FACULDADE DE TECNOLOGIA DE AMERICANA

Curso Superior de Tecnologia em Análise e Desenvolvimento de Sistemas

ALEX DOS SANTOS

JOÃO VICTOR BRAYNER ANTUNES DA SILVA

PAULO GABRIEL RONCHINI

VITOR NIIMI NOVO

DESENVOLVIMENTO DE SOFTWARE PARA MEDICINA OCUPACIONAL

Americana, SP

2017

FACULDADE DE TECNOLOGIA DE AMERICANA

Curso Superior de Tecnologia em Análise e Desenvolvimento de Sistemas

ALEX DOS SANTOS

JOÃO VICTOR BRAYNER ANTUNES DA SILVA

PAULO GABRIEL RONCHINI

VITOR NIIMI NOVO

DESENVOLVIMENTO DE SOFTWARE PARA MEDICINA OCUPACIONAL

Relatório técnico desenvolvido em cumprimento à exigência curricular do Curso Superior de Tecnologia em Análise e Desenvolvimento de Sistemas do Prof. Mestre Anderson Luiz Barbosa

Americana, SP

2017

FICHA CATALOGRÁFICA – Biblioteca Fatec Americana - CEETEPS
Dados Internacionais de Catalogação-na-fonte

N843d NOVO, Vitor Niimi

Desenvolvimento de software para medicina ocupacional. / Vitor Niimi Novo ; Paulo Gabriel Ronchini ; Alex dos Santos ; João Victor Brayner Antunes da Silva. – Americana, 2017.

123f.

Monografia (Curso de Tecnologia em Análise e Desenvolvimento de Sistemas) - - Faculdade de Tecnologia de Americana – Centro Estadual de Educação Tecnológica Paula Souza

Orientador: Prof. Ms. Anderson Luiz Barbosa

1 Desenvolvimento de software 2. Medicina ocupacional I. ROCHINI, Paulo Gabriel II. SANTOS, Alex dos III. SILVA, João Victor Brayner Antunes da IV. BARBOSA, Anderson Luiz V. Centro Estadual de Educação Tecnológica Paula Souza – Faculdade de Tecnologia de Americana

CDU: 681.3.05
614.8

ALEX DOS SANTOS
JOÃO VICTOR BRAYNER ANTUNES DA SILVA
PAULO GABRIEL RONCHINI
VITOR NIIMI NOVO

Desenvolvimento de software para medicina ocupacional

Trabalho de graduação apresentado como exigência parcial para obtenção do titulo de Tecnólogo em Análise e Desenvolvimento de Sistemas pelo CEETEPS/Faculdade de Tecnologia – FATEC/ Americana.
Área de concentração: Desenvolvimento de Software.

Americana, 11 de dezembro de 2017.

Banca Examinadora:

Anderson Luiz Barbosa (Presidente)
Mestre
FATEC - Americana

Wladimir da Costa (Membro)
Mestre
FATEC - Americana

Renato Kraide Soffner (Membro)
Doutor
FATEC - Americana

RESUMO

Esse trabalho tem como objetivo de demonstrar o desenvolvimento de um software que seja capaz de automatizar os processos de empresas da área de Medicina Ocupacional. *Scrum* foi escolhido como *framework* de metodologia de desenvolvimento empregada neste trabalho, assim como, auxiliou no desenvolvimento do projeto, pelo fato que o processo de documentação está sujeito a padrões de Normas Regulamentadoras que podem ser alterados com tempo. Além disso, o processo de compreensão do ambiente de negócio da empresa foi realizado de forma técnica, através da aplicação de técnicas de engenharia de software para coleta de requisitos e documentação (UML). Dessa forma, através da formalização das necessidades do cliente, foi realizado o desenvolvimento do banco de dados e do sistema. O processo de desenvolvimento do software consistiu na escolha de ferramentas que auxiliassem a construção da aplicação *web* (PHP, HTML, CSS e JS). Essas tecnologias foram escolhidas com o objetivo de facilitar a utilização da aplicação e a disponibilidade dos dados de forma gerenciável e organizada. Para realizar o desenvolvimento de um software de maior qualidade, foram aplicadas algumas técnicas de testes de software.

Palavras Chave: Desenvolvimento de software; medicina ocupacional.

ABSTRACT

This work has the objective of demonstrating a software development which is capable of automating the business processes of companies that works in the Ocupational Medicine area. Scrum was the development methodology framework employed in this work, as well as helped in the product development, by the fact that the documentation process which is subjected to Medical Regulatory Standards and can be altered at times. In addition, the company business environment understanding process was accomplished via a technical way, through the application of Software Engineering techniques, for requirement surveying and documentation (UML). This way, through formalization of customer needs, the software and database development were accomplished. The software development process consisted in the choice of certain tools which helped in web application building (PHP, HTML, CSS and JS). Such technologies were chosen with the goal to make the application using more user-friendly and data availability in a manageable and organized way. To make the development of a higher-quality system, some software tests techniques were applied.

Keywords: Software development; Occupational Medicine.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE SIGLAS

ASO	Atestado de Saúde Ocupacional
CGI	*Common Gateway Interface*
CSS	*Cascading Style Sheets*
DER	Diagrama Entidade Relacionamento
DSDM	*Dynamic System Development Method*
FDD	*Feature-Driven Development*
HTML	*Hypertext Markup Language*
IMAP	Internet *Message Access Point*
JS	JavaScript
JSP	JavaServer *Pages*
MER	Modelo entidade relacionamento
NNTP	*Network News Transfer Protocol*
NR	Norma regulamentadora
OOPSLA	*Object-Oriented Programming, Systems, Languages & Applications*
PCMSO	Programa de Controle Médico de Saúde Ocupacional
PDF	*Portable Document Format*
PHP	*Hypertext Preprocessor*
POP	*Post Office Protocol*
SNMP	*Simple Network Management Protocol*
UML	*Unified Modeling Language*
XP	*Extreme Programming*

SUMÁRIO

1 INTRODUÇÃO

O objetivo do projeto é desenvolver uma aplicação que atenda às necessidades de empresas que atuam no ramo de medicina ocupacional, no qual atualmente seus controles são realizados de forma manual (planilhas e documentos de texto).

Para alcançar esse objetivo foram realizadas entrevistas com pessoas que atuam na área, pesquisas sobre legislação e análise de documentações entregues pelo cliente, o que permitiu entender o funcionamento das rotinas internas de uma empresa de medicina ocupacional.

A partir disso foi possível desenvolver um sistema para solucionar os grandes problemas para o gerenciamento da empresa, assim, diminuindo o tempo gasto nas atividades realizadas de modo que auxiliam na tomada de decisões, através da geração de relatórios financeiros permitindo a flexibilização na integração entre os departamentos e a gerência.

No que tange o desenvolvimento, o modelo de processo escolhido, em decorrência da rápida mudança nas regras de negócio foi *Scrum*, no qual cada Sprint abrangeu incialmente o desenvolvimento da documentação, e posteriormente foi adaptada para atender o desenvolvimento do projeto.

A plataforma escolhida para o sistema foi a web, com a diversidade de documentação, frameworks dos mais diversos, a compatibilidade e agilidade oferecida tal plataforma, permitiram que o desenvolvimento fosse realizado de forma mais ágil, e com o a possibilidade de integração com diferentes tecnologias para a geração dos relatórios, além de facilitar a instalação e o acesso ao sistema.

Abaixo uma breve descrição dos capítulos abordados na documentação:

Capítulo 2 - Metodologia: Este capitulo apresentou sobre a metodologia de desenvolvimento escolhida para o projeto, assim como suas devidas aplicações e as definições de papéis de responsabilidade.

Capítulo 3 – Cronograma Ágil: O capitulo abordou como as práticas do *framework Scrum* foram aplicadas no cronograma do projeto.

Capítulo 4 - Empresa: Descreve uma breve informação sobre a empresa, a importância da norma regulamentadora (NR-07) e documentações analisadas para realizar o desenvolvimento do projeto.

Capítulo 5 - Levantamento de requisitos: Este capítulo abordou a importância dos processos de Engenharia de Software, a definição de seus conceitos e como tais conceitos foram aplicados aos processos de engenharia para os processos de desenvolvimento.

Capítulo 6 - UML: Neste capitulo foi abordado os conceitos sobre UML, a sua importância no processo de desenvolvimento, os principais diagramas utilizados no projeto e sua aplicação no projeto

Capítulo 7 - Modelagem do banco de dados: Tem por objetivo elucidar os conceitos relacionados sobre banco de dados, além de demonstrar na prática a modelagem realizada no projeto

Capítulo 8 - Heurísticas de Nielsen: Aborda os princípios de usabilidade do sistema, descrição sobre as heurísticas e a sua implementação.

Capítulo 9 - Ferramentas para o desenvolvimento: Neste capitulo foram abordadas as linguagens e as ferramentas utilizados durante o processo de desenvolvimento.

Capítulo 10 - Infraestrutura de tecnologia da informação: Descreve a infraestrutura necessária na empresa para a implementação do produto.

Capítulo 11 - Plano de teste: Aborda sobre o que é teste de software, sua importância e os testes aplicadas no sistema.

Capítulo 12 - Desenvolvimento: São demostradas as principais telas do sistema, além de uma breve descrição sobre cada uma delas.

Capítulo 13 - Resultados: Descreve os resultados alcançados do sistema desenvolvimento.

Capítulo 14 - Considerações finais: Aborda as dificuldades, melhorias e o resultado final do projeto.

2 METODOLOGIA

O capitulo de Metodologia tem como o objetivo descrever de forma teórica e prática sobre a metodologia utilizada no projeto. Para tal, foi necessário realizar um levantamento teórico que será fundamental para compreensão dos conceitos básicos da engenharia de software que levaram a escolha da metodologia de desenvolvimento do projeto. Levando em consideração todas as etapas que o projeto foi submetido, as responsabilidades de cada integrante, as cerimonias realizadas e a importância da metodologia para alcançar o sucesso do projeto.

2.1 Modelos de Processos de Software

Os modelos de desenvolvimento são recursos disponibilizados do resultado do estudo da Engenharia de Software sobre modelos de processos de software. Os modelos de processos apresentam em essência atividades que fazem parte do processo de software, os produtos gerados do software e os papeis dos envolvidos no desenvolvimento (SOMMERVILLE, 2007, p. 8).

Além do fato que os processos representam um conjunto de atividades e resultados que unidos são capazes de produzir as funcionalidades do software. Sommerville (2007, p.8) defende que existem cerca de quatro atividades fundamentais (fases) de processos que são comuns a todo processo de software, são as seguintes:

- **Especificação**: Definição do que será produzido e das restrições para a operação do software;
- **Desenvolvimento:** Etapa em que o software é projetado e programado;
- **Validação:** Etapa de verificação do software em relação ao que foi requisitado pelo cliente;
- **Evolução:** Etapa de adequação as futuras necessidades do cliente;

As atividades dos processos apresentadas necessitam serem bem definidas e analisada, pois, a escolha inadequada do processo de software (metodologia de desenvolvimento) poderá acarretar no sucesso ou fracasso do projeto.

Sommerville (2007, p.8) ainda enaltece que a maioria dos modelos de processos são baseados nos paradigmas do desenvolvimento de software. Em complemento, Pressman (1995, p.32) realça que os paradigmas da Engenharia de Software são os apresentados a seguir no decorrer do capitulo.

2.2 Metodologias Clássicas

As metodologias clássicas foram muito utilizadas entre a década de 60 até 90 no gerenciamento de projetos, especialmente, no desenvolvimento de softwares. Essas metodologias eram recomendadas para projetos considerados estáveis em relação a mudanças, pois os requisitos para produção do produto eram conhecidos.

Além disso, os projetos que utilizam metodologias tradicionais apresentam escopo, prazos, cronogramas e custo (valor final) muito bem definidos. Isso representa o resultado do tempo gasto no planejamento do projeto. Para que esse detalhamento seja possível, os métodos clássicos utilizam ciclos de vida sequenciais e/ou incrementais, os quais são benéficos para projetos curto ou longos que não apresentam mudanças, pois, se houver mudança, o projeto necessita ser todo replanejado para atender aos novos requisitos, assim representando grandes perdas de recursos durante o andamento do projeto.

2.2.1 Modelo em Cascata

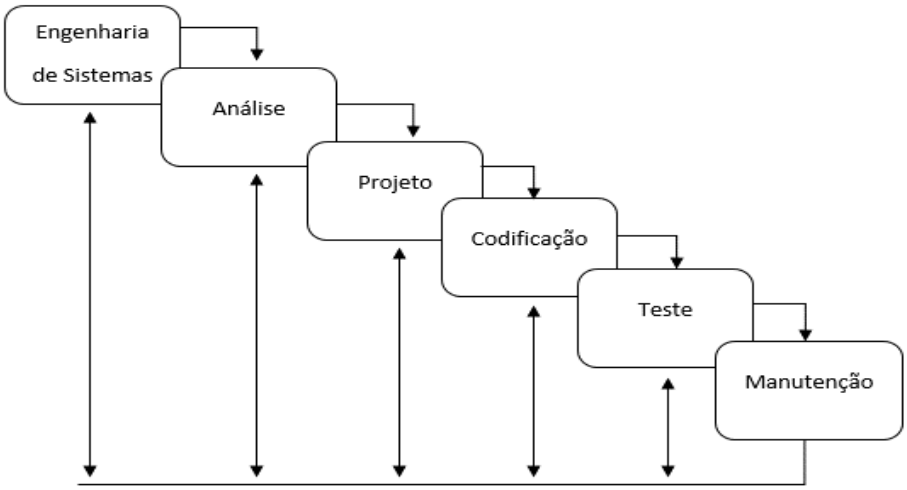
Considerado o modelo de processo mais popularmente utilizado na década de 70, ficou conhecido pela abordagem sequencial e sistemática ao desenvolvimento de software. Representa um modelo em que as fases têm de ser bem definidas e estáveis, pois o modelo não é flexível a mudanças constantes. Além disso, o cliente somente obterá algum resultado após o término do projeto.

Conforme Pressman (1996, p.33-34), as fases do ciclo de vida do modelo podem ser representadas da seguinte maneira:

- Análise e Engenharia de Sistemas: Estabelecimento dos requisitos para todas os elementos do sistema e prossegue com a atribuição de certo conjunto subconjunto desses requisitos ao software;
- Análise de Requisitos de Software: O processo de coleta de requisitos é intensificado e concentrado no software, são documentados e revistos com o cliente;
- Projeto: Comporta os processos de estrutura de dados, arquitetura de software, detalhes procedimentais e caracterização de interface.
- Codificação: O projeto é traduzido para a linguagem especifica da máquina.
- Testes: Realização de testes para verificar a consistências do código gerado, são verificados tanto os aspectos lógicos (interno) como os aspectos funcionais (externos), garantindo a consistência do sistema.
- Manutenção: Pode ocorrer após a entrega do software ao cliente (novos requisitos), ou durante o desenvolvimento através da identificação de erros. Toda mudança necessita ser analisa para acomodar/adaptar o software ao ambiente externo.

O ciclo de vido do modelo em cascata pode ser visualizado na Figura 1.

Figura 1 - Modelo Clássico



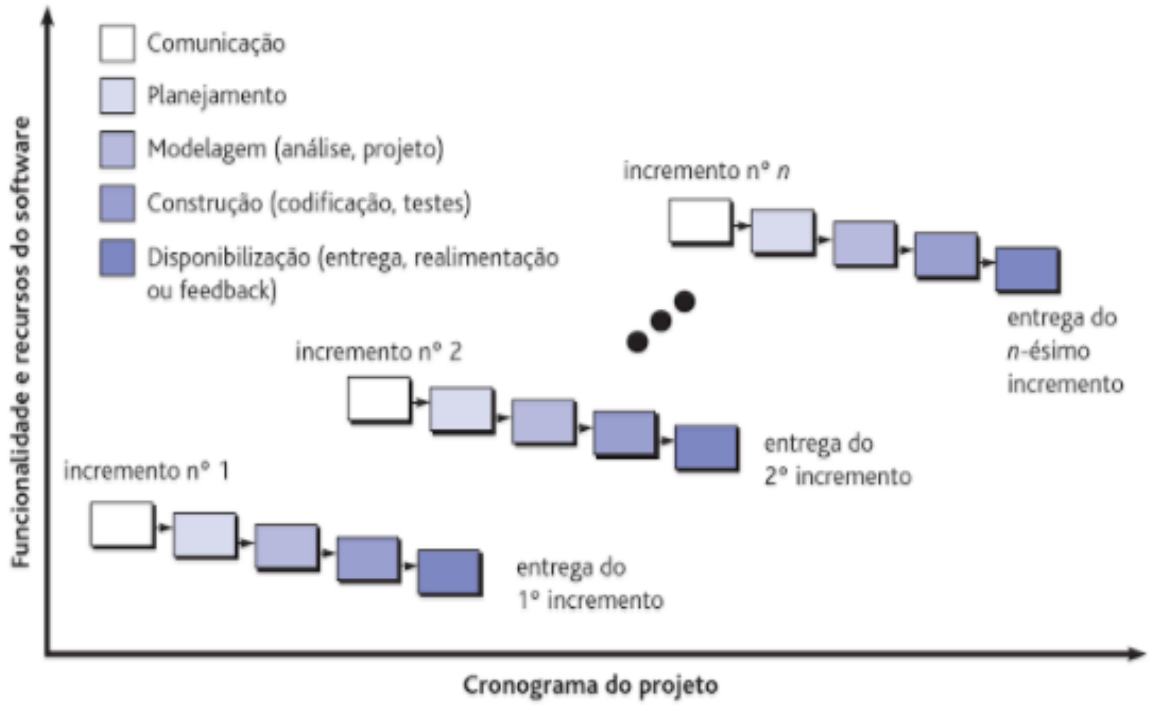
Fonte: Adaptado de Pressman; Maxim (2016).

2.2.2 Modelo Incremental

O modelo de processo incremental é utilizado para o desenvolvimento de software quando é necessário o fornecimento de um determinado conjunto de funcionalidades aos usuários. Pelo fato de combinar os elementos dos processos de sequências lineares (escalonamento) e paralelas, criando assim as sequencias responsáveis por gera os incrementos (produtos) do software que podem ser entregues aos clientes (PRESSMAN; MAXIM, 2016, p.43-44).

As fases do modelo são apresentadas na Figura 2 como Comunicação, Planejamento, Modelagem, Construção e Disponibilização. São disponibilizadas de forma sequencial e escalonada em incrementos, permitindo assim, a eliminação de riscos, comunicação constante com o cliente, entregas rápidas de resultados (MAGELA, 2006, p.31).

Figura 2 - Modelo Incremental



Fonte: (PRESSMAN; MAXIM, 2016, p.44)

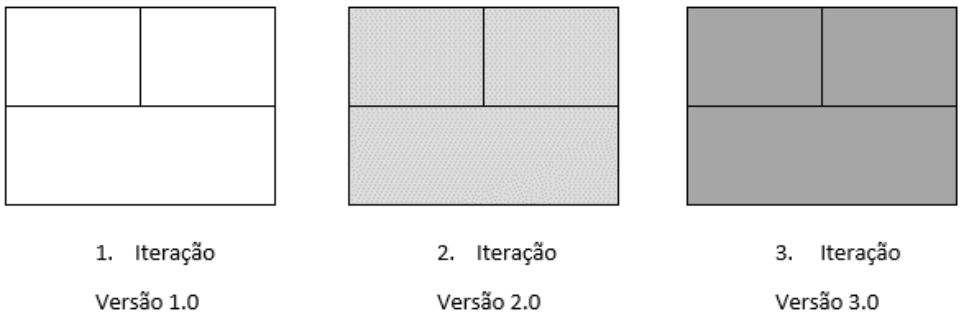
2.2.3 Modelo Iterativo

O modelo iterativo de processos tem o objetivo de aplicar de forma continua todas as atividades do ciclo de vida ao longo de todo o desenvolvimento. Através desse tipo de abordagem a interação com o cliente começa desde o princípio do projeto, assim os riscos podem ser continuamente eliminados durante o desenvolvimento das iterações. No entanto, o fato do modelo permitir a liberação de todas as funcionalidades do sistema de uma única vez, pode acabar deixando a desejar em alguma delas, criando assim, a necessidade mais presente de suporte a cada versão entregue (MAGELA, 2006, p.32).

A

Figura *3* representa o ciclo de vida do modelo iterativo através de versões geradas após o termino de cada iteração que abrange todas as fases do modelo.

Figura 3 - Modelo Iterativo



Fonte: Adaptado de Magela (2006, p.32).

2.2.4 Modelo Iterativo Incremental

O modelo de processo com abordagem iterativa e incremental é baseado no modelo iterativo como no modelo incremental de processos da engenharia de software. A aplicação desse conjunto de incrementos de funcionalidades e iterações graduais torna o modelo menos vulnerável as mudanças durante o desenvolvimento,

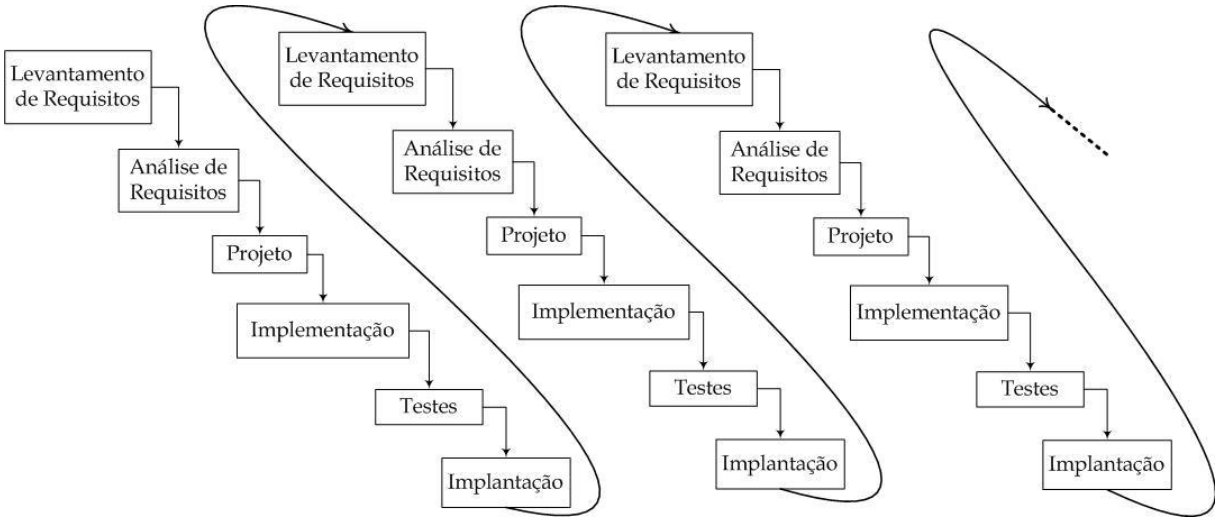
resultante assim em um mecanismo sofisticado de refinamento e abstração que está presente na abordagem do modelo (MAGELA, 2006, p.32-33).

Magela (2006, p.33) ainda apresenta uma breve descrição sobre o ciclo de vida do modelo:

> "[...] podemos estar na atividade de implementação, ainda no ciclo de especificação, e nada impede que tenhamos novos requisitos no ciclo de manutenção. Contudo, o foco e o peso do desenvolvimento vão deslocando-se gradualmente entre os ciclos e entre as atividades".

A Figura 4 representa o ciclo de vida do modelo iterativo incremental de forma didática, pois apresenta onde o modelo iterativo e o modelo incremental é entrelaçado para gerar o modelo em estudo.

Figura 4 - Modelo Iterativo Incremental



Fonte: (DUTRA, 2012).

2.2.5 Modelos Evolucionários

Os modelos evolucionários apresentam uma abordagem que permite trabalhar com projetos complexos e gerar resultados consideráveis em relação as condições especificadas e ao risco inerente ao projeto. Além disso, são modelos que utilizam

iteração no ciclo de vida, possibilitando o versionamento cada vez mais completo a cada iteração concluída (PRESSMAN; MAXIM, 2016, p.83).

Pressman e Maxim (2016) ressaltam dois modelos comuns em processos evolucionários, são os seguintes:

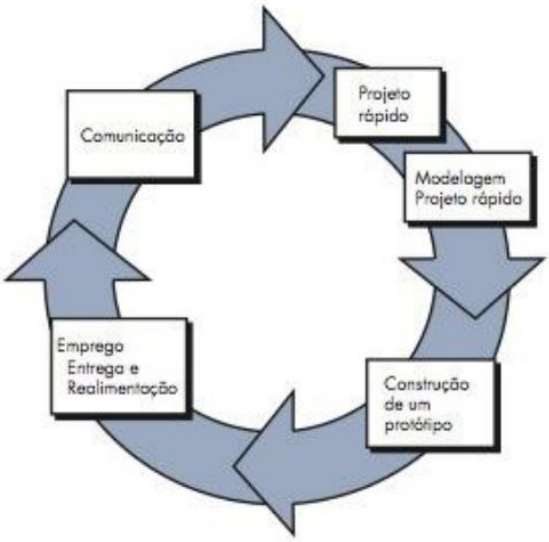
2.2.5.1 Prototipação

O paradigma de prototipação pode ser muito bem empregado quando o cliente tem uma determinada necessidade, mas não tem ideia uma detalhada do que realmente necessita. Neste caso, a criação de um protótipo seria uma ótima escolha, pois, o modelo de prototipação trata de projetos finais que são construídos a partir de protótipos experimentais, que tem o objetivo de coletar os requisitos do cliente (PRESSMAN; MAXIM, 2016, p.87).

O modelo pode ser utilizado isolado (*stand-alone process*) ou com implementação de outros modelos de processo visto anteriormente, pelo fato de que a o paradigma de prototipação auxilia os envolvidos no projeto a compreender o que será desenvolvido, pois trabalha como um mecanismo de compreensão de requisitos do software (PRESSMAN; MAXIM, 2016, p.87).

Para isto, como demonstrado na Figura 5 o ciclo de vida do paradigma inicia com a comunicação tendo o foco na coleta dos requisitos e na definição dos objetivos do projeto, passando rapidamente pela fase de projeto rápido e modelagem que analisarão os aspectos não funcionais (layout), permitindo assim, que a fase de construção do protótipo seja iniciada. Posteriormente, no emprego, entrega e realimentação validar com o cliente se o protótipo atende aos requisitos do cliente (PRESSMAN; MAXIM, 2016, p.87).

Figura 5 - Prototipação



Fonte: (PRESSMAN, MAXIM, 2016, p.87).

2.2.5.2 Modelo em Espiral

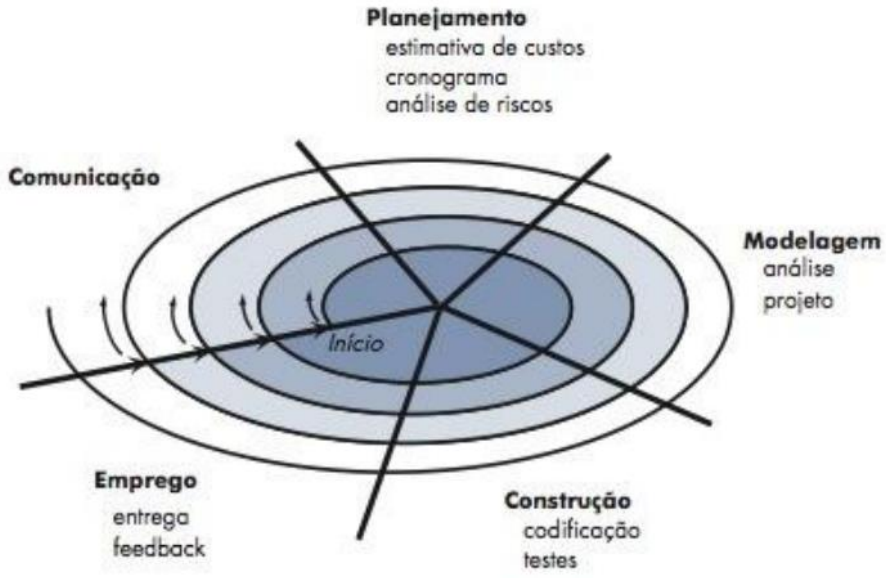
O modelo espiral é uma modelo de processo baseado na natureza iterativa da prototipação, ou seja, apresenta uma abordagem mais realista para o desenvolvimento de software, pelo fato de que o software evoluirá conforme o projeto avança entorno da espiral (versões evolucionarias). Resultando assim, numa melhor comunicação com o cliente, redução de riscos e orçamento. Tudo isto, pois o paradigma permite a aplicação tanto do modelo de prototipação, modelo sequencial como modelo iterativo em sua abordagem (PRESSMAN; MAXIM, 2016, p.89).

A abordagem sequencial permite de foram sistemática a divisão das atividades em etapas. Já a prototipação auxilia na redução de riscos e a coleta de requisitos do cliente através de protótipos. As iterações permitem ao modelo maior flexibilidade de aplicação ao longo de todo ciclo de vida de uma aplicação, podendo o projeto ser considerado iniciado, em andamento, pausado ou em manutenção (PRESSMAN; MAXIM, 2016, p.89).

A Figura 6 representa o ciclo de vida do modelo, onde processo evolucionário inicia no centro da espiral avançando no sentido horário, sendo que a primeira revolução resultará na especificação de produto, as demais, poderão ser resultantes

do desenvolvimento de protótipos, os quais, a cada revolução agregará ao software versões mais sofisticadas do protótipo (PRESSMAN; MAXIM, 2016, p.89), (MAGELA, 2006, p.31).

Figura 6 - Modelo Espiral



Fonte: (PRESSMAN, MAXIM, 2016, p.89)

2.3 Metodologias Ágeis

As metodologias ágeis são baseadas no Manifesto Ágil, assinado por 17 profissionais representantes de métodos especializados no desenvolvimento de software, reuniram-se no Estado de Utah (EUA) em 2001 para discutir e analisar as semelhanças, diferenças, pontos comuns (fortes e fracos) dos métodos utilizados na época.

Tal reunião foi importante para época, pois levou os representantes encontrar um consenso entre seus respectivos métodos e adotarem uma nova abordagem chamada “ágil” (processos leves), que através dos princípios e valores firmados pelo manifesto serve de base para o gerenciamento de projetos de forma mais eficiente e eficaz (CRUZ, 2015, p.79-81).

O Manifesto Ágil (2001) contém princípios ágeis que são seguidos por diversos modelos e boas práticas disseminadas mundialmente, o manifesto busca defender os seguintes fundamentos:

1. Indivíduos e interações mais que processos e ferramentas
2. Software em funcionamento mais que documentação abrangente
3. Colaboração com o cliente mais que negociação de contratos
4. Responder a mudanças mais que seguir um plano

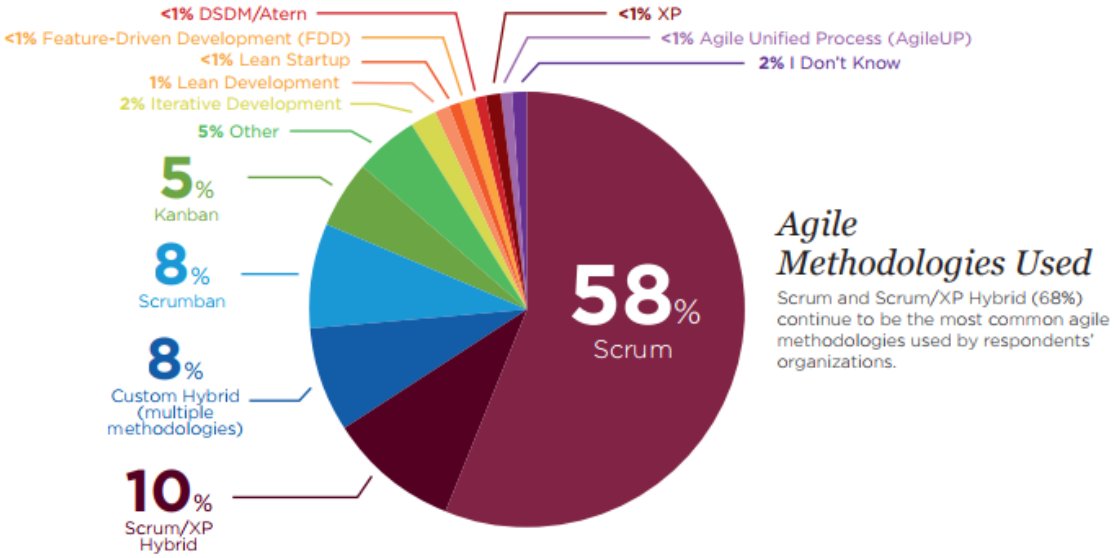
O manifesto representou um marco muito importante para a forma como se desenvolvia software na época, pelo fato de que as ideias apresentadas nesta reunião foram sendo aceitas, adotadas e implementadas nas metodologias de desenvolvimento de projetos que estão no mercado atualmente.

Através dos conceitos provenientes do Manifesto Ágil, novas metodologias foram sendo criadas, algumas delas podem ser visualizadas a seguir:

- *Scrum*
- XP
- DSDM
- FDD
- *Crystal Clear*
- *Adaptive Software Development*

A Figura 7 apresenta o gráfico do relatório de Versionone de 2017 que busca demonstrar a preferência das organizações em porcentagem em relação aos modelos ágeis disponíveis no ano de 2017. No caso, a maioria das organizações optaram por implementar o *framework Scrum* em relação a outros modelos ágeis de desenvolvimento de produtos.

Figura 7 - Métodos e práticas ágeis.



Fonte: (VERSIONONE, 2017, p. 9)

As metodologias ágeis apresentam alguns benefícios, tais como: Escopo atualizados com frequência, equipes multidisciplinares e auto gerenciáveis, ganho na comunicação, inspeções, adaptações e feedbacks constantes, rápida tomada de decisão, redução do risco, produtividade, produtos de acordo com as necessidades do cliente, transparência e visibilidade e, entre outros benefícios (OLIVEIRA, 2003, p. 13-16).

No entanto, para a utilização dessas metodologias, cada projeto deve ser analisado, buscando verificar qual a melhor metodologia ágil que encaixaria no perfil do projeto a ser desenvolvido. Normalmente, são recomendadas para o desenvolvimento de projetos que apresentem alto nível de incertezas (mudanças).

2.4 Framework Scrum

O *Scrum* é um *framework* que foi inicialmente criada em 1990 com o objetivo de auxiliar a gestão do desenvolvimento de produto complexos através da abordagem considerada ágil, que está cada vez mais ganhando aceitação no âmbito de desenvolvimento de software, assim como em outros setores do mercado, pois pode ser aplicado em variados tipos de projetos (SABBAGH, 2014, p. 17, 49).

Um fato interessante é que o nome “*Scrum*” foi baseado numa jogado do rúgbi chamada *Scrummage,* que representa a jogada que dá início ao jogo, objetivando a recolocação da bola em disputa entre os times. Nesta jogada, todos os jogadores do time participam tendo todos o mesmo objetivo, assim, caso um falhe, todos falham. Esta característica é muito presente no *Scrum* pois o trabalho em equipe é fundamental para alcançar o objetivo do projeto (CRUZ, 2015, p. 148).

Através da analogia da jogada, Jeff Sutherland em 1993 enquanto trabalhava na Easel *Corporation* em Massachussets no Estados Unidos, aplicou o *Scrum* em um dos projetos mais complexos (críticos) da organização, resultando assim, no sucesso do projeto. O sucesso foi tão benéfico para a organização que Jeff Sutherland em 1995 apresentou o primeiro Time *Scrum* para Ken Schwaber (CEO da *Advanced Development Methods*), que naquela época estava analisando as vantagens da abordagem empírica para o desenvolvimento de *software* (SABBAGH, 2014, p. 17, 49).

Sendo assim, através da similaridade das ideias de Jeff e Ken, uniram seus esforços para formalizar a definição de *Scrum*, onde em 1995 apresentaram a definição para o mundo no OOPSLA (*Object-Oriented Programming*, *Systems, Languages & Applications* - Congresso de orientação à objetos) (SABBAGH, 2014, p. 17, 49).

Uma grande contribuição para o *Scrum* foi a publicação do artigo “O Novo Jogo no Desenvolvimento de Novos Produtos” (*The New New Product Development Game*) de Nonaka e Takeusgi em 1986. O artigo foi baseado no estudo de diferentes abordagens que as grandes empresas da época aplicavam em suas equipes de desenvolvimento de produtos (SABBAGH, 2014, p. 50).

Através das grandiosas contribuições de Jeff e Ken, foi possível desenvolver o Guia do *Scrum* (“O guia definitivo para o *Scrum*: As Regras do Jogo”), que suas versões são mantidas pelos respectivos criadores e, atualmente, a versão encontra-se na atualização de julho de 2016.

Sendo forma, Schwaber e Sutherland (2016, p.3) apresentam a *framework Scrum* através da seguinte definição:

> *Scrum* não é um processo ou uma técnica para construir produtos; em vez disso, é um *framework* dentro do qual você pode empregar vários processos ou técnicas. O *Scrum* deixa claro a eficácia relativa das práticas de

> gerenciamento e desenvolvimento de produtos, de modo que você possa melhorá-las.

O guia demonstra que o *Scrum* apresenta um conjunto de boas práticas que servem como ferramenta auxiliadora ao desenvolvimento de produtos (projetos), pois permite que outros modelos possam incorporarem a abordagem do modelo, maximizando assim os efeitos da metodologia.

Juntando-se a isso, Schwaber e Sutherland (2016, p. 3) demonstram que o *Scrum* apresenta em sua essência alguns valores fundamentais, são os seguintes:

- Leve;
- Simples de entender;
- Extremamente difícil de dominar.

Quando o Guia *Scrum* aponta que o *framework Scrum* é muito difícil de dominar está fazendo alusão ao fato de que modelo utiliza em seu ciclo de vida diversas técnicas e processos que muitos profissionais pouco experientes poderão ter determinada dificuldade na utilização em seus determinados projetos.

Essa determinada dificuldade pode ser proveniente da estrutura utilizada pelo *Scrum*, pois o framework busca satisfazer um propósito específico, o qual é de fundamental importância para o sucesso do projeto.

A *framework Scrum* foi a metodologia de desenvolvimento de software escolhida para o presente projeto, pelo fato de apresentar uma abordagem ágil, a qual é focada no desenvolvimento de produtos complexos, sendo assim, atende a necessidade do projeto em questão (software) (SABBAGH, 2014, p. 33).

Ademais, o *framework* ainda apresenta uma abordagem aceptível à mudança, ou seja, adaptativa a possíveis mudança que possam ocorrer durante o processo de desenvolvimento do software. Simplesmente, pelo fato de que o projeto está diretamente relacionado ao setor de Medicina Ocupacional, por isso, qualquer mudança de normas (Norma Regulamentadora) que tange as atividades relacionadas aos processos do desenvolvimento do sistema, terão a necessidade de serem adaptados por causas das mudanças que podem ocorrer.

Juntando-se a isso, o presente projeto tem como exigência a formação de uma equipe entre 2 a 4 integrantes, além da escolha de uma metodologia de desenvolvimento de livre escolha da equipe. A escolha do modelo também está relacionada ao cumprimento dessas exigências, pois o *Scrum* comporta equipes pequenas entre 3 a 9 integrantes.

2.4.1 Teoria *Scrum*

A teoria do *Scrum* foi baseada no empirismo, assim confirmando que o conhecimento é proveniente da experiência e do conhecimento adquirido. Dessa forma, pode-se compreender o motivo pelo qual o *Scrum* apresenta uma abordagem iterativa e incremental em relação previsibilidade e o controle dos riscos durante o andamento do projeto (SABBAGH, 2014, p.55).

Para isto, o *Scrum* baseia seu ciclo de vida numa abordagem iterativa e incremental, que permite através da previsibilidade proveniente do empirismo a localização de erros todo momento durante o desenvolvimento. Além de prover versões, que permitem uma previsão de erros que não foram identificados, mas que podem ser analisados e solucionados para a entrega de uma nova versão que cada vez mais atenda às necessidades do cliente (CRUZ, 2015, p. 157).

Segundo Schwaber e Sutherland (2016), o *Scrum* apresenta três pilares fundamentais que estão apoiados no empirismo da metodologia, são os seguintes:

- **Transparência:** Os processos dentro do *Scrum* necessitam ser transparentes, pois estes aspectos dos processos (informação) refletem diretamente na produtividade do Time *Scrum*, pelo fato de que uma comunicação não transparente causaria conflitos, podendo resultar no insucesso do projeto.
- **Inspeção:** A inspeção dos aspectos dos processos busca por variações (problemas), são realizadas com o objetivo de detectar possíveis problemas, para que possam ser sanados através das execuções de correções (adaptações), assim, corrigindo-os desde cedo o andamento do projeto será constante e saudável (CRUZ, 2015, p. 158).

- **Adaptação:** Representa a etapa em que os resultados que foram considerados fora dos padrões pré-estabelecidos, são corrigidos, ou seja, são realizadas adaptações dos aspectos dos processos, permitindo assim, que o processo seja ajustado e possa continuar em andamento (CRUZ, 2015, p. 158).

2.4.2 Valores do *Scrum*

Em essência o *Scrum* demonstra ser simples, pelo de que a *framework* apresenta alguns valores que precisam serem assumidos por todos os interessados no sucesso do projeto. Esses valores são apresentados pelo Guia do *Scrum* da seguinte maneira:

> Quando os valores de comportamento, coragem, foco, transparência e respeito são assumidos e vividos pelo Time *Scrum*, os pilares do *Scrum* de transparência, inspeção e adaptação tornam-se vivos e constroem a confiança para todos. Os membros do Time *Scrum* aprendem e exploram estes valores à medida que trabalham com os eventos, papéis e artefatos do *Scrum* (SCHWABER; SUTHERLAND, 2016, p.5).

Conforme descrito acima, o compromisso e compreensão dos valores do *Scrum* pelos envolvidos no projeto é de suma importância para o sucesso do projeto.

2.4.3 O Time *Scrum*

O Time *Scrum* é composto por três principais papeis de responsabilidade, que são responsáveis por funções/atividades diferentes, mas que trabalham em conjunto para atingir o objetivo do projeto.

De acordo com Schwaber e Sutherland (2016), o Time *Scrum* é constituído pelos seguintes papéis de responsabilidade:

***Product Owner*:**

O *Product Onwer* ou PO (dono do produto), representa o único papel responsável por gerenciar o *Backlog* do Produto, isto é, transmite para o Time de

Desenvolvimento toda a informação referente aos requisitos do cliente através de uma lista de prioridade de requisitos (CRUZ, 2015, p.167).

Além disso, também é responsável por transmitir a visão do produto para todos os envolvidos no projeto, assim, através da compreensão sobre o que é o produto, saberão o que será necessário para o desenvolvimento (SABBAGH, 2014, p. 103-104).

Ademais, é responsável pelo contato entre o cliente, partes interessadas e o Time de Desenvolvimento. Sendo assim, a escolha de um PO que seja acessível é de fundamental importância, pois necessita estar presente para sanar possíveis dúvidas do Time de Desenvolvimento sobre o produto (SABBAGH, 2014, p. 103).

Para o presente projeto o membro do Time *Scrum* escolhido para representar o papel de *Product Onwer* foi um agente externo (representante do cliente). Pelo fato de ser um especialista da área de medicina ocupacional que apresentava a visão completa do produto, assim, foi possível coletar as necessidades do cliente de forma realista e consistente.

***Scrum Master*:**

O *Scrum Master* é o responsável por transmitir ao PO e Time de Desenvolvimento as regras, a teoria e as técnicas do *framework Scrum*. Sendo assim, atua como um facilitador para com os demais papéis do *Scrum*, focando seu trabalho em alinhar o Time *Scrum* com os valores e as práticas ditadas pelo *framework*.

Além disso, pode atuar auxiliando o PO em alcançar um gerenciamento efetivo do *Product Backlog*, na transmissão da Visão do Produto e dos itens do *Backlog* do Produto para Time de Desenvolvimento, na criação e no refinamento do *Backlog* do Produto, na compreensão e prática da agilidade, eventos do *Scrum* e planejamento do produto (SCHWABER; SUTHERLAND, 2016, p.7).

Já com o Time de Desenvolvimento, busca auxiliar no treinamento para alcançar o autogerenciamento, na remoção de impedimentos, na aplicação, compreensão dos eventos do *Scrum* pelo Time de Desenvolvimento e na criação de produtos de alto valor (SCHWABER; SUTHERLAND, 2016, p.7).

E, em relação a organização, busca auxiliar liderando e treinando os envolvidos no projeto da organização a adotar as práticas do *Scrum*, aplicação de técnicas que gerem produtividade ao Time *Scrum* e na comunicação para obtenção de *feedbacks* (SCHWABER; SUTHERLAND, 2016, p.7).

Sendo assim, para o projeto em questão, o membro do Time *Scrum* escolhido para representar o papel de *Scrum Master* foi o integrante que apresentava conhecimentos relacionados ao *framework Scrum*. Dessa forma, através do estudo do paradigma escolhido, buscou conscientizar e auxiliar os demais membros do Time *Scrum* a entenderem os conceitos relacionados ao *framework*. Além de trabalhar como facilitador de impedimentos que atrapalhassem o andamento do projeto e como auxiliador do PO para o gerenciamento do *Product Backlog.*

Time de Desenvolvimento:

O Time de Desenvolvimento é composto por uma equipe multidisciplinar e auto gerenciável, qual tem o objetivo de executar as tarefas provindas do *Backlog* do Produto afim de construir produtos entregáveis em cada Sprint.

De acordo com Schwaber e Sutherland (2016, p. 6), os Times de Desenvolvimento apresentam as seguintes características:

- Auto organizados;
- Multifuncionais (membros especializados e experientes);
- Multidisciplinar (domínios específicos de conhecimento);
- Não há títulos para os membros da equipe;
- Criam o incremento do Produto.

São consideradas equipes multifuncionais e autônomas, pelo fato de que o time escolhe a forma de realizar as tarefas (auto organizáveis), normalmente, de forma colaborativa. Está autonomia, permite ao time uma determinada liberdade de trabalho, pois os membros do time apresentam competências e experiências técnicas suficientes para concluir as atividades que constituem o produto.

Além do fato que os Times de Desenvolvimento *Scrum* não apresentam títulos, pois, são auto gerenciáveis e trabalham de forma colaborativa. Dessa forma, todos os méritos das conquistas não são atribuídos apenas a um indivíduo, mas, para o Time *Scrum*, todos os envolvidos diretamente no desenvolvimento do produto.

Sabendo disso, os integrantes do presente projeto que representaram os membros do Time de Desenvolvimento*,* foram escolhidos pelo fato de que possuíam conhecimentos técnico para o desenvolvimento do produto (software). Sendo assim, através das habilidades e conhecimentos de cada membro da equipe, foi possível formar uma equipe de desenvolvimento multidisciplinar que fosse capaz de produzir em tempo hábil um produto que atendasse as necessidades dos clientes.

A Tabela 1 demonstra as divisões dos papéis do *Scrum* e os respectivos membros responsáveis:

Tabela 1 - Papéis do *Scrum*

Nome	**Papel de responsabilidade**
Alex dos Santos	Time de Desenvolvimento
João Victor B. A. da Silva	*Scrum Master* / Time de Desenvolvimento
Paulo Gabriel Ronchini	Time de Desenvolvimento
Vitor Niimi Novo	Time de Desenvolvimento
Representante do cliente	*Product Onwer*

Fonte: Elaborado pelos autores

Como apresentado na Tabela 1, um dos integrantes do projeto representou e atuou como o *Scrum Master* e membro do Time de Desenvolvimento. Isto, pelo fato que o presente projeto apresenta a exigência de que todos os membros da equipe de estudantes façam suas respectivas contribuições ao desenvolvimento do software (programação).

2.4.4 Eventos *Scrum*

Os eventos são utilizados com o intuito de criar uma rotina que tenha o objetivo de minimizar o risco de cerimônias e entregas fora do planejado. Além disso, os eventos são considerados *time-boxed*, ou seja, todos os eventos necessitam ter uma data muito bem definida de início e término; não podendo ultrapassar ou diminuir a duração estipulada, garantindo assim, que tempo seja gasto de forma produtiva. (SABBAGH, 2014, p. 213).

Além disso, o *framework* define que os eventos que formam uma *Sprints* são como blocos de eventos (*containers*), que segundo o Guia do *Scrum*, os eventos podem ser definidos da seguinte maneira:

> Cada evento no *Scrum* é uma oportunidade de inspecionar e adaptar alguma coisa. Estes eventos são especificamente projetados para permitir uma transparência e inspeção criteriosa. A não inclusão de qualquer um dos eventos resultará na redução da transparência e da perda de oportunidade para inspecionar e adaptar (SCHWABER; SUTHERLAND, 2016, p.8).

2.4.4.1 Sprints

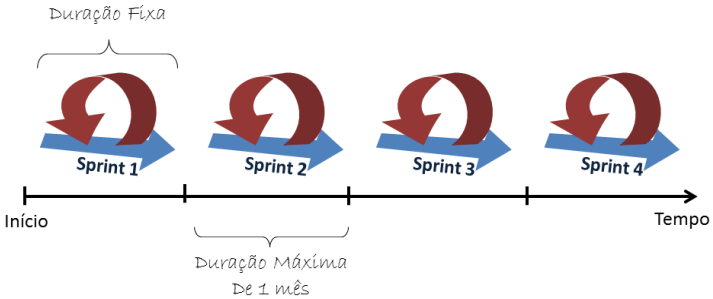
De acordo com o Guia do *Scrum*, as *Sprints* podem ser definidas da seguinte maneira:

> O coração do *Scrum* é a *Sprint*, um *time-boxed* de um mês ou menos, durante o qual um "Pronto", versão incremental potencialmente utilizável do produto, é criado. *Sprints* têm durações coerentes em todo o esforço de desenvolvimento. Uma nova *Sprint* inicia imediatamente após a conclusão da Sprint anterior (SCHWABER; SUTHERLAND, 2016, p.8).

Sendo assim, compreende-se que as *Sprints* apresentam duração bem definidas (*time-boxed*) e, que ao término de cada *Sprint* pode-se analisa o produto proveniente dos eventos internos da *Sprint*, se caso, o resultado for satisfatório como o planejado ou, se necessita de melhorias, deverá ser analisada essas informações resultantes, para que assim, inicie uma nova *Sprint*.

A Figura 8 representa como funciona o processo das *Sprint*, onde todas as *Sprints* apresentam o mesmo tempo de duração, aproximadamente de 2 a 4 semanas. Dessa forma, quando a *Sprint* 1 for finalizada automaticamente inicia-se a *Sprint* seguinte. O tempo de execução das *Sprints* (*time-boxed*) necessita ser estabelecido no início do projeto, além de considerar que a quantidade de *Sprints* atenda as condições para construir o produto do projeto.

Figura 8 - Iterações das *Sprints*



Fonte: (VIERIA, 2017).

Para o presente projeto foram utilizadas *Sprints* de duas semanas de duração. A escolha do *timeboxe* está relacionada a necessidade de satisfazer a data final da entrega do TCC, a quantidade de tarefas que seriam desenvolvidas e o tempo disponível de trabalho de cada integrante do Time *Scrum*. A aplicação das Sprint no projeto será abordada no capitulo CRONOGRAMA AGIL.

2.4.4.2 Reunião de Planejamento da *Sprint*

A Reunião de Planejamento ou *Sprint Planning Meeting*, é uma reunião com tempo de duração determinado, em casos de *Sprints* de duração de um mês, as reuniões em média apresentam 4 horas de duração e, em casos de *Sprints* menores, a duração também terá um tempo de duração menor. Além disso, esta reunião em especial, representa um papel muito importante para a *Sprint*, pois, é através dela que será planejado todo o trabalho que será realizado pelo Time *Scrum* (SCHWABER; SUTHERLAND, 2016, p.9).

Ainda seguindo este ponto de vista, Schwaber e Sutherland (2016, p.9) ressalta que as reuniões de planejamento da *Sprint* são realizadas com todo Time *Scrum*, onde o *Scrum Master* é o responsável dar assistência a equipe, o PO por garantir que os demais participantes entendam o propósito da *Sprint* (Meta), demonstrando com o Time de Desenvolvimento como precisa agir para atingir a Meta da *Sprint*.

Dessa forma, segundo o Schwaber e Sutherland (2016, p. 9), para manter os participantes alinhados com o propósito da *Sprint*, nas reuniões de planejamento são feitas as seguintes perguntas:

- Qual é o objetivo da *Sprint*?
- O que pode ser entregue como resultado do incremento da próxima *Sprint*?
- Como o trabalho necessário para entregar o incremento será realizado?

O objetivo dessas perguntas é enfatizar a previsibilidade de problemas, dessa forma, permitindo que todo o time esteja ciente das ações a serem tomadas em

relação ao caminho para atingir a meta da *Sprint*, eliminando assim empecilhos que atrapalham o sucesso do projeto.

No projeto as Reuniões de Planejamento das *Sprint* foram realizadas em todo o início de cada *Sprint*. Nestas reuniões os integrantes analisavam os *feedbacks* dos evolvidos no projeto e os itens do *Backlog* da *Sprint* que iniciava. Dessa forma, permitia que cada membro do Time de Desenvolvimento pudesse opinar sobre as tarefas que iriam ser executados na *Sprint*.

2.4.4.3 Reunião Diária

As reuniões diárias do *Scrum* têm horários bem definidos, normalmente reuniões rápidas de 15 minutos e, são realizadas antes da inicialização dos trabalhos. Através desta reunião o Time de Desenvolvimento pode inspecionar a situação do desenvolvimento, assim como, verificar existência de problemas que podem resultar na perda de produtividade ou, o não atendimento do prazo e meta da Sprint (CRUZ, 2015, p. 191-192).

De acordo com Schwaber e Sutherland (2016, p.11), existem algumas perguntas que esclarecem e facilitam a identificação de empecilhos, são as seguintes:

- O que eu fiz ontem que ajudou o Time de Desenvolvimento a atender a meta da Sprint?
- O que eu farei hoje para ajudar o Time de Desenvolvimento atender a meta da Sprint?
- Eu vejo algum obstáculo que impeça a mim ou o Time de Desenvolvimento no atendimento da meta da Sprint?

As respostas dessas questões permitem mensurar se o Time de Desenvolvimento está progredindo em direção ao objetivo da Sprint. As reuniões diárias são importantes pois buscam melhorar a comunicação interna, resolução de empecilhos, rápida tomada de decisão, eliminam reuniões desnecessárias e a experiência dos participantes do Time *Scrum* (SABBAGH, 2014, p. 116).

No seguinte projeto as Reuniões Diárias foram realizadas constantemente dentro do ambiente acadêmico (biblioteca, sala de aula e laboratórios) e externamente

(meio digital) para verificação e discussões relacionadas a situação do projeto e de possíveis impedimentos.

2.4.4.4 Revisão da Sprint

A reunião de Revisão da *Sprint*, é executada no final de cada *Sprint* com o intuito de obter o *feedback* do Time *Scrum*, cliente e as partes interessadas sobre o incremento do produto gerado pela Sprint. Esta reunião normalmente tem duração de 4 horas para Sprint de um mês e menos para *Sprints* de menor duração (SABBAGH, 2014, p. 220).

Juntando-se a isso, na reunião de revisão são levantados comentários em relação ao incremento do produto da Sprint, os *feedbacks* serão uteis para que o PO possa coletar as lições aprendidas e utiliza-las para atualizar o *Backlog* do Produto e implementar melhorias nas futuras *Sprints* (SABBAGH, 2014, p. 221).

Segundo o Guia do *Scrum*, as reuniões de revisão das *Sprints* quando finalizadas apresenta os seguintes resultados:

> O resultado da Reunião de Revisão da *Sprint* é um *Backlog* do Produto revisado que define o provável *Backlog* do Produto para a próxima Sprint. O *Backlog* do Produto pode também ser ajustado completamente para atender novas oportunidades (SCHWABER, SUTHERLAND, 2016, p.12).

Conforme apresentado os resultados provindos das Reuniões de Revisões das *Sprints* apresentaram como resultado o *Backlog* do Produto revisado e alinhado para a inicialização da nova Sprint em conformidade com o objetivo do projeto.

No projeto foram realizadas essas reuniões com o intuito de mostrar a todos os envolvidos no projeto os resultados alcançados, com o objetivo de coletar *feedbacks* para possíveis correções do produto. Posteriormente, o Time *Scrum* se reunia para discutir e analisar se o objetivo da *Sprint* foi alcançado, e se organizarem para realizar as devidas correções durante o decorrer da semana.

2.4.4.5 Retrospectiva da Sprint

A Retrospectiva da *Sprint,* é mais um evento fundamental, pois é realizado após a Revisão da *Sprint*, ou seja, ao termino de cada ciclo. Esta reunião tem tempo aproximado de 3 horas para *Sprints* de um mês, onde o *Scrum Master* atua como um facilitador, auxiliador dos demais participantes do Time *Scrum* a compreender o propósito do evento (SABBAGH, 2014, p. 225).

Em complemento, o Guia do *Scrum* apresenta uma definição em relação ao propósito da retrospectiva da *Sprint*:

> Inspecionar como a última *Sprint* foi em relação às pessoas, aos relacionamentos, aos processos e às ferramentas; identificar e ordenar os principais itens que foram bem e as potenciais melhorias; e, criar um plano para implementar melhorias no modo que o Time *Scrum* faz seu trabalho (SCHWABER; SUTHERLAND, 2016, p.12).

Através da orientação do *Scrum Master* para com os participantes da reunião de retrospectiva, melhorias são identificadas e implementadas nas demais iterações através de um plano de melhorias, aplicando-as no *Backlog* do Produto refletira diretamente no *Backlog* das futuras *Sprints*, assim, aplicando o conceito de melhoria continua através das lições aprendidas dos ciclos anteriores.

Neste projeto as Reuniões de Retrospectiva foram realizadas para analisar se o objetivo da *Sprint* foi alcançado, além de verificar os *feedbacks* coletados nas Reuniões de Revisões com os envolvidos. Pois, através da análise das lições aprendidas com a *Sprint* foi possível promover melhorias que fossem úteis para as futuras *Sprints*.

2.4.5 Artefatos *Scrum*

Para Schwaber e Sutherland (2016, p.13), os artefatos podem ser interpretados da seguinte forma:

> Representam o trabalho ou o valor para o fornecimento de transparência e oportunidades para inspeção e adaptação. Os artefatos definidos para o *Scrum* são especificamente projetados para maximizar a transparência das informações chave de modo que todos tenham o mesmo entendimento dos artefatos.

Conforme dito, os artefatos têm o principal objetivo de manter todo o Time *Scrum* informado através da transparência das informações que poderão resultar em rápida tomada de decisão por parte do Time *Scrum*, gerando adaptações e inspeções de incrementos, além de manter o *Backlog* do Produto sempre tangível.

2.4.5.1 Backlog do Produto

O *Backlog* do Produto segundo Schwaber e Sutherland (2016, p.13), pode ser definido da seguinte forma:

> O *Backlog* do Produto é uma lista ordenada de tudo que deve ser necessário no produto, e é uma origem única dos requisitos para qualquer mudança a ser feita no produto. O *Product Owner* é responsável pelo *Backlog* do Produto, incluindo seu conteúdo, disponibilidade e ordenação.

Seguindo este ponto de vista, compreende-se que único responsável por manter e classificar em prioridades os itens do *Backlog* do Produto é o PO. Pois através desse papel que as novas mudanças serão coletadas do Cliente do projeto, realocadas e/ou negociadas com o Time de Desenvolvimento, como o intuito de manter o *Backlog* do Produto sempre atualizado e informado.

Para que o projeto apresente um *Backlog* do produto alinhado como as necessidades de negócio do cliente, o PO utiliza alguns processos de ordenamento dos itens gerados, ou seja, o *Backlog* do Produto é construído de acordo com o grau de importância do item (funcionalidades, valor e custo) que serão desenvolvidos pelo Time de Desenvolvimento (SABBAGH, 2014, p. 113).

Em resumo, o *Backlog* do produto representa um conjunto de itens que representam as necessidades do cliente (requisitos), que passarão por um processo de ordenamento baseado no retorno do investimento do cliente e nas funcionalidades do produto, que são geradas através da negociação entre PO e Time de Desenvolvimento na *Sprint Planning*.

A Figura 9 representa o ordenamento que o PO realiza no *Backlog* do Produto, onde os itens localizados no topo têm apresentam maior prioridade que os abaixo.

Isto, por que o ordenamento é realizado de acordo como o conceito de tempo, custo e conhecimento necessários para o Time de Desenvolvimento construir determinado produto. Para isto, o PO pode utilizar de medidas relativas, que não estão mencionadas no *Scrum*, mas que auxiliam a equipe de desenvolvimento a mensurar o que será gasto em cada item de tarefa.

Figura 9 – *Backlog* do Produto (*Product Backlog*)

Fonte: (VIEIRA, 2017)

Para que o PO construir um *Backlog* do Produto semelhante da Figura 9, não necessita ter conhecimentos avançados em cálculos e estatística avançadas para realizar estimativas dos itens do *Backlog* do Produto, nem mesmo o Time de Desenvolvimento.

Nestes casos, o *Scrum* permite que técnicas simples de estimativas possam ser adaptadas para realidade do projeto, tais como: *Story Poin*t *e Planning pok*er, são técnicas opcionais que podem ser adotadas em auxilio a medição do tempo que cada item do *Backlog* do Produto e velocidade do Time de Desenvolvimento (SABBAGH, 2014, p.118).

Para a aplicação destes conceitos no projeto, o PO com auxílio do *Scrum Master* do projeto foram os responsáveis pela priorização dos itens do *Product Backlog*. Além de transmitirem e negociarem com o Time de Desenvolvimento nas Reuniões de Planejamento da Sprint um conjunto de itens a serem inseridos na *Sprint* (*Backlog* da *Sprint*).

2.4.5.2 Backlog da Sprint

Schwaber e Sutherland (2016, p. 14), definem o *Backlog* da *Sprint* da seguinte maneira:

> O *Backlog* da *Sprint* é um conjunto de itens do *Backlog* do Produto selecionados para a Sprint, juntamente com o plano para entregar o incremento do produto e atingir o objetivo da Sprint. O *Backlog* da Sprint é a previsão do Time de Desenvolvimento sobre qual funcionalidade estará no próximo incremento e sobre o trabalho necessário para entregar essa funcionalidade em um incremento “Pronto”.

Conforme o Guia do *Scrum*, o Time de Desenvolvimento tem o papel de utilizar o *Backlog* da *Sprint* para realizar o desenvolvimento do trabalho necessário para que o incremento do produto atinja o objetivo da *Sprint*. Quando for necessária uma modificação no *Backlog* da Sprint, também será necessário o Time de Desenvolvimento junto ao PO remanejar com será realizado o trabalho para alcançar esse objetivo.

Além disso, o *Backlog* da *Sprint* representa ao final de uma determina *Sprint* uma lista de itens do *Backlog* do Produto em estado pronto, ou seja, durante o decorrer da *Sprint* o produto final começa a ganhar vida, sempre com foco no planejamento e negociação dos itens do *Backlog* da *Sprint* entre o Time de Desenvolvimento e o PO. Normalmente, ocorre antes do início da *Sprint*, na reunião de Planejamento da *Sprint* e em casos excepcionais há a necessidade dos envolvidos analisar a situação através de negociações entre os membros do Time *Scrum*.

No projeto em questão, os *Backlogs* da *Sprints* foram todos analisados nas Reuniões de Planejamento de cada *Sprint* com todo os envolvidos, com o objetivo de discutir sobre os itens que seriam executados na *Sprint* pelo Time de Desenvolvimento.

Para isto, foi utilizada uma ferramenta digital chamada Trello, que tem a mesma funcionalidade que um quadro de tarefas tradicional. Sendo assim, os itens selecionados do *Backlog* do Produto foram inseridos no *Backlog* da Sprint que no aplicativo foi denominada “A Fazer”.

Para atender esta necessidade a ferramenta foi customizada em quatro colunas, apresentadas conforme a seguir:

- **A fazer:** representa os itens *Backlog* da *Sprint*;
- **Fazendo/Realizando:** representa itens que estão sendo executados pelo Time de Desenvolvimento;
- **Em Revisão:** representa os itens que estão em espera para serem considerados prontos;
- **Feito:** representa os itens que já estão finalizados, ou seja, juntos representam o incremento da *Sprint* (produto entregável).

Esta customização descrita pode ser encontrada no capitulo CRONOGRAMA ÁGIL, o que permitiu ao Time de Desenvolvimento maior organização para realizar as tarefas que constituem o produto.

2.4.6 Ciclo de vida do *Scrum*

O ciclo de vida de projetos que o *framework Scrum* é constituído das seguintes etapas importantes:

- *Backlog* do produto
- Planejamento da *sprint*
- *Backlog* da *sprint*
- Execução da *sprint*
- Reunião diária
- Reunião de revisão
- Reunião de retrospectiva

Os tópicos citados foram abordados anteriormente de forma separada e detalhada, pois será necessária a noção básica dos mesmo para a compreensão do funcionamento do ciclo de vida *Scrum*. A Figura 10, representa o ciclo de vida do *Scrum*, onde apresenta todas as etapas que um item do *Backlog* do Produto passa até se tornar um incremento do produto gerado ao termino da *Sprint* em questão.

Figura 10 - Ciclo de vida da Framework *Scrum*

Fonte: (CRUZ, 2015, p. 195)

A Figura 10 demonstra que após as necessidades dos cliente serem transformadas pelo PO com o auxílio do *Scrum Master* em funcionalidades priorizadas (*Backlog* do Produto). Sendo que na Reunião de Planejamento esses itens serão analisados junto ao Time de Desenvolvimento como o objetivo de construir o *Backlog* da Sprint, o qual, contém todas as tarefas que o Time de Desenvolvimento necessita realizar para conseguir entregar o produto da Sprint.

O procedimento descrito na Figura 10 somente pode ser considerado encerrado quando o tempo de contrato terminar ou todas as funcionalidades do produto foram entregues ao cliente. E, em alguns casos, considerado interrompido quando o orçamento ultrapassou o esperado ou contrato cancelado (SABBAGH, 2014, p. 48).

3 CRONOGRAMA ÁGIL

Para o presente projeto foi desenvolvido um cronograma ágil para organização das *Sprints*. Na Figura 11, pode-se observar a aplicação da *Sprint* 3 no cronograma, através desta *Sprint* pode-se entender o funcionamento das demais *Sprint* executadas no projeto. Ou seja, as demais seguem o mesmo modelo, as quais podem serem encontradas no APÊNDICE 1 – QR CODE COM DOCUMENTOS ADICIONAIS. (cronograma completo)

Figura 11 - Cronograma *Sprint* 3

✓	**1.6**	**Sprint #3**	**53**	**14 dias**	**Sáb 16/09/17**	**Sáb 30/09/17**	
✓	**1.6.1**	**Metodologias**		**14 dias**	**Sáb 16/09/17**	**Sáb 30/09/17**	
✓	1.6.1.1	Pesquisar Teorica		14 dias	Sáb 16/09/17	Sáb 30/09/17	JV
✓	1.6.2	**Reunião de Orientação**	55IT	0 dias	Seg 18/09/17	Seg 18/09/17	AB;AS;JV;PG;VN
✓	**1.6.3**	**Desenvolvimento**	55II	**14 dias**	**Sáb 16/09/17**	**Sáb 30/09/17**	
✓	**1.6.3.1**	**Codificação**		**14 dias**	**Sáb 16/09/17**	**Sáb 30/09/17**	
✓	1.6.3.1.1	Telas Independentes		14 dias	Sáb 16/09/17	Sáb 30/09/17	AS;PG;VN
✓	**1.6.3.2**	**Banco de Dados**		**14 dias**	**Sáb 16/09/17**	**Sáb 30/09/17**	
✓	1.6.3.2.1	Criar Scripts do BD	60II	14 dias	Sáb 16/09/17	Sáb 30/09/17	VN
✓	1.6.4	Reunião da Equipe	58IT	0 dias	Qua 20/09/17	Qua 20/09/17	AS;JV;PG;VN
✓	**1.6.5**	**Monografia**	55	**3 dias**	**Seg 25/09/17**	**Qua 27/09/17**	
✓	**1.6.5.1**	**Validação (Orientador)**		**3 dias**	**Seg 25/09/17**	**Qua 27/09/17**	
✓	1.6.5.1.1	**Reunião de Orientação**		0 dias	Seg 25/09/17	Seg 25/09/17	AS;JV;PG;VN;AB
✓	1.6.5.1.2	Correções TCC	66	3 dias	Seg 25/09/17	Qua 27/09/17	VN;PG;AS;JV
✓	1.6.5.1.3	Reunião da Equipe	67IT	0 dias	Qua 27/09/17	Qua 27/09/17	AS;JV;PG;VN
✓	1.7	Término Sprint #3	54;64;6	0 dias	Qui 05/10/17	Qui 05/10/17	

Fonte: Elaborado pelos autores

Dessa forma, o cronograma representou o *Backlog* do Produto do projeto, ou seja, contém todos os itens relacionados as funcionalidades do sistema e as atividades do presente trabalho (relatório de desenvolvimento). Além disso, também são apresentadas as responsabilidades de cada integrante, cerimônias e a representação de cada artefato do *Scrum*.

Sendo que na Figura 12 pode-se observar a aplicação da *Sprint* 3, na qual apresenta seus respectivos itens (*backlog* da *Sprint*). Em auxilio a execução desses

itens do *Backlog* da *Sprint* foi utilizado a ferramenta Trello (Figura 12), que através do layout customizado permitiu maior organização da equipe de desenvolvimento.

Figura 12 - *Sprint* 3

Fonte: Elaborado pelos autores

Através disso, pode-se compreender que cada item do *Backlog* da *Sprint* é movido pelos respectivos membros responsáveis de cada tarefa do Time de Desenvolvimento, onde estes itens são disponibilizados para a equipe através da coluna “A fazer”. Posteriormente, movido para a execução da tarefa na coluna “Fazendo/Realizando”, quando finalizados são enviados para a revisão, sendo colocados na coluna “Em Revisão”.

Por fim, quando todas as correções forem realizadas passarão para a coluna de “Feito”. Dessa forma, quando todos os itens do *Backlog* da *Sprint* forem considerados como “Feito”, poderão ser apresentados e analisados nas cerimonias de revisão da Sprint.

4 EMPRESA

O sistema foi desenvolvido para a Empresa X, localizada na cidade de Americana-SP, prestando serviços nas áreas de medicina e segurança de trabalho, no qual necessitava de ajuda.

Esta empresa necessitava automatizar as tarefas internas, que atualmente esses processos são feitos manualmente e leva muito tempo para serem realizadas, principalmente o processo de geração de ASO e agendamento periódico das consultas.

Devido a esses motivos foram realizadas algumas entrevistas junto a empresa para verificar o que poderia ser feito para melhorar os processos internos, assim, desenvolvendo um produto que atenda às necessidades da empresa.

Além disso, foi necessário realizar algumas pesquisas sobre a área de medicina ocupacional, que envolve a legislação e o processos de PCMSO, para entender como realmente funciona uma empresa de medicina ocupacional.

4.1 Legislação vigente ao desenvolvimento

Existem cada vez mais leis que buscam melhorar as condições de trabalho dentro das empresas buscando cada vez mais diminuir o número de acidentes dos trabalhadores nas mesmas, dentre essas leis podem ser citadas as NR’s que foram instituídas a partir de 1978, dentre as normas e leis que abrangem Medicina e Segurança do Trabalho foram instituídas a fim de oferecer mais segurança para o trabalhador e para o empregador.

Dentre as Normas instituídas pelo Pelas normas Regulamentadoras está o PCMSO siga para Programa de Controle Médico de Saúde Ocupacional.

4.2 PCMSO

O PCMSO é um programa que visa a promoção e preservação da vida do trabalhador no seu ambiente de trabalho.

Cabe a empresa que contrata seu funcionário diretamente ou através de terceiras, implementar as normas que assegurem um ambiente de trabalho seguro para seus funcionários através do PCMSO.

Os programas de prevenção de acidentes através de exames que podem ser no mínimo semestral, podendo ter seu tempo reduzido a critério do médico ou agente de inspeção do trabalho, através do sindicato da categoria ou de negociações coletivas, podem ser realizados pela própria empresa através de um profissional médico interno que irá ser responsável pela execução do PCMSO (BRASIL, 1978, p. 3), ou a mesma desobrigada a manter um profissional médico para realização dos exames exigidos pelo PCMSO, deverá indicar um médico do trabalho para realiza-lo sendo este interno ou externo.

Segundo NR-07 (BRASIL, 1978, pg. 3), o PCMSO deve incluir a realização obrigatória dos seguintes exames médicos.

- Admissional;
- Periódico;
- Retorno ao trabalho;
- Mudança de função;
- Demissional.

4.3 Atestado de Saúde Ocupacional (ASO)

“Para cada exame médico realizado, o “médico emitirá o Atestado de Saúde Ocupacional – ASO, em 2 (duas) vias”. (BRASIL, 1978, p. 3).

A ASO é uma garantia para o trabalhador que garante a ele maior segurança, e a empresa, que garante maior suporte jurídico em ações legais que possam ser movidas por seus ex-funcionário no futuro. A ASO é emitida em duas vias, uma para a empresa e uma para o trabalhador. (BRASIL, 1978, p. 3-4).

A ASO é um documento médico, com valor jurídico que deve conter os seguintes campos: (BRASIL, 1976, p. 4).

- Nome completo do trabalhador, o número de registro de sua identidade e sua função;
- Os riscos ocupacionais específicos existentes, ou a ausência deles, na atividade do empregado, conforme instruções técnicas expedidas pela Secretaria de Segurança e Saúde no Trabalho-SSST;
- Indicação dos procedimentos médicos a que foi submetido o trabalhador, incluindo os exames complementares e a data em que foram realizados;
- O nome do médico coordenador, quando houver, com respectivo CRM;
- Definição de apto ou inapto para a função específica que o trabalhador vai exercer, exerce ou exerceu;
- Nome do médico encarregado do exame e endereço ou forma de contato;
- Data e assinatura do médico encarregado do exame e carimbo contendo seu número de inscrição no Conselho Regional de Medicina.

Os documentos necessários para montagem da ASO são importantes, através deles é realizado a montagem da ASO (Atestado de Saúde Ocupacional) em si.

4.4 Documentações analisadas do ambiente de desenvolvimento

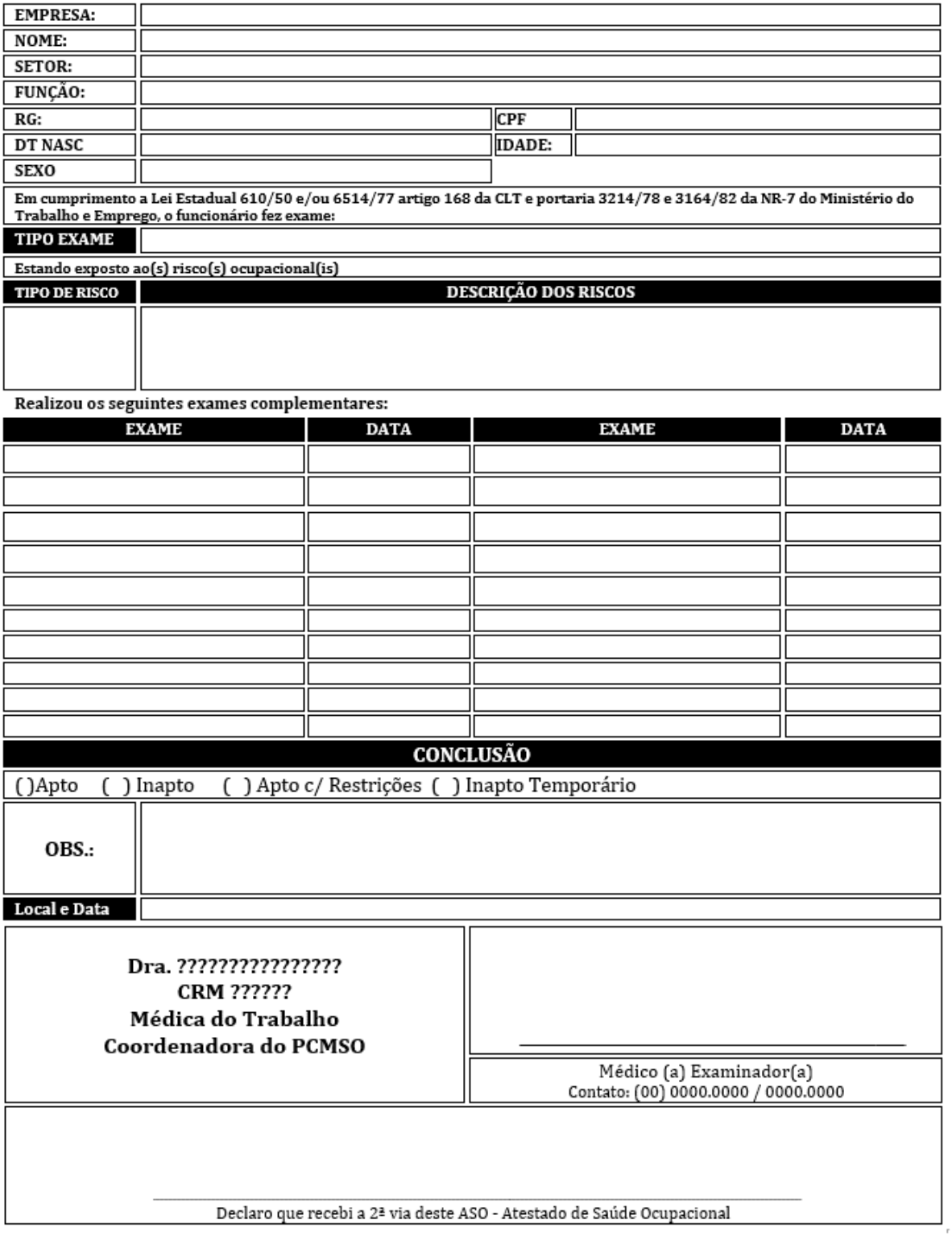
Para o desenvolvimento da aplicação foi necessário analisar alguns documentos que foram fornecidos pelo cliente, para complementar os requisitos que foram obtidos durante as entrevistas junto ao usuário.

Dentre os documentos analisados estavam planilhas e documentos de Word, no qual continham informações essenciais que auxilio na captação dos requisitos. Estes documentos continham informações de clientes, agendas e também financeiras.

A partir destas documentações foi possível incluir alguns requisitos que não foram possíveis de identificar durante as entrevistas com o cliente, assim, foi possível desenvolver o sistema que atenda às necessidades reais da empresa.

Na Figura 13 é possível observar um modelo de ASO que foi fornecido pelo cliente (Não foram inseridos outros documentos devido conter informações do cliente):

Figura 13 - Modelo ASO



EMPRESA:			
NOME:			
SETOR:			
FUNÇÃO:			
RG:		**CPF**	
DT NASC		**IDADE:**	
SEXO			

Em cumprimento a Lei Estadual 610/50 e/ou 6514/77 artigo 168 da CLT e portaria 3214/78 e 3164/82 da NR-7 do Ministério do Trabalho e Emprego, o funcionário fez exame:

TIPO EXAME	

Estando exposto ao(s) risco(s) ocupacional(is)

TIPO DE RISCO	**DESCRIÇÃO DOS RISCOS**

Realizou os seguintes exames complementares:

EXAME	**DATA**	**EXAME**	**DATA**

CONCLUSÃO

( )Apto ( ) Inapto ( ) Apto c/ Restrições ( ) Inapto Temporário

OBS.:	
Local e Data	

Dra. ??????????????
CRM ??????
Médica do Trabalho
Coordenadora do PCMSO

_____

Médico (a) Examinador(a)
Contato: (00) 0000.0000 / 0000.0000

_____

Declaro que recebi a 2ª via deste ASO - Atestado de Saúde Ocupacional

Fonte: Elaborado pelo cliente

5 LEVANTAMENTO DE REQUISITO

Neste capitulo será abordado técnicas da engenharia de software que auxiliam a coleta de requisitos dos clientes, sendo somente possível através da análise criteriosa das necessidades do cliente (Visão de negócio), para assim, após a visão o produto estar estabelecida desenvolver as funcionalidades do produto que satisfaçam as necessidades do cliente.

Os requisitos do presente projeto foram baseados em um projeto da disciplina de laboratório de engenharia de software, no qual foi realizado o reaproveitamento dos requisitos, pois naquela ocasião o software não havia sido finalizado. Por este motivo, foi acordado entre os integrantes do projeto desenvolver um novo sistema com uma abordagem tecnológica diferente.

5.1 Requisitos de Software

Os requisitos de sistema são descrições dos serviços que são fornecidos pelo sistema e as suas restrições operacionais. Esses requisitos refletem as necessidades dos clientes de um sistema que ajuda a resolver algum problema, tais como controlar um dispositivo ou, encontrar informações. O processo de descobrir, analisar, documentar e verificar esses serviços e restrições é chamado de engenharia de requisitos (SOMMERVILLE, 2007, p. 79).

5.2 Requisitos de usuário

Para Sommerville (2007, p. 80), “Requisitos de usuário: são declarações, e uma linguagem natural com diagramas, de quais serviços são esperados do sistema e as restrições sob as quais ele deve operar“.

Estes são feitos para o usuário, e devem apresentar uma linguagem simples compreensível ao mesmo, que, não possui conhecimento técnico detalhado a respeito dos jargões utilizados pela engenharia de software.

Por se tratar de um documento com uma linguagem mais natural que o usuário consiga assimilar, o documento de requisitos pode apresentar vários problemas que dificultam sua clareza, alguns exemplos são citados por Sommerville (2007, p. 85):

- **Falta de Clareza:** ambiguidades e linguagem imprecisa dificultam a leitura do documento de requisitos.
- **Confusão de requisitos:** falta de clareza ao diferenciar requisitos funcionais de funcionais, metas de sistema e informações do projeto.
- **Fusão dos Requisitos:** Requisitos adversos podem ser escritos juntos como apenas um.

Com as reuniões com a cliente tornou-se aparente que o levantamento de requisitos pode se tornar trabalhoso e complexo, demandando várias reuniões com a cliente, em decorrência de no momento de redigir o documento de especificação dos requisitos.

5.3 Requisitos de sistema

Para Sommerville (2007, p. 80), definem, detalhadamente, as funções, os serviços e as restrições operacionais do sistema, estes são versões mais detalhadas e complexas dos requisitos de usuário e são utilizados como ponto de início para o projeto de sistema.

Ambos requisitos de sistema e usuário se utilizam da mesma forma de linguagem para serem escritos, no entanto requisitos de usuário são voltados para não especialistas, requisitos de sistemas, ao contrário de usuário podem ser escritos se utilizando notações mais complexas que descrevem com mais acurácia os requisitos do sistema.

5.4 Requisito funcional

Sommerville (2007, p. 82): “Requisitos funcionais de um sistema devem descrever o que o sistema deve fazer”.

Os requisitos funcionais são dinâmicos, ou seja, são mutáveis, e variam de projeto para projeto, ou até mesmo podem se alterar dependendo do andamento do mesmo projeto, conforme novos requisitos vão aparecendo. Para Sommerville (2007, p. 81): “Os requisitos funcionais descrevem, a função do sistema de forma detalhada, suas entradas, suas saídas e suas exceções”.

Os requisitos funcionais coletados através da análise da necessidade do cliente foram os seguintes:

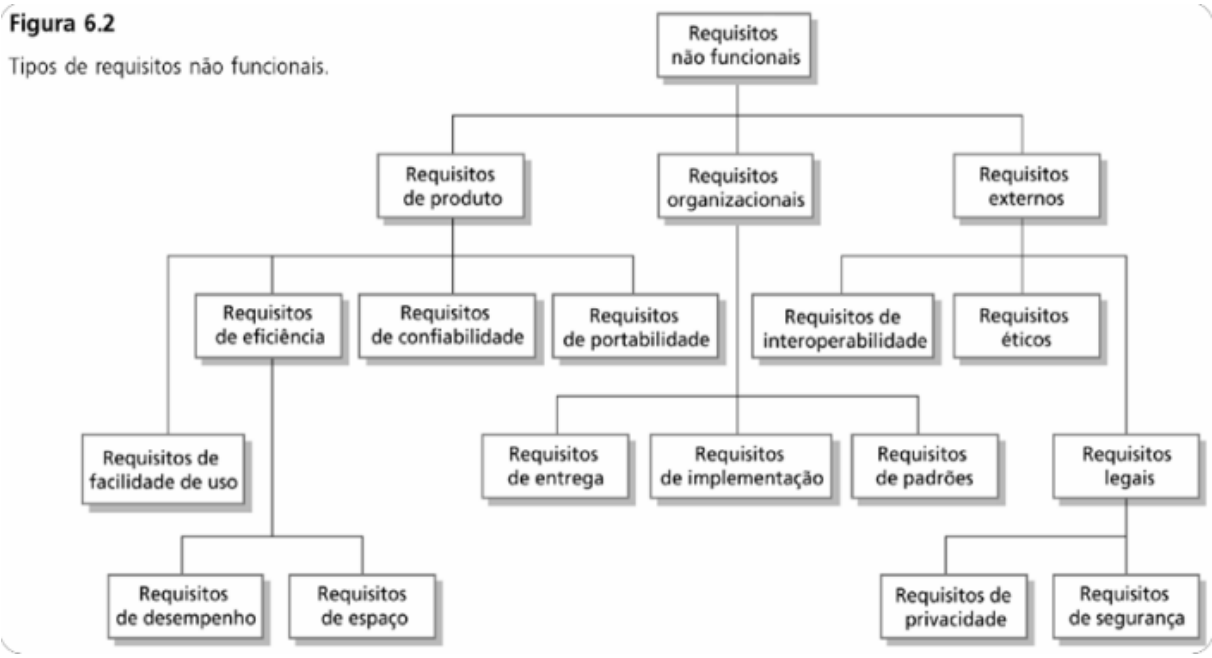
- Manter clientes;
- Manter funcionários do cliente;
- Manter setores do cliente;
- Manter agenda;
- Manter exames;
- Manter riscos;
- Manter fornecedores;
- Criar dashborad;
- Gerar ASO;
- Gerar relatórios financeiros.

5.5 Requisito não funcional

Sommerville (2007, p. 82), ”Requisitos não funcionais são aqueles não diretamente relacionados às funções específicas fornecidas pelo sistema”. Podem estar relacionados a funções não relacionadas com as operações principais que o sistema deve desempenhar, como confiabilidade e outras propriedades emergentes do sistema que podem impactar na experiência do usuário ao utilizar o sistema.

Estes requisitos são tão importantes quanto, requisitos funcionais a falta ou não operabilidade destes requisitos, pode significar que o sistema é inútil ao seu usuário final.

Figura 14 - Tipos de requisitos não funcionais.



Fonte: (SOMMERVILLE, 2007, p.82).

Juntando-se a isso Sommerville (2007, p. 83) ainda ressalta que: “Abrange todos os requisitos derivados de fatores externos ao sistema e seu processo de desenvolvimento.” E também afirma que “[...] requisitos legais que devem ser seguidos para assegurar que o sistema funcione dentro da lei e requisitos éticos.”

No sistema vigente o maior requisito externo encontrado, foram requisitos legais que devem ser seguidos corretamente para a montagem da ASO. A ASO deve seguir por padrão o que é dito pelo Ministério do Trabalhado seguindo o que é dito na Norma Regulamentadora – NR-07.

Os requisitos não funcionais coletados através da análise da necessidade do cliente foram os seguintes:

- Interface amigável, para facilitar o uso para usuário;
- Sistema tem de ser seguro, afim de proteger os dados do cliente;
- Sistema deve ser rápido (desempenho);
- Sistema deve ter documentação do sistema (Manual do usuário);
- Desenvolvedores devem oferecer suporte e treinamento aos usuários;
- O sistema deve ser desenvolvido com paradigmas de orientação a objetos, afim de reutilizar os códigos.

6 FERRAMENTA PARA DOCUMENTAÇÃO DO SISTEMA – UML

A linguagem UML (*Unified Modeling Language* – Linguagem de Modelagem Unificada) é uma linguagem utilizada na modelagem de software, ela é visual baseada no paradigma orientado a objeto. A linguagem UML não é uma linguagem de programação e sim uma linguagem de modelagem que tem como função, auxiliar engenheiros de software a definir, estrutura lógica, requisitos, comportamento, entre outras características, vale destacar que a UML não é um processo de desenvolvimento, mas sim utilizada por outros processos (GUEDES, 2009, p. 19).

Até meados dos anos 90, existiam três métodos de modelagem populares, que eram, o método de Booch, o método OMT de Jacobson e o método OOSE de Rumbaugh, esses três métodos foram unidos para a criação da linguagem UML, primeiramente a união se deu entre os métodos Booch e o OMT, resultando no que foi chamado de Método Unificado no final de 1995, foi quando então que Rumbaugh com seu método OOSE, se juntou a Booch e Jacobson, criando a primeira versão da UML em 1996. Atualmente a UML está na versão 2.2, porém já existe a versão 2.3 beta (GUEDES, 2009, p. 19-20).

Booch, Rumbaugh e Jacobson (2006, p. 5) dizem que muitas equipes de desenvolvimento não se atentam ao planejamento do software e partem já para a codificação do sistema e acreditam que com bastante trabalho, se consiga superar os problemas durante o desenvolvimento do sistema, porém apenas mais trabalho extra não é garantia de que o código esteja correto, ou dependendo do tamanho do projeto, que o mesmo seja finalizado. A principal ferramenta para um software de qualidade é a modelagem.

A modelagem possui quatro objetivos segundo Booch, Rumbaugh e Jacobson (2006, p. 6):

- Os modelos ajudam a visualizar o sistema como ele é ou como desejamos que seja;
- Os modelos permitem especificar a estrutura ou o comportamento de um sistema;
- Os modelos proporcionam um guia para a construção de um sistema;
- Os modelos documentam as decisões tomadas.

Guedes (2009, p. 21) destaca que um projeto de software por menor que seja, sempre pode crescer, ou seja, aumentar a complexidade ou tamanho, isso significa que os sistemas são dinâmicos, isso se deve ao fato de os sistemas estarem sujeitos a mudanças impulsionadas por fatores, tais como, desejo dos clientes em melhorar o sistema ou adicionar uma nova funcionalidade, mudança de estratégias de negócios nas empresas, a criação ou a mudança de leis e impostos já existentes por parte do governo, etc. Para que essas mudanças possam acontecer com mais facilidade é necessário que o sistema seja bem documentado e a modelagem é uma forma de se documentar um software.

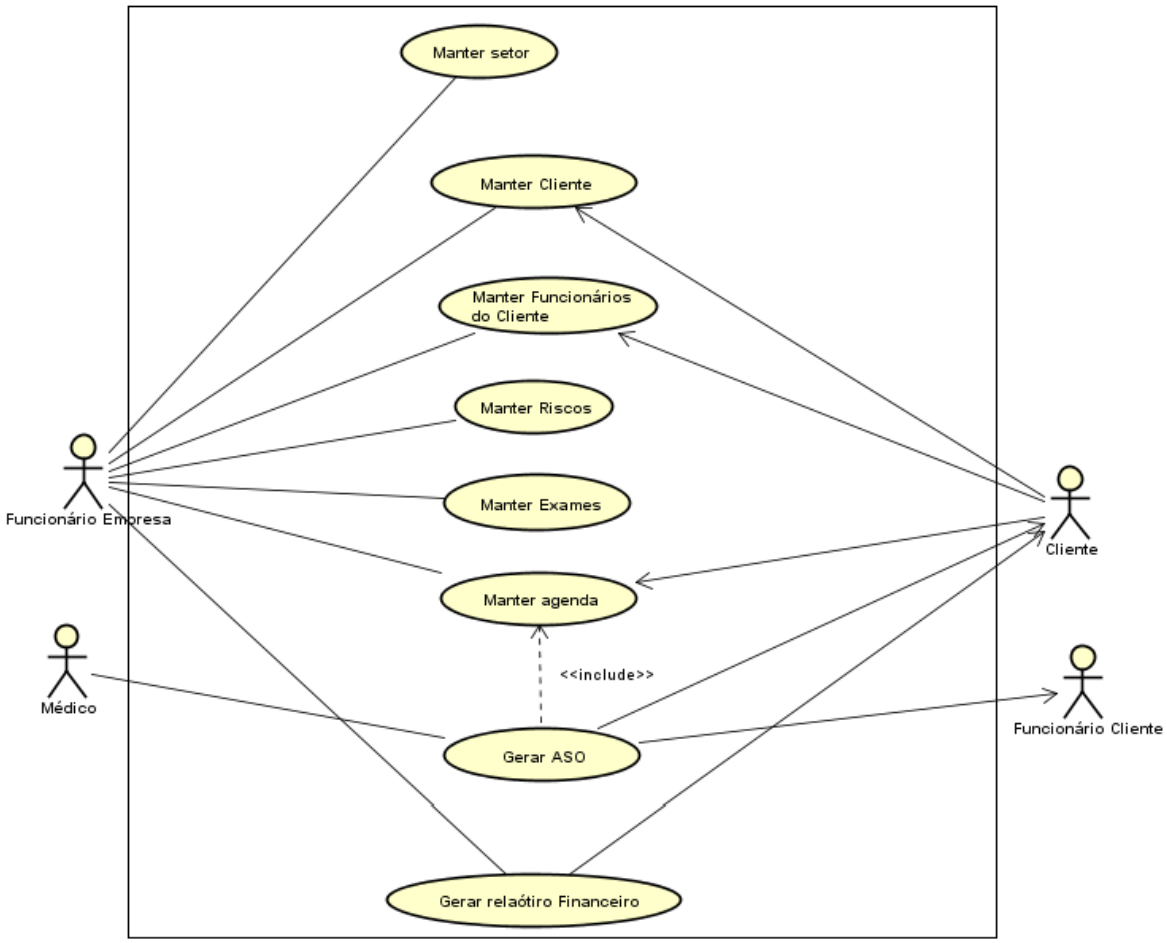
6.1 Diagrama de Caso de Uso

O diagrama de caso de uso é um dos diagramas existentes na UML, neste diagrama são mostrados os casos de uso, atores e os seus relacionamentos, este diagrama é utilizado para mostrar uma visão geral do contexto dos casos de uso que compõem o sistema. É importante para visualizar, especificar e documentar o comportamento de um elemento (esses elementos seriam sistemas, subsistemas e classes) (BOOCH, RUMBAUGH, JACOBSON, 2007, p. 241).

De acordo com Guedes (2009, p. 55), o diagrama de caso de uso tenta demonstrar o comportamento externo do sistema (as suas funcionalidades) para qualquer usuário, apresentando o sistema na perspectiva do usuário, o diagrama de caso de uso é o mais abstrato entre os diagramas da UML e por isso é o mais flexível e informal. Por ser um diagrama mais voltado ao usuário, geralmente é utilizado no processo de levantamento e análise de requisitos, servindo como base também para outros diagramas.

Na Figura 15 é possível observar o diagrama de caso de uso desenvolvido para representar o sistema, onde os funcionários da empresa e o médico tem acesso as funções presentes no sistema, tanto para enviar como visualizar informações. Já o cliente e seu funcionário possuem somente acesso algumas opções, sendo que ele não pode visualizar informações de algumas delas (Manter Clientes, Manter Empregados e Manter agenda e gerar ASO).

Figura 15 - Diagrama de caso de uso



Fonte: Elaborado pelos autores.

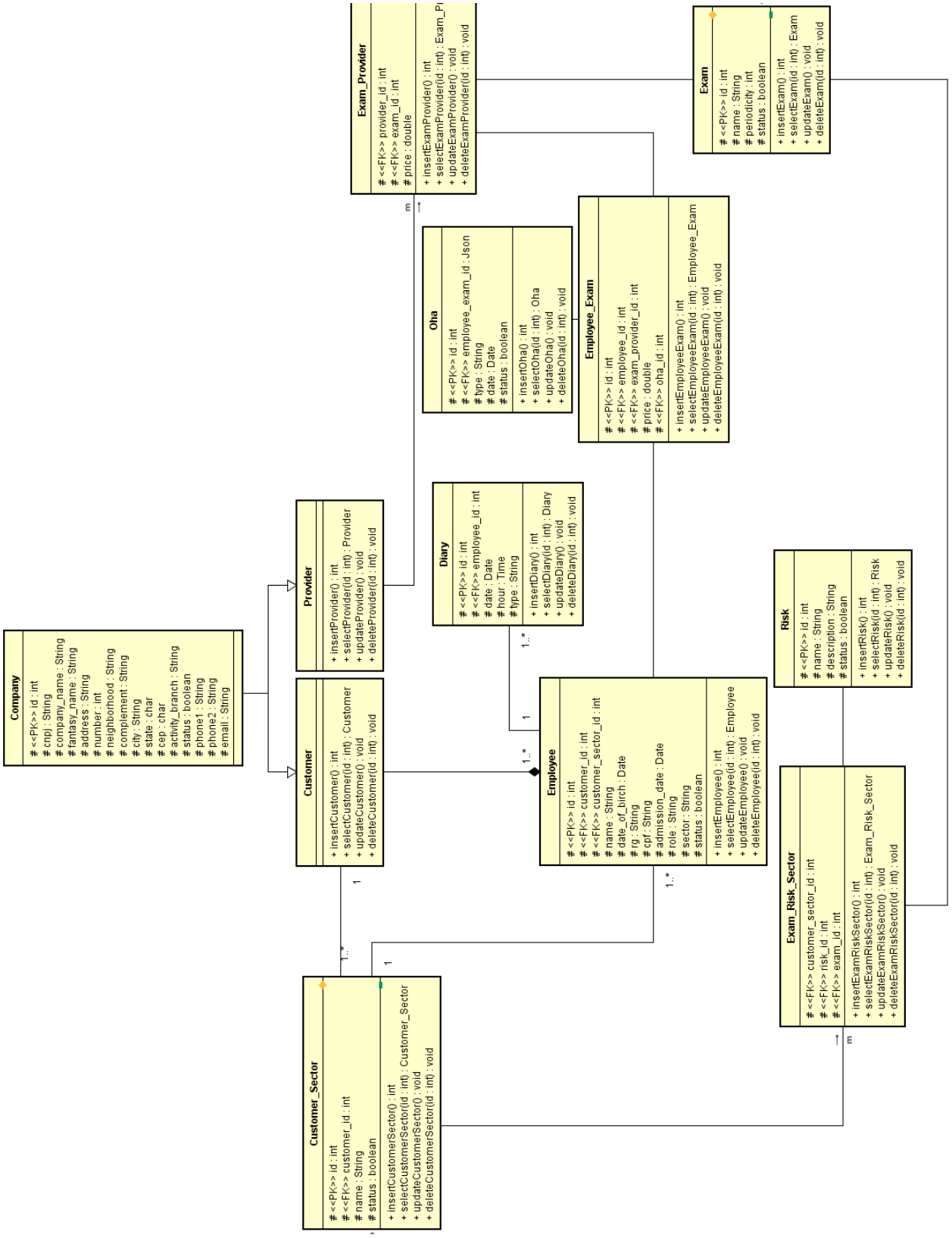
6.2 Diagrama de Classe

O Diagrama de Classes, tem como objetivo descrever os diferentes objetos que compõem um sistema, bem como o relacionamento estático destes objetos, as propriedades e as operações de uma classe (FOWLER, 2004, p. 52). O Diagrama de Classes é um dos mais importantes da UML, servindo de base para a construção de outros diagramas da UML, este diagrama é composto basicamente pelas classes e os seus relacionamentos (GUEDES, 2009, p. 106).

Booch, Rumbaugh e Jacobson (2007, p. 49) nos dá a seguinte definição sobre classe: “As classes são os blocos de construção mais importantes de qualquer sistema orientado à objeto. Uma classe é uma descrição de um conjunto de objetos que compartilham os mesmos atributos, operações, relacionamentos e semântica”.

O diagrama de classes está sendo representado na Figura 16, onde é possível observar as classes com seus métodos e as relações entre elas, assim, é possível obter uma organização no código, pois todas as classes com seus métodos estão representadas nesse diagrama, o que facilita o processo de codificação.

Figura 16 - Diagrama de classe



Fonte: Elaborado pelos autores.

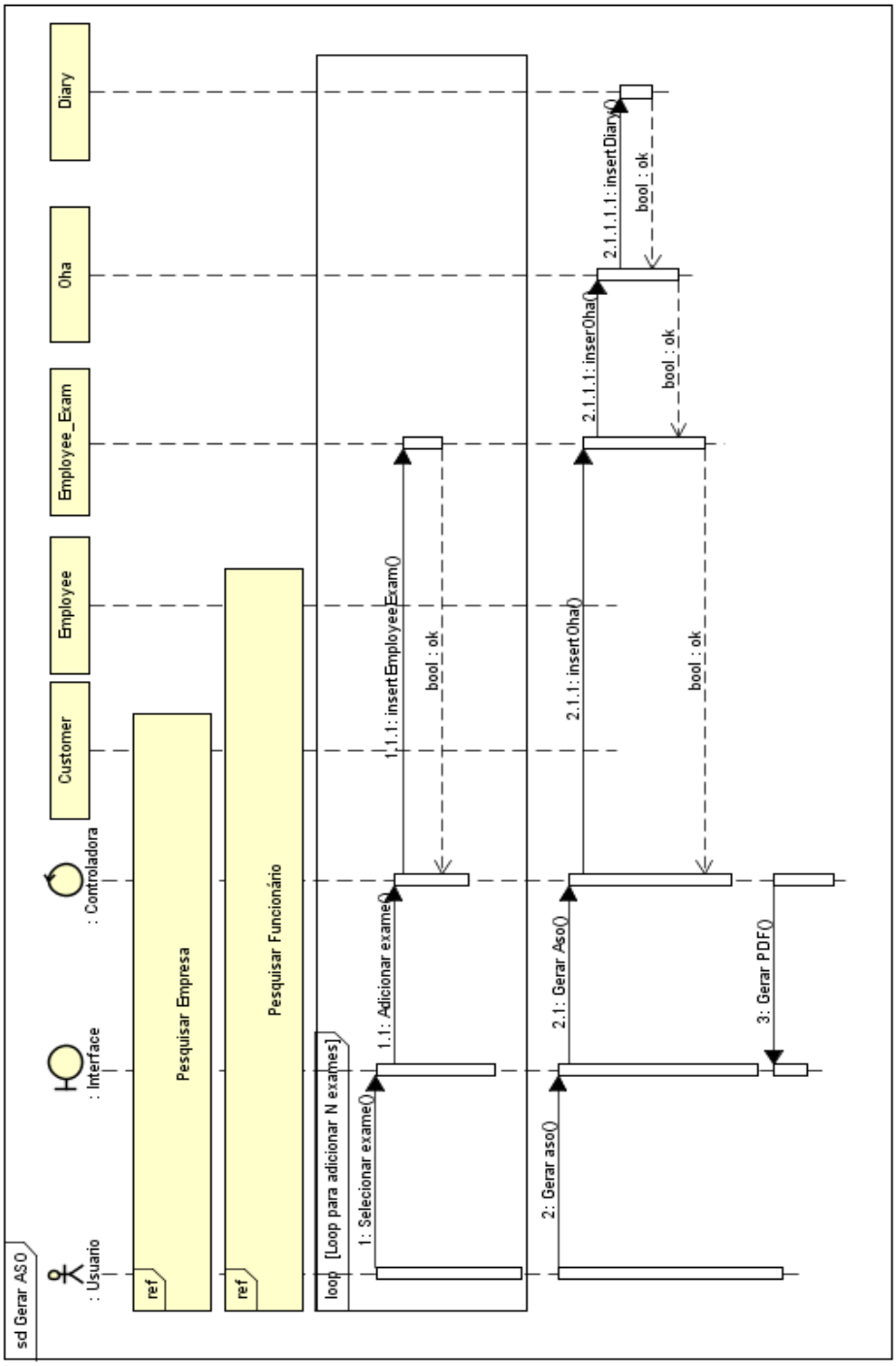
6.3 Diagrama de Sequência

O diagrama de sequência mostra as interações entre os objetos (troca de mensagens) em determinada situação e mostra a ordem com que as interações acontecem. Os objetos são identificados por linhas tracejadas verticais com o seu nome no topo, as mensagens são representadas por setas que cruzam as linhas verticais dos objetos, a linha de tempo também é vertical, aumentado para baixo a medida com que as mensagens são trocadas (VARGAS, [2007?], p. 7).

Um diagrama de sequência pode documentar um caso de uso específico, pois geralmente um caso de uso está relacionado com um diagrama de sequência, em algumas ferramentas CASE é possível fazer o diagrama de sequência a partir do diagrama de caso de uso (GUEDES, 2009, p. 200).

O diagrama de sequência pode ser observado na Figura 17, nele demonstrado como é feito o processo para gerar uma ASO e quais tabelas serão utilizadas para salvar ou buscar as informações, o processo começa com pesquisas dos dados dos funcionários, logo após existe um loop que adiciona “n” exames na ASO, e por fim faz a geração da ASO imprimindo um PDF. As funções de pesquisar empresa e funcionário podem ser encontradas no APÊNDICE 1 – QR CODE COM DOCUMENTOS ADICIONAIS.

Figura 17 - Diagrama de sequência (Gerar Aso)



Fonte: Elaborado pelos autores

6.4 Diagrama de Atividades

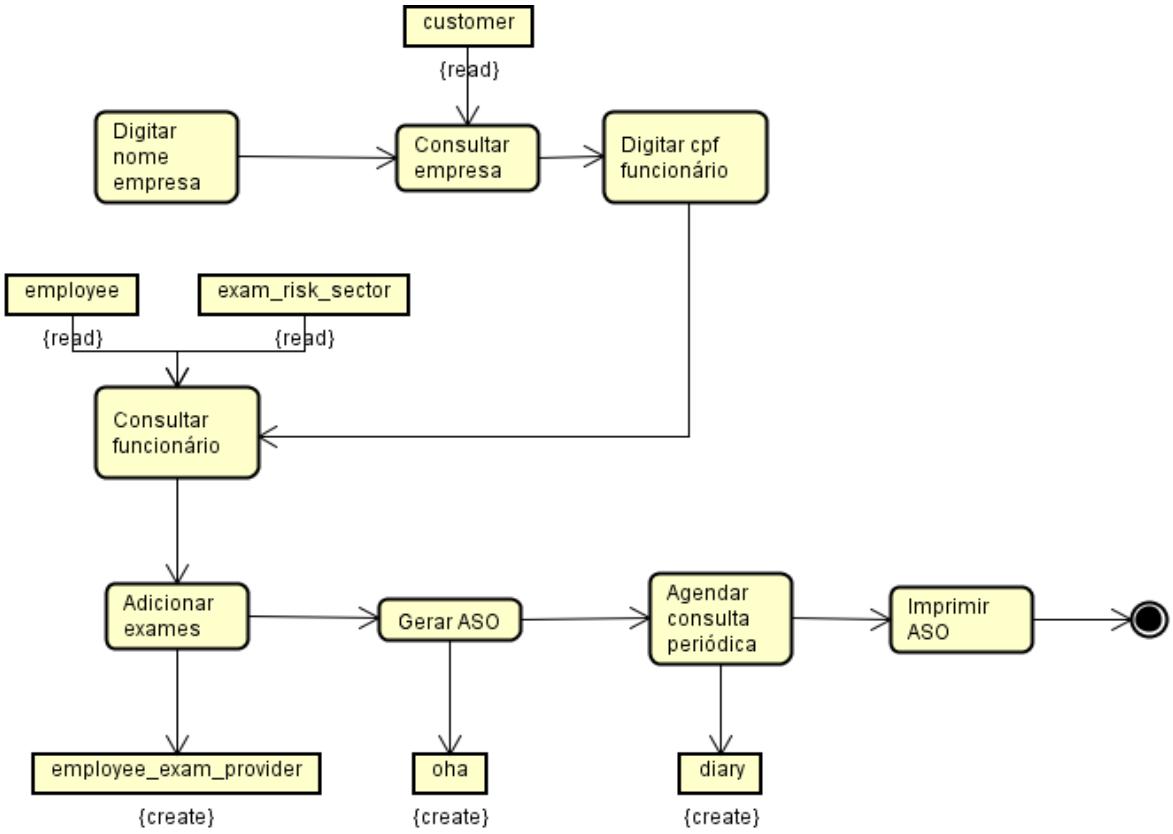
De acordo com Silva (2007, p. 155): “O diagrama de atividades é voltado a detalhar o comportamento de um programa, isto é, descrevê-lo enquanto em

execução”. Essas descrições podem ser de alto ou baixo nível de abstração sobre o que é executado pelo programa (SILVA, 2007, p. 155).

Um diagrama de atividades pode ser utilizado para modelar, como o nome diz, atividades, essas atividades vão desde um método ou um algoritmo, até um processo completo (GUEDES, 2009, p. 285).

Na Figura 18 na representação do diagrama de atividades, é possível observar os fluxos para geração de uma ASO. Ela tem início ao informar os dados da empresa e de seu funcionário corretamente, logo em seguida existe a possibilidade de adicionar exames que o funcionário deve realizar, por fim, é agendado uma nova consulta e é impresso a ASO.

Figura 18 - Diagrama de atividades (Gerar ASO)



Fonte: Elaborado pelos autores.

7 MODELAGEM DO BANCO DE DADOS

Neste ponto do projeto serão demonstradas as técnicas utilizadas na modelagem do banco de dados do projeto. Para isto, foi desenvolvido o modelo conceitual, lógico e físico para o desenvolvimento da aplicação em questão. No entanto, durante o decorrer deste capitulo será apresentado alguns conceitos chaves que auxiliarão no entendimento das técnicas utilizadas para realizar a modelagem do banco de dados.

7.1 Definição de banco de dados

Um banco de dados é “uma coleção de dados persistentes, usadas pelos sistemas de aplicação de uma determinada empresa”. Entretanto, Elmasri e Navathe (2011, p. 23), definem que banco de dados é “uma coleção de dados relacionados”.

No entanto, essas definições são muito generalistas e, acabem não enaltecendo pontos importantes que um banco de dados apresenta na realidade. Para tal, Elmasri e Navathe (2011, p. 23), apontam algumas características especificas que um banco de dados necessita contém, são as seguintes:

> Um banco de dados representa algum aspecto do mundo real, às vezes chamado de minimundo ou de universo de discurso (UoD – *Universe of Discurse*). As mudanças no minimundo são refletidas no banco de dados.
> Um banco de dados é uma coleção logicamente coerente de dados com algum significado inerente. Uma variedade aleatória de dados não pode ser corretamente chamada de banco de dados.
> Um banco de dados é projetado, construído e populado com dados para uma finalidade específica. Ele possui um grupo definido de usuários e algumas aplicações previamente concebidas nas quais esses usuários estão interessados.

A partir dessas características apontadas, pode-se dizer que um banco de dados possuí uma fonte de dados, possui um grupo de pessoas interessadas em seu conteúdo e interações com eventos que acontecem no minimundo.

Um exemplo interessante que muitas das vezes passa despercebido, mas retrata essa realidade, é quando um cliente de uma agencia bancaria deseja tirar um extrato de sua conta, certamente estará consultando os dados de suas

movimentações bancárias, que, por sua vez, esses dados certamente estão armazenados em um determinado banco de dados (ELMASRI, NAVATHE, 2011, p. 23).

A seguir, Tosing (2006, p. 21-24) defende algumas vantagens da utilização do banco de dados, sendo as seguintes:

- Independência dos dados;
- Integridade, consistência e compartilhamento de dados: isso significa que é possível fazer checagem para ver se os dados estão corretos e que eles podem ser compartilhados em um ambiente multiusuário;
- Redundância de dados pode ser minimizada: redundância de dados é quando um dado está presente em mais de uma tabela, há casos em que é necessário a redundância, mas a casos que sua presença é apenas desperdício de recursos.
- Utilização de padrões: por exemplo, quando várias aplicações acessam o mesmo banco de dados, sempre se fará referência a uma tabela ou atributo com o mesmo nome que consta no dicionário de dados;
- Garantia da completude de uma transação: ou seja, quando é feita uma ou muitas operações no banco de dados (por exemplo uma inserção) e esta operação não seja realizada com sucesso, deve-se retornar ao estado anterior afim de que se mantenha a integridade dos dados.

As vantagens apresentadas permitem uma melhor organização dos dados, garantindo assim uma maior qualidade no gerenciamento das informações e, consequentemente maior nível de compreensão, manutenção, íntegridade e disponibilidade das informações ao usuário final.

7.2 Projeto de banco de dados

O projeto de banco de dados representa uma etapa de suma importância em qualquer desenvolvimento de sistema, o qual, durante o decorrer do tempo ganhou seu espaço, tanto como atividade especial que merece um cuidado mais próximo quanto nas etapas referentes ao planejamento do sistema. Pois, antes de desenvolver qualquer parte de um projeto faz necessária um planejamento detalhado (MACHADO, 2008, p. 15).

Sendo assim, a primeira etapa do projeto de banco de dados é o levantamento e análise de requisitos, onde os projetistas fazem uma entrevista com o cliente que solicitou o sistema, assim, gerando os requisitos escritos de forma concisa.

É importante que os requisitos estejam bem detalhados e completos, pois dentre estes requisitos estão os requisitos de dados (relacionamento dos dados que serão armazenados) e os requisitos funcionais, que representam as operações feitas no banco de dados (ELMASRI, NAVATHE, 2011, p. 132).

Heuser (2009, p. 29) defende que um projeto de banco de dados é dividido em três fases, as quais serão apresentadas a seguir durante o decorrer da seção.

7.3 Modelo conceitual (MER)

O modelo conceitual ou modelo entidade relacionamento (MER) representa o primeiro modelo a ser desenvolvido quando o assunto é projeto de banco de dados, pois, este modelo tem o principal objetivo de descrever de forma simples a realidade do problema que o projeto está sendo construído, ou seja, representar um determinado contexto de negócio que será armazenado através de processos em um banco de dados (MACHADO, 2008, p. 20).

Nesta fase, o modelo busca auxilia o analista a correlacionar as informações coletadas no levantamento de dados pela análise de requisitos, em prol do entendimento e do atendimento das necessidades de armazenamento do cliente ou da organização (HEUSER, 2009, p.29).

Ainda segundo Machado (2008, 2008, p.21), o modelo pode ser definido da seguinte forma:

> O objetivo do modelo conceitual e descrever de forma simples e facilmente compreendida pelo usuário final as informações de um contexto de negócios, as quais devem ser armazenadas em um banco de dados.
>
> E uma descrição de alto nível (macrodefinição), mas que tenha a preocupação de captar e retratar toda a realidade de uma organização, processo de negócio, setores, departamento e etc.

No entanto, Heuser (2009, p. 25) ressalta uma definição mais técnica sobre o modelo conceitual:

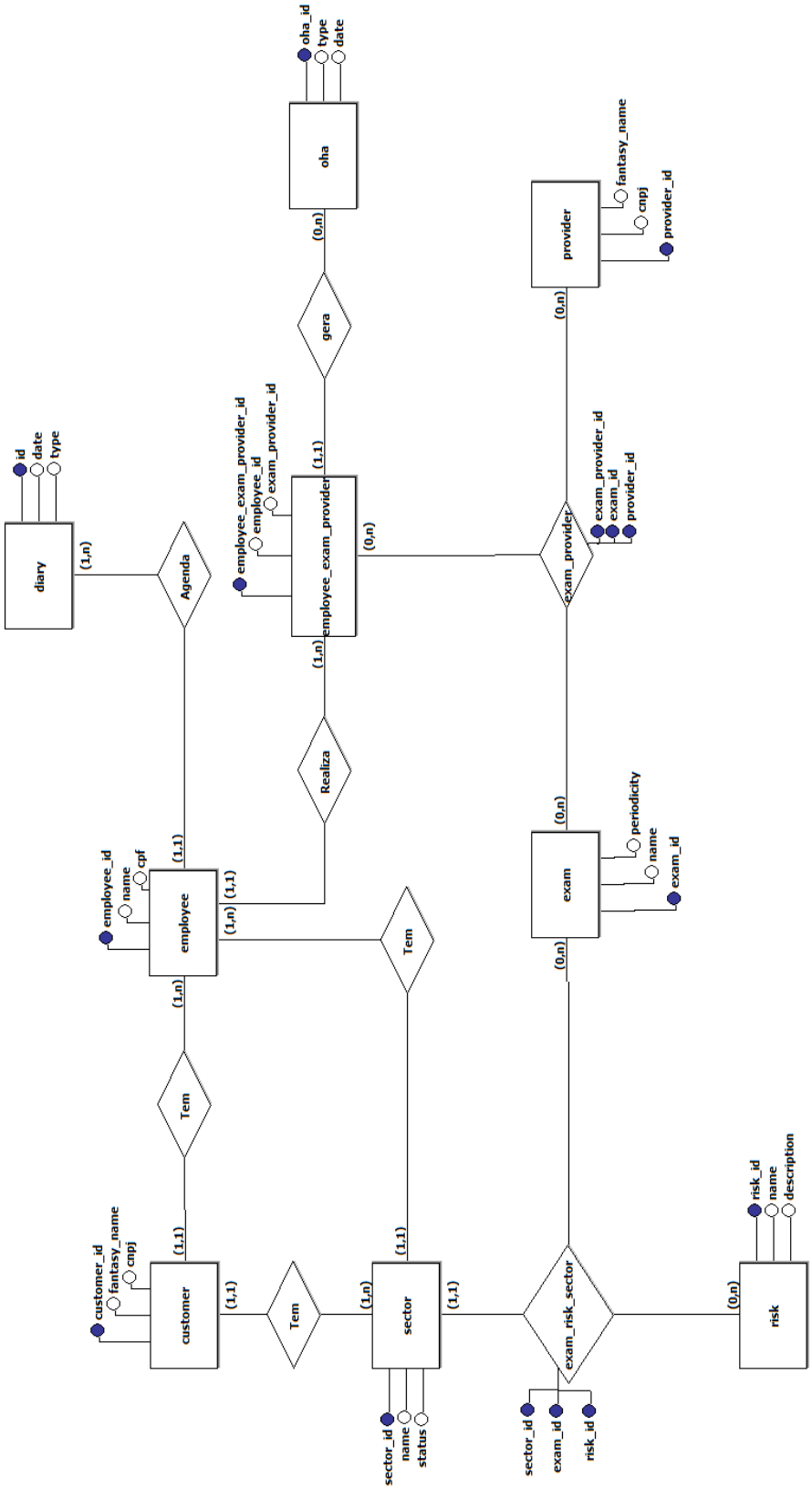
> Um modelo conceitual é uma descrição do banco de dados de forma independente de implementação em um SGBD. O modelo conceitual registra

> que dados podem aparecer no banco de dados, mas não registra como estes dados estão armazenados a nível de SGBD.

Através desses conceitos apresentados Machado (2008) e Heuser (2009), é possível compreender a realidade em que um projeto de banco de dados do software enfrentara na fase de concepção. Sendo assim, são realizadas analises da realidade do negócio ou processos que permitem identificar as necessidades (requisitos) que realmente serão de grande relevância serem levantados para construir o modelo conceitual.

Na Figura 19, é possível observar o modelo conceitual do banco de dados que foi projetado para o sistema desenvolvido, o qual, contém as tabelas com suas chaves primárias e relacionamentos (Não foi inserido todos atributos devido a sua quantidade, mas eles serão mostrados mais à frente no DER).

Figura 19 - MER



Fonte: Elaborado pelos autores.

7.4 Modelo lógico (DER)

Já o modelo logico, somente poderá ser iniciado após a criação do modelo conceitual, pois o modelo abordado nesta secção é desenvolvido com objetivo de definir o relacionamento lógico dos dados presentes no modelo conceitual. Ou seja, necessita dos dados que foram coletados através da abstração da realidade do negócio (minimundo), para que seja feito o relacionamento logico desses dados. Nesta fase, o modelo apresenta uma abordagem generalista mais voltada para os Sistema Gerenciador de Banco de Dados (SGBD) (MACHADO, 2008, p. 21).

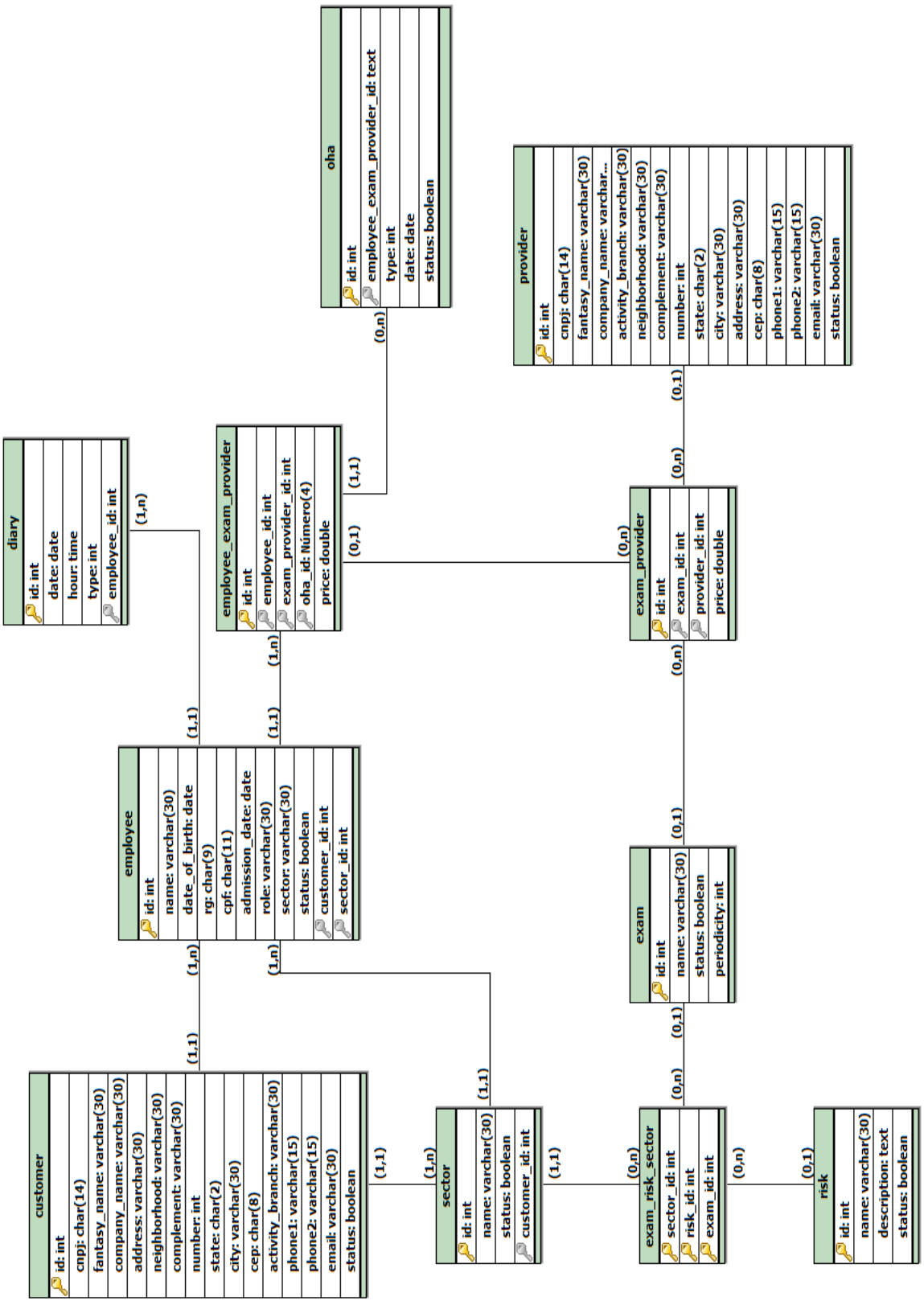
Segundo Heuser (2009, p.26), o modelo logico é “uma descrição de um banco de dados no nível de abstração visto pelo usuário do SGBD. Assim, o modelo lógico é dependente do tipo de SGBD que está sendo usado”.

Dessa forma, com os dados do minimundo real do negócio coletados (modelo conceitual), pode-se seguir agora para uma abordagem mais tecnológica e, com uma linguagem mais voltada para interpretação das restrições referentes ao SGBD que será utilizado. Dessa maneira, os analistas podem identificar facilmente como a correlação das abordagens e das restrições com o relacionamento será constituído (MACHADO, 2008, p. 22).

Para isto, foi desenvolvido o DER (Diagrama Entidade Relacionamento) que tem a função de descrever o modelo logico através de simbologias referentes aos seguintes elementos: entidades (proprietária dos atributos), atributos (tipo e tamanho), *primary key* (chave primaria) e *foreign key* (chave estrangeira). Através das simbologias que o diagrama fornece qualquer analista que entenda do assunto poderá compreender o esquema lógico do banco de dados.

Para o projeto de banco de dados em questão, foi desenvolvido o seguinte diagrama entidade relacionamento (Figura 20).

Figura 20 - DER



Fonte: Elaborado pelos autores.

7.5 Modelo físico

O modelo físico somente é produzido após a conclusão do modelo lógico, visto que representando a etapa final do projeto de banco de dados. Além disso, o modelo é utilizado nessa fase para descrever as estruturas de armazenamento de dados, as quais, conforme Machado (2008, p. 22) são definidas da seguinte forma:

- Tipo e tamanho de campo;
- Índices;
- Domínio de preenchimento desses campos;
- Nomenclaturas;
- Exigência de conteúdo;
- Gatilhos (*Triggers*).

As estruturas apresentadas são importantes na construção do modelo físico pois quando analisadas de forma detalhada pelos analistas, pode-se construir um projeto de banco de dados econômico em recursos computacionais, ou seja, resultando na utilização eficiente dos recursos do SGBD escolhido.

8 HEURÍSTICAS DE NIELSEN

As heurísticas de Nielsen tratada nesta seção fazem referência ao estudo dos princípios de usabilidade que foram desenvolvidos através do estudo da engenharia de usabilidade. Tais princípios são representações de métricas que ditam as condições necessárias para determinada interface ser considerada de fácil utilização pelos usuários, assim, transmitindo uma experiência de utilização agradável e amigável.

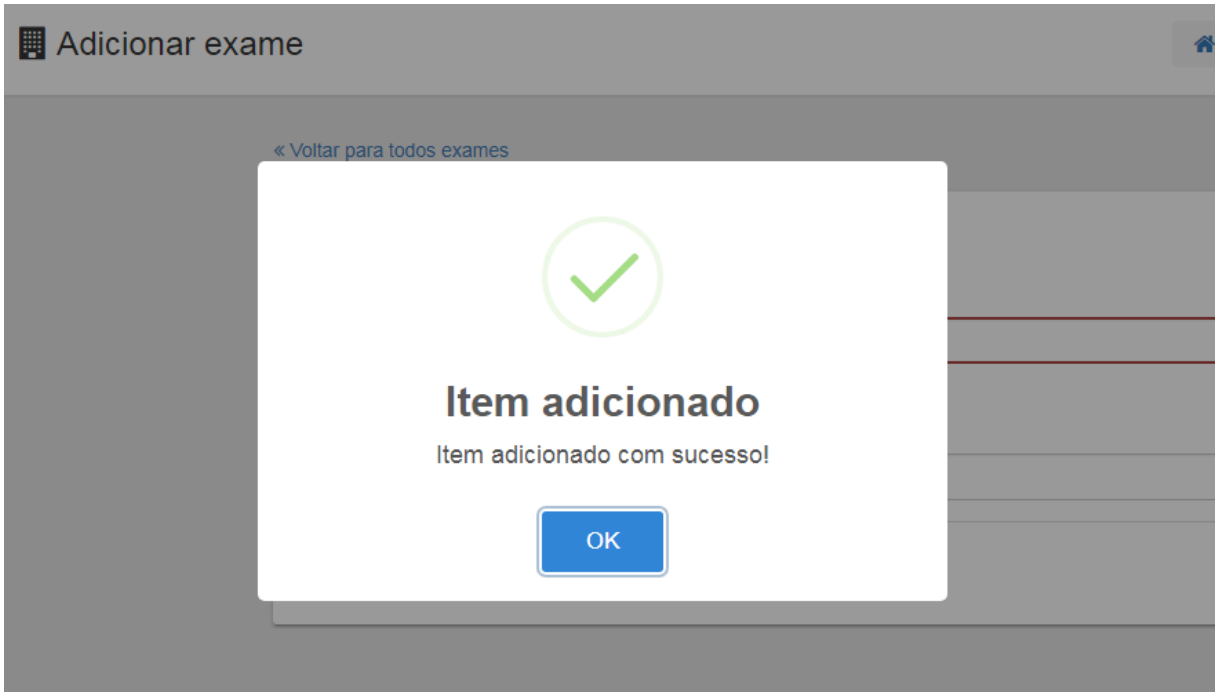
8.1 Visibilidade de Status do Sistema

De acordo com o ANEXO 1 - HEURÍSTICAS, a visibilidade de status do sistema necessita manter o usuário sempre informado dos acontecimentos dentro sistema, sendo permitido a utilização de avisos adequados que informa em tempo hábil o usuário do sistema.

Como esclarecido pela a definição acima, manter o usuário sempre informado do que está acontecendo a cada ação que o mesmo realizar no sistema, demonstra ser de grande influência na experiência de navegação que o usuário terá.

Para tal, a heurística em questão, foi aplicada nas notificações do sistema como demonstrado na Figura 21. Dessa forma, o usuário recebe feedbacks constantes (alertas) a cada ação realizada, podem ser de ações de inserção, edição ou remoção de alguma informação do sistema.

Figura 21 – Aviso de inserção



Fonte: Elaborado pelos autores.

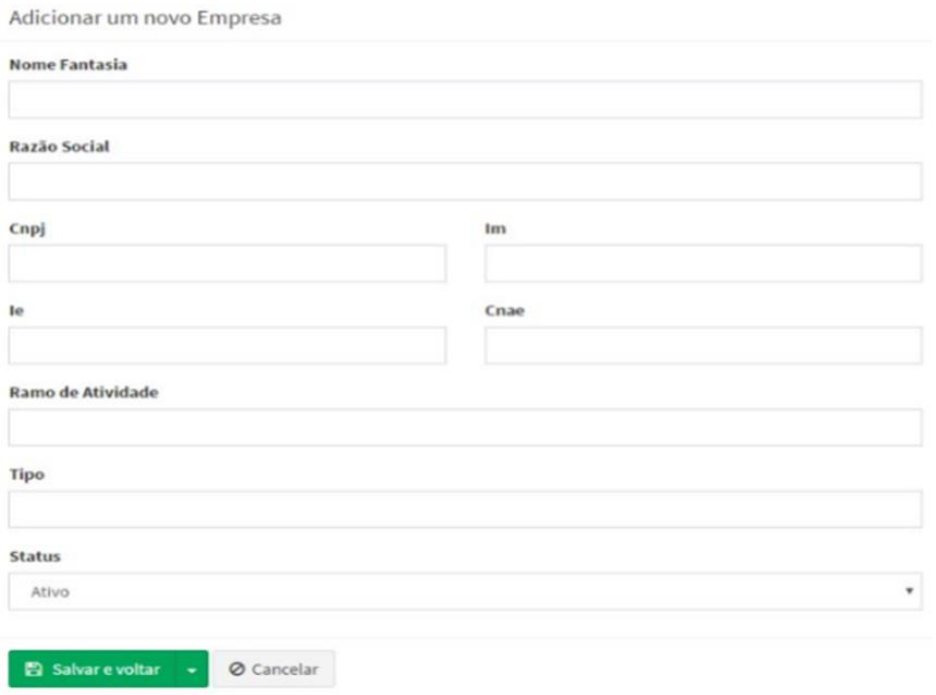
8.2 Relacionamento entre a interface do sistema e o mundo real

Segundo o ANEXO 1 - HEURÍSTICAS, o relacionamento entre as interfaces do sistema e o mundo real necessitam ter a mesma linguagem dos usuários, favorecendo a utilização de palavras, frases e conceitos familiares ao invés de termos técnicos. Assim, as informações serão transmitidas de forma natural e logica para o receptor (usuário).

Ou seja, a linguagem textual das interfaces do sistema necessita estar de acordo com o mesmo nível da linguagem utilizada pelos usuários, assim, permitindo que a experiência com a utilização do sistema seja agradável e amigável com os usuários.

Sendo assim, a heurística em questão foi aplicada no sistema de medicina ocupacional, através da análise de formulários de pericias medicas utilizadas pelos colaboradores da empresa. Desta forma, as interfaces do sistema foram desenvolvidas com mesmo designer e linguagem técnica contidas nos formulários. Como pode ser observado na Figura 22.

Figura 22 - Cadastro empresa



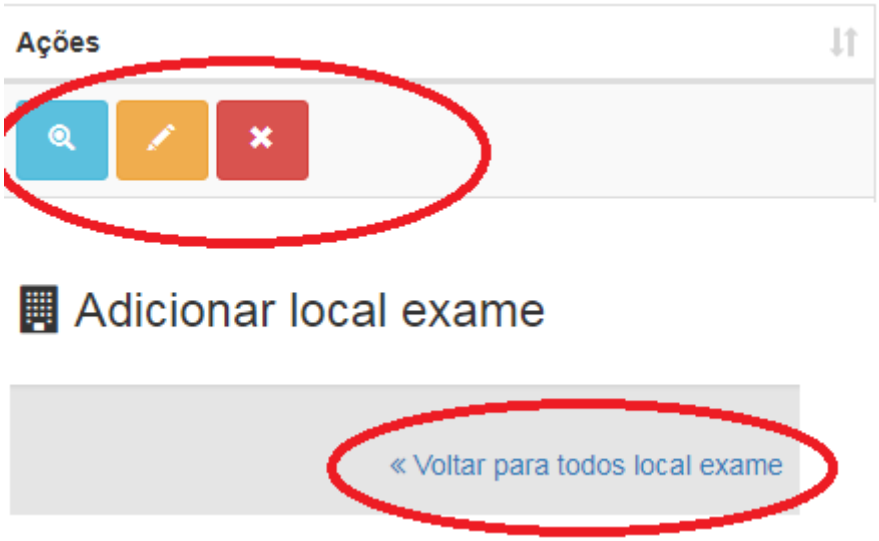
Fonte: Elaborado pelos autores.

8.3 Liberdade e Controle do Usuário

A heurística de liberdade e controle do usuário apresentada no ANEXO 1 - HEURÍSTICAS, demonstra que os sistemas necessitam apresentar funções que permitam aos usuários terem opções de escolha em caso de seleção acidental de funções do sistema. Assim, retornando ao estado de normalidade, através de possíveis funções de saída (“*undo*” e “*redo*”) disponíveis no sistema para a utilização do usuário.

Para tal heurística, foram implantadas saídas de emergências, para facilitar retornar à página anterior ou posterior. Além disso, também foram introduzidos botões para que os usuários possam desfazer ou editar ações no sistema. Tais informações mencionadas podem ser visualizadas na Figura 23:

Figura 23 – Botões de saída



Fonte: Elaborado pelos autores.

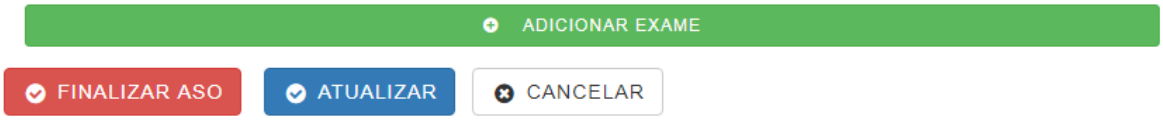
8.4 Consistência e Padrões

Conforma o ANEXO 1 - HEURÍSTICAS, a heurística de consistência e padrões ressalta que o sistema precisa ser muito bem definido em relação a utilização de palavras, situações ou ações que significam a mesma coisa, pois, os usuários não podem ter dúvidas e suposições durante a experiência de navegação no sistema.

Como dito, as interfaces do sistema não podem permitir suposições ou dúvidas em relação algum tipo de ação realizada pelo usuário. Dessa forma, as interfaces necessitam estar organizadas e padronizadas, assim, entendendo as funções principais de uma determinada interface as demais, necessariamente, seguiram os mesmos padrões de layout, botões, menus e etc.

Para tal, está heurística foi aplicada no design dos botões, os quais, para botões de ações distintas apresentam colocação diferenciada, isso facilita o reconhecimento das funcionalidades de cada botão do sistema, além de permitir uma experiência mais agradável e amigável para os usuários, demostrado na Figura 24.

Figura 24 – *Design* botões



Fonte: Elaborado pelos autores.

8.5 Prevenção de Erros

A heurística de prevenção de erros de acordo com o apresentado no ANEXO 1 - HEURÍSTICAS, demonstra ser um fator de grande importância para o sucesso do sistema. Para isto, um sistema necessita ser validado por processos de verificações de erros (plano de testes), que minimizara a possibilidade de erros no sistema. Sendo assim, sistema que desfrutam destas heurísticas de usualidade, permitem aos usuários confirmarem determinada ação antes que seja decretar como executada, assim conferindo maior segurança para os usuários e a integridade do sistema.

Como apresentado, o sistema necessita ser capaz de emitir ao usuário mensagens de alertas ou erros, além de apresentar uma linguagem que os usuários compreendam e, também, opções de escolhas que facilitem a tomada de decisão provinda dos usuários.

Para tal, está heurística foi aplicada em janelas de confirmação de exclusão de registros, como demostrado na Figura *25*. Assim, torna-se possível minimizar erros provenientes de descuidos dos usuários ao deletar uma determinada informação do sistema.

Figura 25 - Tela de confirmação

Fonte: Elaborado pelos autores.

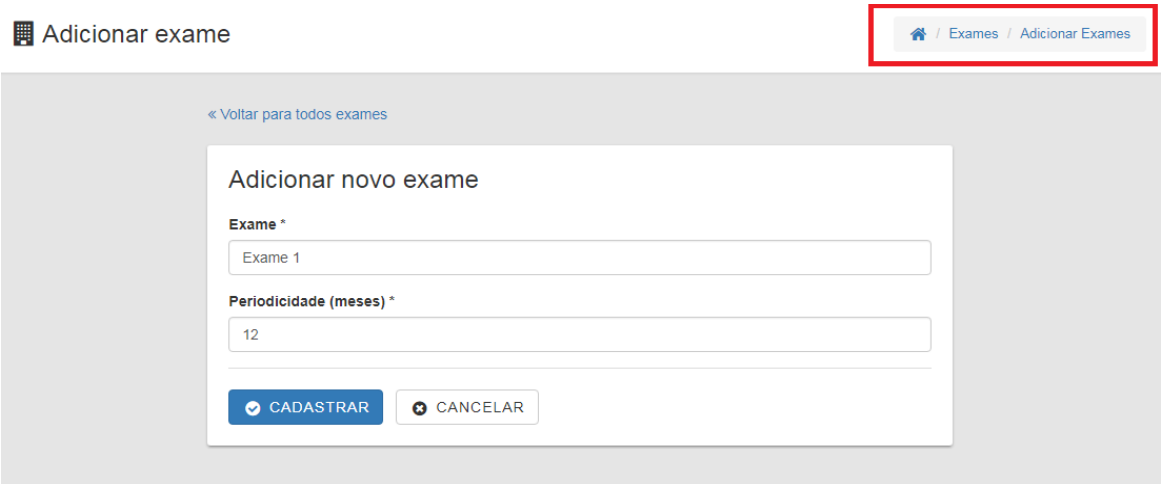
8.6 Reconhecimento ao invés de Lembrança

Segundo o ANEXO 1 - HEURÍSTICAS, sistemas que fazem uso da heurística em questão, apresentam as instruções de utilização do sistema sempre de forma visível aos usuários. Assim, facilita a navegação no sistema, pois a cada interface que o usuário navega os objetivos, ações e opção sempre são disponibilizadas de foram visíveis e intuitivas. Dessa forma, os usuários não apresentam sobre carga de memória durante a utilização do sistema, pois, não há necessidade do uso da lembrança.

Dessa forma, o sistema necessita apresentar determinada padronizarem em suas interfaces para que o reconhecimento seja tenha realmente um efetivo. Assim, o usuário pode utilizar o sistema sem ter a necessidade de lembrar o que fazer em determinada situação do sistema, pois as interfaces foram desenvolvidas para serem intuitivas aos olhos dos usuários.

Esta heurística se faz perceptível no sistema quando o usuário navega entre as interfaces (telas). Como demonstrando na Figura 26, é possível o usuário identificar facilmente a página atual, além das páginas anteriores.

Figura 26 - *Breadcrumb*



Fonte: Elaborado pelos autores.

8.7 Flexibilidade e eficiência de uso

A heurística de flexibilidade e eficiência apresentada no ANEXO 1 - HEURÍSTICAS**,** está relacionada a aceleradores presentes no sistema, que permitem aos usuários mais avençados usufruir de opções ou atalhos que permitem maior flexibilidade, agilidade e performance do sistema.

Quando se diz “aceleradores” estamos referenciando a teclas de atalhos, pois permitem maior produtividade e eficiência nas realizações de atividade no sistema. Normalmente, os “atalhos” são mais utilizados por usuários mais avançados, entretanto, a disponibilidade dos mesmos faz necessário para qualquer nível de usuário do sistema.

Está heurística não foi implementada no sistema, pois não havia a necessidade de criar atalhos, uma vez que o acesso as telas e o preenchimento das informações já são facilmente identificados e intuitivos.

8.8 Designer estético e minimalista

Conforme ressaltado no ANEXO 1 - HEURÍSTICAS, o designer estético e minimalista de um sistema necessita ser o mais objetivo possível na transmissão da informação para o usuário, pois, informações irrelevantes e raramente necessárias diminuem a visibilidade dos elementos da interface, dificultando a navegação dos usuários.

Os diálogos permitem necessitam apresentar a transmitir informação necessária para o usuário da forma mais direta possível, assim minimizando os efeitos de interferências de comunicação entre interface e usuário.

A heurística em questão foi aplicada na tela de *login* do usuário, nela contém somente informações necessárias para que o usuário consiga acessar o sistema. Como pode ser visualizado na Figura 27:

Figura 27 - Tela *login*

Fonte: Elaborado pelos autores.

8.9 Suporte para o usuário, diagnosticar e recuperar erros

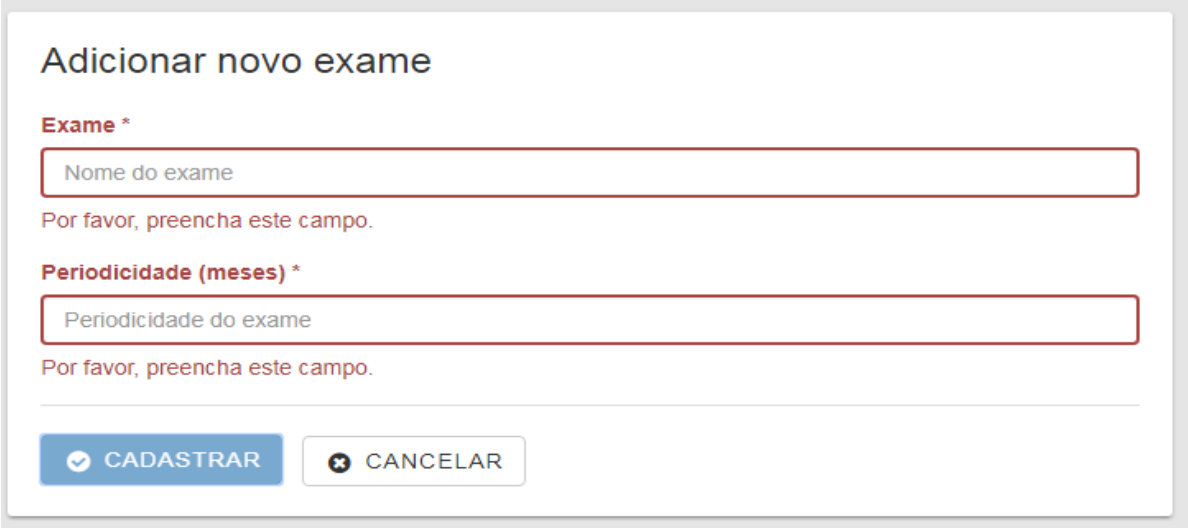
De acordo o ANEXO 1 - HEURÍSTICAS, a heurística em questão trata da transmissão de mensagens de suporte a erros permite que os usuários entendam seus respectivos erros e possam soluciona-los o mais rápido possível. Para isto, a linguagem utilizada nos diálogos necessitam ser a mesma que a dos usuários, os

diagnostico de erros do sistema necessitam ser efetivos e a possível solução necessita ser de simples compreensão.

A mensagem de suporte a erros permite que os usuários entendam seus respectivos erros e possam soluciona-los o mais rápido possível. Para isto, a linguagem utilizada nos diálogos necessitam ser a mesma que a dos usuários, os diagnostico de erros do sistema necessitam ser efetivos e a possível solução necessita ser de simples compreensão.

Na Figura 28 é possível observar a aplicação desta heurística, na qual foi aplicada na interface de formulário, onde o usuário precisa preencher todos os campos obrigatórios. No entanto, havendo algum campo incompleto ou incorreto, o sistema alertara o usuário os campos que necessitam ser preenchidos corretamente.

Figura 28 - Tela de erro



Fonte: Elaborado pelos autores.

8.10 Ajuda e Documentação

A heurística de ajuda e documentação do sistema segundo exposto no ANEXO 1 - HEURÍSTICAS, todo sistema computacional necessita disponibilizar formas de auxiliar os usuários a sanar possíveis duvidas recorrentes da interação com o sistema,

podendo ser recurso físico ou digital. Tais documentos, precisam apresentar passos concretos, serem objetivos, acessíveis e estarem na linguagem do usuário.

Como exposto, todo sistema computacional necessita disponibilizar formas de auxiliar os usuários a sanar possíveis duvidas recorrentes da interação com o sistema, podendo serem em recurso físico ou digital.

Além disso, a necessidade de manuais e ferramentas de help integradas ao sistema são de grande auxilio ao usuário. No entanto, necessitam estar acessíveis na linguagem utilizada pelos usuários.

Para está heurística foi desenvolvido um manual do usuário, nele contém todas as informações necessárias para a utilização o sistema em uma linguagem que o usuário possa compreender de forma simples. Este manual pode ser acessado no APÊNDICE 1 – QR CODE COM DOCUMENTOS ADICIONAIS.

9 FERRAMENTAS PARA O DESENVOLVIMENTO

Neste capítulo serão apresentadas as tecnologias e ferramentas de desenvolvimento que foram utilizadas no projeto, as escolhas foram feitas levando em consideração que as mesmas fossem gratuitas e que atendessem as necessidades do projeto.

9.1 Programação

Nesta seção serão apresentas as tecnologias de desenvolvimento *web* aplicadas no projeto.

9.1.1 Linguagem HTML

O HTML vem da sigla *Hypertext Markup Language* que significa Linguagem de Marcação de Hipertexto e é umas das principais linguagens para o desenvolvimento web, sendo considerada a linguagem básica da internet (EIS, 2011).

A linguagem HTML foi desenvolvida por Tim Berners-Lee no início dos anos 90, mais precisamente quando o navegador *Mosaic* começou a ganhar força, foi neste instante que outros desenvolvedores de browsers começaram a usar o HTML como mesma convenção (FERREIRA, EIS, [200-?] p. 7). Entre 1993 e 1995 o HTML teve algumas versões, o grande problema foi que elas não tinham nenhum padrão, apenas no ano de 1997 a W3C criou uma versão padronizada, essa versão era a 3.2 (FERREIRA, EIS, [200-?], p. 8).

Para GUIMARÃES (2005) a linguagem HTML é voltada para:

“- Estruturação de documentos;

-Apresentação visual de documentos em um navegador (“browser”). “

Um documento HTML é composto por uma hierarquia de elementos, estes elementos são demarcados por duas *tags*, a primeira *tag* possuí a seguinte formação: <nome-da-*tag* atributos (opcional) > (GUIMARÃES, 2005).

Os atributos da *tag* inicial definem alguma característica do elemento. A *tag* final possuí a seguinte estrutura:</nome-da-*tag*> (GUIMARÃES, 2005).

O conteúdo dos elementos que ficam entre as *tags* final e inicial, podem ser textos ou outros elementos HTML, lembrando que a raiz de todo documento HTML é a *tag* <*html*> (GUIMARÃES 2005).

A grande vantagem do HTML é esta linguagem ter a característica de ser interoperável, ou seja, um código escrito em HTML pode ser acessado por diferentes plataformas, independentemente de sua capacidade técnica, isso aumenta o acesso de suas informações e diminui os custos de desenvolvimento (FERREIRA, EIS, [200-?], p. 8).

9.1.2 Linguagem CSS

O CSS vem da sigla *Cascading Style Sheets* e tem como finalidade controlar a aparência de uma página web, como por exemplo, a altura, a largura, a cor, fonte, margem, espaço entre diferentes elementos, entre outros (DUCKETT, 2010, p. 255). Isso é feio a partir de regras que são definidas para cada elemento de uma página web, com isso o site apresenta uma aparência muito mais atraente (DUCKETT, 2010, p. 255). Antigamente o HTML possuía alguns atributos que definiam a aparência da página, porém a w3c decidiu que o CSS deva assumir essa função (DUCKETT, 2010, p. 255).

Em 1994, Hackon Lie propôs a criação do CSS, diante das dificuldades encontradas na complexidade do HTML de possuir atributos que definiam a formatação das informações, como foi dito acima e também pelo problema de compatibilidade entre os navegadores da época o que fazia uma página funcionar corretamente em um navegador e em outro não. Em 1996 surge a primeira versão do CSS (CSS *level* 1) e em 1998 surge o CSS *level* 2 (BARROS; SANTOS, 2008, p. 3). Atualmente existe a versão 3 do CSS.

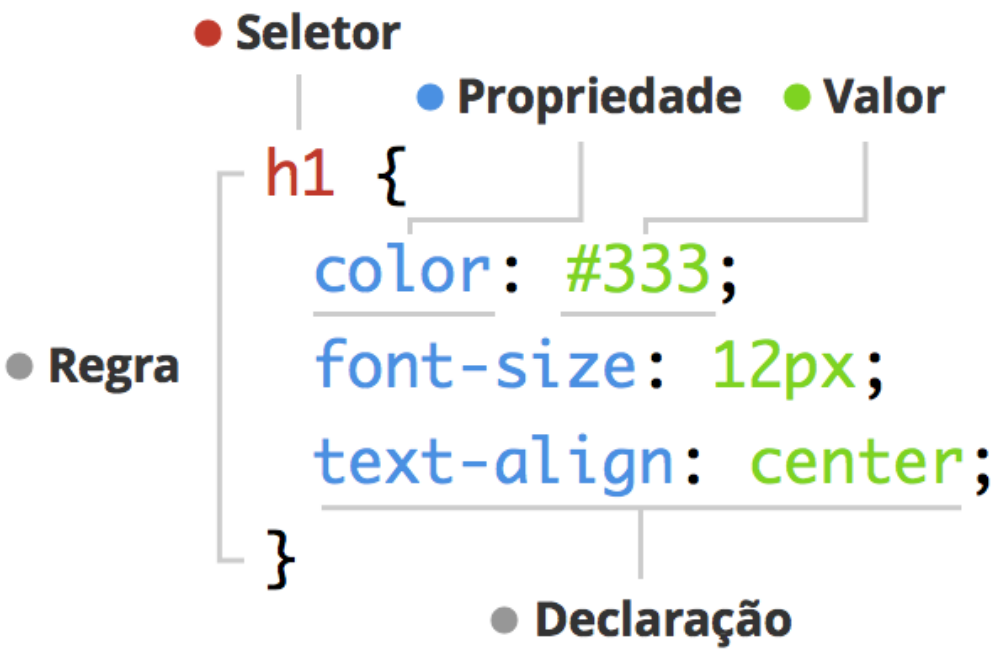
As vantagens de utilizar o CSS são a maior facilidade de manutenção nos códigos, pois a codificação do layout e das informações são separadas, o que permite também apresentar uma informação várias vezes com o visual diferente, outra

vantagem do CSS é a sua maior variedade de opções de formatação e também é que a maioria dos navegadores o suportam (BARROS; SANTOS, 2008, p. 3 - 4).

Como já foi dito, o CSS atribui regras para cada elemento do HTML, uma regra é dividida em duas partes, a primeira é chamada de *seletor* que indica em qual elemento HTML será aplicado a regra, a segunda parte é a *declaração* onde estão as propriedades que serão atribuídas ao elemento deve aparecer (DUCKETT, 2010, p. 256 - 257). Cada propriedade possui um valor que a define (DUCKETT, 2010, p. 256 - 257).

A Figura 29 mostra a estrutura de uma regra CSS:

Figura 29 - Regra CSS



Fonte: SANTOS (2014)

9.1.3 Linguagem PHP

O PHP é uma linguagem de programação criada por Rasmus Lerdorf no ano de 1994, de início o PHP era apenas um conjunto de scripts desenvolvidos em C em eu Rasmus utilizava para monitorar o acesso ao seu currículo na Internet, com o passar do tempo outras pessoas começaram a usar o PHP, foi aí que em 1995

Rasmus liberou o código do PHP, fazendo com que mais desenvolvedores participassem do projeto (DALL'OGLIO, 2016, p. 1).

Em 1997 após várias reescritas em seu código, foi lançada a segunda versão, o PHP 2, neste momento cerca de 1% dos domínios da Internet eram feitos em PHP (aproximadamente 60 mil domínios), na mesma época os estudantes Andi Gutmans e Zeed Suraski, ajudaram Rasmus a aprimorar o PHP, com essa parceria surgiu o PHP 3 em 1998. Esta nova versão ganhou novas funcionalidades, como por exemplo, conexão com vários bancos de dados, suporte a orientação à objeto e uma nova API, neste momento o PHP já estava em 10% dos domínios da Internet (DALL'OGLIO, 2016, p. 2).

O PHP 4 lançado em 2000, trazia consigo o núcleo Zend Engine que tinha sido desenvolvido por Andi e Zeed e trazia melhorias, como, maior suporte a servidores web, abstração da API e seções. Quatro anos mais tarde a versão cinco do PHP foi lançada com melhoramentos ao suporte a orientação à objeto, estima-se que atualmente cerca de 80% dos domínios da Internet sejam em PHP (DALL'OGLIO, 2016, p. 2).

A linguagem PHP é uma linguagem de *script*, ou seja, não é gerado um arquivo no momento da execução e sim interpretado e executado no momento de sua requisição (TOSING, 2008, p. 3).

O PHP gera páginas web dinamicamente, como as páginas web dinâmicas necessitam de um ambiente *client-server* (cliente-servidor) para funcionar, logo o PHP roda no lado servidor (TOSING, 2008, p.3).

De acordo com Soares (2007, p. 4) o PHP pode fazer qualquer coisa que um sistema CGI (*Common Gateway Interface*) faz, ou seja, enviar receber *cookies*, capturar dados de formulários, gerar páginas dinamicamente. Ainda segundo Soares, o PHP tem suporte à um variado número de banco de dados, entre eles podemos citar, Oracle, mySQL, Sysbase, PostgreSQl, etc, o PHP tem suporte a protocolos IMAP, SNMP, NNTP, POP3 e HTTP.

A seguir a Tabela 2 mostra o comparativo de preço entre as tecnologias que segundo Soares (2007, p. 1) pertencem ao mesmo nicho de mercado.

Tabela 2 - Comparativo de preços entre as tecnologias do mercado

Item	ASP	*Cold Fusion*	JSP	PHP
Desenvolvimento	US$ 0-480	US$ 395	US$ 0	US$ 0
Servidor	US$ 620	US$ 1.295	US$ 0-595	US$ 0
RDBMS	US$1.220-4220	US$0-~10.000	US$0-~10.000	US$ 0
Suporte de incidente	US$ 0-245	US$ 0-75	US$ 0-75	US$ 0

Fonte: SOARES (2007, p. 5)

9.1.4 JavaScript

A linguagem JavaScript segundo Grillo e Fortes (2008, p. 3) é uma linguagem dinâmica, que roda no navegador do cliente (*cliente-side*) e que possui características de orientação à objetos.

JavaScript é uma linguagem de script, isso significa que ó código é digitado e executado por um interpretador, sem a necessidade de se criar um arquivo binário, ou seja, caso se queira alterar algo no programa, é necessário apenas alterar o código para que a alteração tenha efeito, como o código é executado por um interpretador, pode-se chamar o JavaScript de linguagem interpretada (GRILLO; FORTES, 2008, p. 4).

Antes de receber o nome de JavaScript, outros nomes foram adotados, como Mocha e LiveScript até o seu lançamento em 1995 com o nome de JavaScript (GRILLO; FORTES, 2008, p. 4).

O JavaScript permite diretamente do computador do cliente, validar formulários, alterar valores de elementos HTML, criar elementos HTML, etc. Os códigos ficam embutidos em um documento HTML, como o script roda do lado cliente a execução é mais rápida, pois não se perde tempo com latência da rede (GRILLO; FORTES, 2008, p. 4).

9.1.5 Bootstrap

Bootstrap é um framework *front-end* que fornece uma grande variedade plug-ins em JavaScript para a criação de componentes, como *carousel*, *slideshow, menu-dropdown*, *modal*, entre outros, também o Bootstrap facilita a criação de sites com tecnologia mobile (responsivo) sem a necessidade de se criar códigos CSS, diminuindo o tempo de desenvolvimento (MUSSEL, 2017, p. 4).

Segundo Mussel (2017, p. 4) o Bootstrap possuí as seguintes características:

- Possui uma interface super amigável e moderna;
- Atualmente possui uma grande diversidade de temas;
- Grande quantidade de *plugins* adaptados ou desenvolvidos para o framework;
- Sistema responsivo;
- Integração com qualquer linguagem de programação;
- Um dos frameworks mais utilizados no desenvolvimento de portais e sistemas do mundo;
- Totalmente FREE!

O Bootstrap foi anunciado em 2011 em um artigo publicado no Twitter por Mark Otto (criador do Bootstrap) e Jacob Thornton, Otto que estava trabalhando como desenvolvedor no Twitter, relatou que os desenvolvedores utilizavam bibliotecas com que tinham mais afinidade, isso gerava uma inconsistência na integração e na manutenção das aplicações, foi então que o Bootstrap surge como solução para esses problemas (SILVA, 2015, p. 21).

Para usar o Bootstrap é necessário baixa-lo no site *http://getbootstrap.com*, adicionar os arquivos bootstrap.min.css e bootstrap.min.js ao projeto que está sendo desenvolvido e criar um link CSS e outro JS dentro da *tag* <*head*> do documento HTML, ai é só aproveitar suas funcionalidades (MUSSEL, 2017, p. 4).

É importante também que se tenha instalado o JQuery para que o Bootstrap funcione corretamente (MUSSEL, 2017, p. 4).

9.2 Ferramentas

Nesta seção serão apresentadas as ferramentas(softwares) utilizados para auxiliar o desenvolvimento e gerenciamento do projeto.

9.2.1 MySQL WorkBench

Para o gerenciamento do banco de dados foi utilizado o MySQL WorkBench, ferramenta de propriedade da empresa Oracle. O MySQl WorkBench possui uma versão *community* que satisfaz as necessidades do projeto.

No site do MySQL(2017, *tradução nossa*) se encontra uma definição do MySQL WorkBench:

> MySQL WorkBench é um conjunto de ferramentas visuais para arquitetos de banco de dados, desenvolvedores e DBAs. MySQL WorkBench fornece modelagem de dados, desenvolvimento em SQL e um completo conjunto de ferramentas para configuração de servidores, administração de usuários, backup e muito mais. MySQL WorkBench está disponível para Windows, Linux e Mac OS X.

As funcionalidades apresentadas acima, como por exemplo, a modelagem de dados e desenvolvimento em SQL e também a disponibilidades para Linux, Windows, foi de grande utilidade para o desenvolvimento do projeto.

9.2.2 Astah

Para o modelagem dos diagramas UML(Diagrama de Classe, Caso de Uso, Sequência, Máquina de Estados e Diagrama de Atividades) foi utilizado a ferramenta Astah *Community.*

Rosin(2011, p. 20) faz a seguinte descrição sobre o Astah: “Astah *Community* (ASTAH, 2010) é uma ferramenta de modelagem gratuita para projeto de sistemas orientados a objeto. É baseada nos diagramas e na notação da UML 2.0 gera código em Java.”

Como dito acima, o Astah *Community* é uma ferramenta gratuita, o que ajudou no abatimento de custos no projeto.

9.2.3 Git

O controle de versionamento é importante para manter uma organização dos arquivos que compõem um projeto de software, principalmente no caso de alteração de requisitos, onde é necessário alterar alguma parte do código fonte do sistema, ou quando existe várias pessoas que compõem a equipe de desenvolvimento e é necessário unir todos os arquivos e códigos produzidos.

No site do Git([201-?], *tradução nossa*) se encontra a seguinte definição para o Git:

> Git é uma distribuição de sistema de controle de versão livre e open source projetado para lidar com tudo desde pequenos a grandes projetos com velocidade e eficiência.
> Git é fácil de aprender e tem uma pequena pegada com desempenho rápido. Ela ultrapassa ferramentas SCM como Subversion, CVS, Perforce, e ClearCase com características como ramificação local barata, convenientes staging areas e múltiplos fluxos de trabalho

Tendo esta característica de fácil aprendizagem e ser *open source* o Git foi ideal para fazer o controle de versionamento local de arquivos.

9.2.4 Bitbucket

Como já foi dito, o Bitbucket tem como função manter o versionamento dos arquivos do sistema remotamente, em se tratando de uma equipe de quatro pessoas que desenvolveram diferentes funcionalidades do sistema, foi de suma importância um sistema como Bitbucket que faça, além do controle de versão, mas também a junção dos trabalhos de cada membro.

No site oficial do Bitbucket (2017, Tradução nossa) se encontra uma pequena definição sobre o Bitbucket: “Sistema de controle de versão distribuída que facilita a colaboração com sua equipe. A única solução colaborativa do Git que escala massivamente.”

9.2.5 Xampp

No site do XAMPP (2017) tem uma definição sobre este programa:

> O XAMPP é o ambiente de desenvolvimento mais popular.
> O XAMPP é completamente gratuito, de fácil de instalar a distribuição Apache, contendo MySQL, PHP e Perl. O pacote de código aberto do XAMPP foi criado para ser extremamente fácil de instalar e de usar.

Como este trabalho trata-se de um sistema desenvolvido em PHP que possui um banco de dados gerenciado pelo MySQL e como o PHP é uma linguagem *server-side* foi necessário um programa que simulasse um servidor web em uma máquina cliente para que se pudesse desenvolver e testar o sistema.

Portanto sendo o XAMPP uma plataforma de desenvolvimento gratuita contendo o MySQL e o PHP, ele foi de muita importância para o desenvolvimento do projeto.

9.2.6 Trello

O Trello é uma ferramenta que faz a organização de tarefas facilitando o gerenciamento de uma equipe em relação a distribuição de tarefas e prazos de entrega.

O Trello permite a criação de quadros com uma lista de cartões, estes cartões estão relacionados a alguma tarefa (independente se esta tarefa foi ou será feita), nestes cartões é possível adicionar pessoas que estão relacionadas a tarefa, anexar algum arquivo, definir prazo de término, adicionar *checklist*, adicionar etiqueta, além disso é possível arrastar um cartão de um quadro para outro (TRELLO, 2017).

Portanto o Trello foi essencial para a organização da equipe na divisão de tarefas e para as entregas no prazo determinado.

9.2.7 MS-Project

Houve a necessidade de se utilizar uma ferramenta para o gerenciamento do projeto, a ferramenta escolhida para isso foi o Ms-Project.

No site da Microsoft (2017) são mostradas informações sobre o Ms-Project:

> O PPM (Gerenciamento de portfólio de projetos) da Microsoft o ajuda a iniciar e executar os projetos rápida e facilmente. Os modelos incorporados, as ferramentas de programação familiares e o acesso a partir de vários dispositivos aumentam a produtividade de gerentes e equipes de projetos.

No projeto ele foi de suma importância, sendo uma ferramenta complementar junto com o Trello para controlar o cronograma e as entregas de trabalho.

9.2.8 Sublime Text

Para a codificação do sistema foi necessário utilizar uma ferramenta que fosse gratuita e permitisse a programação em diferentes linguagens, por exemplo o PHP e o HTML, para esta tarefa foi utilizado o Sublime Text 3.

Segundo informações do site do desenvolvedor do Sublime Text (2017, tradução nossa): “Sublime Text pode ser baixado e avaliado gratuitamente, no entanto, uma licença deve ser comprada para uso continuado. Atualmente não há limite de tempo para a avaliação.”

Com o limite indeterminado de avaliação da IDE, foi possível desenvolver o sistema sem gerar custos e atendendo as necessidades da equipe.

10 INFRAESTRUTURA DE TECNOLOGIA DA INFORMAÇÃO

Nesta seção será abordado a infraestrutura necessária para a implementação de software dentro da empresa e também das licenças necessárias para desenvolver o produto.

10.1 Infraestrutura de redes

Para utilizar a aplicação é necessário que todas as máquinas da rede (aquelas que irão utilizar o sistema) possuam conexão com a internet, pois a aplicação não será hospedada no local.

Nesse quesito a empresa atualmente já possui a infraestrutura necessária para o funcionamento do sistema, uma vez que, todos os computadores que acessarão o sistema já possuem internet.

10.2 Link de internet

O software necessita de conexão de internet pois será hospedado online, assim, é importante que a empresa contrate 2 links de internets. Contratando 2 links de internet a disponibilidade da aplicação será maior, pois pode haver queda de conexão em um dos links.

É interessante que os links contratados sejam diferentes, exemplo, uma conexão de fibra ótica e a outra via rádio, pois caso esteja chovendo é bom mudar para conexão via cabo e se o poste da operado do link tiver algum problema mudar para a conexão via rádio.

É recomendado que as conexões possuam uma velocidade de 5 Mbs, para que a aplicação funcione sem problemas de lentidão, assim, permitindo uma melhor experiência aos usuários.

10.3 Aquisições de equipamentos e periféricos

Em relação aos computadores da empresa não será necessário um investimento tão grande, pois a aplicação será rodada na web, sendo assim, não existe necessidade de uma máquina muito potente, pois o único aplicativo que deve ser instalado para acessar a aplicação é um navegador.

Na Figura 30 e Figura 31 é possível observar os navegadores dos dispositivos móveis e computadores que são compatíveis com o sistema:

Figura 30 - Dispositivos móveis compatíveis

	Chrome	Firefox	Safari	Android Browser & WebView	Microsoft Edge
Android	Supported	Supported	N/A	Android v5.0+ supported	N/A
iOS	Supported	Supported	Supported	N/A	N/A
Windows 10 Mobile	N/A	N/A	N/A	N/A	Supported

Fonte: BOOTSTRAP (2017).

Figura 31 - Navegadores desktops compatíveis

	Chrome	Firefox	Internet Explorer	Microsoft Edge	Opera	Safari
Mac	Supported	Supported	N/A	N/A	Supported	Supported
Windows	Supported	Supported	Supported	Supported	Supported	Not supported

Fonte: (BOOTSTRAP, 2017).

10.4 Aquisições de licenças de softwares

Para o desenvolvimento do sistema foram utilizadas diversas ferramentas que foram descritas na seção FERRAMENTAS PARA O DESENVOLVIMENTO, dentre as aplicações utilizadas, somente o Microsoft Project possui à necessidade de comprar a licença, os softwares restantes são gratuitos ou possuem uma versão *free*.

No site da Microsoft uma licença da versão 2016 online do Project está sendo comercializada no valor de R$ 116,20, sendo essa taxa cobrada mensalmente, também é possível encontrar a versão para desktop no valor de R$ 4.999,00.

11 PLANO DE TESTES

Segundo Rios e Moreira (2006, p. 8) o teste de software é: "[...]é um processo que visa a sua execução de forma controlada, com o objetivo de avaliar o seu comportamento baseado no que foi especificado[...]"

Para Koscianski e Soares (2006, p. 337):

> O objetivo do teste é encontrar defeitos, revelando que o funcionamento do software em uma determinada situação não está de acordo com o que foi esperado. Um teste bem-sucedido identifica defeitos que ainda não foram descobertos e que podem ser, então, corrigidos pelo programador. Quando a atividade de teste é planejada de maneira sistemática e rigorosa, pode ser utilizada como um dos parâmetros para estimar a confiabilidade e a qualidade do software construído.

Como foi mostrado acima, o teste de software é de extrema importância para se determinar se um software atende a todos os requisitos do cliente, seja estes requisitos funcionais ou de qualidade e identifica a presença de "*bugs*" ou erros de programação.

A atividade de teste de software não se restringe apenas no produto de software pronto, mas sim, durante todo o seu processo de desenvolvimento, conforme diz Delamaro, Maldonado e Jino (2007, p. 2):

"Atividades de VV&T não se restringem ao produto final. Ao contrário, podem e devem ser conduzidas durante todo o processo de desenvolvimento do software, desde a sua concepção, e englobam diferentes técnicas."

No projeto desenvolvido pelo grupo foram utilizados dois tipos de teste: o teste de caixa-preta e o teste de caixa-branca que saram descritos a seguir.

11.1 Teste de Caixa Branca

Segundo Pressman (1995, p.791-792):

> O teste de caixa branca do software baseia-se num minucioso exame dos detalhes procedimentais. Os caminhos lógicos através do software são testados, fornecendo-se casos de teste que propõem à prova conjuntos específicos de condições e/ou laços. O “status do programa” pode ser examinado em vários pontos para determinar se o status esperado ou estabelecido corresponde ao status real.

Ou seja, o teste de caixa branca se mostra de grande valia, uma vez que através dele é possível detectar *bugs* no sistema que não poderiam ser descobertos utilizando apenas testes de caixa preta, sendo uma ferramenta importante para a identificação de erros.

Segundo Pressman (1995, p. 793), os testes de software podem ser derivados para que consigam alcançar determinados objetivos, como:

- Verificar as decisões lógicas (*true* ou *false*);
- Executar os laços de repetição dentro das limitações operacionais;
- Garantir validez das estruturas de dados;
- Verificar os caminhos independentes em uma parte do programa tenha sida executa ao menos uma vez.

No projeto desenvolvido pelo grupo, foram utilizadas duas ferramentas para realizar os testes de caixa branca, são elas: teste de caminho básico e teste de estrutura de controle, que serão descritas nos tópicos abaixo.

11.1.1 Teste de caminho básico

Segundo Pressman (1995, p.793-794):

> O método de caminho básico possibilita o projetista de teste derive uma medida da complexidade lógica de um projeto procedimental e use essa medida como guia para definir um conjunto básico de caminhos de execução. Os casos de teste derivados para exercitarem o conjunto básico têm a garantia de executar cada instrução do programa pelo menos uma vez durante a atividade de teste.

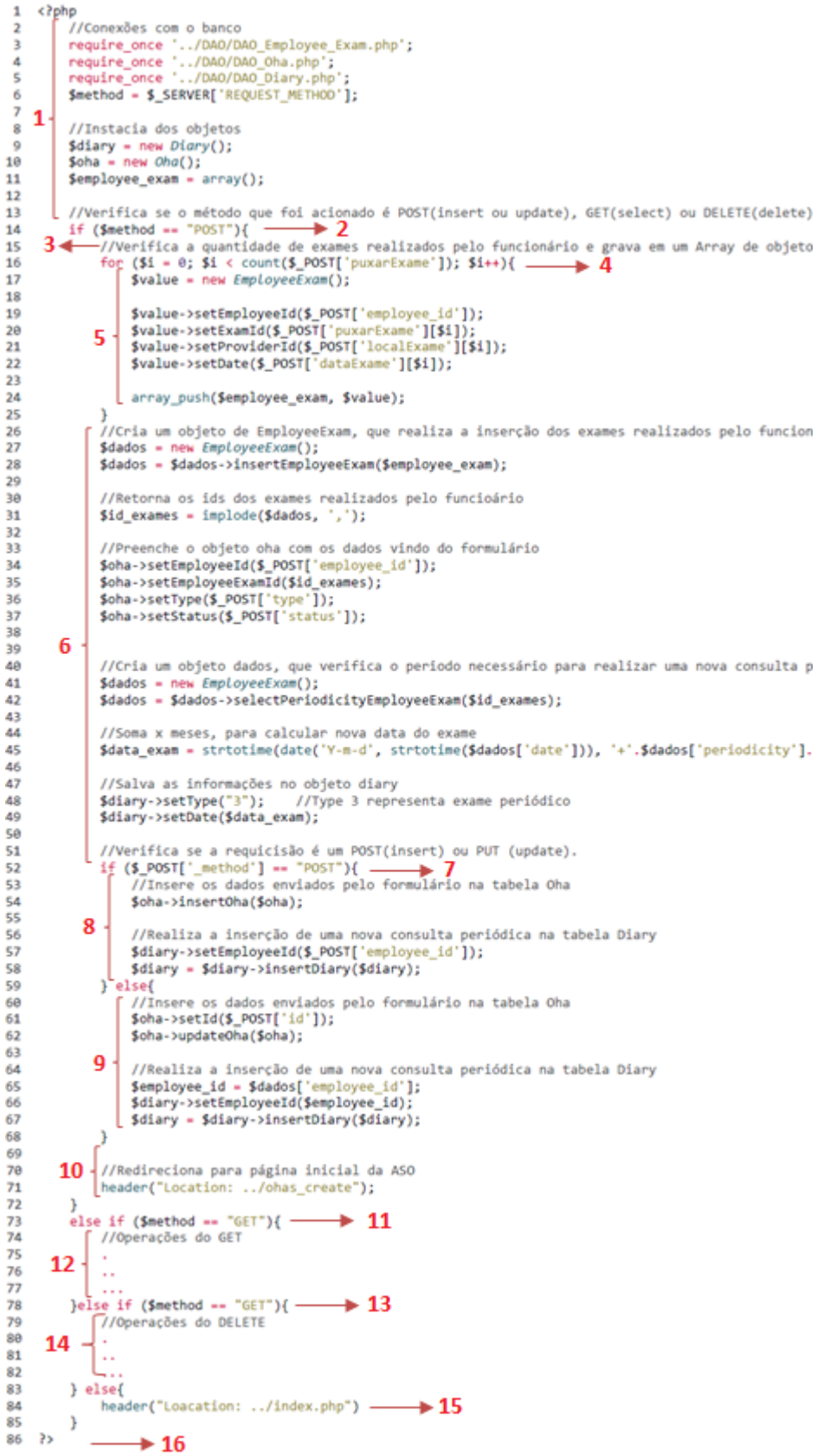
Dessa forma, é possível testar todos os caminhos possíveis de uma parte do sistema, permitindo verificar se todas as opções estão sendo executados corretamente. Assim, facilitando no processo de codificação uma vez que o desenvolvedor sabe todos os caminhos que o sistema irá tomar.

Nas figuras abaixo é possível observar a aplicação deste teste, na Figura 32 é possível observar o código fonte para geração de uma ASO divido em nós, isto permite verificar todos locais que o software poderá passar e que são necessários testar.

Na Figura 33, é possível observar o fluxo dos caminhos que podem ser executados., além de mostrá-los. Também demostra o cálculo de ciclo ciclomática, que indica a complexidade lógica de programa, este ciclo indica a quantidade de vezes

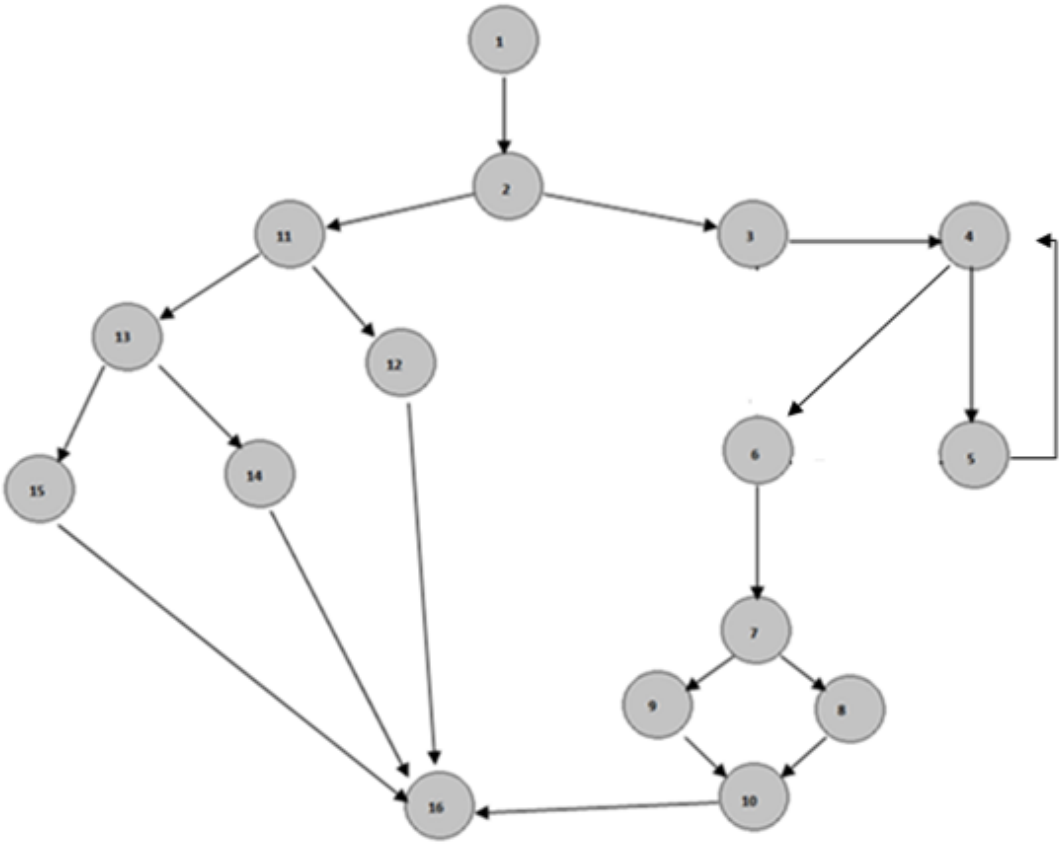
que deve ser realizado o teste para verificar se a aplicação está executando conforme os requisitos.

Figura 32 - Código fonte - identificando nós



Fonte: Elaborado pelos autores.

Figura 33 – Grafo de fluxo - caminhos independentes



Cálculo de compexidade ciclomática:

V(G) = E - N +2
V(G) = 20 - 16 + 2
V(G) = 6

Onde:
E -> número de ramos
N -> número de nós do grafo

Fonte: Elaborado pelos autores.

11.1.2 Teste de estrutura de controle

Segundo Pressman (1995, p. 807-808), o teste de caminho básico (descrito na seção acima) não é suficiente para abranger todos os testes, com o auxílio do teste de estrutura de controle é possível abranger uma área maior de testes, afim de melhorar a qualidade dos testes de caixa branca.

Dentro dos testes de estrutura existem alguns métodos para realizar os testes, são eles:

- Teste de condição: Neste tipo de teste são postos em prova as condições lógicas com intuito de verificar se existe erros de operadores relacionais, expressões aritméticas, operadores booleanos e parênteses.
- Teste de ciclos: São testados nesta parte os ciclos dos programas (*loops*), com intuito de verificar a validade dos laços construídos.

A utilização destes testes, permitiu verificar se aplicação estava sendo executada de forma correta, sem erros de lógicas que poderiam ser causados devido problemas na construção de condições e laços de repetições.

Na Figura 34 é possível observar um trecho de código do software no qual foi aplicado as técnicas acima para validação da lógica inserida. Este trecho tem por intuito buscar exames realizados pelo cliente para gerar relatórios financeiros para o usuário.

Figura 34 - Aplicação de teste de estrutura de controle

```
446         public function selectReportSummaryEmpresaOha($ids, $data_inicial, $data_final){
447             $conn = new Connection();
448             $db = $conn->getDb();
449 
450             // Realiza Consulta no banco
451             $consulta = $db->prepare("SELECT COUNT(EE.price) as qtd, EX.name as exame, EE.price as
452                 EMPLOYEE_EXAM EE INNER JOIN EMPLOYEE E ON EE.employee_id = E.id
453                 INNER JOIN CUSTOMER C ON E.customer_id = C.id
454                 INNER JOIN PROVIDER P ON EE.provider_id = P.id
455                 INNER JOIN EXAM EX ON EE.exam_id = EX.id
456                 WHERE EE.id IN (".$ids.") AND P.status = 2 AND EE.date BETWEEN ? AND ? GROUP BY lo
457 
458 
459             $consulta->execute(array($data_inicial, $data_final));
460             $oha = array();
461 
462             //Grava os resultados do banco em uma variável
463             foreach ($consulta as $value) {
464                 $oha[] = $value;
465             }
466 
467             //Organiza o array com o intuito de facilitar a geração do relatório
468             $nova = array();
469             for ($i=0, $cont=1; $i < count($oha); $i++) {
470                 if ($i == 0){
471                     array_push($nova, array(
472                         "local_exame" =>    $oha[$i]["local_exame"],
473                         "cnpj" =>    $oha[$i]["cnpj"],
474                         "endereco" =>    $oha[$i]["endereco"],
475                         "bairro" =>    $oha[$i]["bairro"],
476                         "cidade" =>    $oha[$i]["cidade"],
477                         "cep" =>    $oha[$i]["cep"],
478                         "estado" =>    $oha[$i]["estado"],
479                         "telefone" =>    $oha[$i]["telefone"],
480                         "email" =>    $oha[$i]["email"],
481                         "exams" => array(array(
482                             "cliente" =>    $oha[$i]["cliente"],
483                             "funcionario" => $oha[$i]["funcionario"],
484                             "exame" =>    $oha[$i]["exame"],
485                             "preco" =>    $oha[$i]["preco"],
486                             "qtd" =>    $oha[$i]["qtd"],
487                             "total" =>    $oha[$i]["total"],
488                             "data" =>    $oha[$i]["data"],
489                         ))
490                     ));
491                     continue;
492                 }
493                 if ($nova[count($nova)-1]['cnpj'] == $oha[$i]['cnpj']){
494                     array_push($nova[count($nova)-1]['exams'], array(
495                         "cliente" =>    $oha[$i]["cliente"],
496                         "funcionario" => $oha[$i]["funcionario"],
497                         "exame" =>    $oha[$i]["exame"],
498                         "preco" =>    $oha[$i]["preco"],
499                         "qtd" =>    $oha[$i]["qtd"],
500                         "total" =>    $oha[$i]["total"],
501                         "data" =>    $oha[$i]["data"],
502                     ));
503                 }
504                 else{
505                     array_push($nova, array(
506                         "local_exame" =>    $oha[$i]["local_exame"],
507                         "cnpj" =>    $oha[$i]["cnpj"],
508                         "endereco" =>    $oha[$i]["endereco"],
509                         "bairro" =>    $oha[$i]["bairro"],
510                         "cidade" =>    $oha[$i]["cidade"],
511                         "cep" =>    $oha[$i]["cep"],
512                         "estado" =>    $oha[$i]["estado"],
513                         "telefone" =>    $oha[$i]["telefone"],
514                         "email" =>    $oha[$i]["email"],
515                         "exams" => array(array(
516                             "cliente" =>    $oha[$i]["cliente"],
517                             "funcionario" => $oha[$i]["funcionario"],
518                             "exame" =>    $oha[$i]["exame"],
519                             "preco" =>    $oha[$i]["preco"],
520                             "qtd" =>    $oha[$i]["qtd"],
521                             "total" =>    $oha[$i]["total"],
522                             "data" =>    $oha[$i]["data"],
523                         ))
524                     ));
525                 }
526             }
527 
528             $conn = null;
529             $db = null;
530 
531             return $nova;
532         }
```

Fonte: Elaborado pelos autores.

11.2 Teste de caixa preta

Segundo Koscianski e Soares (2006, p. 343):

> O teste de caixa-preta é também chamado de teste funcional. O nome "caixa-preta" é usado também na engenharia, no problema de identificação de sistemas, e advém do fato de que ao analisar o comportamento de um objeto, ignora-se totalmente sua construção interna.

Como foi dito acima o teste de caixa-preta não se interessa pela construção interna do software, mas sim, verifica se os padrões de entrada e saída estão de acordo com o que foi requisitado.

Como este teste verifica as entradas e as saídas de dados, trata-se, portanto, de um teste voltado mais para o usuário, pois este não está interessado em como o sistema foi feito ou qual é sua estrutura interna, como diz também Koscianski e Soares (2006, p. 344): "Esse tipo de teste reflete, de certa forma, a ótica do usuário, que está interessado em se servir do programa sem considerar os detalhes de sua construção."

Para este projeto, o teste de caixa preto foi aplicado de maneira a complementar o teste de caixa branca, para Pressamn (2011, p. 432):

"O teste de caixa preta possibilita descobrir uma classe de erros diferente daquela obtida com métodos de caixa branca.".

Abrangendo os principais requisitos funcionais do software, as principais funcionalidades do mesmo, a forma como o mesmo lhe dá com entradas e saídas específicas, e, operações de banco de dados, de modo a identificar problemas que pudessem afetar a experiência do usuário final.

Na Tabela 3 é possível observar a aplicação do teste de caixa preta no projeto:

Tabela 3 - Teste de caixa preta

TESTE DE CAIXA PRETA						
	Operação	Objetivo	Entrada	Procedimento	Descrição	Resultado Esperado
Funcionalidade	Cadastro	Cadastrar Usuário	Dados do usuário	Entrar com os dados do usuário para o cadastro	Cadastrar novo usuário no sistema	Cadastro efetuado com sucesso
		Cadastro de Funcinário	Dados do funcionário	Entrar com os dados essênciais do Funcionário	Cadastrar novo funcionário	Cadastro efetuado com sucesso
		Cadastrar Risco	Informações pertinentes ao risco	Entrar com os dados do risco do Funcionário	Cadastrar a relação risco funcionário com sucesso	Cadastro efetuado com sucesso
		Cadastrar ASO	Dados do exame, empresa e funcionário	Entrar com os dados do tipo de exame, a empresa e o funcionário	Castrar os dados do exame que foi realizado, a empresa que se presatou serviço e o funcionário para a qual se presta serviço para a geração do Atestado de Saúde Ocupacional	Cadastro Efetuado com sucesso
	Atualização	Atualizar Usuário	Dados do usuário	Inserir novos dados para a atualização do Usuário	Inserir e atualizar os dados do usuário do sistema.	Atualização efetuada com sucesso
		Atualizar Funcionário	Dados do funcionário	Inserir novos dados para a atualização do Funcionário	Inserir e atualizar os dados do funcionário da empresa ao qual se presta serviço	Atualização efetuada com sucesso
		Atualizar Risco	Informações do risco	Inserir novas informações para atualização do Risco	Inserir e atualizar os dados da relação funcionário x empresa no sistema.	Atualização efetuada com sucesso
		Atualizar ASO	Informações do exame, empresa e funcionário.	Inserir novas informações para atualização da Atestado de Saúde Ocupacional	Inserir e atualizar os dados referentes a montagem da ASO	Atualização etetuada com sucesso.
	Deleção	Deletar Usuário	Informar identificador	Deletar usuário do sistema	Delção logica do usuário do sistema através do seu identificador	Deleção realizada com sucesso
		Deletar Funcionário	Informar identificador	Deletar funcionário do sistema	Deleção lógica do funcionário e seus dados através do seu identificador.	Deleção realizada com sucesso
		Deletar Risco	Informar identificador	Deletar risco do sistema	Deleção lógica da relação risco x funcionário através do identificador	Deleção realizada com sucesso
		Deletar ASO	Informar identificador	Deletar ASO do sistema	Deleção lógica do ASO do sistema	Deleção realizada com sucesso
	Exibição	Listar usuários	scripts de busca	Listar usuário do sistema	Listar todos os usuário do sistema	Listagem bem sucedida
		Listar funcionários	scripts de busca	Listar funcionários	Listar todos os funcionarios cadastrados no sistema	Listagem bem sucedida
		Listar risco	scripts de busca	Listar riscos	Listar todos os riscos da relação funcionário x empresa cadastrados no sistema	Listagem efetuada com sucesso
	Editar dados	Editar dados da ASO	Informações do Atestado de Saúde Ocupacional	Inserir os dados da ASO	Inserir dados da ASO que em um tenham sido deixado em branco ou tenham sido omitidos	Edição feita com sucesso
Interface	Entrada de dados	Testar Entrada inválida	Caracteres Inválidos	Inserir caracteres inválidos/ incompatíveis	Verificar se ao inserir caracteres incompatíveis com os campos é aceito pelo sistema	Alertar erro
		Testar entrada nula	Vazio	Deixar campos vazios	Verificar se o sistema aceita campos obrigatórios que estejam vazios.	Alertar erro
		Testar entrada incompleta	Caracteres	Deixar campos incompletos	Verificar se o sistema aceita compos incompletos	Alertar erro
Comportamento	Logar no sistema	Testar login	Usuário e senha	Logar no sistema	Verificar se o sistema aceita usuários e senhas inesistentes ou	Impedir acesso
		Testar acesso	Usuário e senha	Logar no sistema	Verificar se o sistema aceita que usuário tenha acesso a conteúdo do administrador e vide versa	Bloquear acesso
	Vazamento de informações	Verificar segurança	Forças erros	Gerar mensagem de erro	Verificar se mensagem de erro não exibem informações confidenciais	Impedir vazamento de informações
		Verificar html	scripts de busca	Verificar código html	Verificar html em busca de dados sensíveis que possam comprometer o usuário	Impedir que dados sensíveis sejam exibidos em html
		Verificar dados sensíveis	Inseção de dados	Verificar URL	verificar se dados sensíveis estão sendo passados explicitamente pela URL	Impedr que dados sejam passados explicitamente pela URL
	Tratamento de Entrada/Saída	injeção de Scripts	Injeção de determinados scripts	Injetar sripts via console	Verificar se a aplicação aceita injeção de scripts como parte das entradas	Impedir a injeção de scripts como parte das entradas
		Injeção de SQL	Injeção de comando SQL	Injeção de sql via console	Verificar se a apliação processa comandos sql injetados por terceiros	Impedir que a aplicação aceita Sql injection

Fonte: Elaborado pelos autores.

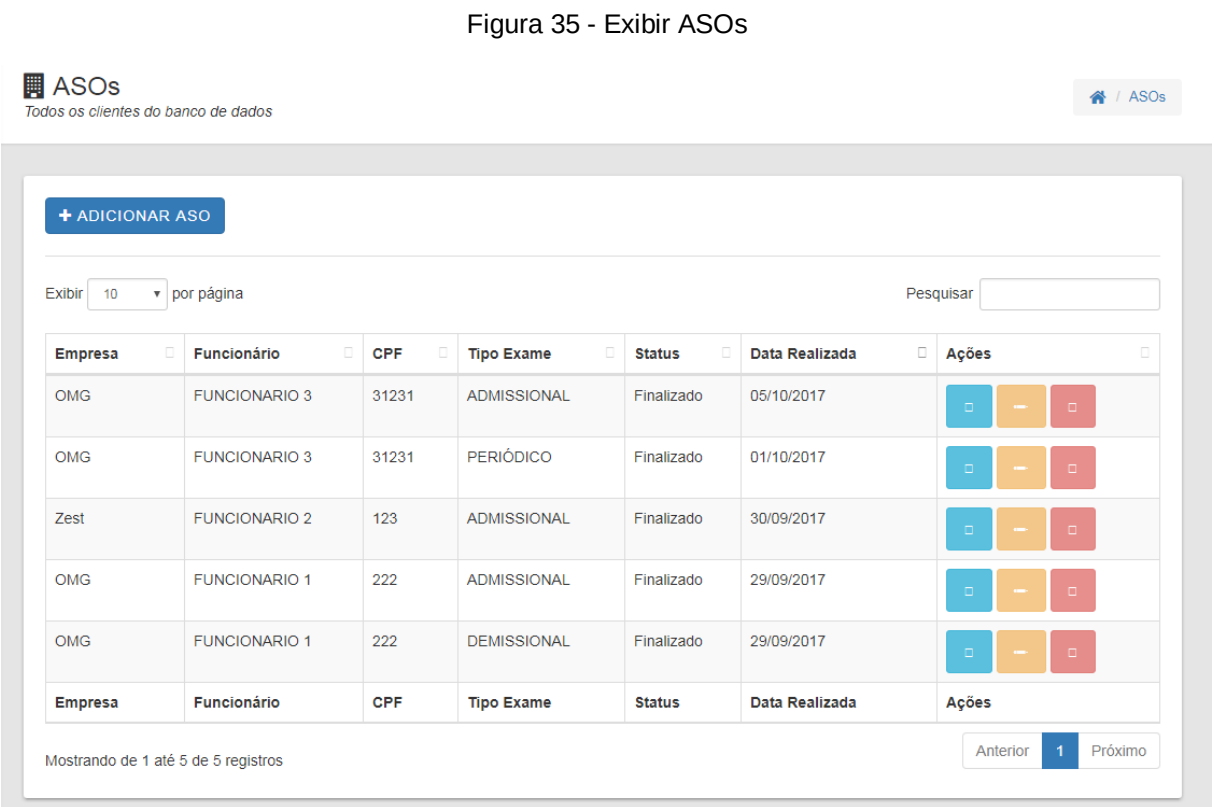
12 DESENVOLVIMENTO

Neste capítulo serão mostradas as principais telas desenvolvidas, além de uma breve descrição dela. Não serão mostradas todas as telas devido a quantidade, porém elas podem ser encontradas no manual do usuário que está no APÊNDICE 1 – QR CODE COM DOCUMENTOS ADICIONAIS**.**

12.1 Apresentação de telas do sistema desenvolvido

Na Figura 35 é possível observar a tela que lista todas as ASOs geradas pelo sistema, também permite que o usuário possa acessar outras telas (adicionar, editar e deletar). Nessa tela também conta com um campo de pesquisa o qual permite o usuário pesquisar dados de uma ASO especifica.

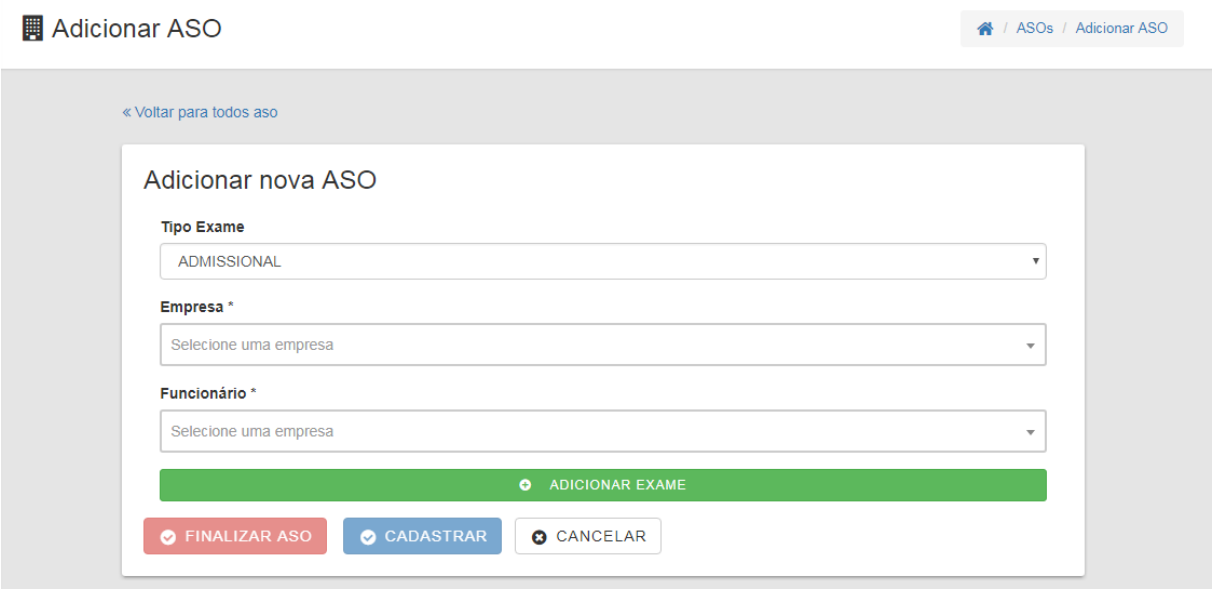
Figura 35 - Exibir ASOs



Fonte: Elaborado pelos autores.

Na Figura 36 é possível observar a tela para gerar uma ASO, no qual é preciso entrar com os dados da empresa e do funcionário e também um botão para adicionar exames que serão realizados. Ao acessar essa tela o usuário tem duas opções, a primeira é salvar a ASO, o que permite o usuário alterar alguns dados que não foram informados quando foi gerado o documento, e a outra opção é finalizar, que encerra a ASO e gera um pdf contendo as informações do estado paciente e não sendo possível alterar os dados.

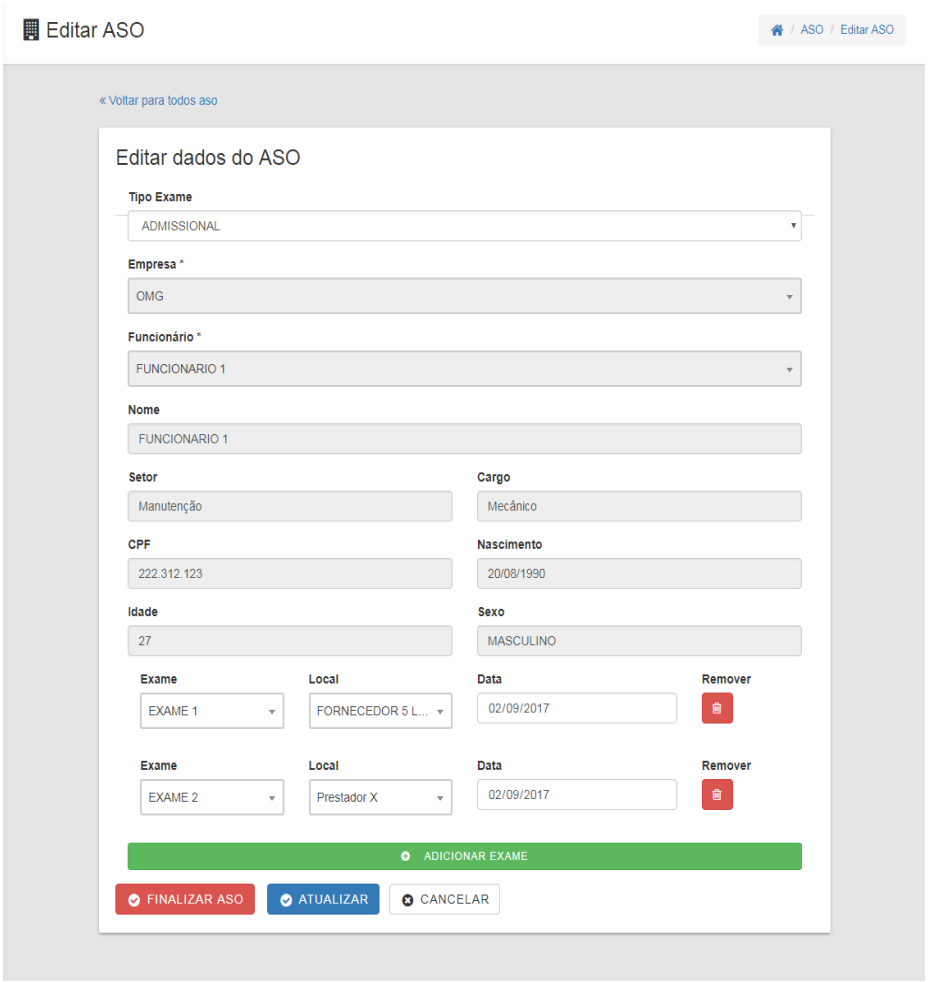
Figura 36 - Adicionar ASO



Fonte: Elaborado pelos autores.

Na Figura 37 é exibida a tela de editar (somente é possível editar ASOs que não foram finalizadas), nela é possível somente editar informações dos exames que serão realizados, não sendo possível editar informações do funcionário.

Figura 37 - Editar ASO



Fonte: Elaborado pelos autores.

Na).

Figura 38 é possível observar uma ASO gerada pelo sistema, nela consta todas as informações necessárias de acordo com a NR-07 (citado no capitulo

13 CRONOGRAMA ÁGIL

Para o presente projeto foi desenvolvido um cronograma ágil para organização das *Sprints*. Na Figura 11, pode-se observar a aplicação da *Sprint* 3 no cronograma, através desta *Sprint* pode-se entender o funcionamento das demais *Sprint* executadas no projeto. Ou seja, as demais seguem o mesmo modelo, as quais podem serem encontradas no APÊNDICE 1 – QR CODE COM DOCUMENTOS ADICIONAIS. (cronograma completo)

Figura 11 - Cronograma *Sprint* 3

✓	**1.6**	**Sprint #3**	**53**	**14 dias**	**Sáb 16/09/17**	**Sáb 30/09/17**	
✓	**1.6.1**	**Metodologias**		**14 dias**	**Sáb 16/09/17**	**Sáb 30/09/17**	
✓	1.6.1.1	Pesquisar Teorica		14 dias	Sáb 16/09/17	Sáb 30/09/17	JV
✓	1.6.2	**Reunião de Orientação**	55IT	0 dias	Seg 18/09/17	Seg 18/09/17	AB;AS;JV;PG;VN
✓	**1.6.3**	**Desenvolvimento**	55II	**14 dias**	**Sáb 16/09/17**	**Sáb 30/09/17**	
✓	**1.6.3.1**	**Codificação**		**14 dias**	**Sáb 16/09/17**	**Sáb 30/09/17**	
✓	1.6.3.1.1	Telas Independentes		14 dias	Sáb 16/09/17	Sáb 30/09/17	AS;PG;VN
✓	**1.6.3.2**	**Banco de Dados**		**14 dias**	**Sáb 16/09/17**	**Sáb 30/09/17**	
✓	1.6.3.2.1	Criar Scripts do BD	60II	14 dias	Sáb 16/09/17	Sáb 30/09/17	VN
✓	1.6.4	Reunião da Equipe	58IT	0 dias	Qua 20/09/17	Qua 20/09/17	AS;JV;PG;VN
✓	**1.6.5**	**Monografia**	55	**3 dias**	**Seg 25/09/17**	**Qua 27/09/17**	
✓	**1.6.5.1**	**Validação (Orientador)**		**3 dias**	**Seg 25/09/17**	**Qua 27/09/17**	
✓	1.6.5.1.1	**Reunião de Orientação**		0 dias	Seg 25/09/17	Seg 25/09/17	AS;JV;PG;VN;AB
✓	1.6.5.1.2	Correções TCC	66	3 dias	Seg 25/09/17	Qua 27/09/17	VN;PG;AS;JV
✓	1.6.5.1.3	Reunião da Equipe	67IT	0 dias	Qua 27/09/17	Qua 27/09/17	AS;JV;PG;VN
✓	1.7	Término Sprint #3	54;64;6	0 dias	Qui 05/10/17	Qui 05/10/17	

Fonte: Elaborado pelos autores

Dessa forma, o cronograma representou o *Backlog* do Produto do projeto, ou seja, contém todos os itens relacionados as funcionalidades do sistema e as atividades do presente trabalho (relatório de desenvolvimento). Além disso, também são apresentadas as responsabilidades de cada integrante, cerimônias e a representação de cada artefato do *Scrum*.

Sendo que na Figura 12 pode-se observar a aplicação da *Sprint* 3, na qual apresenta seus respectivos itens (*backlog* da *Sprint*). Em auxilio a execução desses

itens do *Backlog* da *Sprint* foi utilizado a ferramenta Trello (Figura 12), que através do layout customizado permitiu maior organização da equipe de desenvolvimento.

Figura 12 - *Sprint* 3

Fonte: Elaborado pelos autores

Através disso, pode-se compreender que cada item do *Backlog* da *Sprint* é movido pelos respectivos membros responsáveis de cada tarefa do Time de Desenvolvimento, onde estes itens são disponibilizados para a equipe através da coluna “A fazer”. Posteriormente, movido para a execução da tarefa na coluna “Fazendo/Realizando”, quando finalizados são enviados para a revisão, sendo colocados na coluna “Em Revisão”.

Por fim, quando todas as correções forem realizadas passarão para a coluna de “Feito”. Dessa forma, quando todos os itens do *Backlog* da *Sprint* forem considerados como “Feito”, poderão ser apresentados e analisados nas cerimonias de revisão da Sprint.

EMPRESA).

Figura 38 - Modelo ASO gerado pelo sistema

EMPRESA:	GO TRANSPORTE		
NOME:	HELENA DA SILVA		
SETOR:	MANUNTENçãO		
FUNÇÃO:	FUNILEIRA		
RG:	12.938.21	**CPF:**	213.213.213
DTA NASC:	12/08/1989	**IDADE:**	28
SEXO:	FEMININO		

Em cumprimento a Lei Estadual 610/50 e/ou 6514/77 artigo 168 da CLT e portaria 3214/78 e 3164/82 da NR-7 do Ministério do Trabalho e Emprego, o funcionário fez exame:

TIPO EXAME:	**ADMISSIONAL**

Estando exposto ao(s) risco(s) ocupacional(is)

TIPO DE RISCO	**DESCRIÇÃO DOS RISCOS**
Fisico	Umidade
Fisico	Ruido / Ambiente de Trabalho

Realizou os seguintes exames complementares:

EXAME	**DATA**	**EXAME**	**DATA**
AVALIAçãO CLINICA	12/01/2017	ACUIDADE VISUAL	12/01/2017
ESPIROMETRIA	13/01/2017	RAIO X	13/01/2017

CONCLUSÃO

( ) Apto ( ) Inapto ( ) Apto c/ Restrições ( ) Inapto Temporário

OBS.:	
Local e Data	AMERICANA,

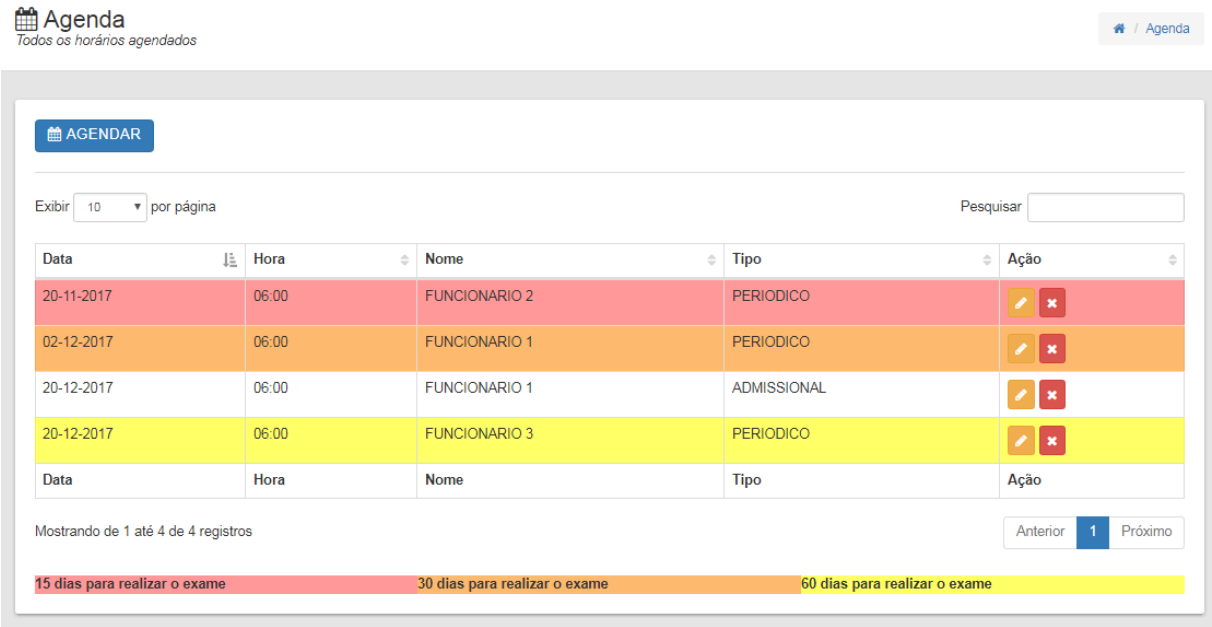
Dra. XXXXXXXXX **CRM 0000000** **Médica do Trabalho** **Coordenadora do PCMSO**	_____ Médico(a) Examinador(a) Contato: (00) 0000.0000 / 0000.0000

_____
Declaro que recebi a 2ª via deste ASO - Atestado de Saúde Ocupacional

Fonte: Elaborado pelos autores.

A tela de agenda é exibida na Figura 39**,** ela contém informações dos próximos exames do funcionário, além de destacar com cores diferentes os exames periódicos que a empresa necessita fazer. Esta tela permite o usuário agendar, reagendar e deletar as consultas.

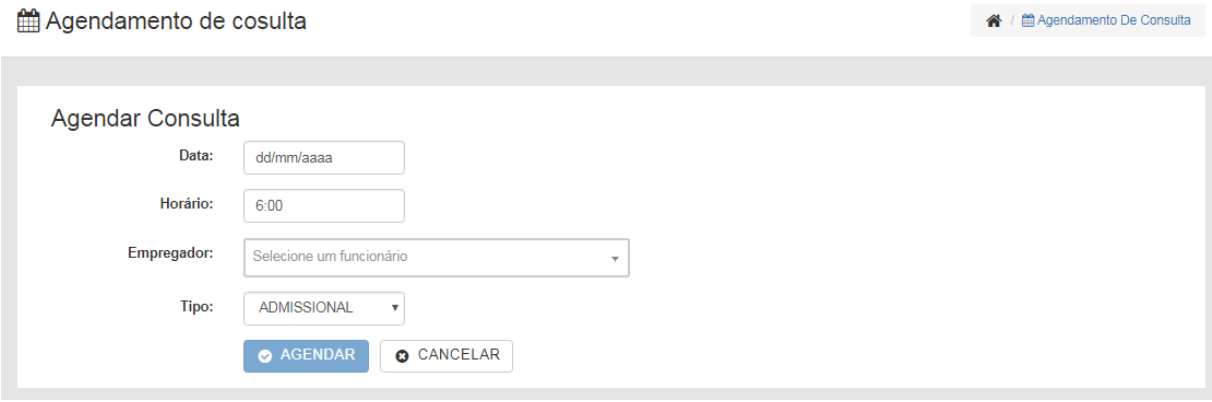
Figura 39 - Agenda



Fonte: Elaborado pelos autores.

A tela de agendar consulta é exibida na Figura 40**,** nela é possível cadastrar os funcionários que irão realizar consultas, sendo obrigatório preencher todos os dados para realizar o agendamento.

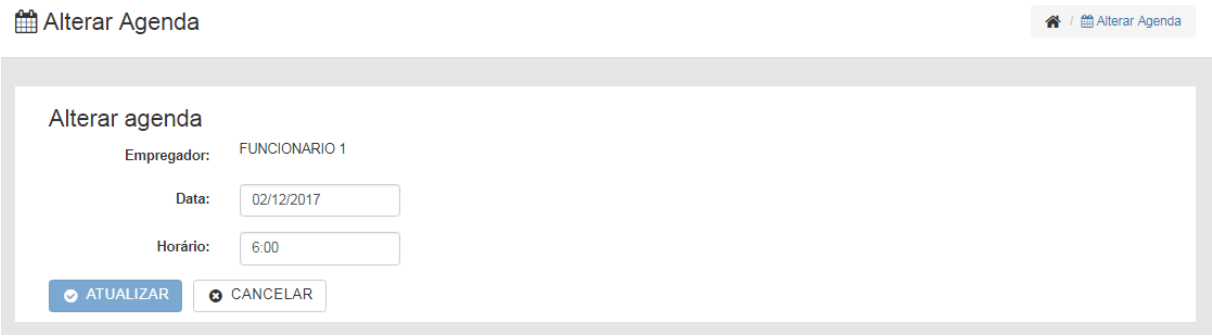
Figura 40 - Agendar consulta



Fonte: Elaborado pelos autores.

A tela de editar consulta é exibida na Figura 41**,** nela é possível alterar a data e horário da consulta dos funcionários, sendo obrigatório preencher todos os dados para realizar o agendamento.

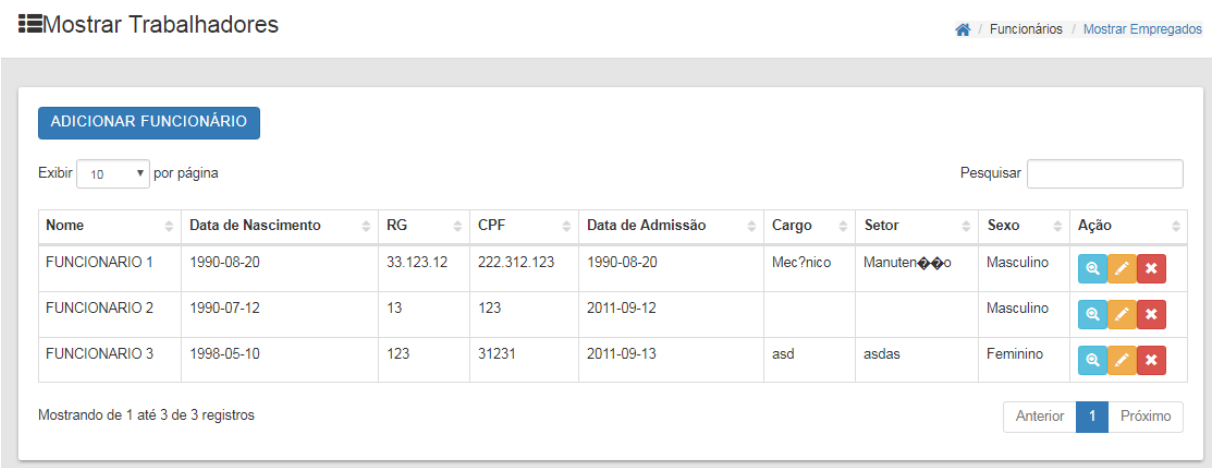
Figura 41 - Alterar agenda



Fonte: Elaborado pelos autores.

Na Figura 42 é exibido a tela de trabalhadores, que contém uma lista com todos os trabalhadores cadastrados no sistema, sendo possível acessar as opções de adicionar, editar e deletar. Assim como as outras telas, é possível pesquisar funcionários pelo campo de busca.

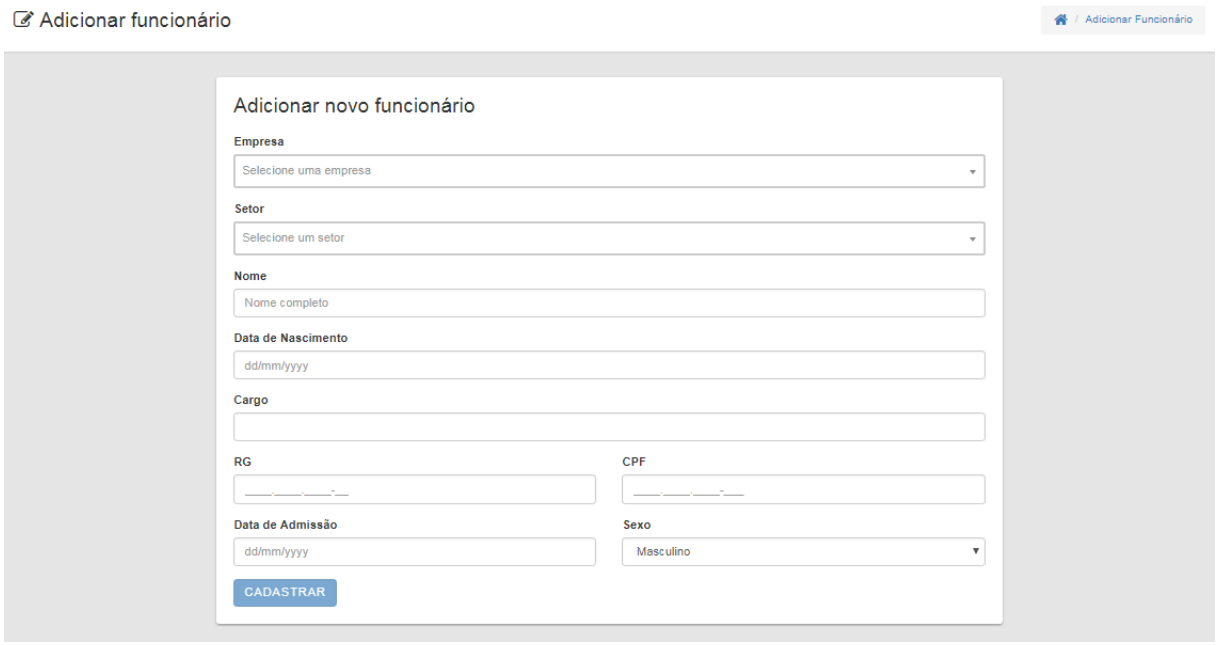
Figura 42 - Trabalhadores



Fonte: Elaborado pelos autores.

Na Figura 43 é exibido a tela de adicionar funcionário, nela é possível cadastrar todas as informações do trabalhador, que serão utilizadas quando o usuário for gerar a ASO, assim, fazendo a busca pelos dados cadastrados no sistema.

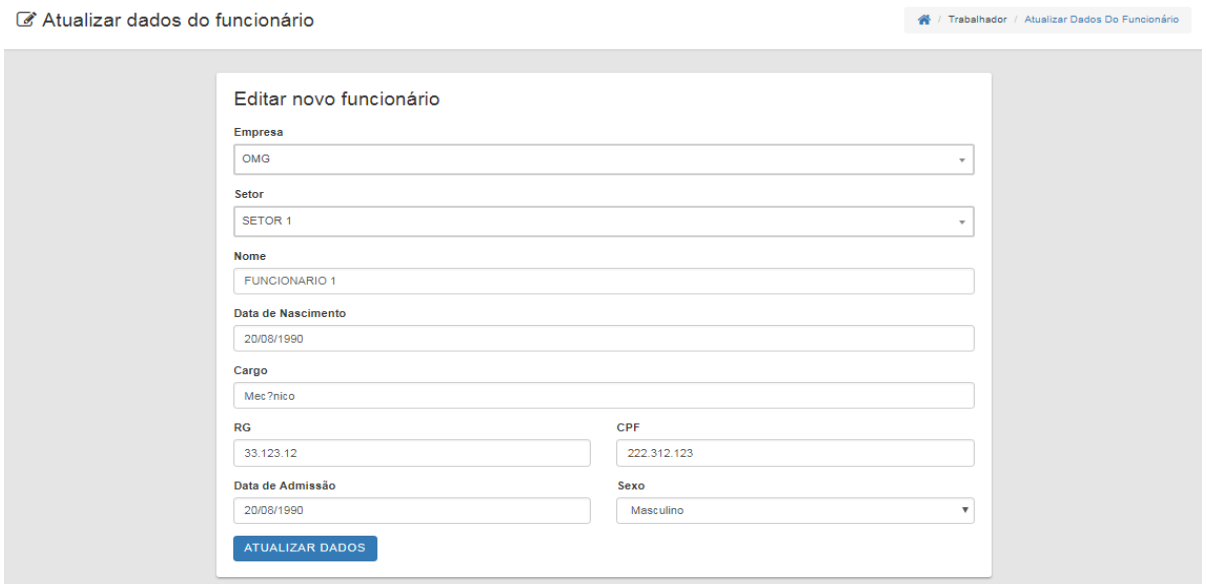
Figura 43 - Adicionar funcionário



Fonte: Elaborado pelos autores.

Na Figura 44 é exibido a tela de editar funcionários, nela é possível atualizar dados pessoais dos trabalhadores, isso pode acontecer quando existe uma mudança de cargo do funcionário.

Figura 44 - Editar funcionário



Fonte: Elaborado pelos autores.

14 RESULTADOS

O objetivo do presente projeto foi desenvolver uma aplicação (software) capaz de otimizar e automatizar algumas tarefas diárias de empresas do ramo de medicina ocupacional, processos estes de documentação que eram realizados de forma manual através de papéis, documentos e planilhas eletrônicas.

Dessa forma, através dos benefícios da automatização dos processos internos, a empresa apresentaria uma documentação organizada, gerenciada e disponível, que amplificariam a produtividade dos processos da organização na tomada de decisões.

Para isto, foram aplicados os conhecimentos adquiridos ao longo do curso, tais como: programação, banco de dados, engenharia de software e gestão de projetos. Para que fosse possível desenvolver um sistema que atendesse as necessidades do cliente do projeto.

Abaixo pode ser visualizada uma lista que contém as principais solicitações do cliente que são importantes serem implementadas nas versões futuras do projeto:

- Manter agenda: A atividade de agenda foi aprovada pelo cliente, no entanto, sugeriu alguma maneira de avisar os clientes a necessidade de uma nova consulta dos funcionários, podendo ser emitido um relatório por empresa informando as pessoas (funcionários) que necessitam realizar o exame periódico, ou também uma maneira de enviar e-mails para os clientes e /ou para a empresa de forma automática;
- Manter Clientes: Para esta funcionalidade foi proposto que a tela de “relacionar os riscos com os setores” fosse adicionada a descrição dos riscos, uma vez que podem existir riscos com nomes iguais, porém com descrições diferentes (exemplo: Risco sobre o ambiente de trabalho e ruídos são classificados como físico).
- Manter prestadores: Essa funcionalidade foi aprovada, assim, não sendo necessária modificações, apenas sendo necessária a correção de pequenos *bugs* que podem ser identificados durante o uso do software.
- Gerar ASO: Essa funcionalidade foi aprovada, no entanto, foi sugerido algumas novas características, como, permitir a customização da ASO de acordo com os padrões do cliente.

- Relatórios financeiros: Essa funcionalidade foi aprovada pelo cliente, sendo necessário somente a correção de *bugs* que podem ser encontradas no decorrer do uso.

Além das funcionalidades solicitadas durante a reunião com o cliente, também foram sugeridas algumas dicas em relação a comercialização do produto. Pois, segundo o cliente a inciativa do projeto desenvolvido (acadêmico) apresenta potencial que atente as necessidades de empresas do setor.

No entanto, foi ressaltado que o sistema ainda necessita de algumas melhorias significativas para poder iniciar no mercado. Dessa forma, foi sugerida a disponibilização do software à empresa de medicina ocupacional de forma gratuita, como o intuito de analisar os *feedbacks* dos clientes e, assim, poder modelar o software para atender as diferentes necessidades de negócio.

Sendo assim, o desenvolvimento de um produto que seja flexível e, atenda às necessidades dos clientes, faz do software cada vez mais completo. Podendo assim, até mesmo ser inserido no mercado para competir com sistemas semelhantes, que não são flexíveis, pelo fato de que os clientes acabam tendo que pagar por módulos que não cabem a suas regras de negócio.

15 CONSIDERAÇÕES FINAIS

Os resultados alcançados com o desenvolvimento do software para medicina ocupacional foram considerados satisfatórios, mesmo com alguns imprevistos o produto desenvolvido pela equipe alcançou o objetivo, atendendo assim as principais funcionalidades do sistema, sendo necessário apenas a realização de algumas melhorias solicitadas pelos clientes do projeto.

Alguns empecilhos encontrados foram ocasionados por diversos fatores, tais como, dificuldades em relação ao conhecimento técnico (tecnologia da informação e medicina ocupacional), alocação de tempo para o trabalho e projeto pessoais, inexperiência em projetos desse porte (realidade com clientes reais) e na alocação de tarefas de outras disciplinas (projetos paralelos) que exigiram dos estudantes alocação de tempo para desenvolve-los.

Outro motivo que causou dificuldade para a equipe, foi o modelo de documentação disponibilizado pela faculdade, que, apresentava algumas necessidades. Por tratar-se de um modelo recentemente proposto pela faculdade, houve grande dificuldade por parte da equipe em relação ao que deveria ser exposto no relatório final ou não.

O conjunto das dificuldades no projeto contribuíram para a ocorrência de pequenos atrasos, os quais, necessitaram serem analisados e priorizados pelos integrantes da equipe para que a meta não fosse prejudicada.

Caso o projeto tivesse início novamente a equipe buscaria estudar mais sobre a metodologia *Scrum*, uma vez que os membros tinham conhecimento parcial para sua implementação no desenvolvimento. Outro ponto importante seria tentar aprender mais sobre as plataformas de desenvolvimento *web* (PHP, HTML, CSS e JS), e por fim, aprender mais sobre boas práticas em desenvolvimento de software, com objetivo de garantir maior padronização.

Para melhorias futuras do sistema, seria interessante inserir algumas funcionalidades solicitadas pelo cliente (citadas em resultados) e desenvolver um módulo para integrar a área de segurança de trabalho junto com a de medicina ocupacional, dessa forma, automatizando ainda mais os processos da empresa, uma vez que esses setores estão diretamente relacionados.

Em resumo, as lições aprendidas com este projeto ofereceram a todos os envolvidos uma grande experiência acadêmica e pessoal, pois permitiu vivenciar atividades relacionadas à Tecnologia da Informação em auxilio a outras áreas do conhecimento. Essas lições ficarão marcadas na vida dos integrantes do projeto, tanto profissional quanto pessoal.

REFERÊNCIAS

BARROS, Isabelle Guimarães M. O. de; SANTOS Carlos Felipe Araujo dos. **Apostila de Introdução ao CSS.** 2008. Disponível em: <https://www.telecom.uff.br/pet/petws/downloads/tutoriais/css/css2k80912.pdf>. Acesso em: 25 nov. 2017.

BITBUCKET. BitBucket, 2017. Disponível em:<https://bitbucket.org/product>. Acesso em: 17 de Set. 2017.

BOOCH, Grady; RUMBAUGH, James; JACOBSON, Ivar. **UML:** guia do usuário. 2ª. ed. Rio de Janeiro: Elsevier (Campus), 2007.

BOOTSTRAP. **Navegadores e dispositivos**. Disponível em: <http://getbootstrap.com.br/v4/getting-started/browsers-devices/>. Acesso em: 15 set. 2017.

BRASIL. Ministério do Trabalho. NR 7 - Programa de controle médico de saúde ocupacional (PCMSO), 1978, 16p. **Portaria GM n.º 3.214, de 8 de junho de 1978**. Disponível em: <http://trabalho.gov.br/images/Documentos/SST/NR/NR7.pdf>. Acesso em: 14 set. 2017, às 12h26min.

CONCEIÇÃO, Leonardo Ferreira da. **Scrum, a metodologia ágil**. Disponível em: <https://uvagpclass.wordpress.com/2017/05/24/scrum-a-metodologia-agil/>. Acesso em: 19 nov. 2017.

CRUZ, Fábio Rodrigues. **Artefatos Scrum**. Disponível em: <http://www.fabiocruz.com.br/frameworkscrumold/artefatos-scrum/>. Acesso em: 19 nov. 2017.

CRUZ, Fábio Rodrigues. **Scrum e Agile em projetos**: guia completo. Rio de Janeiro: Brasport, 2015.

DALL'OGLIO, Pablo. **PHP:** programando com orientação à objeto. 2ª.ed. São Paulo: Novatec, 2015.

DELAMARO, Márcio Eduardo; MALDONADO José Carlos; JINO Mario. **Introdução ao teste de software.** Rio de Janeiro: Elsevier, 2007.

DUCKETT, Jon. **Introdução à Programação Web com HTML, XHTML e CSS.** 2ª ed. Rio de Janeiro: Editora Ciência Moderna, 2010. p. 255-257.

DUTRA, Alexssandrolima. **Vida Interativa e Incremental de Desenvolvimento de Software!**, 2012. Disponível em : <http://alexpagernet.blogspot.com.br/2012/08/vida-interativa-e-incremental-de.html>. Acesso em: 25 nov. 2017.

EIS, Diego. **O Básico:** o que é HTML?. Disponível em:<https://tableless.com.br/o-que-html-basico/>. Acesso em: 25 nov. 2017.

ELMASRI, Rames; NAVATHE, Shamkant B.. **Sistemas de banco de dados**. 6ª.ed. São Paulo: Pearson, 2011. p. 23, 132.

FERREIRA, Elcio; EIS, Diego. **HTML5**: curso w3c escritório Brasil. Disponível em: <http://www.w3c.br/pub/Cursos/CursoHTML5/html5-web.pdf>. Acesso em: 25 nov. 2017.

FOWLER, Martin. **UML Essencial:** um breve guia para a linguagem-padrão de modelagem de objetos. 3ª ed. São Paulo: Bookman, 2004.

GIT. **Git**: everything is local, 201-?. Disponível em:<https://git-scm.com/>. Acesso em: 16 set. 2017.

GRILLO, Filipe Del Nero; FORTES, Renata Pontin de Mattos. **Aprendendo JavaScript.** 2008. Disponível em: <http://conteudo.icmc.usp.br/CMS/Arquivos/arquivos_enviados/BIBLIOTECA_113_ND_72.pdf>. Acesso em: 15 jun. 2017.

GUEDES, Gilleanes T. A.. **UML 2:** uma abordagem prática. São Paulo: Novatec Editora, 2009. p. 19-21.

GUIMARÃES, Célio. **Introdução a Linguagens de Marcação**: HTML, XHTML, SGML, XML. 2005. Disponível em:<http://www.ic.unicamp.br/~celio/inf533/docs/markup.html>. Acesso em: 25 nov. 2017.

HEUSER, Carlos Alberto. **Projeto de banco de dados**. Porto Alegre: bookman, vol. 4, 2009. p. 25-29.

KOSCIANSKI, André; SOARES, Michel dos Santos. **Qualidade de software.** São Paulo: Novatec, 2007.

MACHADO, Felipe Nery Rodrigues. **Projeto e implementação de banco de dados**. 2ª ed. São Paulo : Érica, 2008.

MANIFESTO AGIL, Manifesto para o desenvolvimento ágil de software. Disponível em:<https://www.manifestoagil.com.br>. Acesso em: 16 dez. 2017.

MAGELA, Rogério. **Engenharia de Software Aplicada**: Princípios. Rio de Janeiro: Alta Books, 2006.

MICROSOFT. **Project**. Disponível em:<https://products.office.com/pt-br/project/project-management>. Acesso em: 17 set. 2017

MICROSOFT. **Planos e preços**. Disponível em: <https://products.office.com/pt-br/project/compare-microsoft-project-management-software?tab=tabs-1>. Acesso em: 04 set. 2017.

MUSSEL, Marcelo Corrêa. **Aplicações para WEB.** 2017. Disponível em:< http://aplicacoesweb.esy.es/gallery/apostilabootstrap.pdf >. Acesso em: 15 Jun. 2017

SILVA, Maurício Samy. Bootstrap 3.5.5. São Paulo: Novatec Editora, 2015. p. 21.

MySQL. MySQL WorkBench, 2017. Disponível em:<https://www.mysql.com/products/workbench/>. Acesso em: 16 de Set. 2017.

NETBEANS. **NetBeans IDE**. Disponível em: <https://netbeans.org/features/index.html>. Acesso em: 17 de Set. 2017.

OLIVEIRA, Ebenezer Silva. **Uso de Metodologias Ágeis no Desenvolvimento de Software**. Belo Horizonte, 2003. Disponível em: <http://www.infobitsolucoes.com/antigos/trabalhos/monografias/Monografia-EbenezerSilvaOliveira.pdf>. Acesso em: 25 out. 2017.

PICANÇO JR, Péricles Luiz; DELAZARI, Luciene Stamato. **Avaliação da usabilidade de interfaces de sistemas vgi na tarefa de inserção de feições**. Disponível em: <http://www.scielo.br/scielo.php?script=sci_arttext&pid=S1982-21702016000300492#B2>. Acesso em: 9 set. 2017.

PRESSMAN, Roger S. **Engenharia de Software**. Tradução: José Carlos Maldonado. São Paulo: Pearson Makron Books,1995.

PRESSMAN, R. S.; MAXIM, B. R. **Engenharia de software**: uma abordagem profissional. Tradutores: Arivaldo Griesi. 7. ed. Porto Alegre: AMGH, 2011.

PRESSMAN, R. S.; MAXIM, B. R. **Engenharia de software**: uma abordagem profissional. Tradutores: Arivaldo Griesi, João Eduardo Nóbrega Tortello, Mario Moro Fechhio. 8. ed. Porto Alegre: AMGH, 2016.

RIOS, Emerson; MOREIRA, Trayahú. **Teste de software.** Rio de Janeiro: Alta Books, 2006.

ROSIN, Roberto. **Sistema para composição e correção de listas de questões de múltipla escolha**. 2011. Trabalho de Conclusão de Curso. Universidade Tecnológica Federal do Paraná.

SABBAGH, Rafael. Scrum: **Gestão ágil para projetos de sucesso**. São Paulo: Editora Casa do Código, 2014.

SANTOS, Marcel dos. **Aprendendo HTML** : a linguagem da Web. 2014. Disponível em : <http://pensandonaweb.com.br/aprendendo-html-a-linguagem-da-web/>. Acesso em: 25 nov. 2017.

SCHWABER, K. e SUTHERLAND, J. 2013 Guia do Scrum. 2013. Disponível em: <http://goo.gl/ah2AaM>. Acesso em 15 out 2017.

SILVA, Ricardo Pereira. **UML 2:** em modelagem orientada a objetos. Florianópolis: Visual Books, 2007

SOARES, Bruno Augusto Lobo. **Aprendendo a linguagem PHP.** Rio de Janeiro: Ciência Moderna, 2007.

SUBLIME TEXT. Download, 2017. Disponível em: <https://www.sublimetext.com/3>. Acesso em: 22 nov. 2017.

SOMMERVILLE, Ian. **Engenharia de Software**. Tradutores: Selma Shin Shimizu Melnikoff, Reginaldo Arakari, Edilson de Andrade Barbosa. São Paulo: Perrson Addison Wesley. 8ª edição, 2007.

SUBLIME. **Buy**. Disponível em: <https://www.sublimetext.com/buy?v=3>. Acesso em: 04 set. 2017.

TOSING, Sérgio Luiz. **PHP com Ajax na web 2.0:** com muitos exemplos práticos. Rio de Janeiro: Ciência Moderna, 2008.

TOSING, Sérgio Luiz. **MYSQL Aprendendo na Prática**. Rio de Janeiro: Ciência Moderna, 2006. 304p.

TRELLO. **Tour**. Disponível em: <https://trello.com/tour>. Acesso em: 17 set. 2017.

VARGAS, Thânia Clair de Souza. **A história de UML e seus diagramas.** Disponível em: <https://projetos.inf.ufsc.br/arquivos_projetos/projeto_721/artigo.tcc.pdf>. Acesso em: 05 out 2017.

VERSIONONE. **11th Annual State of Agile Report**. Disponível em: <https://explore.versionone.com/state-of-agile/versionone-11th-annual-state-of-agile-report-2>. Acesso em: 20 out. 2017.

VIEIRA, Denisson. **Scrum**: A Metodologia Ágil Explicada de forma Definitiva. Disponível em: <http://www.mindmaster.com.br/scrum/>. Acesso em: 19 nov. 2017.

XAMPP. **O que é o XAMPP?** Disponível em: <https://www.apachefriends.org/pt_br/index.html>. Acesso em: 16 set. 2017.

ANEXO 1 - HEURÍSTICAS

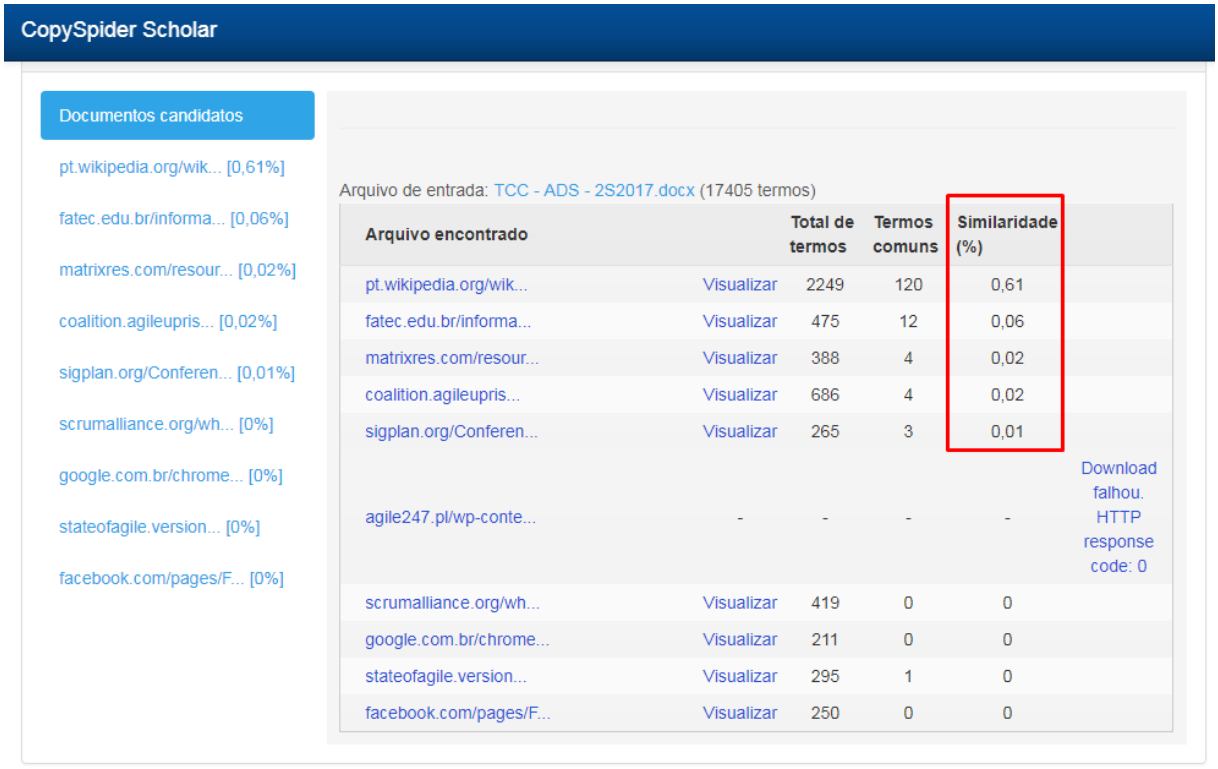
Figura 45 - Heurísticas de Nielsen

Visibilidade do estado do sistema
O sistema deve sempre manter os usuários informados sobre o que acontece no sistema, através de avisos apropriados e em tempo hábil.
Mapeamento entre o sistema e o mundo real
O sistema deve usar a linguagem dos usuários, com palavras, frases e conceitos familiares, em vez de termos técnicos. Seguindo convenções do mundo real, fazendo as informações aparecerem em uma ordem natural e lógica.
Liberdade e controle do usuário
Os usuários frequentemente escolhem as funções do sistema acidentalmente e deve haver uma indicação clara de saída da função indesejada sem ter que migrar de tela. Suporte com as funções de *undo* e *redo*.
Consistência e Padrões
Os usuários não devem ter que supor se palavras diferentes, situações ou ações significam a mesma coisa. Seguir convenções da plataforma.
Prevenção de erros
Melhor do que boas mensagens de erro é um projeto cuidadoso que impeça um problema ocorrer em primeiro lugar. Eliminar as condições passíveis de erros ou verificá-los e apresentar aos usuários uma opção de confirmação antes de se comprometer com a ação.
Reconhecer em vez de relembrar
Minimizar a carga de memória do usuário, fazendo objetos, ações e opções visíveis. O usuário não deve ter que lembrar informações de uma parte do diálogo para outra. Instruções de utilização do sistema devem estar visíveis ou facilmente recuperáveis sempre que necessário.
Flexibilidade e eficiência de uso
Aceleradores podem frequentemente melhorar a interação para todos os usuários do sistema, mesmo que sejam invisíveis aos usuários iniciantes.
Design estético e minimalista
Os diálogos não devem conter informações irrelevantes ou raramente necessárias. Diminui a visibilidade relativa dos elementos da interface.
Suporte para o usuário reconhecer, diagnosticar e recuperar erros
Mensagens de erro devem ser expressas em linguagem simples (sem códigos), indicar com precisão o problema e construtivamente sugerir uma solução.
Ajuda e documentação
Qualquer informação deve ser fácil de pesquisar, focada na tarefa do usuário, listar passos concretos a serem realizados, e não ser muito grande.

Fonte: (PICANÇO; DELAZARI, 2017).

ANEXO 2 – FERRAMENTA ANTI-PLÁGIO

Figura 46 - *CopySpider Scholar*



Fonte: Gerado pelo programa *Copyspider*

APÊNDICE 1 – QR CODE COM DOCUMENTOS ADICIONAIS

Figura 47 - Código QR

Fonte: Elaborado pelos autores

Neste código QR é possível acessar a pasta do Google *Drive* com toda a documentação do trabalho de conclusão de curso, além de documentos extras (diagramas UML, código fonte, script do baco e ATAs).