto houveram alterações em alguns requisitos, o que geraram alterações também em alguns diagramas, principalmente o diagrama de

casos de uso que norteia o projeto. Pelo RUP ser orientado por Caso de Uso, qualquer alteração no DCU influencia também nos demais diagramas.

As mudanças foram estudadas e analisadas antes de serem alteradas. A última mudança realizada foi a inclusão de mais uma funcionalidade no sistema em gerenciar usuários. Antes de alterar o projeto, houve uma análise para declarar se a mudança é pertinente ou não. Contada a real necessidade da funcionalidade, a mudança foi feita. Outras mudanças foram feitas no Banco de Dados, como o atributo 'limite de vagas' que foi acrescentado posteriormente.

Todas as modelagens foram devidamente estruturadas, juntamente com os requisitos especificados, para que estivessem de acordo com a expectativa do cliente.

Após os dados colhidos o documento de Requisitos foi elaborado e finalizado, a verificação do cliente para aprovação final dá o marco para início do processo de codificação e a conclusão dessa pesquisa.

2.1.7. Análise de Resultados

O documento foi construído discorrendo sobre as análises apresentadas, constituído de informações textuais e ilustrativas, através de gráficos e diagramas para que com minuciosidade reproduzisse as características do sistema.

Algumas dificuldades foram encontradas na elaboração, como as mudanças durante o processo, já que uma mudança pode acarretar várias outras.

Suprindo as expectativas, o documento mostrou ser realmente eficiente quanto ao seu objetivo, o seu nível de detalhes não se mostra exacerbado nem maçante ao leitor, mas suficiente para se ter um entendimento preciso do sistema.

No ponto de vista do *stakeholder*, o documento despertou o interesse em adquirir o sistema, dando-lhe maior clareza do futuro *software*. Após uma nova reunião, o documento foi validado e aprovado pelo cliente, dando-lhe a garantia que as funcionalidades estão pertinentes ao desejado, descobrindo que a ideia da informatização seria um solução viável para seu comércio.

Nos apêndices ao final da pesquisa encontram-se o ERS acompanhado da prototipação das interfaces.

CONSIDERAÇÕES FINAIS

A presente pesquisa apresentou as vantagens em se obter um documento de Especificação de Requisitos como sendo um grande aliado não só ao cliente mas também ao desenvolvedor na busca de um *software* de qualidade.

Para atingir o objetivo geral da pesquisa foram dispostas a incumbência de atingir aos objetivos específicos, idealizados pela seguinte análise:

Aprimorar os conhecimentos acerca da regra de negócio da empresa e das necessidades dos *stakeholders*:

De uma análise intensa da realidade da empresa, foi possível uma melhor análise do problema em busca de soluções que promovessem uma melhora no ambiente de atuação. As regras de negócios foram compreendidas e utilizadas na produção do ERS tendo a cautela de não ferir nenhum princípio da empresa e do cliente.

Avaliar o ambiente da empresa para possível aplicação da proposta.

O ambiente foi analisado e discutido para que se encontrasse a problematização e que tivesse um melhor entendimento dessa. Descobriu-se que a falta de um sistema que proporcionasse um controle para melhor gestão dos seus clientes impactava negativamente no atendimento e na qualidade do serviço prestado. Com essa análise foi constatada a viabilidade da proposta.

Realizar um levantamento de requisitos do software para a proposta de documentação do RUP:

Através de entrevistas os requisitos foram levantados para que a proposta fosse elaborada. Essas entrevistas foram importantes para explanar a compreensão das necessidades do cliente com a intensão de levar uma melhor proposta para suprir suas expectativas.

Criar o documento de Especificação de Requisitos:

O documento foi criado como sendo o artefato principal da monografia. Disposto de gráficos e diagramas para ilustrar as funcionalidades do futuro *software*. Tendo o cuidado de respeitar a metodologia RUP e o modelo oferecido pela IBM[22].

Verificar a especificação de forma a assegurar se todos os requisitos foram definidos:

Os requisitos sofreram alterações no decorrer do processo, isso se deu devido ao amadurecimento do projeto, que foi o motivo das verificações constantes dos requisitos para acompanhar a sua evolução.

Validar o documento para aprovação junto ao cliente:

O documento foi validado e aprovado pelo cliente, dando-lhe a garantia que as funcionalidades estão pertinentes ao desejado, o que proporcionou ter uma visão melhor do *software*, descobrindo que a ideia da informatização seria uma solução viável para seu comércio.

Haja vista a problemática da pesquisa, na possibilidade da criação de um documento de Especificação de Requisitos de *Software*, segundo o RUP, para melhorar a qualidade do *software* a ser desenvolvido para o Restaurante Monalisa, foram examinadas as hipóteses abordadas.

H0 – Seria inviável a criação de um Documento de Especificação de Requisitos de Software, seguindo os padrões da metodologia RUP aplicada ao Restaurante Monalisa, pois não haveria nenhuma melhora no entendimento do software a ser construído e consequentemente na sua qualidade.

De acordo com a pesquisa está hipótese foi constatada como nula, comprovada através da validação do cliente em ter tido uma melhor compreensão do *software*. O documento foi resultado de análises, verificações e correções até que se chegasse a um resultado de excelência, pensando na qualidade do *software* a ser produzido.

---

[22] Manual disponível em: <ttps://www.ibm.com/developerworks/community/wikis/form/api/wiki/W0dcb4f3d0760_48cd_9026_a90843b9da06/page/Support%20for%20reviews%20and%20rating>

H1 – A documentação de Requisitos proporcionaria ao cliente e ao desenvolvedor a possibilidade de analisar o futuro *software*, expondo dúvidas, opiniões e possíveis mudanças antes do processo de codificação.

Com a proposta de documentação, a análise do *software* foi bem detalhada dando oportunidade tanto ao cliente quanto ao analista de entender o *software*, antes desse ser construído, isso proporcionou ao cliente a facilidade em mudar e/ou alterar os requisitos sem maiores complicações. Portanto, pode-se ponderar essa hipótese como válida.

H2 - A configuração de mudanças conforme a Engenharia de Requisitos possibilitaria um controle de versões, afim de evitar mudanças futuras em excesso, reduzindo consequentemente os gastos.

Foram descobertas mudanças que sem a documentação só seriam detectadas durante a implementação do *software*, o que aumentaria os gastos do projeto e insatisfação do cliente. A configuração de mudanças controla através de *baselines* as modificações de forma que sejam acompanhadas de acordo com a evolução do projeto. Sendo assim a hipótese foi validada.

H3 – A proposta de documentação traria uma visão ampla do negócio auxiliando a tomada de decisão relacionada ao desenvolvimento, melhorando a produtividade, evitando o retrabalho e a perda de tempo no processo de desenvolvimento do *software*.

A hipótese foi validada por constatar que essa análise produzida antes da codificação ajuda o desenvolvedor a entender o sistema como um todo, evitando o trabalho de pausar a codificação devido a mudanças excessivas. A documentação serve como um manual para que o desenvolvedor transpasse do papel para a prática.

H4 – A documentação de requisitos facilitaria para o programador no desenvolvimento do *software* por conter informações que o orienta em todo processo, permitindo aos profissionais controle sobre processo de desenvolvimento, acerca dos custos, dos níveis de qualidade desejado e os prazos estimados.

O detalhamento das informações do futuro *software*, dispostas na documentação de requisitos traz aos desenvolvedores mais confiança em saber que uma análise de viabilidade e um estudo aprofundado do projeto já foram realizados, aumentando as chances de sucesso no desenvolvimento. A hipótese, portanto, foi validada.

H5 - A documentação de requisitos serviria como um contrato entre as ambas partes, resguardando o desenvolvedor e o cliente, inibindo futuras reclamações por estabelecer uma espécie de acordo entre os envolvidos.

A hipótese foi validada pela seguinte observação: a presença de um contrato garante segurança entre as partes, por ser um registro de um acordo mútuo. O cliente com a documentação em mãos, o precaveu de quaisquer complicações posteriores, dando-lhe a garantia que o *software* por ele aprovado na documentação, será o *software* implementado em sua empresa.

O Documento de Especificação de Requisitos produzido foi promissor, conseguindo portanto alcançar seus objetivos, entendendo que todo o esforço em aplicar os conceitos da Engenharia de Requisitos antes do processo de codificação traz inúmeros benefícios.

O modelo de Especificação de Software do RUP foi competente e proveitoso, porém a IBM[23] deve propor constantes atualizações para o modelo SRS, essas atualizações são precisas para que possa melhorar ainda mais seu resultado.

É importante ressaltar que o investimento necessário para a boa especificação de requisitos não representa custos supérfluos, mas sim indispensáveis para qualidade, produtividade, e diminuição de gastos excessivos do futuro *software*.

Ao término desse estudo, posteriormente, o cliente pretende ter em mãos o *software*, entendendo a necessidade deste para o bom funcionamento da gestão do Restaurante. Outras funcionalidades poderão ser acrescidas ao software como a gestão de vendas e a gestão financeira, podendo ocasionar em trabalhos futuros.

[23] IBM – Empresa que comercializa o RUP atualmente.

REFERÊNCIAS

BEZERRA, Eduardo. *Princípios de Análise e Projeto de Sistemas com UML.* 2ª ed. Rio de Janeiro: Elsevier, 2007.

CINTRA, Caroline Carbonell. *A implantação de um Processo de Engenharia de Requisitos Baseado no Processo Unificado da Rational (RUP) Alcançando Nível 3 de Maturidade da Integração de Modelos de Capacidade e Maturidade (CMMI) Incluindo a Utilização de Práticas de Métodos Ágeis*. 2006. 160 f. Dissertação mestrado – Universidade Federal do Rio Grande do Sul, Porto Alegre, 2006. Disponível em < https://www.lume.ufrgs.br/bitstream/handle/10183/8128/000568343.pdf?sequence=1 > Acesso em: 20 mai. 2017.

DIDIER, Ana Cláudia Veras Beltrão. *WRE-Process: Um processo de Engenharia de Requisitos Baseado no RUP.* 2003. 232 f. Pós graduação – Universidade Federal de Pernambuco, Recife, 2003. Disponível em < http://repositorio.ufpe.br/bitstream/handle/123456789/2477/arquivo4654_1.pdf?sequence=1&isAllowed=y> Acesso em: 18 ago. 2017.

ELMASRI, Ramez; NAVATHE, Shamkant B. *Sistemas de Banco de Dados.* 4ª ed. São Paulo: Addison Wesley, 2005.

FILHO, Wilson de Pádua Paula. *Engenharia de Software: Fundamentos, métodos e padrões.* 2ª ed. Rio de Janeiro: LTC Editora, 2003.

LAMOUNIER, Hudson. Disponível em <http://www.devmedia.com.br/atividades-basicas-ao-processo-de-desenvolvimento-de-software/5413#> Acesso em 15 mar. 2017.

MEDEIROS, Higor. Disponível em< http://www.devmedia.com.br/principios-da-engenharia-de-software/29630 > Acesso em 28 mar. 2017.

Modelo de Especificação de Software. Disponível em <https://www.cti.ufu.br/sites/cti.ufu.br/files/cti-especificacao-requisito-software-em%20conformidade-com-cmmi.pdf> Acesso em 03 abr. 2017.

OLIVEIRA, Johnatan Alves de; SOUZA, Priscila Pereira de; FIGUEIREDO, Eduardo. *Uma Avaliação de Ferramentas de Modelagem de Software.* Iniciação científica – Universidade Federal de Minas Gerais (UFMG), Belo Horizonte –MG. Disponível em < http://labsoft.dcc.ufmg.br/lib/exe/fetch.php?media=smes.pdf> Acesso em 24 nov. 2017.

PFLEEGER, Shari Lawrence. *Engenharia de Software: Teoria e Prática.* 2ª ed. São Paulo: Prentice Hall, 2004.

PRESSMAN, Roger S. *Engenharia de Software.* 6ª ed. São Paulo: McGraw-Hill, 2006.

SIBERSCHATZ, Abraham; KORTH, Henry F.; SUDARSHAN.S. *Sistema de Banco de Dados.* 5ª ed. Rio de Janeiro: Elsevier, 2006.

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

SPÍNOLA, Rodrigo Oliveira et al. *Qualidade de Software.* Engenharia de Software Magazine, s.l, Ano 1, p.46-52.28-37, v.1, n.1 ,s.d. 2007. Disponível em <http://sidneyvieira.kinghost.net/abas/disciplinas/download/ESI/es_alta.pdf> Acesso em: 03 abr. 2017.

Virtual Medical Home: *Software Requirement Specification*. 2009. Disponível em< https://www.ibm.com/developerworks/community/wikis/form/api/wiki/W0dcb4f3d0760_48cd_9026_a90843b9da06/page/Support%20for%20reviews%20and%20rating> Acesso em: 16 mai. 2017.

Disponível em<https://www.projectbuilder.com.br/blog-pb/entry/projetos/como-fazer-um-business-case-eficiente-do-seu-projeto> Acesso em 23 out. 2017.

Disponível em <https://balsamiq.com/> Acesso em 23 nov. 2017.

Disponível em < http://argouml.tigris.org/> Acesso em 23 nov. 2017.

Disponível em < https://products.office.com/pt-br/visio/online-diagram-business-solutions?&tab=tabs-1> Acesso em 23 nov. 2017.

Disponível em <https://products.office.com/pt-br/visio/visio-online> Acesso em 24 nov. 2017.

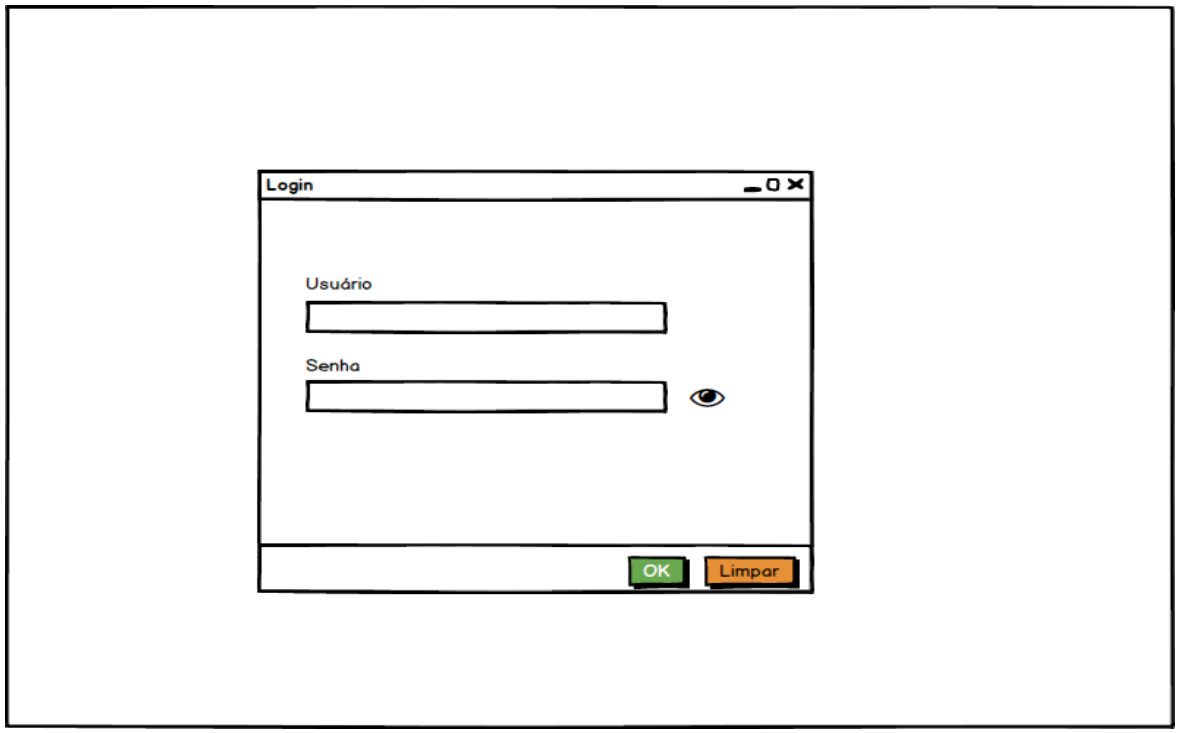
Disponível em <https://www.mysql.com/products/workbench/> Acesso em 23 nov. 2017.

Disponível em <https://www.mysql.com/products/workbench/design/> Acesso em 23 nov. 2017.

APÊNDICE 1

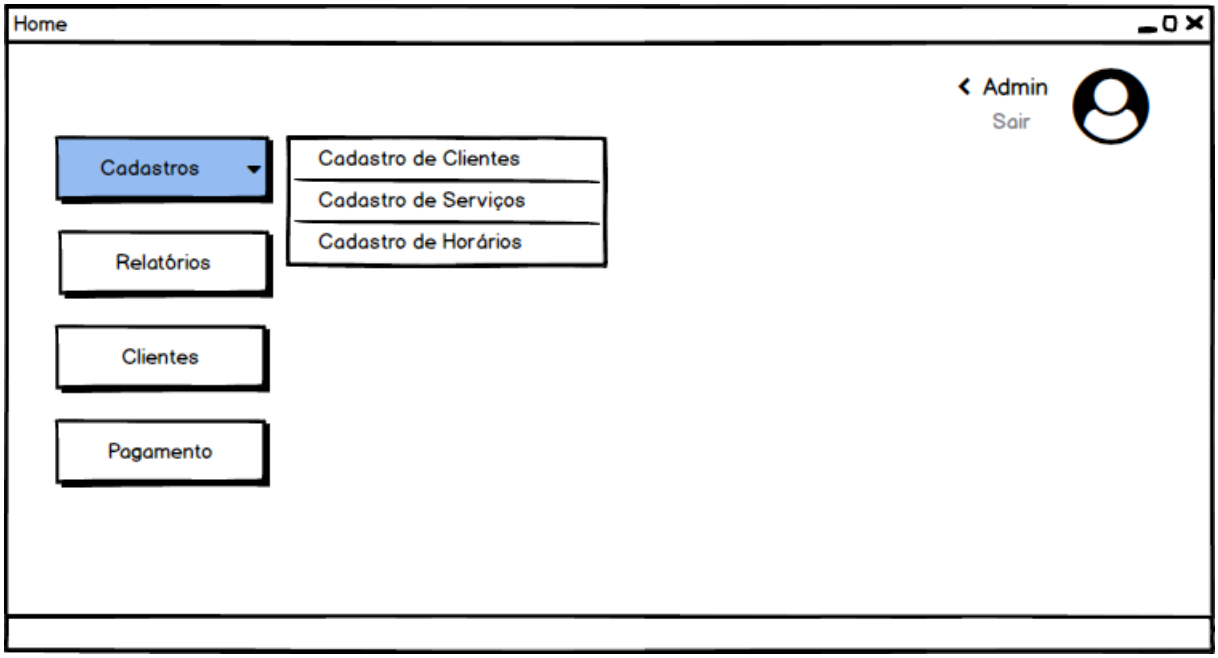
Prototipação das Interfaces

Figura 24 – Tela de *Login*



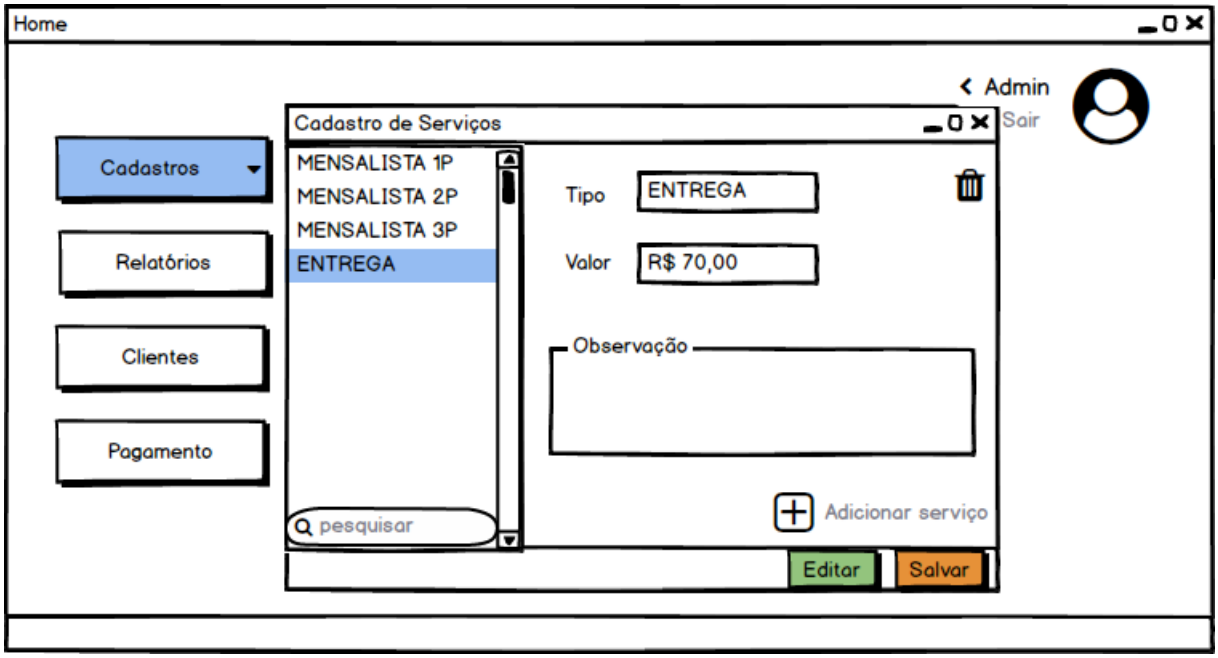
Fonte: Da própria autora

Figura 25 – Tela do menu principal



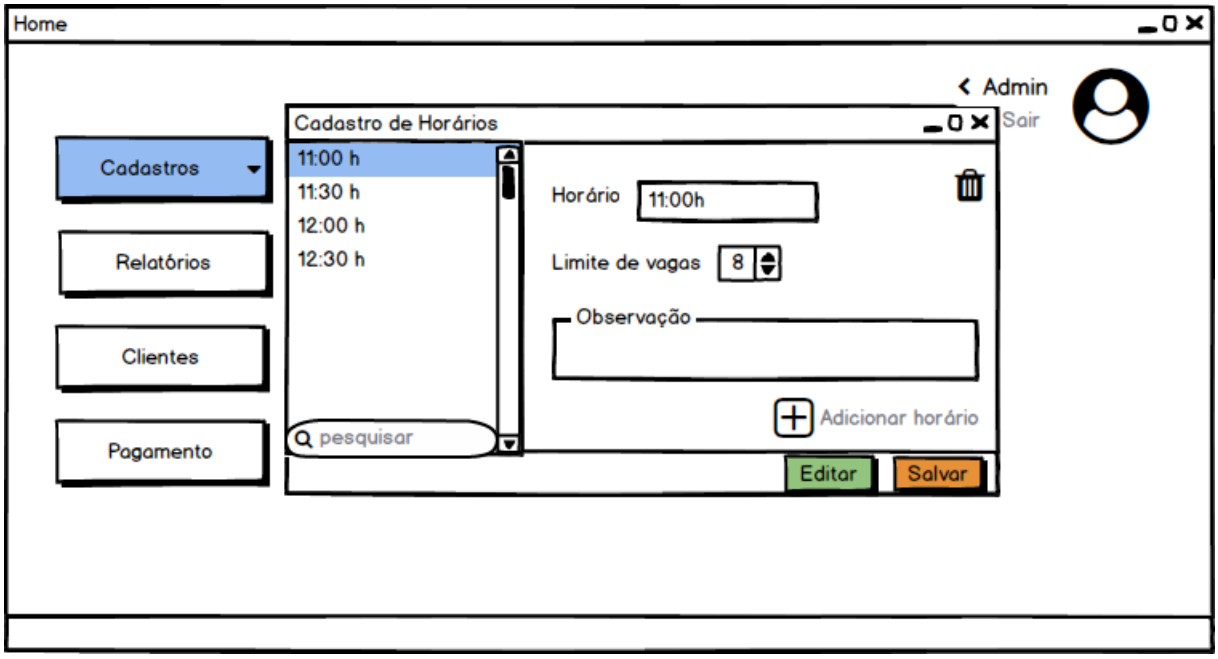
Fonte: Da própria autora

Figura 26 – Tela de Gerir Serviços



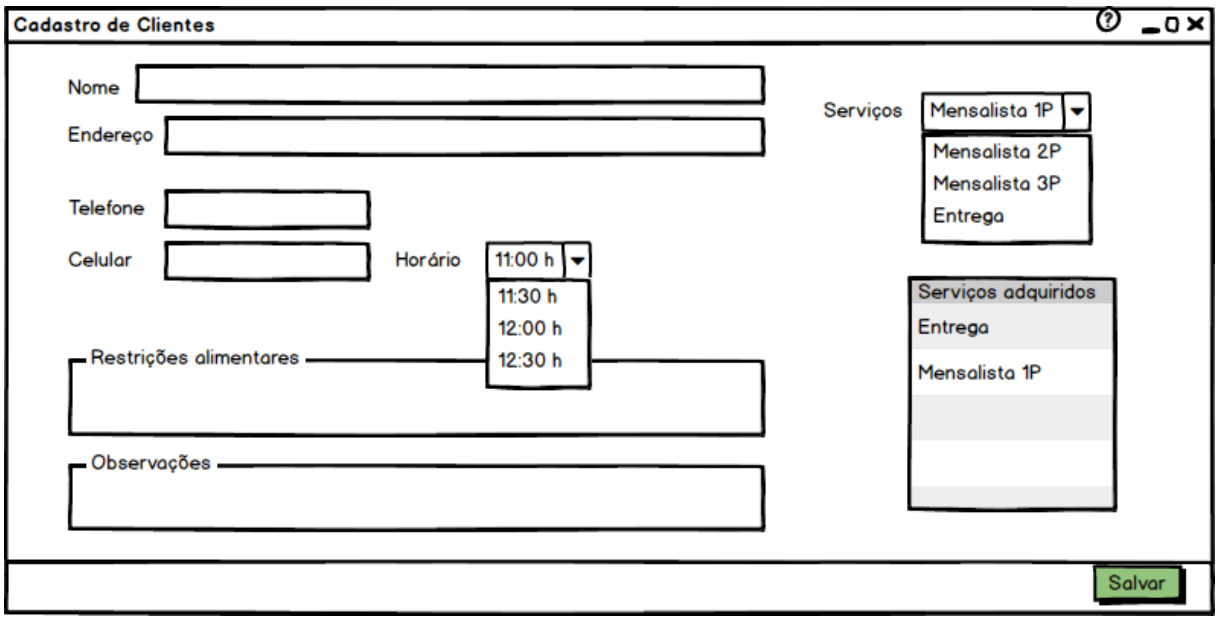
Fonte: Da própria autora

Figura 27 – Tela de Gerir Horário



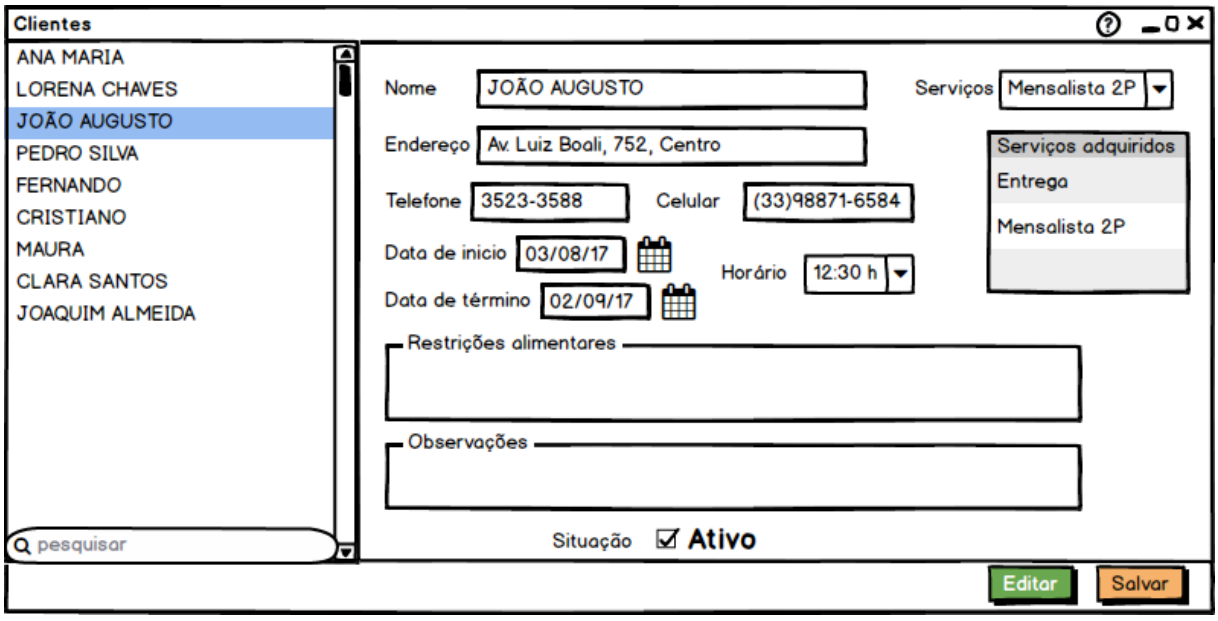
Fonte: Da própria autora

Figura 28 – Tela de Cadastro de Clientes



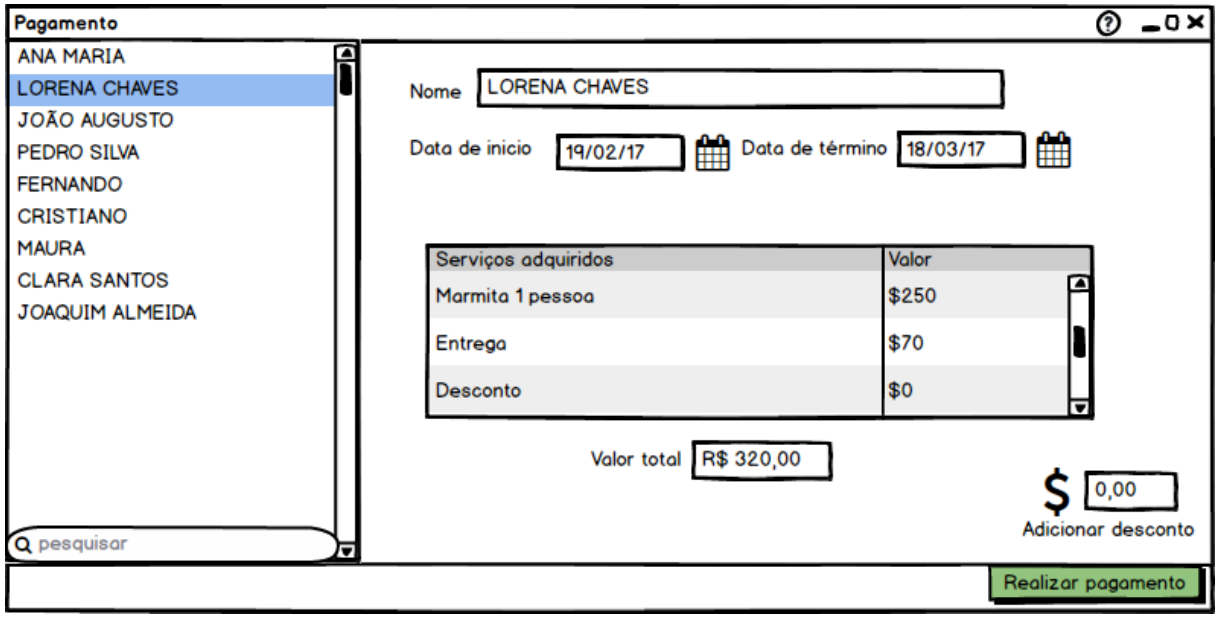
Fonte: Da própria autora

Figura 29 – Tela de Gerir Clientes



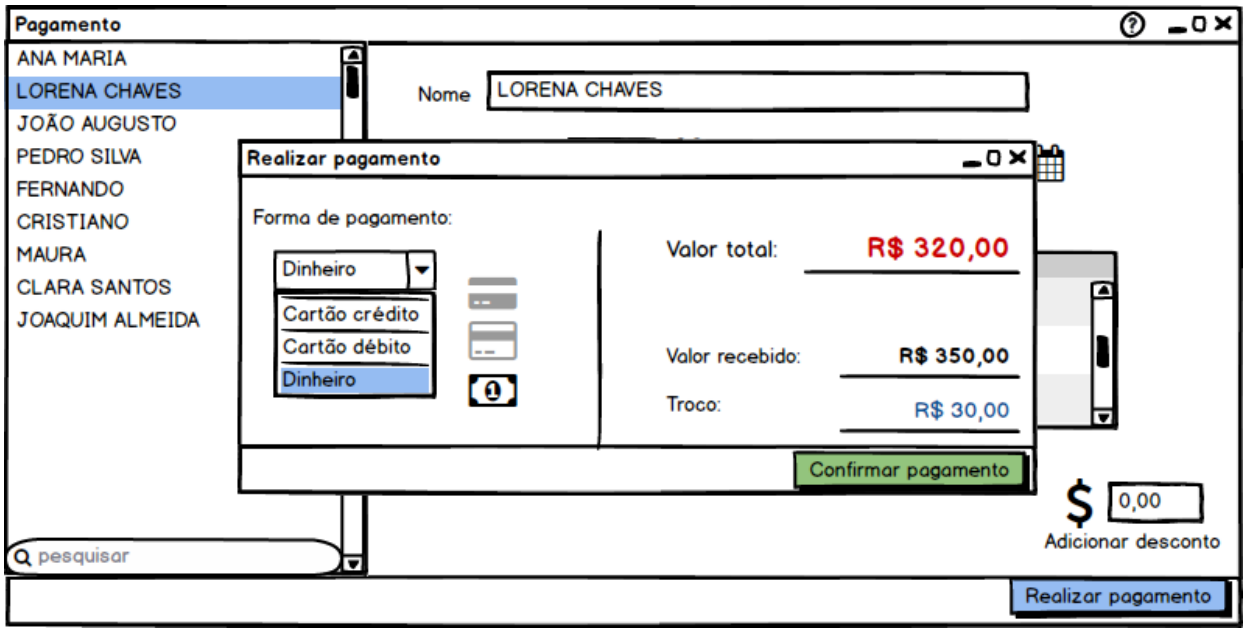
Fonte: Da própria autora

Figura 30 – Tela de Controle de Pagamentos



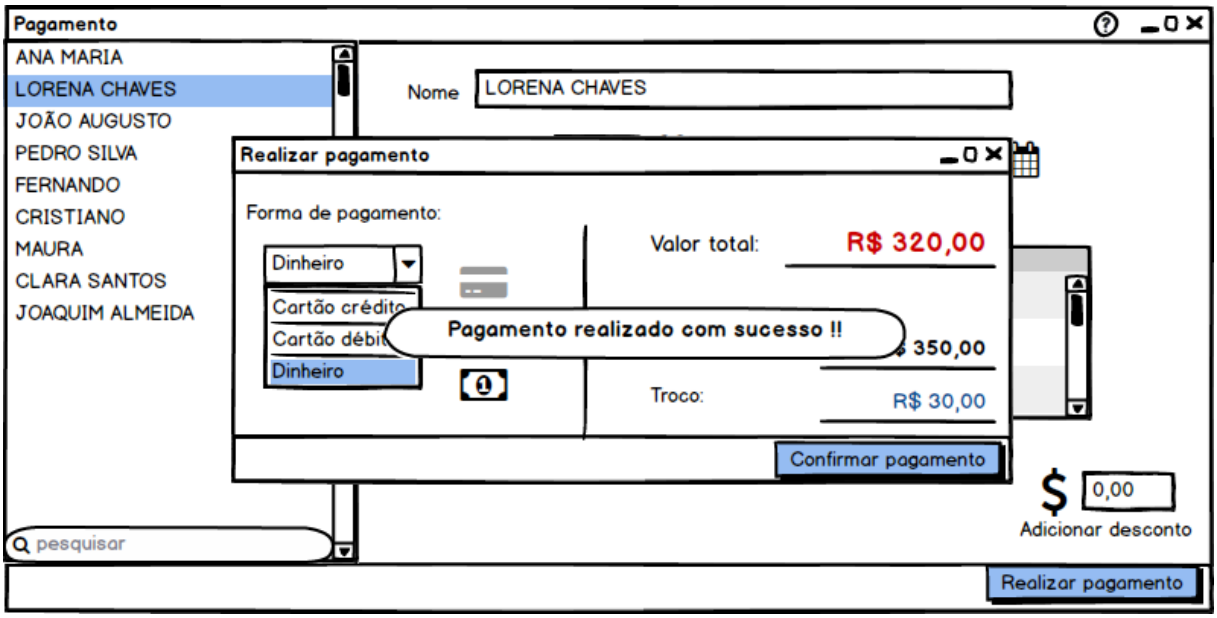
Fonte: Da própria autora

Figura 31 – Tela de Realizar pagamento



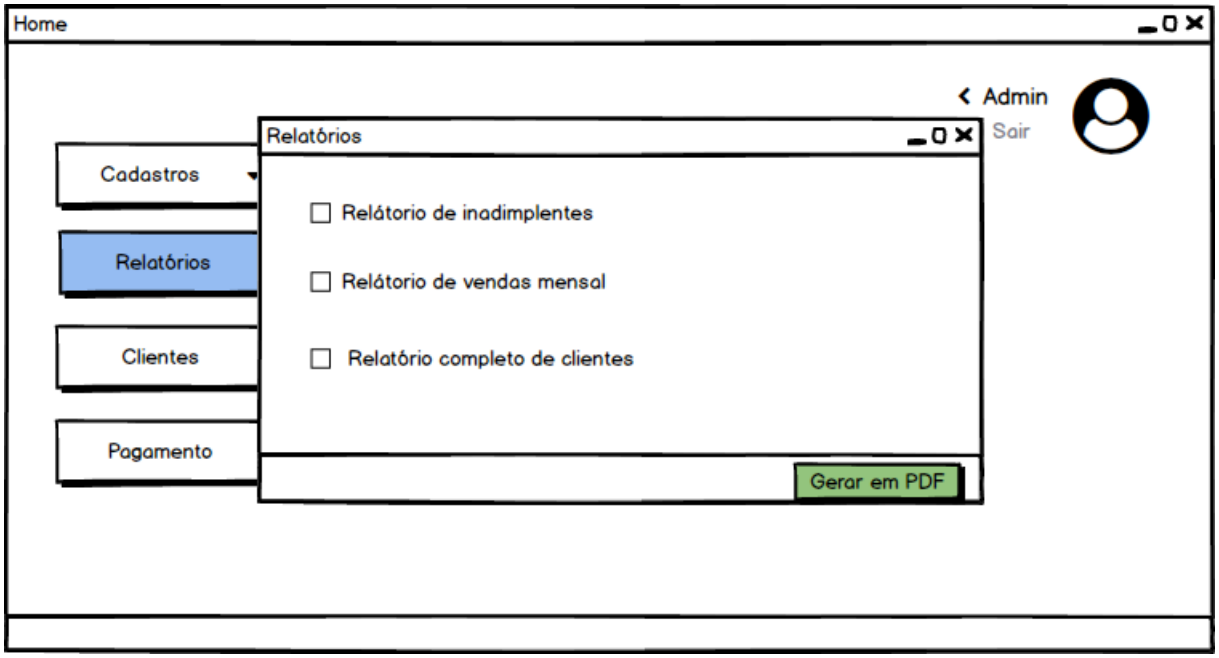
Fonte: Da própria autora

Figura 32 – Tela de Pagamento Confirmado



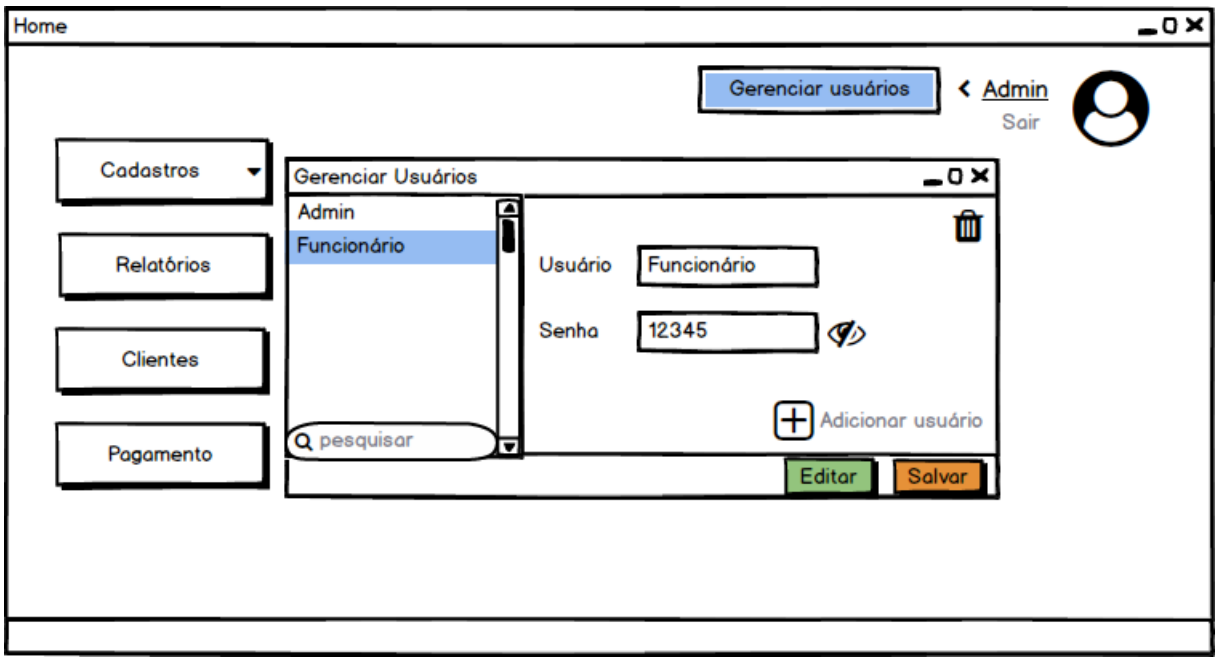
Fonte: Da própria autora

Figura 33 – Tela de Gerar Relatórios



Fonte: Da própria autora

Figura 34 – Tela de Gerenciar Usuários



Fonte: Da própria autora

APÊNDICE 2

Especificação de Requisitos de Software

Versão 1.0

Autora:

Sálua Jawhari Duarte

SUMÁRIO

1 INTRODUÇÃO

1.1. Metodologia

O *Rational Unified Process* reúne elementos de todos os modelos genéricos de processos, suporta iterações e ilustra boas práticas em especificação e design. O RUP é normalmente descrito de três perspectivas:

Uma perspectiva dinâmica que mostra as fases do modelo ao longo do tempo.

Uma perspectiva estática que mostra as atividades do processo que são promulgadas.

Uma perspectiva prática que sugere boas práticas a serem usadas durante o processo.

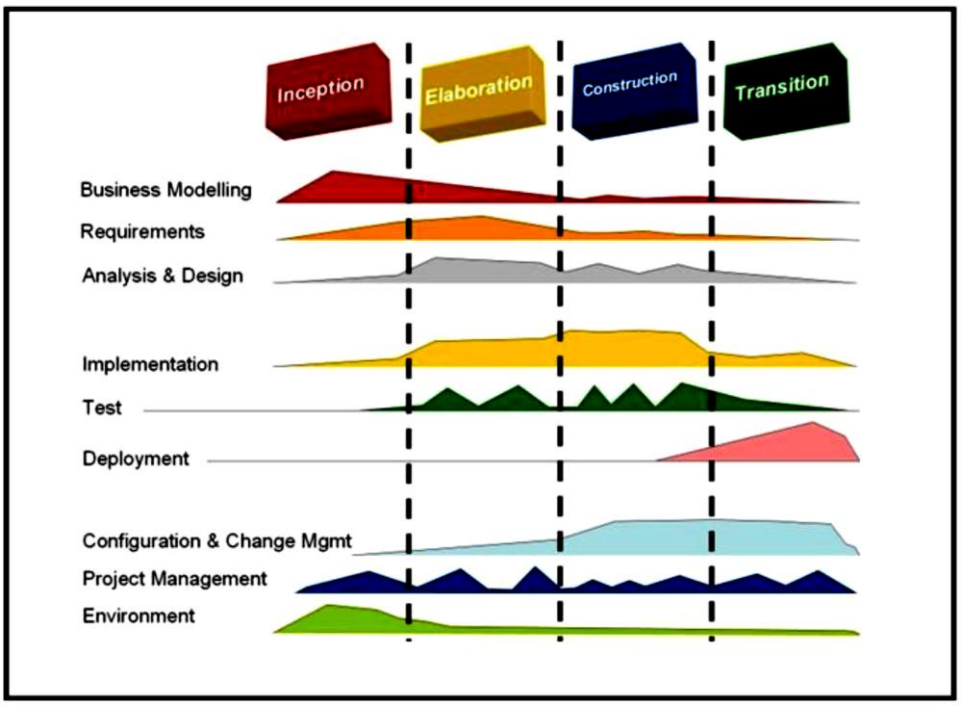


Figura 1 - Fases do RUP

As diferentes fases do RUP são:

Concepção

O objetivo da fase inicial é estabelecer um *business case* para o sistema. Identificando todos os externos entidades que irão interagir com o sistema e definir essas interações. Esta informação é usada para avaliar a contribuição do sistema para as empresas.

Elaboração

Os objetivos da fase de elaboração são desenvolver uma compreensão do domínio do problema, estabelecer um quadro arquitetônico, desenvolver plano de projeto e identificar os principais riscos do projeto.

Construção

Esta fase está preocupada com o *design*, programação e teste do sistema. Partes do sistema são desenvolvidas em paralelo e integrado durante esta fase.

Transição

Esta é a fase final do RUP e está preocupada com o movimento do sistema da comunidade de desenvolvimento para a comunidade de usuários e fazê-lo funcionar em ambiente real.

1.2. Proposta

A aplicação tem como proposta a gestão de clientes do Restaurante Monalisa. Um dos seus serviços voltado para clientes mensalistas, dá abertura ao registro desse documento. Visando aproveitar as oportunidades de crescimento para ampliar os seus negócios, a solução da informatização da empresa está sendo estudada para posteriormente investir no desenvolvimento de um software de qualidade, adequado para uma gestão eficaz da empresa.

Esse documento especifica os requisitos contemplados pelas regras de negócio do Restaurante Monalisa. O software auxilia o proprietário na administração dos seus serviços, explanando dados dos clientes, situação financeira, relatórios periódicos para análise do seu negócio. Todas as informações disponíveis em uma aplicação, organizando o dia-a-dia da empresa, centrando em aumentar potenciais competitivos para crescimento no mercado.

1.3. Escopo

- Tem dois usuários básicos: Funcionário e Administrador
- Operadora de cartão interage com o sistema no pagamento.
- O funcionário pode manter os funcionários, os horários e os serviços, cadastrando, atualizando e removendo esses dados.
- O Administrador tem funções exclusivas de gerar os relatórios e gerenciar usuários.

1.4. Definições, Acrônimos e Abreviações

Admin
Administrador. Ele tem autoridade de emitir relatórios e controlar usuários.

UML
Unified Modeling Language. Uma linguagem gráfica usada no desenvolvimento orientado a objetos, que inclui diversos tipos de modelos que fornecem diferentes visões de um sistema. A UML tornou-se um padrão de fato para a modelagem orientada a objetos.

JDK
Java SE Development Kit. O JDK é um conjunto de utilitários que inclui o Java *Runtime Environment*, o compilador Java e as APIs Java, para desenvolver programas em Java, tanto para os novos programadores quanto para os experientes.

IDE
Ambiente de Desenvolvimento integrado para desenvolvimento de software.

MYSQL
O MySQL é o banco de dados de código aberto mais conhecido no mundo. Com comprovado desempenho, confiabilidade e facilidade de uso, o *My*SQL tornou-se a principal opção de banco de dados para aplicativos baseados na Web

JDBC
É uma marca registrada da Sun *Microsystem*, embora seja geralmente entendido como um acrônimo de Java Data Base *Connectivity*.

1.5. Ferramentas usadas

Arquitetura de aplicação – JDK
Java SE *Development* Kit (JDK) é um conjunto de utilitários que permitem criar programas para plataforma Java. O pacote fornecido pela Oracle, vem todo o ambiente necessário para a criação e execução de aplicações Java.
Java é uma linguagem de programação orientada a objeto criada pela empresa americana Sun *Microsystems* na década de 90, atualmente ela pertence a Oracle. Sua principal característica é ser multiplataforma. Mas apesar de pertencer a uma grande empresa de software a linguagem Java é um *software* de código aberto e que é mantida pela Oracle em conjunto com a comunidade de usuários e empresas do ecossistema Java.

Plataforma de Banco de Dados – MYSQL
O MySQL é o banco de dados de código aberto mais conhecido no mundo. Com comprovado desempenho, confiabilidade e facilidade de uso, o MySQL tornou-se a principal opção de banco de dados para aplicativos baseados na Web

Ferramenta de desenvolvimento – Eclipse IDE
O Eclipse é uma ferramenta IDE que compreende vários tipos de linguagem e que aceita a instalação de *plugins* para emular o desenvolvimento da plataforma.

Ferramenta de Design – Argo UML, Visio
Argo UML
ArgoUML é a principal ferramenta de modelagem UML de código aberto e inclui suporte para todos os diagramas UML 1.4 padrão. Ele é executado em qualquer plataforma Java e está disponível em dez idiomas.

Visio
O *Microsoft Visio* é um software para desenhar uma grande variedade de diagramas. Estes incluem fluxogramas, gráficos organizacionais, projetos, plantas, diagramas de fluxo de dados, diagramas de fluxo de processos, processos de negócios,

modelagem, diagramas de raias, mapas 3D e muitos mais. Ele é um produto da *Microsoft*, vendido como um complemento para o MS *Office*.

1.6. Referências

- Java - https://www.java.com/pt_BR/download/faq/develop.xml
- Engenharia de Software, 8ª edição, Ian Sommerville.
- Oracle - https://www.oracle.com/br/mysql/index.html
- Sistemas de Banco de Dados, 4ª edição, Elmasri;Navathe.
- Tech tudo - http://www.techtudo.com.br/tudo-sobre/java-jdk.htm
- Devmedia - http://www.devmedia.com.br/conhecendo-o-eclipse-uma-apresentacao-detalhada-da-ide/25589
- Argo UML – http://argouml.tigris.org/
- Lucidchart - https://www.lucidchart.com/pages/pt/o-que-%C3%A9-microsoft-visio

1.7. Tecnologias a serem usadas

- JDK: Java SE Development Kit
- Eclipse IDE
- MYSQL

1.8. Visão Geral

Sistema existente:

Não existe sistema na empresa.

Sistema proposto:

- Cadastro de clientes, serviços
- Facilidade em editar os dados para atualização.
- Reunir organizadamente os dados e a situação do cliente.

Nosso plano:

- Cadastro de clientes, serviços e horários.
- Controle de clientes inadimplentes.
- Controlar limites de vagas por horário.
- Comunicação com a operadora de cartão.

2. DESCRIÇÃO GERAL

2.1. Perspectiva do Produto

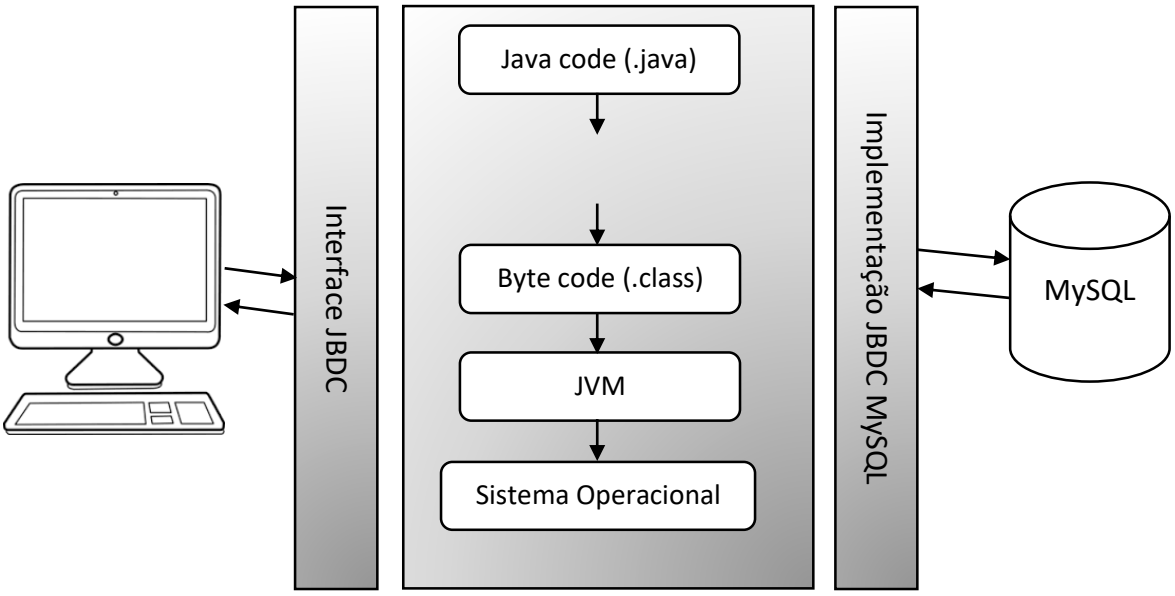


Figura 2 - Perspectiva do Produto

2.2. Interface do Software

Cliente/Servidor
Sistema Operacional (qualquer)
Servidor de Banco de Dados
MySQL, Sistema Operacional (qualquer)
Desenvolvimento Final
JDK (JVM, Java C, Eclipse IDE), MySQL, OS (Windows).

2.3. Interface do Hardware

Requisitos mínimos

Cliente/Servidor			
	Processador	**RAM**	**Espaço do disco**
JDK 7	Intel Pentium II – 1 GHz	128 MB	124 MB
MySQL		256 MB	500 MB (excluindo o tamanho dos dados)

Requisitos recomendados

Cliente/Servidor			
	Processador	**RAM**	**Espaço do disco**
JDK 8	Intel Core i3 - 2,3 GHz	512MB	200 MB
MySQL		512 MB	500 MB (excluindo o tamanho dos dados)

2.4. Interface de Comunicação

- O funcionário e/ou administrador usará o Sistema Operacional que roda o JVM.

2.5. Restrições

- O cliente só se torna ativo quando tem ao menos um serviço e um horário cadastrado.
- O login e a senha são usados para a identificação dos usuários.
- O sistema está projetado para trabalhar com servidor único.
- As cobranças de pagamento são feitas somente pelos clientes ativos do sistema.
- O limites de vagas por horário devem ser obedecidos senão o cadastro/alteração não será efetuada.

2.6. ER Diagrama

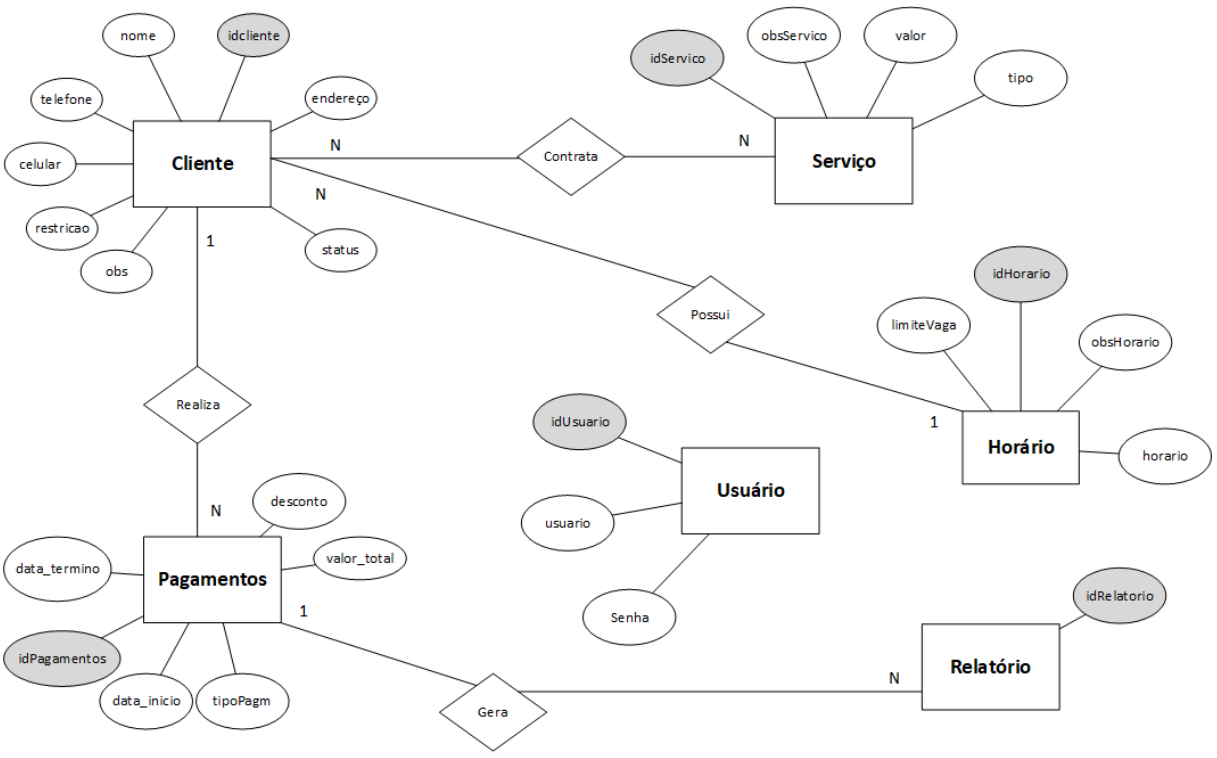


Figura 3 - Diagrama ER

2.7. Levantamento do Modelo de Caso de Uso

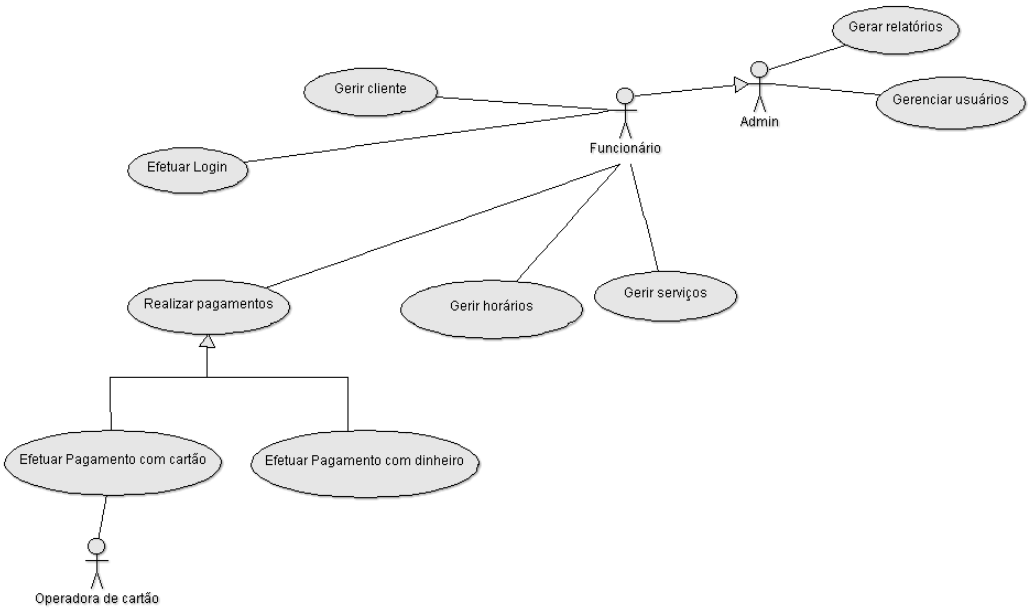


Figura 4 - Modelo de Caso de Uso

2.8. Design da Arquitetura

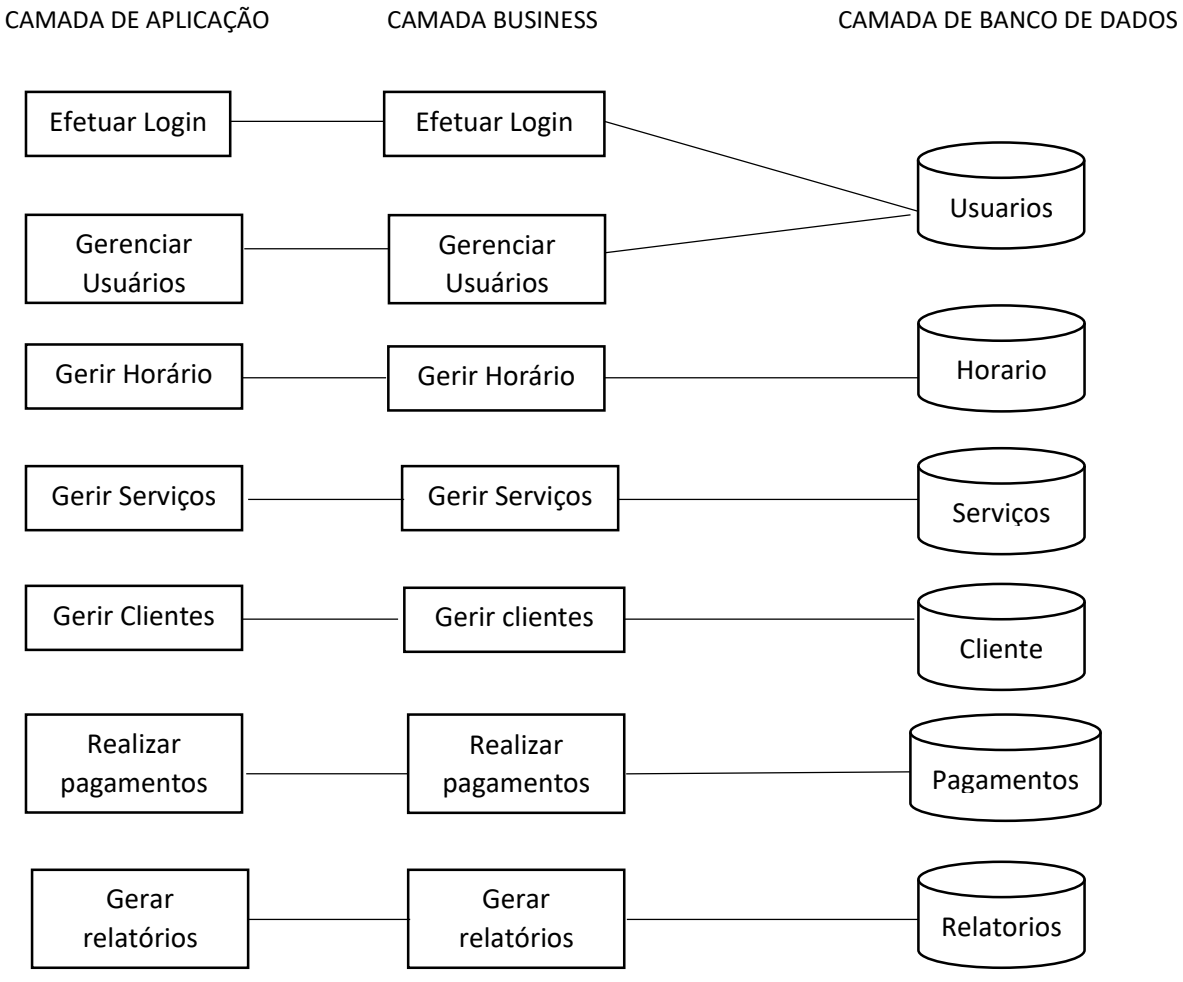


Figura 5 - Diagrama de Arquitetura

2.9. Design do Banco de dados

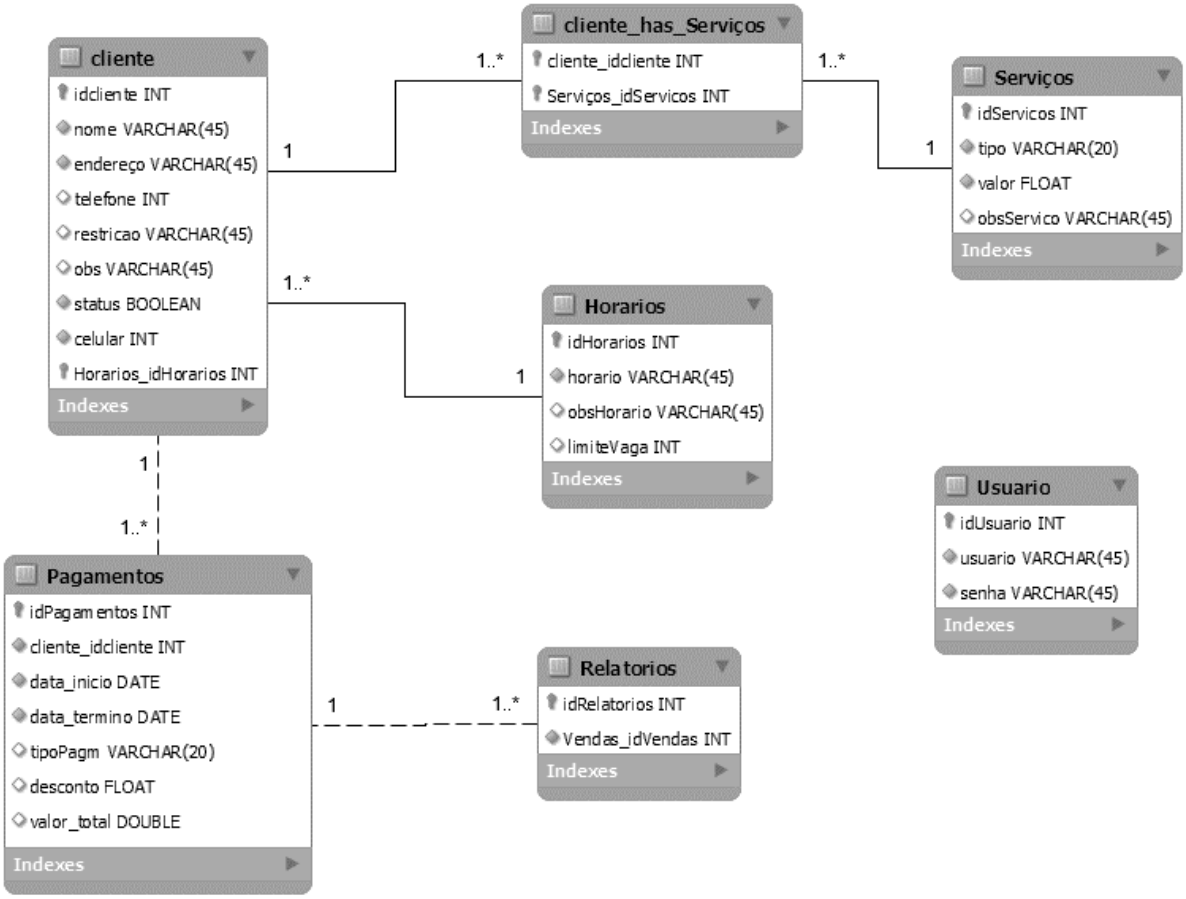


Figura 6 - Modelo de Banco de Dados

3. REQUISITOS ESPECÍFICOS

3.1. Relatórios de Caso de Uso

Gerir Cliente (CSU01)
Sumário: O funcionário irá manter clientes, podendo realizar consultas, cadastros e alterações de dados.
Ator primário: Funcionário
Pré-condições: O funcionário deve estar logado no sistema.
Fluxo Principal: 1. O sistema exibe o menu da tela principal com as funcionalidades específicas. 2. O funcionário seleciona a opção desejada [1], [2], [3]. 3. O caso de uso é encerrado.
Fluxo Alternativo (2A): Cadastrar Cliente [1] a. O funcionário seleciona a opção ‘Cadastros’. b. Em ‘Cadastros’ o funcionário seleciona a opção ‘Cadastro de Clientes’. c. O sistema exibe a tela ‘Cadastros de clientes’. d. O funcionário solicita os dados para ser preenchidos.

e. O sistema habilita o botão ‘Salvar’.
f. O funcionário salva os dados.
g. O sistema cadastra o cliente.
h. O caso continua a partir do passo 3

Fluxo Alternativo (2B): Atualizar Cliente [2]
a. O funcionário seleciona a opção ‘Clientes’
b. O sistema exibe a tela ‘Clientes’
c. O funcionário pesquisa pelo o nome, o cliente desejado.
d. Na tela, o sistema, exibe todas as informações do cliente.
e. O funcionário clica no botão ‘Editar’
f. Todos os campos são habilitados para edição, pelo sistema.
g. O funcionário seleciona e altera os dados preenchidos.
h. O sistema habilita a opção ‘Salvar’
i. O funcionário valida os dados.
j. O sistema atualiza os dados
k. O caso continua a partir do passo 3.

Fluxo Alternativo (2C): Inativar Cliente [3]
a. O funcionário seleciona a opção ‘Clientes’
b. O sistema exibe a tela ‘Clientes’
c. O funcionário pesquisa pelo o nome, o cliente desejado.
d. Na tela, o sistema, exibe todas as informações do cliente.
e. O funcionário clica no botão ‘Editar’
f. Todos os campos são habilitados para edição, pelo sistema.
g. O funcionário seleciona o campo ‘Situação’ e desmarca a função Ativo.
h. O sistema habilita a opção ‘Salvar’.
i. O funcionário clica em ‘Salvar’.
j. O sistema inativa o cliente.
k. O caso continua a partir do passo 3

Fluxo de exceção: No fluxo alternativo [1] e [2], os campos nome, celular, serviços e horário em branco
a. O sistema envia uma mensagem de advertência na tela.

Fluxo de exceção: No fluxo alternativo [1] não há horários cadastrados
a. Ir para o caso de uso: Gerir horário

Fluxo de exceção: No fluxo alternativo [1] e [2] limite de vagas por horários excedido
a. O sistema envia uma mensagem de advertência na tela.
b. Incluir outro horário disponível.

Pós-condições: O cliente foi cadastrado/atualizado/inativado com sucesso. O sistema manterá o usuário logado, exibindo a tela inicial do usuário. Os dados estarão atualizados nas tabelas do BD.

Histórico: Criado por Sálua Jawhari Duarte em 21/09/2017

Gerir Horários (CSU02)
Sumário: O funcionário irá manter horários, fazer cadastros e alterações de dados dos horários.
Ator primário: Funcionário
Pré-condições: O funcionário deve estar logado no sistema.
Fluxo Principal: 1. O sistema exibe o menu da tela principal com as funcionalidades específicas. 2. O funcionário seleciona a opção ‘Cadastros’ 3. Em ‘Cadastros’, o funcionário seleciona a opção ‘Cadastro de horários’. 4. O sistema exibe a tela ‘Cadastros de horários’. 5. O funcionário seleciona a opção desejada [1], [2], [3]. 6. O caso de uso é encerrado.
Fluxo Alternativo (5A): Cadastrar Horário [1] a. O sistema habilita o botão ‘Adicionar horário’ e ‘Editar’ b. O funcionário clica no botão ‘Adicionar horário’ c. O sistema desabilita as funções ‘Editar’ e ‘Salvar’. d. O funcionário preenche os campos para cadastro. e. O sistema habilita o botão ‘Salvar’. f. O funcionário salva os dados. g. O sistema cadastra o horário. h. O caso continua a partir do passo 3 **Fluxo Alternativo (5B):** Atualizar Horário [2] a. O funcionário pesquisa pelo horário desejado. b. Na tela são exibidas todas as informações do horário. c. O sistema habilita o botão ‘Adicionar horário’ e ‘Editar’ d. O funcionário clica no botão ‘Editar’ e. Todos os campos são habilitados para edição, pelo sistema. f. O sistema desabilita as funções ‘Adicionar horário’ e ‘Salvar’. g. O funcionário seleciona e altera os dados preenchidos. h. O sistema habilita a opção ‘Salvar’ i. O funcionário valida os dados. j. O sistema atualiza os dados k. O caso continua a partir do passo 3. **Fluxo Alternativo (5C):** Remover Horário [3] a. O funcionário pesquisa pelo horário desejado. b. Na tela são exibidas todas as informações do horário. c. O sistema habilita o botão ‘Editar’ d. O funcionário clica no botão ‘Editar’ e. Todos os campos são habilitados para edição, pelo sistema. f. O sistema habilita o ícone da lixeira. g. O funcionário clica no ícone da lixeira. h. O sistema envia uma mensagem de confirmação. i. O funcionário confirma a ação e o horário é removido. j. O caso continua a partir do passo 3.
Fluxo de exceção: No fluxo alternativo [1] e [2] o campo horário estiver em branco.

a. O sistema envia uma mensagem de advertência na tela.
Pós-condições: O horário foi cadastrado/atualizado/removido com sucesso. O sistema manterá o usuário logado, exibindo a tela inicial do usuário. Os dados estarão atualizados nas tabelas do BD.
Histórico: Criado por Sálua Jawhari Duarte em 25/09/2017

Gerir Serviços (CSU03)
Sumário: O funcionário irá manter serviços, fazer cadastros e alterações de dados dos serviços.
Ator primário: Funcionário
Pré-condições: O funcionário deve estar logado no sistema.
Fluxo Principal: 1. O sistema exibe o menu da tela principal com as funcionalidades específicas. 2. O funcionário seleciona a opção ‘Cadastros’. 3. Em ‘Cadastros’, o funcionário seleciona a opção ‘Cadastro de serviços’. 4. O sistema exibe a tela ‘Cadastros de serviços’. 5. O funcionário seleciona a opção desejada [1], [2], [3]. 6. O caso de uso é encerrado.
Fluxo Alternativo (5A): Cadastrar Serviços [1] a. O sistema habilita o botão ‘Adicionar serviço’ e ‘Editar’ b. O funcionário clica no botão ‘Adicionar serviço’ c. O sistema desabilita as funções ‘Editar’ e ‘Salvar’. d. O funcionário preenche os campos para cadastro. e. O sistema habilita o botão ‘Salvar’. f. O funcionário salva os dados. g. O sistema cadastra o serviço. h. O caso continua a partir do passo 3 **Fluxo Alternativo (5B):** Atualizar Serviços [2] a. O funcionário pesquisa pelo serviço desejado. b. Na tela são exibidas todas as informações do serviço. c. O sistema habilita o botão ‘Adicionar serviço’ e ‘Editar’ d. O funcionário clica no botão ‘Editar’ e. Todos os campos são habilitados para edição, pelo sistema. f. O sistema desabilita as funções ‘Adicionar serviço’ e ‘Salvar’. g. O funcionário seleciona e altera os dados preenchidos. h. O sistema habilita a opção ‘Salvar’ i. O funcionário valida os dados. j. O sistema atualiza os dados k. O caso continua a partir do passo 3. **Fluxo Alternativo (5C):** Remover Serviços [3] a. O funcionário pesquisa pelo serviço desejado. b. Na tela são exibidas todas as informações do serviço. c. O sistema habilita o botão ‘Editar’ d. O funcionário clica no botão ‘Editar’ e. Todos os campos são habilitados para edição, pelo sistema. f. O sistema habilita o ícone da lixeira.

g. O funcionário clica no ícone da lixeira. h. O sistema envia uma mensagem de confirmação. i. O funcionário confirma a ação e o serviço é removido. j. O caso continua a partir do passo 3.
Fluxo de exceção: No fluxo alternativo (1) e (2) o campo serviço estiver em branco. a. O sistema envia uma mensagem de advertência na tela.
Pós-condições: O serviço foi cadastrado/atualizado/removido com sucesso. O sistema manterá o usuário logado, exibindo a tela inicial do usuário. Os dados estarão atualizados nas tabelas do BD.
Histórico: Criado por Sálua Jawhari Duarte em 25/09/2017

Realizar pagamentos (CSU04)
Sumário: O funcionário irá receber pagamentos, abrindo e renovando pacotes de serviço.
Ator primário: Funcionário
Pré-condições: O funcionário deve estar logado no sistema. Para realizar pagamentos o cliente deve estar cadastrado e ativo no sistema.
Fluxo Principal: 1. O sistema exibe o menu da tela principal com as funcionalidades específicas. 2. O funcionário seleciona a opção ‘Pagamentos’ 3. O ator pesquisa pelo nome do cliente desejado para atendimento. 4. Na tela são exibidas as informações da situação financeira do cliente. 5. O funcionário confere as informações e preenche as datas de início e término dos pacotes e/ou acrescenta demais informações. 6. O sistema habilita o botão ‘Realizar pagamento’. 7. O funcionário clica no botão ‘Realizar pagamento’. 8. O sistema exibe uma tela com as opções de forma de pagamento. 9. O funcionário seleciona a opção desejada [1], [2], [3]. 10. O funcionário confere os dados e clica em confirmar pagamento. 11. O sistema exibe uma mensagem de ‘Pagamento realizado com sucesso’. 12. O caso de uso é encerrado.
Fluxo Alternativo (9A): Efetuar pagamento com dinheiro [1] a. O funcionário seleciona ‘Dinheiro’. b. O funcionário registra o valor recebido c. O sistema registra o valor recebido em dinheiro. d. O sistema informa o troco a ser repassado ao cliente e. O caso continua a partir do passo 10. **Fluxo Alternativo (9B):** Efetuar pagamento com cartão de débito [2] a. O funcionário seleciona ‘Cartão de débito’. b. O sistema faz comunicação com o sistema da operadora de cartão. c. O sistema solicita autorização de pagamento da operadora de cartão.

d. O caso continua a partir do passo 10. **Fluxo Alternativo (9C):** Efetuar pagamento com cartão de crédito [2] a. O funcionário seleciona ‘Cartão de crédito’. b. O sistema faz comunicação com o sistema da operadora de cartão. c. O sistema solicita autorização de pagamento da operadora de cartão. d. O caso continua a partir do passo 10.
Fluxo de exceção (5): Data de início e término em branco ou inválidos. a. O sistema envia uma mensagem de advertência na tela. b. O caso continua a partir do passo 5. **Fluxo de exceção (3):** Cliente não cadastrado. a. O funcionário volta ao menu principal b. Incluir o caso de uso: Gerir Clientes (CSU01) c. O caso continua a partir do passo 3 **Fluxo de exceção:** No fluxo alternativo [1] e [2], pagamento não autorizado pela operadora de cartão. a. O sistema envia uma mensagem de alerta para o usuário b. O caso continua a partir do passo 8.
Pós-condições: O pagamento/renovação foi realizado com sucesso. O sistema manterá o usuário logado, exibindo a tela inicial do usuário. Os dados estarão atualizados nas tabelas do BD.
Histórico: Criado por Sálua Jawhari Duarte em 25/09/2017

Efetuar Login (CSU05)
Sumário: O usuário irá logar e ter acesso ao sistema
Ator primário: Funcionário e/ou administrador
Fluxo Principal: 1. O sistema exibe a tela de login. 2. O usuário informa nome e senha. 3. O sistema valida o usuário e senha. 4. O usuário tem acesso ao sistema. 5. O caso de uso é encerrado.
Fluxo de exceção: Usuário ou senha em branco ou inválidos. c. O sistema envia uma mensagem de advertência na tela. d. O caso continua a partir do passo 2
Pós-condições: O usuário tem acesso ao sistema.
Histórico: Criado por Sálua Jawhari Duarte em 25/09/2017

Gerar Relatórios (CSU06)
Sumário: O administrador vai ter acesso aos relatórios do sistema e poder emiti-los.
Ator primário: Administrador
Pré-condições: O administrador deve estar logado no sistema.
Fluxo Principal: 1. O sistema exibe a tela do menu principal. 2. O ator seleciona a opção ‘Gerar relatório’. 3. O sistema a janela com as opções a ser selecionadas. 4. O ator seleciona a opção desejada. 5. O sistema gera o PDF com o relatório solicitado. 6. O caso de uso é encerrado.
Fluxo de exceção: Não há cadastros no sistema a. O sistema emite uma mensagem de informação ao usuário
Pós-condições: O administrador tem acesso ao pdf tanto para consulta em tela ou impressa.
Histórico: Criado por Sálua Jawhari Duarte em 25/09/2017

Gerenciar Usuários (CSU07)
Sumário: O administrador irá manter usuários, fazer cadastros e alterações de dados dos usuários.
Ator primário: Administrador
Pré-condições: O administrador deve estar logado no sistema.
Fluxo Principal: 2 O sistema exibe o menu da tela principal com as funcionalidades específicas. 3 O administrador seleciona a opção desejada ‘Gerenciar Usuários’ 4 O sistema exibe a tela de Gerenciar Usuários. 5 O funcionário seleciona a opção desejada [1], [2], [3]. 6 O caso de uso é encerrado.
Fluxo Alternativo (4A): Cadastrar Usuário [1] a. O sistema habilita o botão ‘Adicionar usuário’ e ‘Editar’ b. O administrador clica no botão ‘Adicionar usuário’ c. O sistema desabilita as funções ‘Editar’ e ‘Salvar’. d. O administrador preenche os campos para cadastro. e. O sistema habilita o botão ‘Salvar’. f. O administrador salva os dados. g. O sistema cadastra o usuário. h. O caso continua a partir do passo 3 **Fluxo Alternativo (2B):** Atualizar Usuário [2] a. O administrador pesquisa pelo usuário desejado. b. Na tela são exibidas todas as informações do usuário. c. O sistema habilita o botão ‘Adicionar usuário’ e ‘Editar’ d. O administrador clica no botão ‘Editar’ e. Todos os campos são habilitados para edição, pelo sistema.

f. O sistema desabilita as funções ‘Adicionar usuário’ e ‘Salvar’.
g. O administrador seleciona e altera os dados preenchidos.
h. O sistema habilita a opção ‘Salvar’
i. O administrador valida os dados.
j. O sistema atualiza os dados
k. O caso continua a partir do passo 3.

Fluxo Alternativo (2C): Remover Usuário [3]
a. O administrador pesquisa pelo usuário desejado.
b. Na tela são exibidas todas as informações do usuário.
c. O sistema habilita o botão ‘Editar’
d. O administrador clica no botão ‘Editar’
e. Todos os campos são habilitados para edição, pelo sistema.
f. O sistema habilita o ícone da lixeira.
g. O administrador clica no ícone da lixeira.
h. O sistema envia uma mensagem de confirmação.
i. O administrador confirma a ação e o usuário é removido.
j. O caso continua a partir do passo 3.

Fluxo de exceção: No fluxo alternativo [1] e [2] o campo usuário ou senha estiver em branco.
a. O sistema envia uma mensagem de advertência na tela.

Pós-condições: O usuário foi cadastrado/atualizado/removido com sucesso. O sistema manterá o usuário logado, exibindo a tela inicial do usuário. Os dados estarão atualizados nas tabelas do BD.

Histórico: Criado por Sálua Jawhari Duarte em 23/10/2017

3.2. Diagrama de Atividade

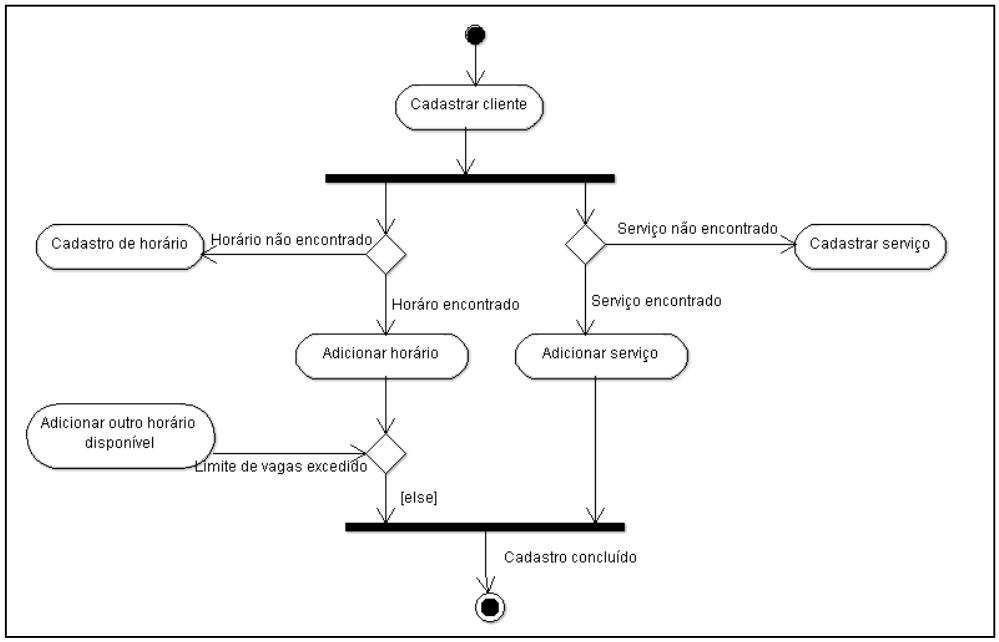


Figura 7 - Cadastrar Cliente

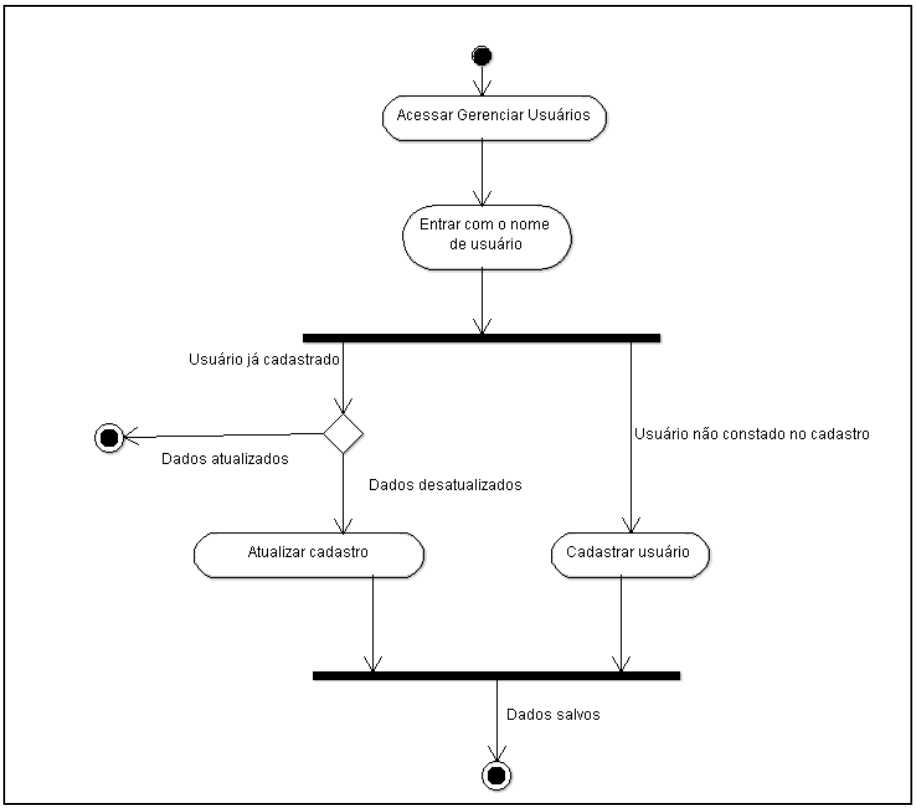


Figura 8 - Gerenciar usuário

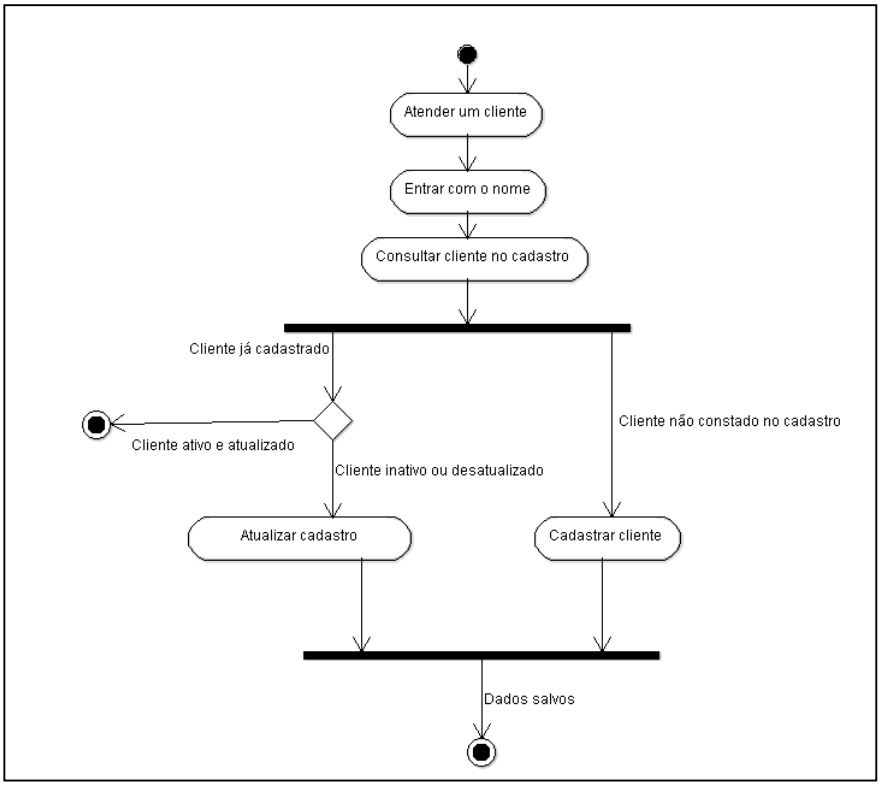


Figura 9 - Gerir Cliente

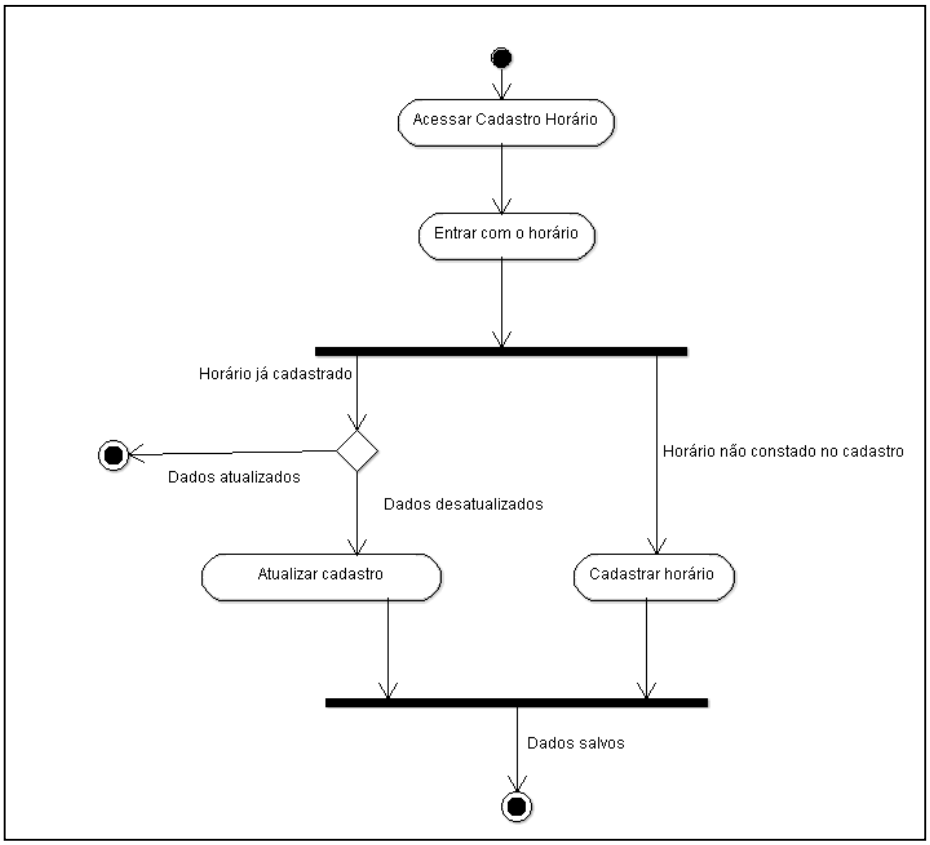


Figura 10 - Gerir Horário

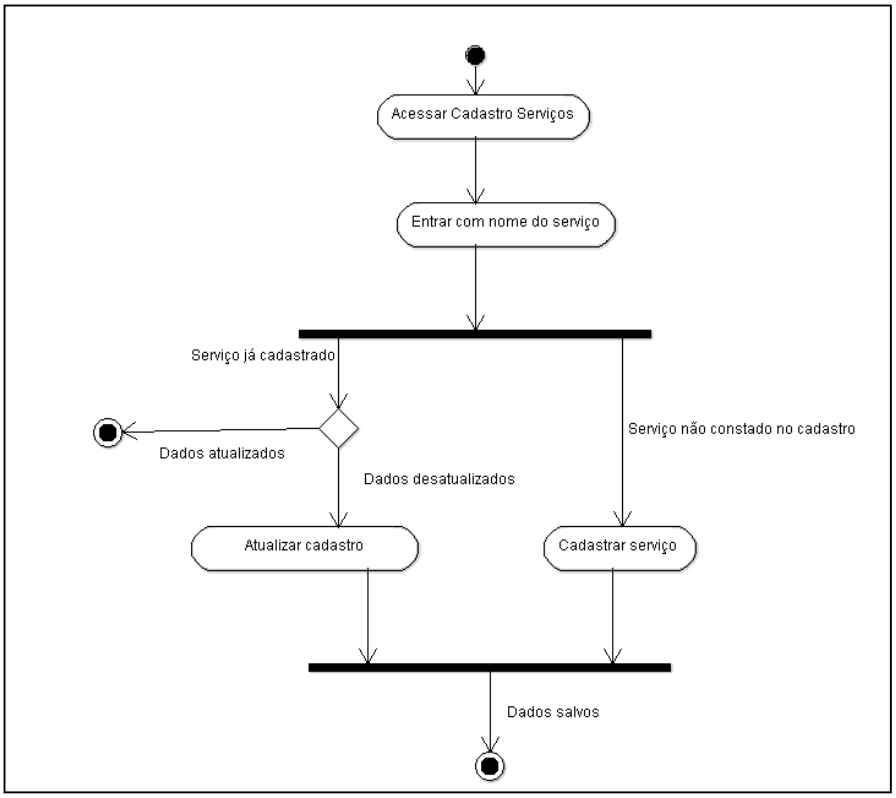


Figura 11 - Gerir Serviço

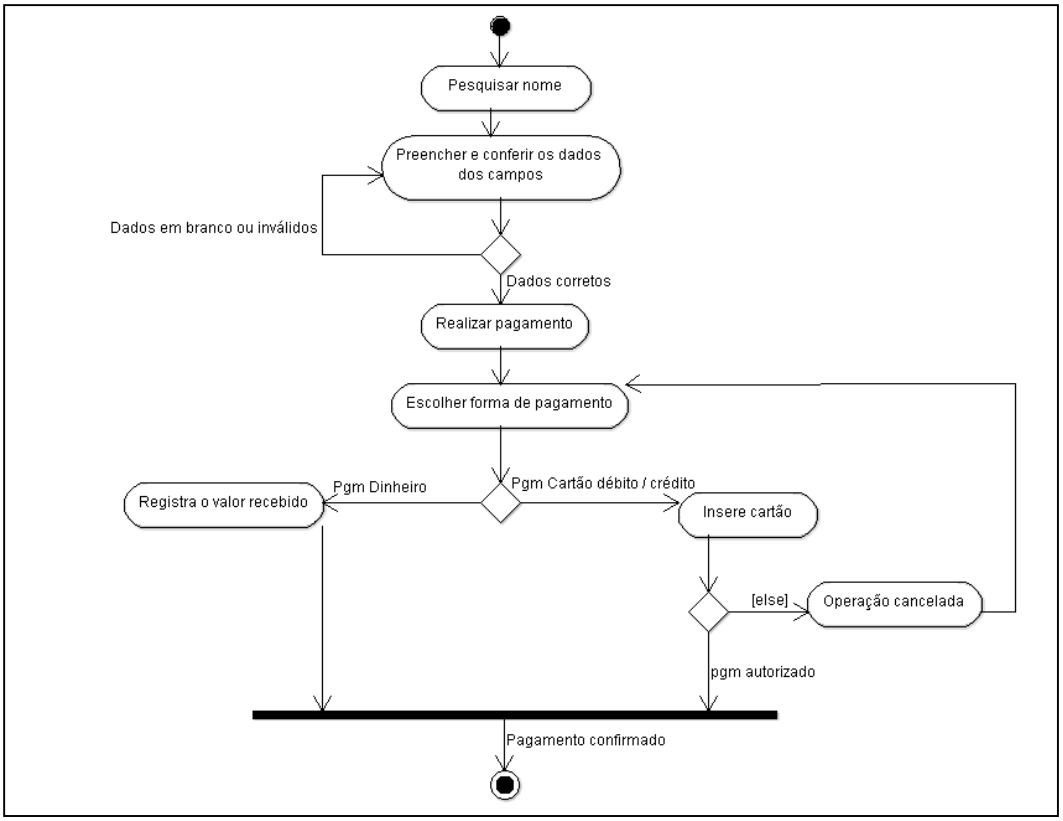


Figura 12 - Receber pagamento

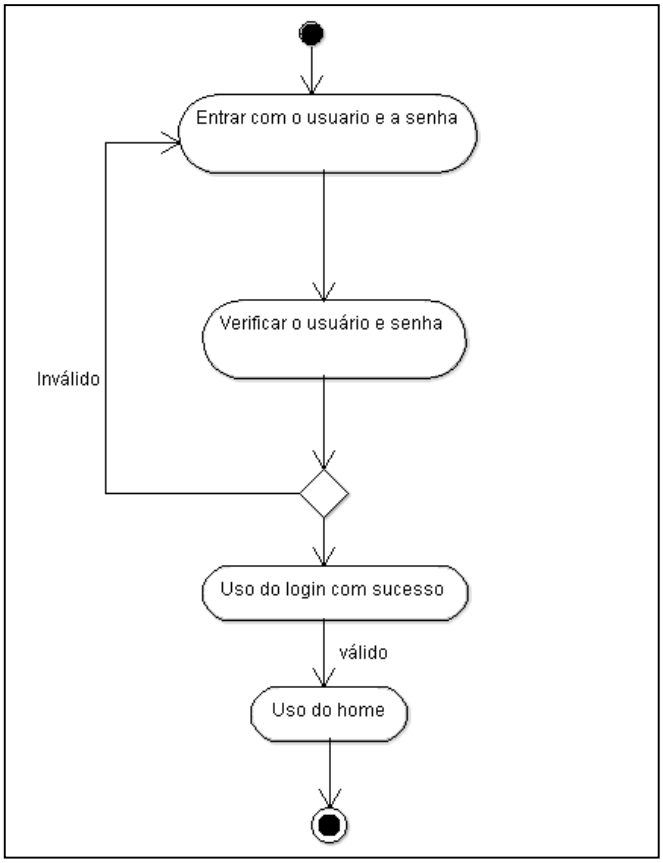


Figura 13 - Efetuar Login

3.3 Diagrama de Sequência

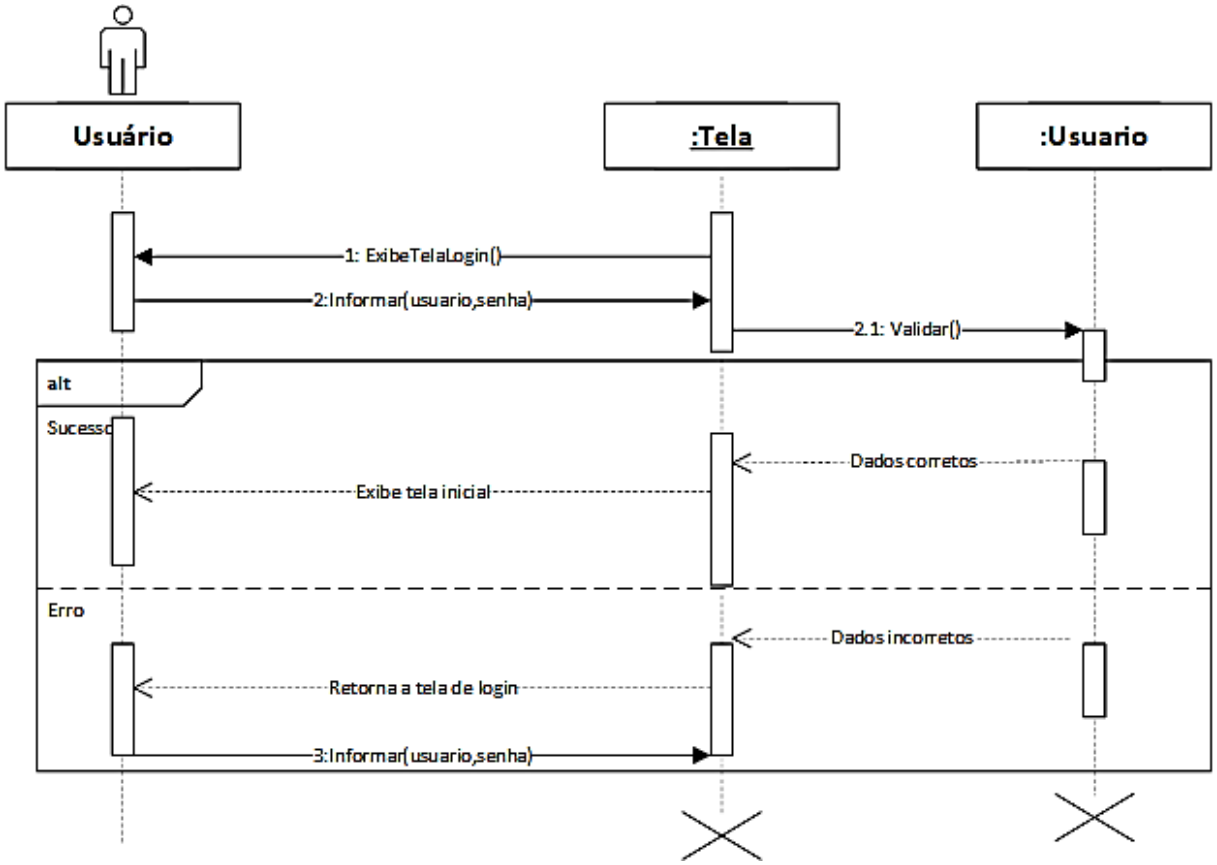


Figura 14 - Efetuar Login

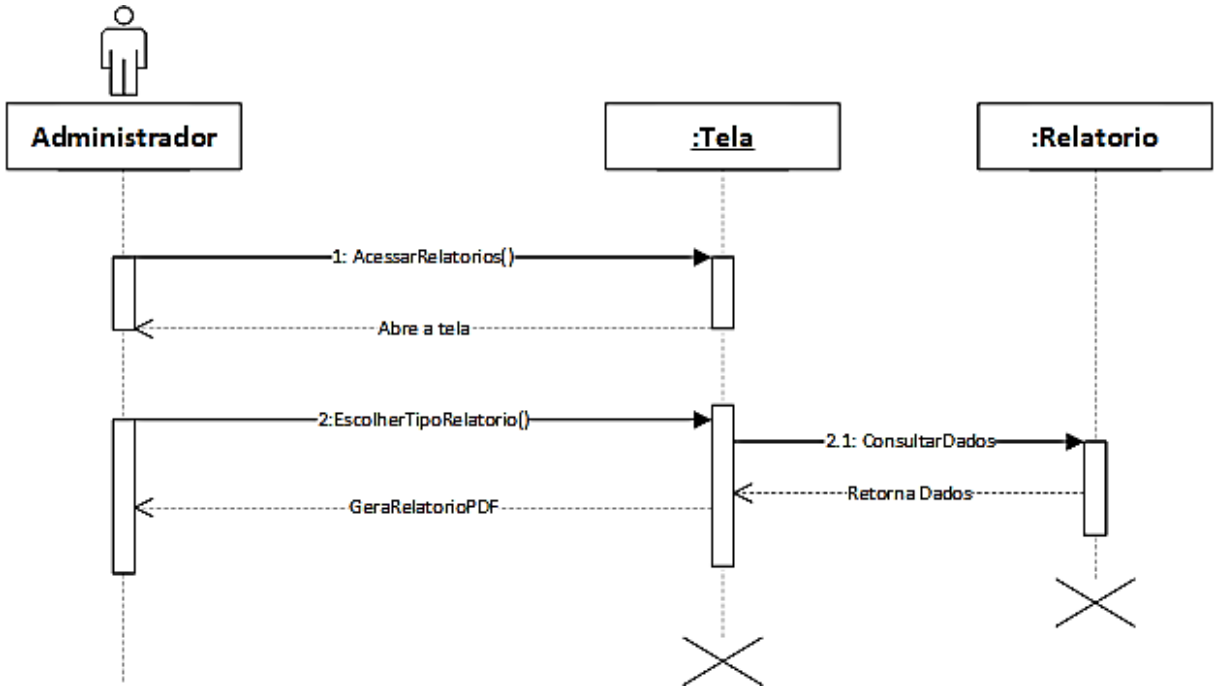


Figura 15 - Gerar relatório

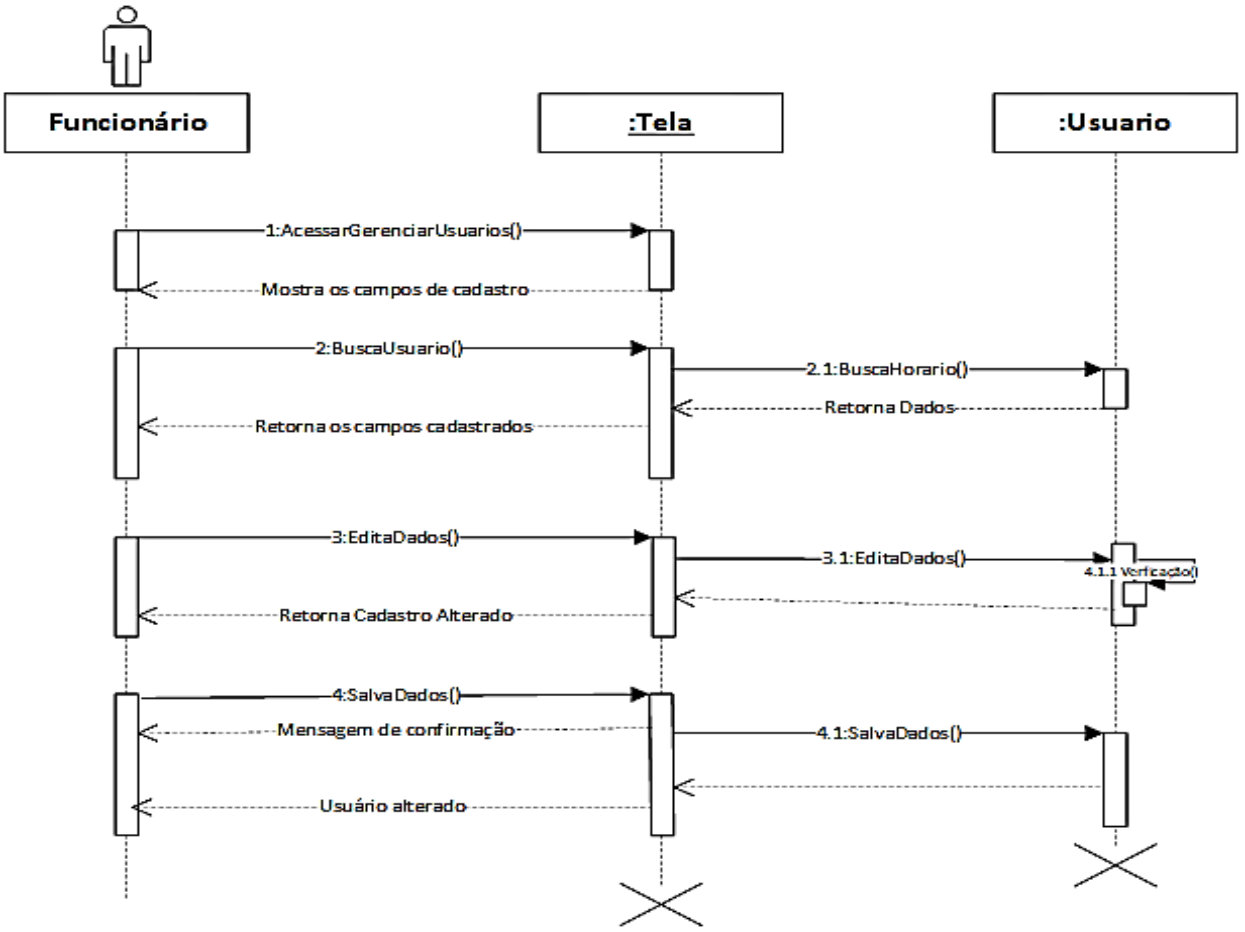


Figura 16 - Atualizar Usuário

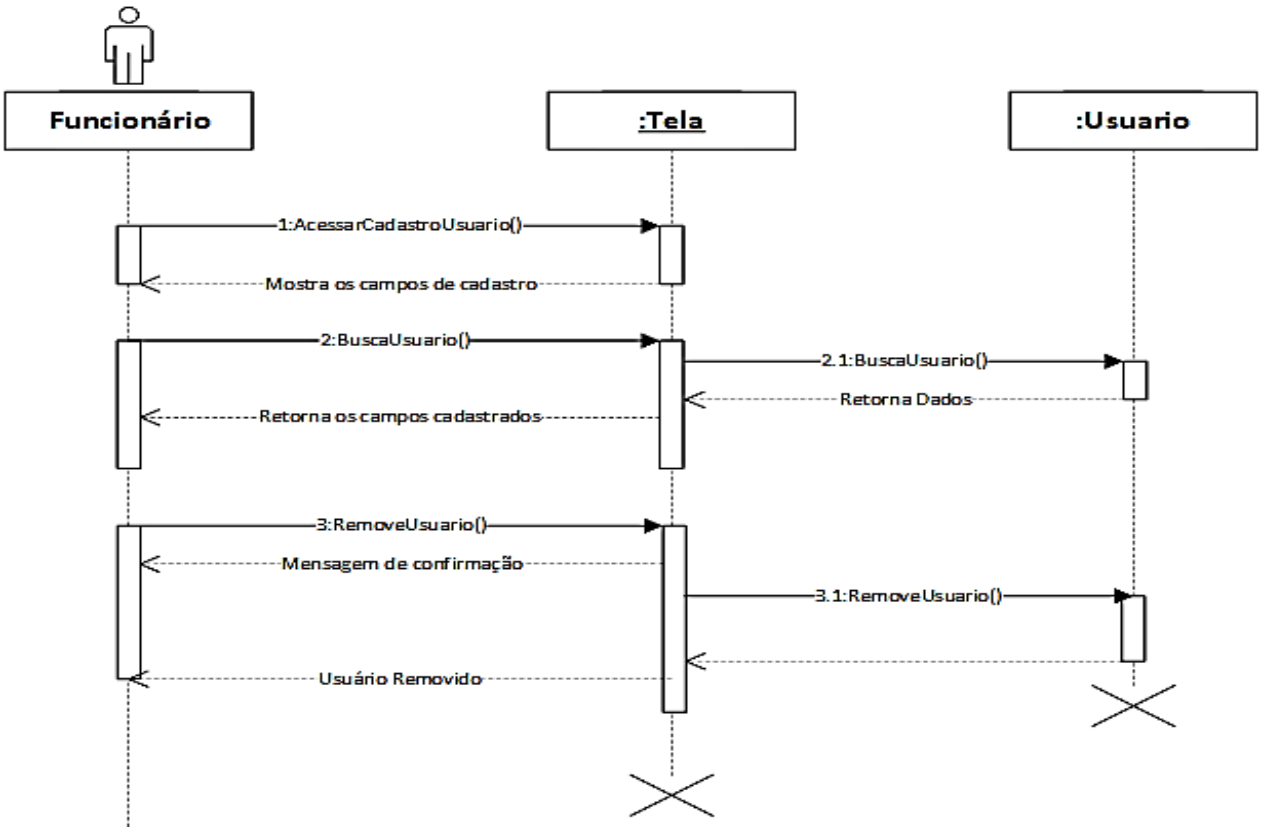


Figura 17 - Remover Usuário

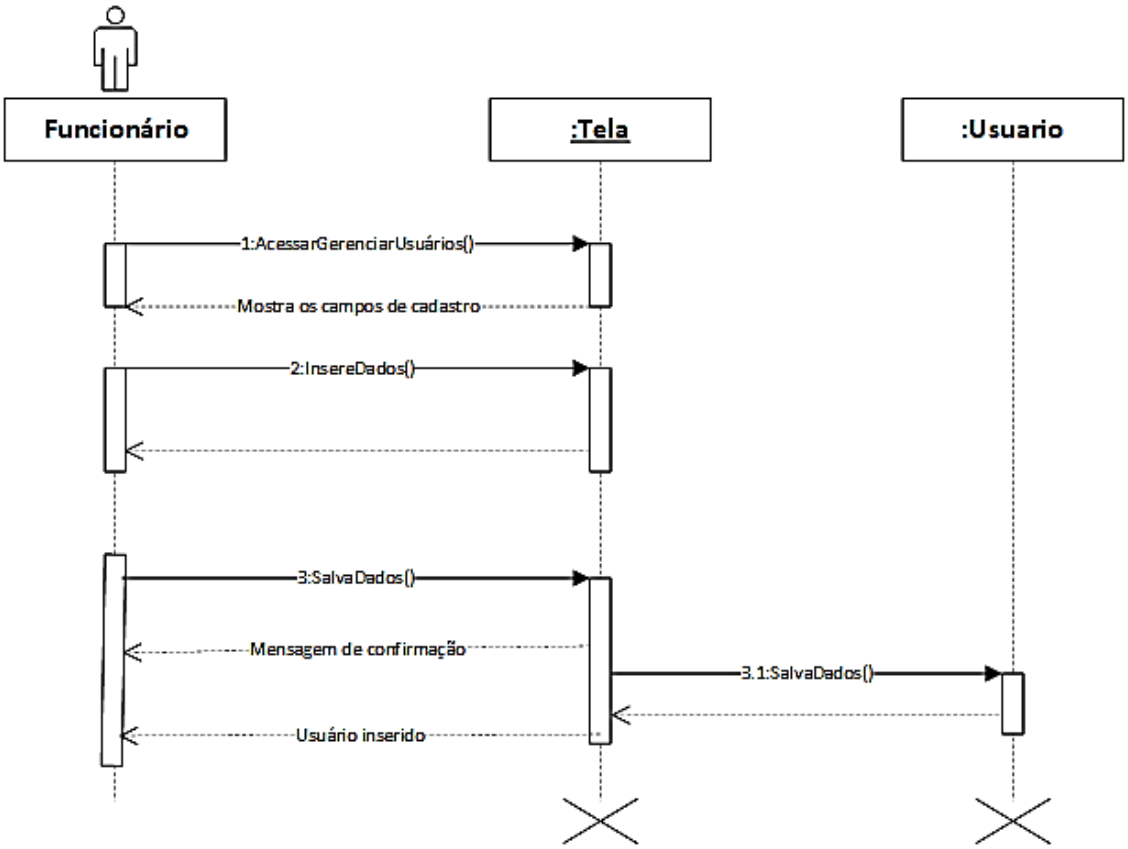


Figura 18 - Cadastrar Usuário

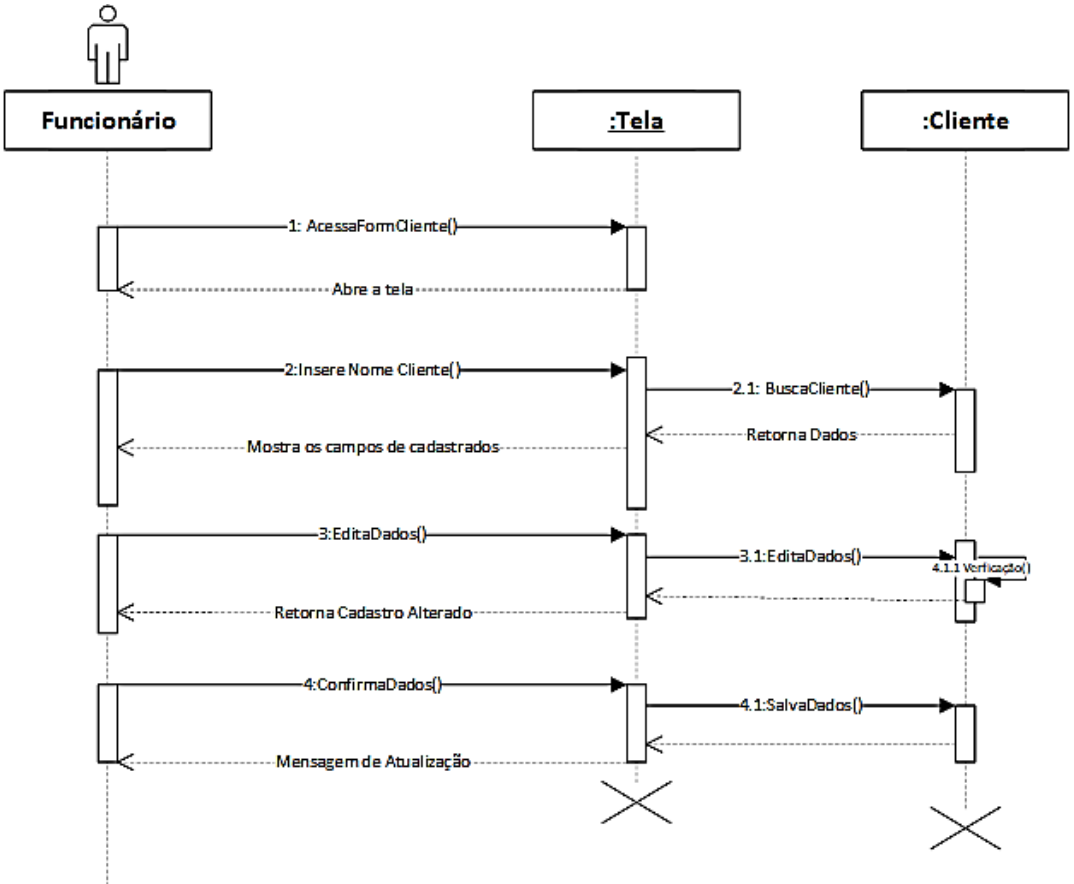


Figura 19 - Atualizar Cliente

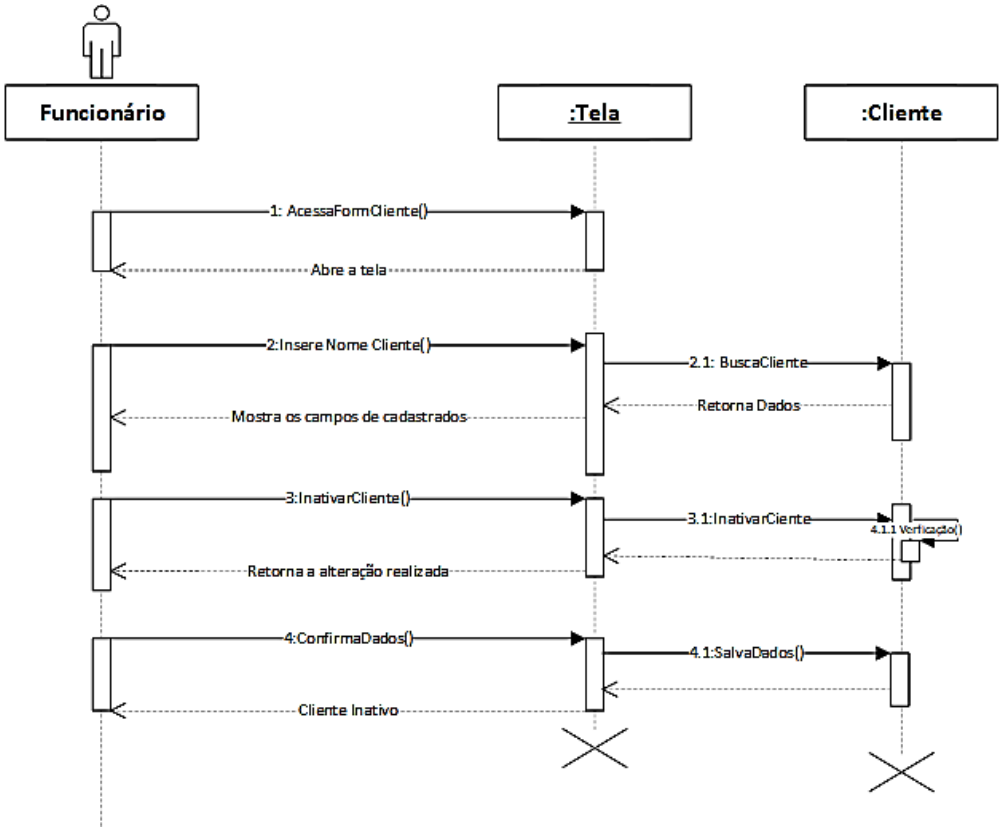


Figura 20 - Inativar Cliente

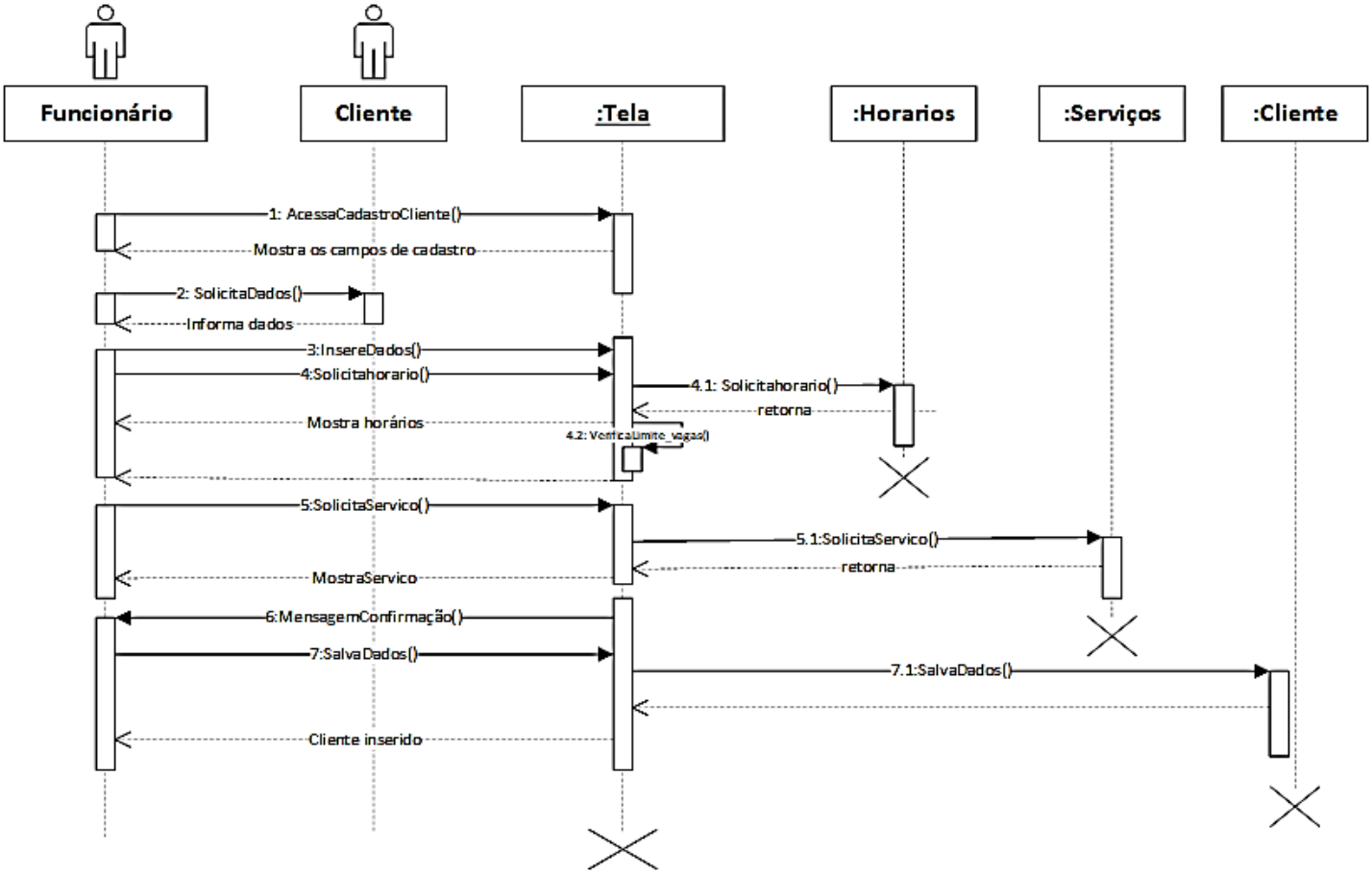


Figura 21 - Cadastrar Cliente

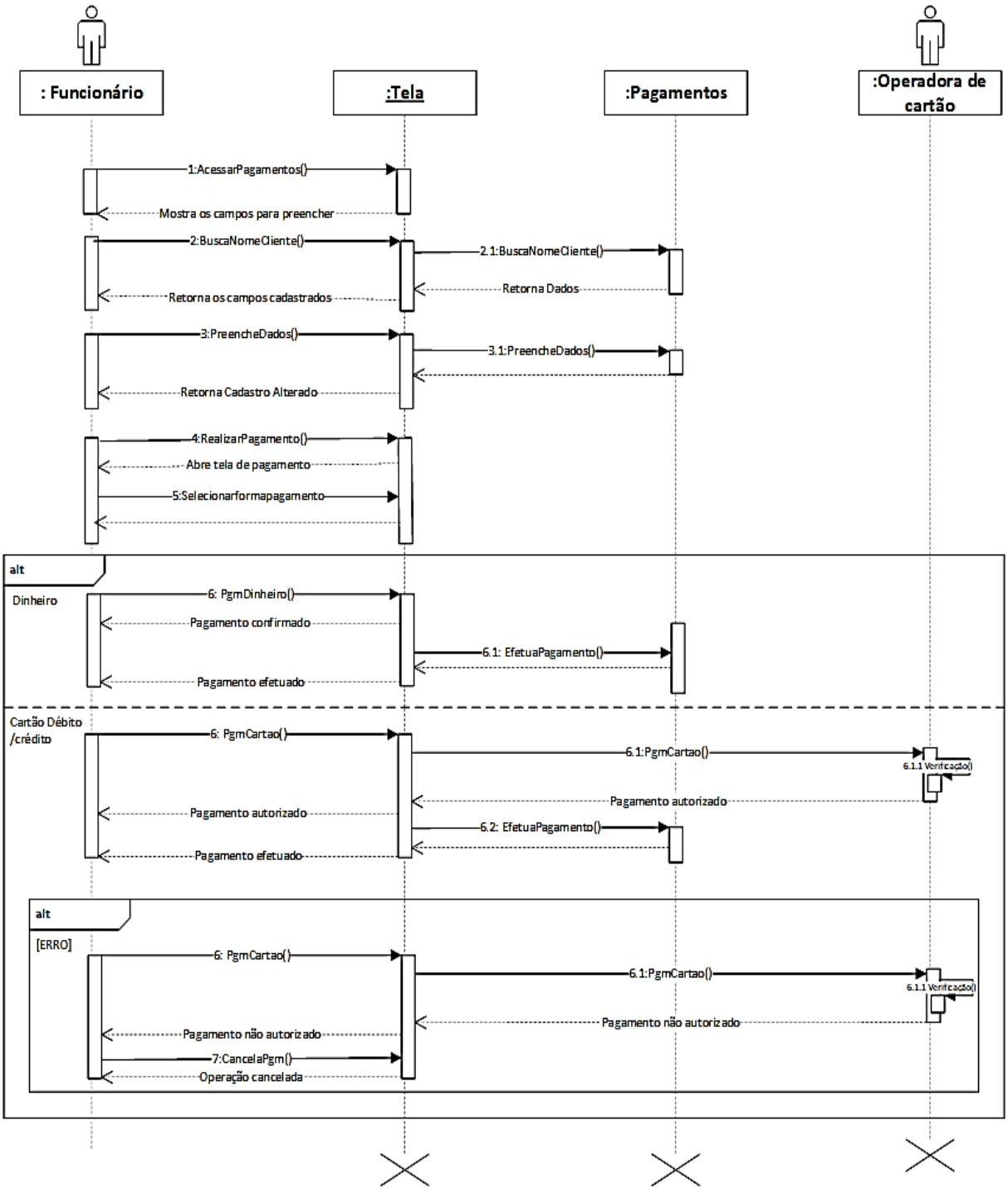


Figura 22 - Realizar pagamento

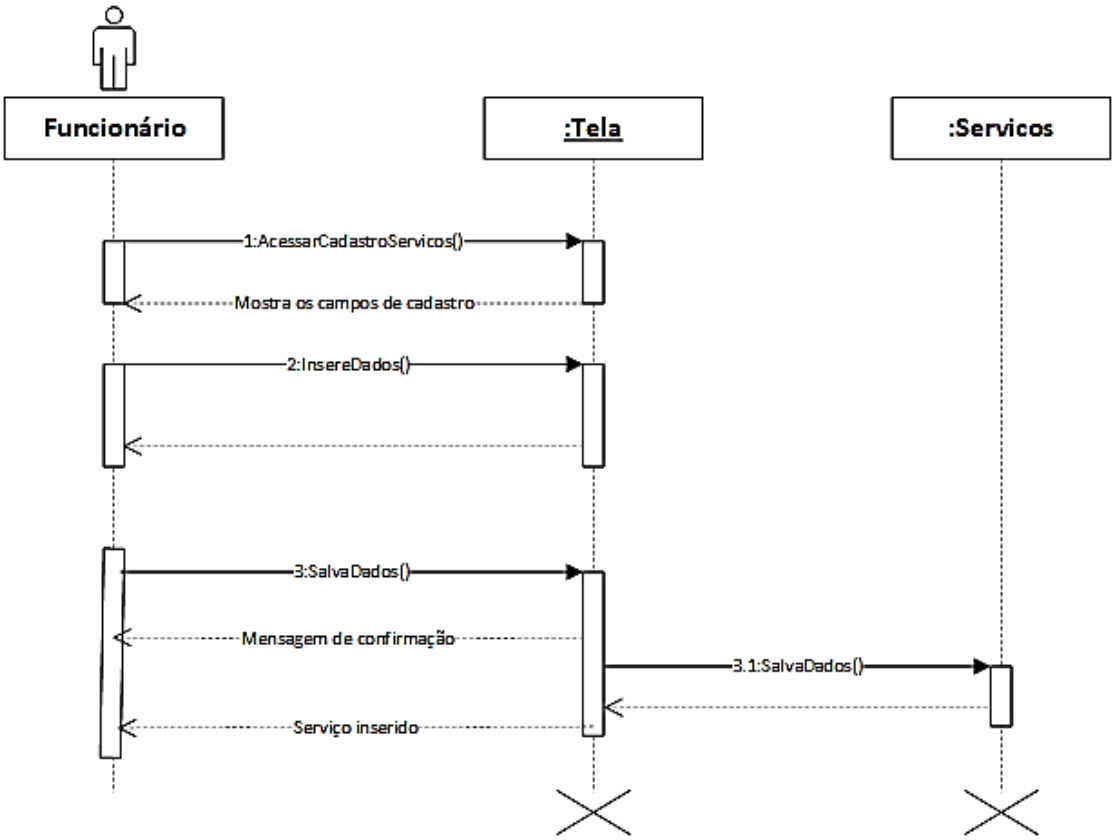


Figura 23 - Cadastrar Serviço

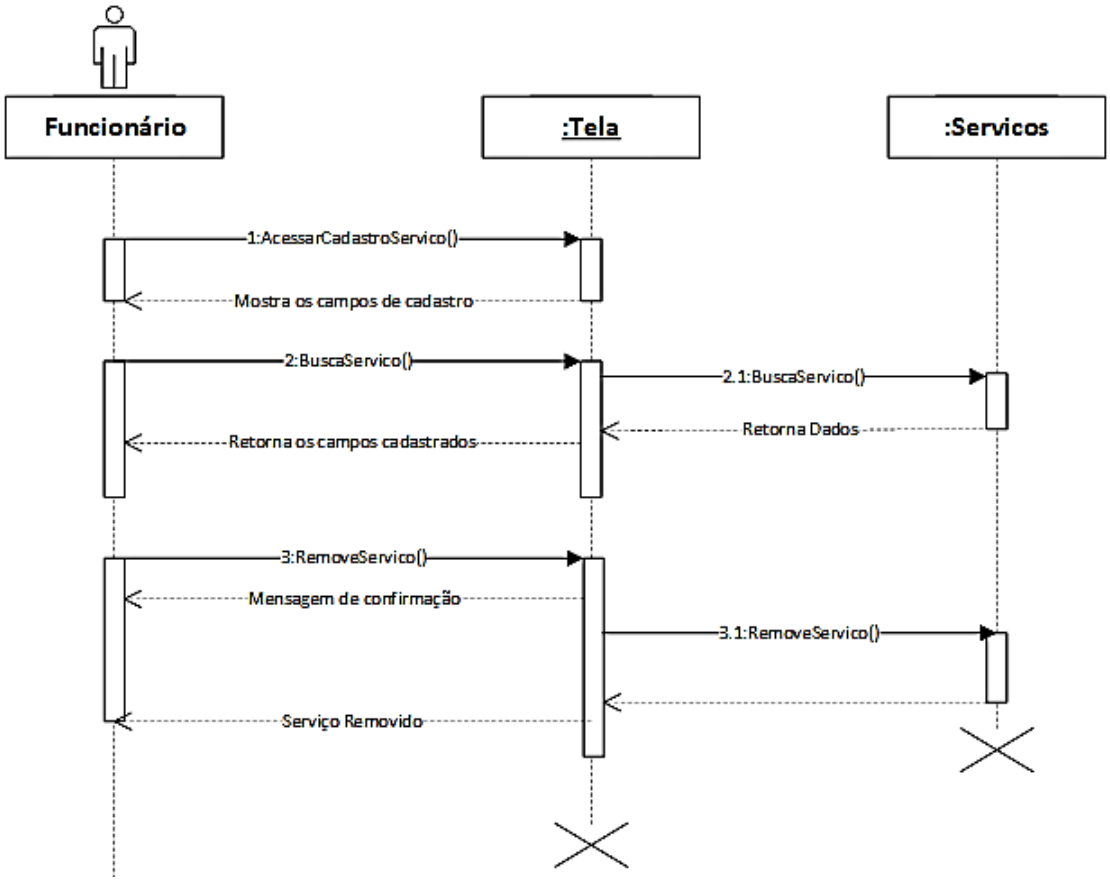


Figura 24 - Serviço Removido

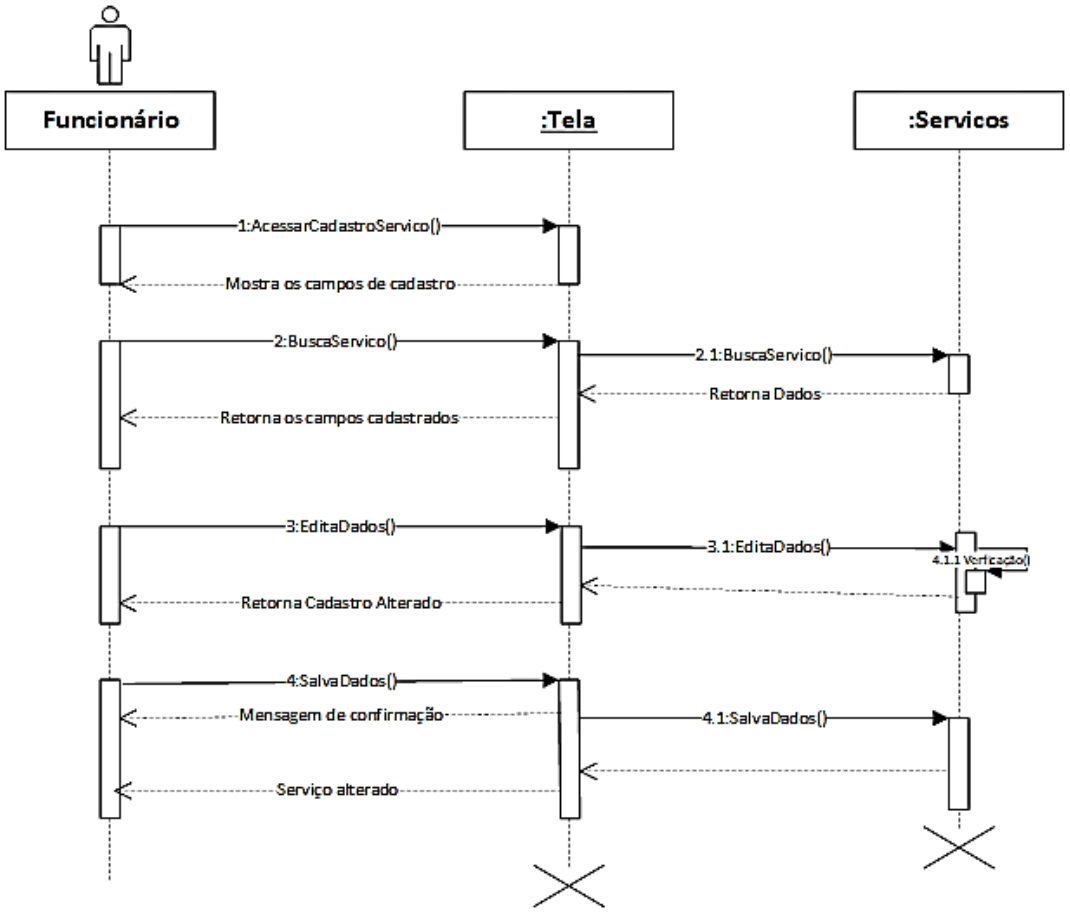


Figura 25 - Serviço alterado

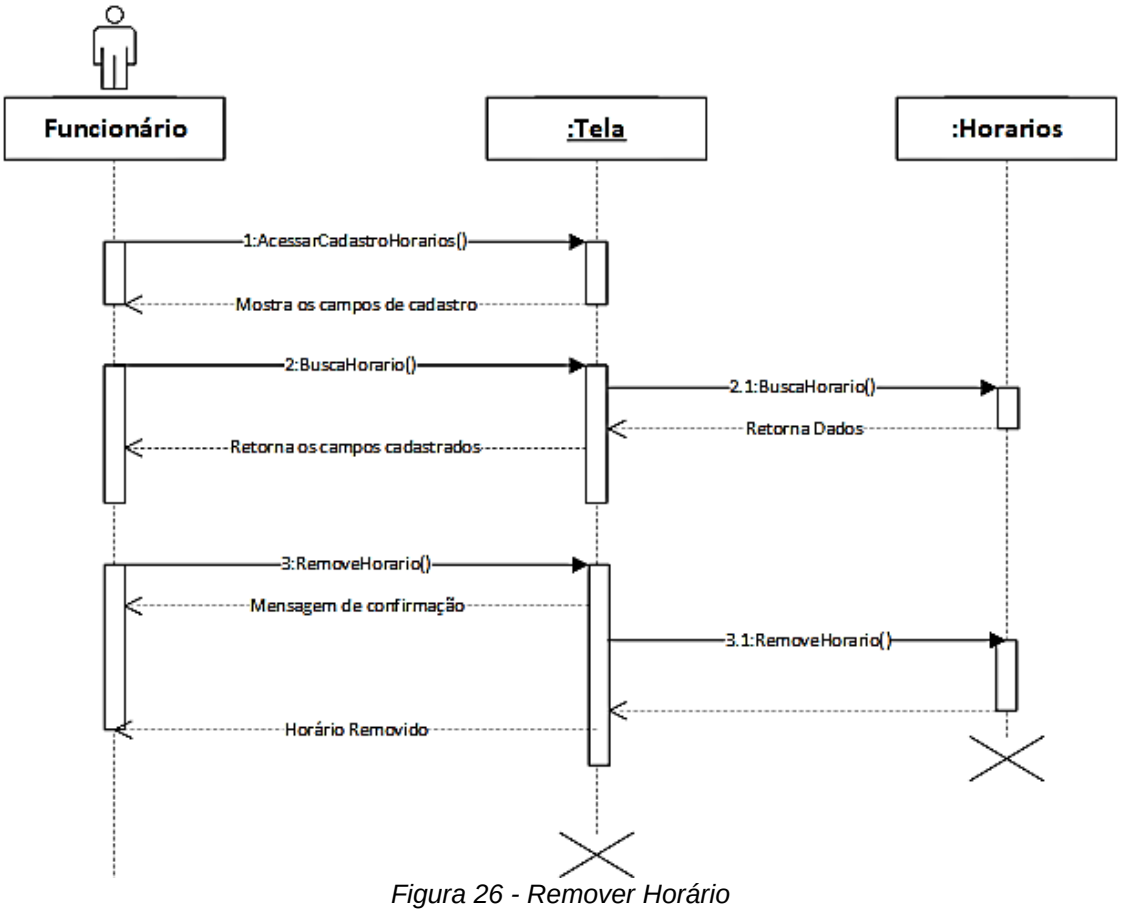


Figura 26 - Remover Horário

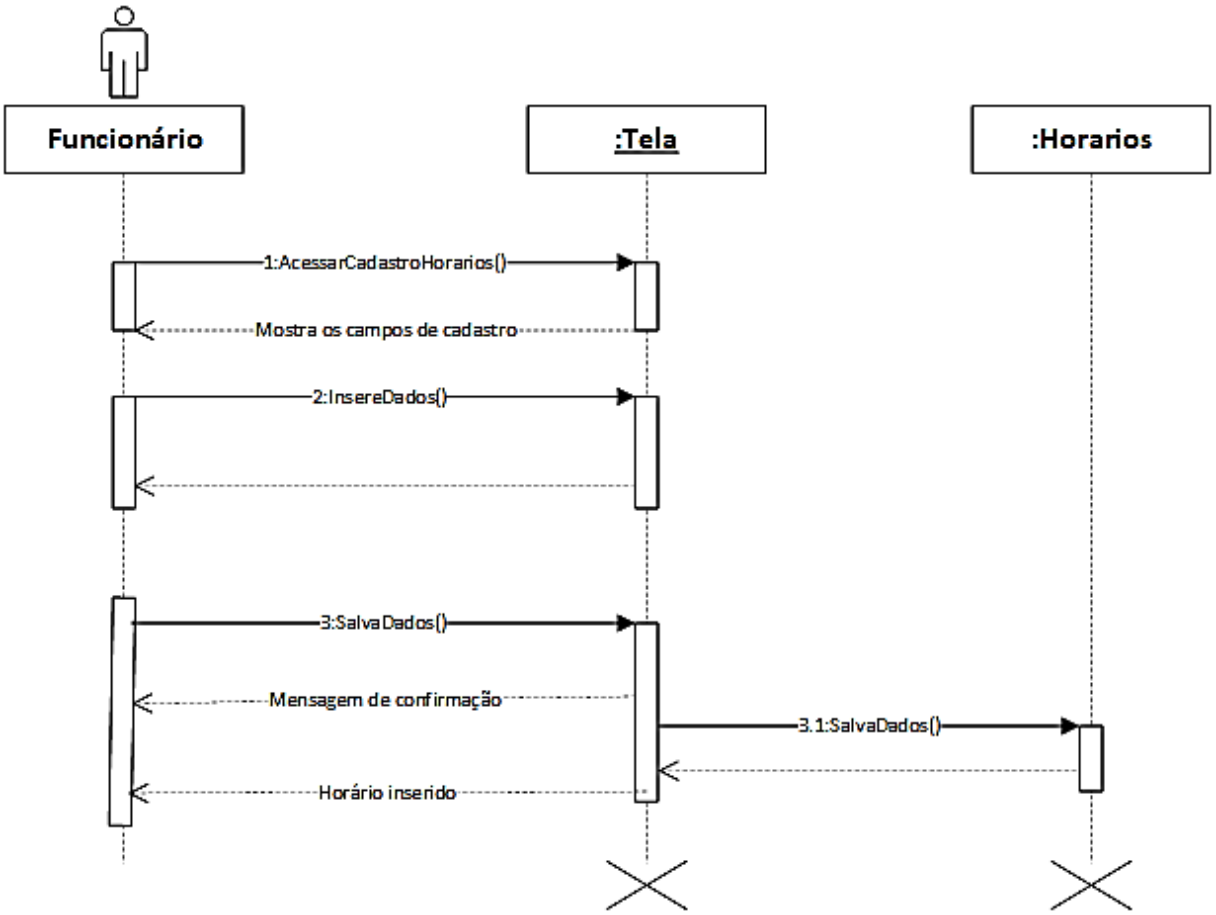


Figura 27- Cadastrar Horário

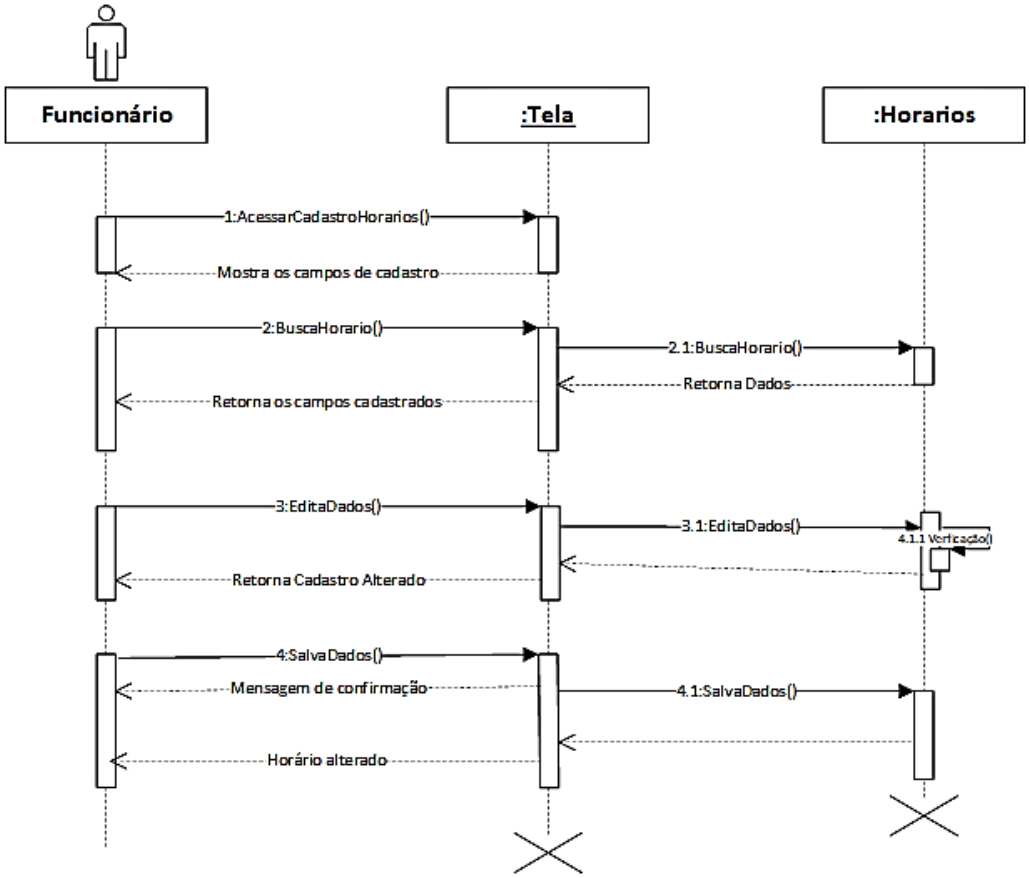


Figura 28- Cadastrar Horário