UNIVERSIDADE DE SÃO PAULO
FACULDADE DE MEDICINA DE RIBEIRÃO PRETO

Carlos de Brito

***Controle eletrônico das etapas do processo de
planejamento de pacientes em Radioterapia***

Ribeirão Preto
Julho / 2017

Carlos de Brito

***Controle eletrônico das etapas do processo de
planejamento de pacientes em Radioterapia***

Dissertação apresentada à Universidade de São Paulo, para a obtenção de Título de Mestre no Programa Mestrado Profissional em Ciências das Imagens e Física Médica.

Área de Física Médica, Opção: Radioterapia e Física Médica.

Orientador: Prof. Dr. Harley F. Oliveira

Ribeirão Preto

Julho / 2017

AUTORIZO A REPRODUÇÃO E DIVULGAÇÃO TOTAL OU PARCIAL DESTES TRABALHOS, POR QUALQUER MEIO CONVENCIONAL OU ELETRÔNICO, PARA FINS DE ESTUDO E PESQUISA, DESDE QUE CITADA A FONTE.

FICHA CATALOGRÁFICA

Brito, Carlos de.

Controle eletrônico das etapas do processo de planejamento de pacientes em Radioterapia / Carlos de Brito ; orientador, Harley Francisco de Oliveira. – 2017.

47 f. : 24 il.

Dissertação (Mestrado) – Faculdade de Medicina de Ribeirão Preto da Universidade de São Paulo. Departamento de Clínica Médica. Área de Concentração: Investigação Biomédica (Programa Mestrado Profissional em Ciências das Imagens e Física Médica – Área de Física Médica, Opção: Radioterapia e Física Médica).

Versão original.

1) Java; 2) Framework; 3) IDE; 4) Eclipse; 5) persistência; 6) dados; 7) Mapeamento; 8) Relacional; 9) Objeto; 10) MOR; 11) SQL; 12) arquitetura; 13) desenvolvimento; 14) relacionamento; 15) interface; 16) MVC; 17) Workbench; 18) API; 19) JPA; 20) HIBERNATE; 21) entidade; 22) relacionamento; 23) MySQL; 24) planejamento; 25) radioterapia; 26) requisitos

FOLHA DE APROVAÇÃO

Brito, Carlos de

Controle eletrônico das etapas do processo de planejamento de pacientes em Radioterapia.

Dissertação apresentada à Universidade de São Paulo,
para a obtenção de Título de Mestre no Programa
Mestrado Profissional em Ciências das Imagens e Física
Médica – Área de Física Médica, Opção: Radioterapia e
Física Médica.

Aprovado em: ____/____/____

Banca examinadora:

Prof. Dr. _____

Instituição: _____ Assinatura: _____

Prof. Dr. _____

Instituição: _____ Assinatura: _____

Prof. Dr. _____

Instituição: _____ Assinatura: _____

Dedicatória

*À minha filha Ana Carolina B. de Brito, à
minha maravilhosa esposa, Joice Batista de
Brito, que sempre me incentivou para a
realização dos meus ideais, encorajando-me a
enfrentar todos os momentos difíceis da vida.*

*Com muito carinho, a minha mãe Maria de
Lourdes V. de Brito, pela compreensão apoio
e contribuição para minha formação
acadêmica*

Agradecimentos

À Deus pela dádiva da vida, e por me sustentar nos momentos mais difíceis.

Aos meus pais, em especial meu pai que hoje se encontra com Deus, mas sempre me incentivou na continuação do curso, sendo junto com minha mãe verdadeiros amigos, companheiros e confidentes, que hoje sorriem orgulhosos ou choram emocionados, que se doaram inteiros e muitas vezes renunciaram aos seus sonhos para que eu pudesse realizar um sonho. Minha eterna gratidão vai além dos meus sentimentos, pois a vocês foi cumprido o dom Divino, o dom de ser Pai, o dom de ser Mãe.

RESUMO

Brito, Carlos de. **Controle eletrônico das etapas do processo de planejamento de pacientes em Radioterapia**. 2017. 47f. Dissertação (Mestrado Profissional em Ciências das Imagens e Física Médica); orientador, Harley Francisco de Oliveira. – 2017

Em diversos serviços de radioterapia brasileiros há a premente necessidade de sistematização de processos que ainda hoje, mesmo com acesso à tecnologia da informação, é realizada manualmente. Este fato acontece com o controle dos pacientes em planejamento radioterápico onde os dados de todas as etapas do processo são inseridos em um quadro físico conforme executadas. Esses dados consistem desde o agendamento da tomografia de planejamento até a liberação do mesmo para o tratamento.

O objetivo deste estudo é o desenvolvimento de um *software* para controle eletrônico de pacientes em processo de planejamento radioterápico, substituindo o quadro físico utilizado atualmente. Foi utilizado para este estudo a técnica de elicitação de requisitos, que resulta em um documento de requisitos que serve de base para a modelagem do sistema através de diagramas *UML* e modelagem conceitual do banco de dados *MySQL* e posteriormente a utilização da técnica de mapeamento objeto relacional (MOR). A consistência nos dados permite acesso seguro e confiável à esses dados, facilita a obtenção de dados estatísticos e agiliza o processo de planejamento, já que de posse de estatísticas consistentes é possível a identificação de gargalos no processo e sua resolução com maior rapidez.

O resultado deste estudo é um software e banco de dados que integra todas as informações do processo de planejamento radioterápico.

Palavras-chave: *Java, Framework, IDE, Eclipse, persistência, dados, Mapeamento, Relacional, Objeto, MOR, SQL, arquitetura, desenvolvimento, relacionamento, interface, MVC, Workbench, API, JPA, HIBERNATE, entidade, relacionamento, MySQL, planejamento, radioterapia, requisitos.*

ABSTRACT

Brito, Carlos de. **Electronic control of the steps of the planning process of patients in Radiotherapy**.2017. 47f. Dissertation (Professional Master's Degree in Image Sciences and MedicalPhysics); Advisor, Harley Francisco de Oliveira. 2017

In Brazilian radiotherapy departments there is a pressing need for systematization of processes, that even today with the access of information technology is performed manually. The control of the patients in radiotherapy planning with the data of all the steps of the process are inserted in a physical board as they are executed.

These data consist of scheduling since the planning tomography until the release of the patient for the treatment.

The objective of this study is the development of a software for electronic control of patients in radiotherapy planning process, replacing the physical board currently used.

The requirements for the elicitation technique was used for this study, which results in a requirements document that serves as the basis for modeling the system through UML diagrams and conceptual modeling of the MySQL database and after using the relational object mapping technique (MOR). The data consistency allows secure and reliable access to these data, facilitates the collection of statistical data and streamlines the planning process, this consistent statistics allow the identification of limitation in the process and faster resolution.

The result of this study is a software and database that integrates all the information of the radiotherapy planning process.

KeyWords: *Java, Framework, IDE, Eclipse, persistence, data, Mapping, Relational, Object, MOR, SQL, architecture, development, relationship, interface, MVC, Workbench, API, JPA, HIBERNATE, entity, MySQL, planning, radiotherapy, requirements.*

Lista de Figuras

Figura 1- Quadro Controle Planejamientos	13
Figura 2 - Eclipse Tela de Codificação.....	12
Figura 3- Eclipse Tela de Design	13
Figura 4- MySql Workbench - Tela modelagem gráfica.....	17
Figura 5- Tela Editor SQL.....	17
Figura 6 Estrutura ORM - Fonte: Devmedia 2011	18
Figura 7 Diagrama de Classes	22
Figura 8 - Estrutura de pacotes do Projeto	23
Figura 9 - persistence.xml	24
Figura 10 - Criando Schema Banco de dados.....	27
Figura 11 - Tabela Pacientes	28
Figura 12 - Método Salvar Paciente.	29
Figura 13 - Registro Pacientes no Banco de dados	30
Figura 14 - Busca por id.	30
Figura 15 - Tela Principal com menus.....	31
Figura 16 - Cadastro de Físicos	31
Figura 17 - Cadastro de Médicos.....	32
Figura 18 - Cadastro Clínica de origem.....	32
Figura 19 - Cadastro Região Anatômica	33
Figura 20 - Cadastro de Aparelho.....	33
Figura 21 - Cadastro sistema de Gerenciamento	33
Figura 22 - Tela Principal em Funcionamento	34
Figura 23 - Relatório de planejamentos por Período	35
Figura 24 - Modelo Conceitual Banco de Dados	47

Lista de Siglas

IDE	Ambiente Integrado de Desenvolvimento (do inglês: <i>Integrated Development Environment</i>).
SQL	Linguagem de Consulta Estruturada utilizada na manipulação de dados (do inglês: <i>Structured Query Language</i>).
ORM	Mapeamento do objeto relacional (do inglês: <i>Object Relational Mapper</i>).
MOR	Mapeamento de objeto relacional.
MVC	visão modelo e controle é um padrão que separa o modelo de negócio do restante da aplicação (do Inglês: <i>Model View Controller</i>).
SGBD	Sistema gerenciador de banco de dados.
JDT	Ferramentas de desenvolvimento Java (do Inglês: <i>Java Development Tools</i>).
JPA	do Inglês: <i>Java Persistence API</i> .
API	Interface de programação de aplicações (do Inglês: <i>Application Programming Interface</i>).
JSF	<i>API para desenvolvimento de interfaces</i> (do Inglês: <i>Java Server Faces</i>).
EJB	do Inglês: <i>Enterprise Java Beans</i> , <i>API</i> utilizado fornecer um desenvolvimento rápido e simplificado de aplicações <i>Java</i> , com base em componentes distribuídos, transacionais, seguros e portáteis.
JAAS	<i>Service</i> , é a <i>API</i> padrão do <i>Java</i> para controle de acesso e autorização em aplicações <i>JavaEE</i> (do Inglês: <i>Java Authentication and Authorization</i>).
UML	Linguagem de modelagem unificada (do Inglês: <i>Unified Modeling Language</i>).
DAO	Objeto de acesso a dados (do Inglês: <i>Data Access Object</i>).
3DRT	Radioterapia conformacional em 3 Dimensões.
TC	Tomografia Computadorizada (do Inglês: <i>Computed tomography</i>).
RMN	Ressonância magnética.
TPS	Planejamento computadorizado de tratamento.
IMRT	Radioterapia de intensidade modulada.
CRM	Conselho regional de medicina.
ABFM	Associação brasileira de física médica.
PET/CT	Tomografia por emissão de pósitrons.

SUMÁRIO

INTRODUÇÃO	10
JUSTIFICATIVA	13
OBJETIVOS	10
ORGANIZAÇÃO DO TRABALHO	10
1 MÉTODO	10
1.1 PRINCIPAIS FERRAMENTAS	10
1.2 IDE ECLIPSE	11
1.3 LINGUAGEM DE DESENVOLVIMENTO JAVA	13
1.3.1 ORIENTAÇÃO À OBJETOS	14
1.4 MySQL WORKBENCH 6.3	16
2 PERSISTÊNCIA DE DADOS	18
2.1 CONCEITUAÇÃO DO MAPEAMENTO OBJETO RELACIONAL	18
2.2 INTRODUÇÃO AO JPA E HIBERNATE	19
3 DESENVOLVIMENTO DA APLICAÇÃO	20
3.1 DOCUMENTO DE REQUISITOS	20
3.2 DESENVOLVIMENTO DA APLICAÇÃO	21
3.3 BANCO DE DADOS	27
3.3.1 CRIANDO AS TABELAS	28
3.3.2 CRIANDO O <i>EntityManager</i>	29
3.3.3 PERSISTINDO OBJETOS	29
3.3.4 BUSCANDO OBJETOS PELO IDENTIFICADOR	30
4 RESULTADOS	31
5 CONCLUSÃO	36
BIBLIOGRAFIA	37
APÊNDICE A	40
APÊNDICE B	47

INTRODUÇÃO

Cerca de 60% dos pacientes oncológicos são submetidos à radioterapia, que ao lado da quimioterapia e da cirurgia compõe o tripé do tratamento do câncer (PELIZZON, 2015). A radioterapia emprega feixe de radiação ionizante capaz de destruir células tumorais. Doses fracionadas de radiação são administradas em múltiplos campos direcionados ao volume alvo tumoral, com o menor prejuízo possível às células normais adjacentes. Porém, apesar da máxima proteção permitida a este tecido normal, a radioterapia é responsável por efeitos colaterais provenientes deste tratamento. Alguns aspectos radiobiológicos dos tecidos permitem elaborar “*constraints*” ou dose-limite em acordo com a sensibilidade inerente a cada tecido normal. Se esta dose de tolerância não for respeitada, podem ocorrer efeitos tóxicos e irreversíveis em maior frequência que aquelas aceitáveis (SEGRETO, 2000).

As diferentes técnicas de radioterapia variam conforme a tecnologia envolvida no planejamento e administração da radiação. A primeira a ser desenvolvida, a técnica convencional (2DRT), utiliza radiografias simples e referências anatômicas para delimitar o volume alvo. Possui custo relativamente inferior às demais técnicas, pode ser rapidamente planejada e realizada, sendo amplamente disponível. Sua desvantagem é a impossibilidade de identificação e avaliação da dose no volume alvo e tecidos normais adjacentes. Com isso, a entrega de dose é realizada em grandes volumes e margens amplas, com maior potencial de efeitos colaterais e impossibilidade de aumentar, em muitas situações, a dose de radiação administrada.

A radioterapia conformacional 3D (3DRT) utiliza imagens de simulação adquiridas por tomografia computadorizada (TC), Ressonância magnética (RNM), ou tomografia por emissão de pósitrons (PET/CT) para delinear o volume alvo e os órgãos a serem preservados através de sistema de planejamento computadorizado de tratamento (TPS). Possibilita tratamentos individualizados e com maior segurança na administração da dose de radiação.

A radioterapia de intensidade modulada (IMRT) é uma variação da técnica conformada, desenvolvida na década de 80, baseada em software capaz de modular a intensidade do feixe de radiação a fim de melhorar a cobertura do volume alvo ao mesmo tempo em que reduz a dose nos órgãos de risco (OAR), minimizando as toxicidades do tratamento (HERMANTO et al., 2007; MACDONALD et al., 2007; NARAYANA et al.,

2006) As desvantagens da IMRT decorrem da complexidade do planejamento, o que exige equipe capacitada, adequados equipamentos e o aumento do tempo de planejamento e tratamento.

Para os planejamentos 3DRT e IMRT é necessário o estabelecimento de processos que devem ser realizados para a adequada execução do tratamento. O controle de cada uma das etapas torna-se fundamental na garantia da qualidade da radioterapia.

O controle dos processos do planejamento radioterápico ainda hoje é feito manualmente através de um quadro físico, onde os dados são inseridos manualmente não podendo ser utilizados para coleta de dados estatísticos posteriormente, pois quando o planejamento é finalizado e o paciente inicia o tratamento, as informações são apagadas do quadro. Alguns serviços que possuem estes dados armazenados os fazem por duplicidade de informação, digitadas por membro da equipe em planilhas, o que pode gerar inconsistências dos dados.

Um serviço de radioterapia dispõe de uma série de recursos que devem ser geridos e organizados com a finalidade de oferecer aos pacientes a segurança e qualidade na administração da dose de radiação. Para atingir estes objetivos é fundamental que utilizemos sistema de informação. Neste sentido, os sistemas de informação são essenciais para o funcionamento, organização e gestão dos recursos e dados.

A interface visual e desenvolvimento de banco de dados *MySQL* que é um banco de dados relacional, isto é, possui relacionamento entre tabelas, robusto e de licença gratuita, também amplamente utilizado tanto para aplicações web como desktop. É utilizado também para a persistência dos dados o modelo objeto relacional, onde a persistência de dados em orientação a objetos consiste em manter objetos no banco de dados. As linguagens orientadas a objetos fornecem uma maneira natural e intuitiva de visualizar o processo de design do software ao modelar objetos por seus atributos e comportamentos assim como são descritos no mundo real (DEITEL, 2010).

O mapeamento objeto relacional (MOR) integra os recursos da linguagem de programação orientada a objetos com bancos de dados relacionais como *Oracle*, *MySQL*, *Sybase* entre outros (BARRY, 2014).

A técnica de mapeamento objeto relacional permite fazer uma relação dos objetos com os dados que os mesmos representam. Sua utilização vem crescendo muito nos últimos anos, e esse crescimento se deve principalmente pelo fato de muitos desenvolvedores não se sentirem a vontade de escrever código em linguagem de consulta estruturada (SQL) e pela

produtividade que esta técnica lhes proporciona. Existem ótimos MOR's como *Hibernate*, que será utilizado neste projeto, *NHibernate*, *Entity Framework*, *Core Data*, etc. (SOUZA, 2011).

Elicitação dos requisitos que o algoritmo deve possuir e especificação do documento de requisitos, onde a engenharia de requisitos estabelece o processo de definição de requisitos como um processo no qual o que deve ser feito é elicitado, modelado e analisado. Este processo deve lidar com diferentes pontos de vista, e usar uma combinação de métodos, ferramentas e pessoal. O produto desse processo é um modelo, do qual um documento de requisitos é produzido. Este processo acontece num contexto previamente definido a que se denomina universo de informação, que define o que uma solução computacional deve contemplar, e como será a apresentação dos dados.

Um algoritmo computacional é um conjunto de regras e procedimentos lógicos definidos que levam à solução de um determinado problema em uma quantidade finita de passos, e que será utilizado para o desenvolvimento do sistema computacional onde será utilizada a linguagem *Java*, que é uma linguagem de programação puramente orientada a objetos, isto é, tudo são classes ou instâncias com exceção dos seus tipos primitivos, é uma linguagem que permite *multithreading* onde é possível o desenvolvimento de aplicações/programas capazes de executar múltiplas rotinas concorrentemente. *Java* é independente de plataforma pois os programas são compilados para uma forma intermediária chamada de *bytecode* onde através de uma máquina virtual *java* (*Java Virtual Machine - JVM*) são executados, por exemplo um único *bytecode* pode ser executado em sistemas *Linux* ou *Windows* sem a necessidade do código ser compilado duas vezes, uma para cada sistema, sendo necessário apenas utilizar a máquina virtual que é responsável por essa tradução do *bytecode* para linguagem de máquina e por fim a execução do programa.

A ferramenta de modelo objeto relacional *jpa/Hibernate* é utilizada para persistência dos dados, com as vantagens da simplificação do processo de persistência que permite que o desenvolvedor do software faça todo o acesso ao banco de dados sem a necessidade de conhecimento da linguagem SQL (*do inglês Structured Query Language*) que é a linguagem de desenvolvimento de banco de dados, onde com algumas linhas de código essa ferramenta faz todo o trabalho de persistência dos dados no banco escolhido.

Desta forma, o desenvolvimento de um *software* que sirva de painel eletrônico e que substitua o quadro físico, permite acessibilidade e perenidade aos dados do planejamento radioterápico, além de promover um maior controle das informações do planejamento. Fornece dados estatísticos onde é possível avaliar a qualidade do serviço realizado, contribuindo para identificação de possíveis gargalos do processo. Cada vez mais a

preocupação no desenvolvimento de softwares permite interfaces intuitivas e de fácil operação. O conceito de pervasividade computacional, que é a capacidade de inserção da tecnologia no cotidiano das pessoas de maneira imperceptível, sem a necessidade do conhecimento técnico envolvido, tendência cada vez mais expressiva em nossa sociedade.

JUSTIFICATIVA

As etapas do processo de planejamento radioterápico ainda hoje são feitas manualmente através de um quadro/lousa físico, onde se escreve ou altera-se o status de cada etapa com caneta e após finalizado o processo de cada paciente, o mesmo é apagado para dar lugar ao próximo planejamento, dificultando o processo estatístico, e também há redundância no processo, já que todos esses dados são inseridos também em uma planilha eletrônica para se poder consultar posteriormente e dessa forma ser realizada estatísticas que se façam necessárias. Mas o fato dos dados serem inseridos em lugares diferentes e por pessoas diferentes muitas vezes, já que não há uma única pessoa responsável pela inserção desses dados pois todos os físicos médicos do serviço radioterápico realizam essa tarefa, pode causar inconsistência nos dados, que são armazenados de maneira manual sem nenhum mecanismo que garanta a qualidade e fidelidade dos dados armazenados. A figura 1 exemplifica o quadro utilizado para o controle.

PACIENTE	RG	MÉDICO	REGIÃO	DATA CT	CONT	ALVO	PLANO	APROV	IMPRES	1º ASS	2º ASS	BLOCO	LANTIS	OBSERVAÇÃO
Maria Silva de Camargo	05540560	Felipe	MAMA	12/10	X10	OK	OK	OK	OK	EU				
Selma A. dos Santos	13773095	Felipe	MAMA	21/10	X10	OK	OK	OK	OK	EU				
Lucia Aparecida da Costa	04552813	Felipe	MAMA	14/10	X10	OK	OK	OK	OK	EU	03			AOB
Marcelina dos Reis	0708820K	Felipe	MAMA	21/10	X10	OK	OK	OK	OK	EU				
NA														
LEFEMENTE S. BRITO	13638574	VIVI	RETO	23/08	X10	OK	OK	OK	OK	ISA	02	29/8	020	POVARDA. DE FELI
A. S. RODRIGUES	013559	EFRAIM	MAMA	20/10	X10									
DOS SANTOS	0734963	EFRAIM	MAMA	20/10	X10									
ERI AP. R. CAMADO	0203480143	VIVI	MAMA	17/10	X10	OK	OK	OK	OK	ISA	03	27/8	020	P125110
TELIUO R. DE ALMEIDA	13277436	RAUO	PROSTATA	13/10	X10									
ALESSANDRA S. DOS SANTOS	0718003A	VIVI	UTERO	11/10	X10	OK	OK	OK	OK	CFB	EDE	13/10	CFB	
PAULO AP. VENANCIO	1383355	FELIPE	RETO	20/08	X10	OK	OK	OK	OK	CFB	EDE	30/9		
REGINALDO PAULO	15587876	DIOGO	CERVICAL	05/10	X10	OK	OK	OK	OK	EU				
CONCEIÇÃO VITOR DA SILVA	13786233	VIVI	MAMA	24/10	X10									
JOILA RODRIGUES	0145058K	VIVI	LINGUA	11/10	X10	OK	OK	OK	OK	ISA	03	34/10	020	
MARCIO FUMINA	0024071A	VIVI	PROSTATA	17/10	X10									
MARCIA S. C. MUNHOZ	0550609A	DIAS	ESTOMAGO	05/10	X10	OK	OK	OK	OK	EU				
JOHANE L. JONHA	1377018B	JUNIO	CERVICAL	08/09	X10	OK	OK	OK	OK	CFB	EDE	13/10	CFB	P125110
TE LOPES FILIOBINO	0429892A	VIVI	UTERO	20/07	X10	OK	OK	OK	OK	AND	03	14/9	020	
R. BOTA SILVA	138692B	VIVI	UTERO											

Figura 1- Quadro Controle Planejamentos

OBJETIVOS

Desenvolvimento de um algoritmo e posteriormente um *software* desenvolvido em *java* e com sistema gerenciador de banco de dados (SGBD) que utilize um banco de dados relacional para controlar eletronicamente todas as etapas do processo de planejamento de pacientes em um serviço de radioterapia, de maneira que a integridade dos dados seja garantida pela estrutura oferecida pelo SGBD, e que permita consultas posteriores fornecendo dessa maneira dados consistentes para realização de estatística que pode agilizar o processo de planejamento e também possibilitando a identificação de possíveis gargalos podendo assim serem sanados mais rapidamente melhorando inclusive a qualidade do atendimento prestado.

ORGANIZAÇÃO DO TRABALHO

No capítulo 1 é apresentado ao leitor de maneira mais geral os métodos e as ferramentas utilizadas para desenvolvimento deste trabalho. Já no capítulo 2 é feita uma introdução à persistência de dados, uma contextualização do modelo de mapeamento objeto relacional e apresentada de maneira introdutória o *framework* de persistência *JPA/Hibernate*.

No capítulo 3 é apresentado de maneira prática como foi implementado o modelo de mapeamento objeto relacional na persistência de dados do sistema desenvolvido. Já no capítulo 4 é apresentado os resultados obtidos, com figuras mostrando as telas da aplicação e resultados obtidos. Já na conclusão é apresentado as considerações sobre a aplicação, utilização da ferramenta de persistência e possibilidades para futuras atualizações do sistema. E por último e não menos importante apresentamos as referências bibliográficas, que são as fontes pesquisadas para a realização deste trabalho, onde o leitor se houver interesse em se aprofundar em algum assunto tratado de maneira mais superficial, tem a possibilidade de realizar leituras adicionais e adquirir o conhecimento desejado.

1 MÉTODO

Trata-se de projeto de desenvolvimento tecnológico, mais especificamente um algoritmo que com ajuda de uma ferramenta computacional visa otimizar o controle das etapas do processo de planejamento de pacientes em radioterapia. Os parâmetros clínicos e de planejamento utilizados foram:

1. NOME DO PACIENTE
2. NÚMERO DE IDENTIFICAÇÃO DO PACIENTE
3. MÉDICO RESPONSÁVEL
4. PATOLOGIA / LOCAL DA RADIOTERAPIA
5. DATA DA TOMOGRAFIA DE PLANEJAMENTO
6. REALIZAÇÃO DO CONTORNO DOS ORGÃOS DE RISCO
7. DEFINIÇÃO DO VOLUME ALVO
8. FINALIZAÇÃO DO PLANO
9. APROVAÇÃO DO PLANO
10. IMPRESSÃO DAS FICHAS E PARAMETROS TÉCNICOS
11. CONTROLE DE QUALIDADE DO PLANO
12. 1ª E 2ª ASSINATURA DOS FÍSICOS RESPONSÁVEIS
13. CHECAGEM DOS DADOS EM SISTEMA DE GERENCIAMENTO
14. DATA DE INÍCIO DA RADIOTERAPIA
15. OBSERVAÇÕES

1.1 PRINCIPAIS FERRAMENTAS

As principais ferramentas utilizadas neste trabalho foram a *IDE Eclipse* como ambiente de desenvolvimento, o *SGBD MySql*, a ferramenta de desenvolvimento visual do banco de dados *MySql* o *MySql Workbench*, a linguagem de programação orientada a objetos *Java* e a ferramenta de persistência de dados que utiliza o modelo objeto relacional *JPA/Hibernate*.

1.2 IDE ECLIPSE

O *Eclipse* é uma plataforma de desenvolvimento de *software* livre extensível, baseada em *Java*. Por si só, é simplesmente uma estrutura e um conjunto de serviços para desenvolvimento de aplicativos de componentes de *plug-in*. Felizmente, o *Eclipse* vem com um conjunto padrão de *plug-ins*, incluindo as amplamente conhecidas Ferramentas de Desenvolvimento *Java* (JDT). O Projeto *Eclipse* foi originalmente criado pela *International business machines Corporation* (IBM) em novembro de 2001 e suportado por um consórcio de fornecedores de *software*. A *Eclipse Foundation* foi criada em janeiro de 2004 como uma organização sem fins lucrativos independente para atuar como organizadora da comunidade *Eclipse*. Foi criada para permitir o surgimento de uma comunidade ao redor do *Eclipse* independente de fornecedor, transparente e aberta. Hoje, a comunidade do *Eclipse* é composta por pessoas e organizações de uma seção transversal do segmento de mercado de *software* (IBM DEVELOPERWORKS, 2012).

A *Eclipse Foundation* gerencia e dirige o desenvolvimento contínuo do *Eclipse*. A fundação fornece serviços para a comunidade, mas não emprega os desenvolvedores de *software* livre (denominados *committers*), que realmente trabalham nos projetos *Eclipse*. Os *committers* do *Eclipse* são normalmente empregados pelas organizações ou são desenvolvedores independentes que trabalham como voluntários em um projeto de *software* livre (IBM DEVELOPERWORKS, 2012).

No *eclipse* é possível criar toda a aplicação inclusive a parte visual, das telas, botões, etc., através de linhas de código, ele possui uma ferramenta chamada de *intellisense*, onde digitando o início do comando e pressionando-se as teclas *control* e espaço no teclado, aparece sugestões referente ao comando desejado, possui também através dos seus menus, opções que criam uma estrutura genérica de determinados comandos onde o mesmo vem praticamente pronto e que para terminar basta apenas trocar os nomes genéricos pelos criados no desenvolvimento do sistema, a figura abaixo apresenta o editor de código do eclipse.

Figura 2 - Eclipse Tela de Codificação

Possui também uma aba chamada *design* onde é possível criar toda a parte visual da aplicação com esquemas prontos e fornece atalhos as funções, economizando linhas de código e em consequência tempo de desenvolvimento, onde por exemplo, clica-se em um botão e posiciona-o na região da tela que está sendo desenvolvida e personaliza-se esse botão de forma gráfica sem a necessidade de digitar todo o código necessário para a criação do mesmo.

A figura a seguir mostra a aba design para exemplificar.

Figura 3- Eclipse Tela de Design

1.3 LINGUAGEM DE DESENVOLVIMENTO JAVA

Java é uma linguagem de programação de computador que é amigável para o aprendizado de iniciantes e adequada para programadores experientes utilizarem na construção de sistemas de informação empresariais importantes. O *Java* tornou-se a linguagem preferida para implementar aplicações baseadas na internet e softwares para dispositivos que se comunicam por rede (DEITEL, 2010).

Java é uma linguagem puramente orientada a objetos, tudo são classes ou instâncias de classes, com exceção de seus tipos primitivos de dados, não possui ponteiros, isto é, não permite a manipulação direta de endereços de memória, possuindo um mecanismo automático de gerenciamento de memória o *Garbage Collector*, permite também *multithreading*, isto é, possui recursos que permite o desenvolvimento de aplicações capazes de executar múltiplas rotinas concorrentemente. Outra vantagem é que o *Java* é independente de plataforma, pois os programas em *java* são compilados para uma forma intermediária chamado de *bytecode*, que é posteriormente interpretado e executado por uma máquina virtual *java*, permitindo que uma plataforma (*Windows*, *Linux*, etc.), possuindo essa máquina virtual é capaz de executar o sistema.

Os programas *Java* normalmente passam por cinco fases para serem executados:

- **Fase 1 (Edição):** Consiste em editar um arquivo com código em *Java* e salvá-lo com a extensão *.java*;
- **Fase 2 (Compilação):** Comando *javac* compila o programa. O compilador *Java* traduz (*.java* => *.class*);
- **Fase 3 (Carga):** Carrega o programa na memória antes de ser executado. Carregador de classe, pega o arquivo(s) *.class* que contém os *bytecodes*.
- **Fase 4 (Verificação):** O verificador assegura que os *bytecodes* são válidos e não violam as restrições de segurança de Java.
- **Fase 5 (Execução):** A JVM máquina virtual *Java* (Interpretador) interpreta (em tempo de execução) o programa, realizando assim a ação especificada pelo programa (DEITEL, 2010).

1.3.1 ORIENTAÇÃO À OBJETOS

Neste capítulo será apresentada uma introdução ao conceito de orientação a objetos, para que o leitor deste trabalho esteja mais familiarizado com o processo de desenvolvimento do *software* objetivo deste trabalho.

A produção de *software* para se conservar o padrão de qualidade exige a adoção de métodos de desenvolvimento que atendam essa dinâmica do mercado.

A orientação a objeto é um dos métodos mais utilizados, oferecendo recursos para mais qualidade no desenvolvimento de software, e com seus conceitos traduzir as características do mundo real para o mundo computacional. Abaixo é feita uma introdução aos principais recursos da orientação à objetos:

- **Abstração:** A abstração consiste na separação dos objetos de uma totalidade, proporcionando uma melhor utilização dos recursos da orientação a objetos, onde são isolados os objetos que se deseja representar do ambiente complexo em que se situam, e utiliza-se nesses objetos somente as características que são relevantes para o problema em questão;
- **Encapsulamento:** É um dos pilares da orientação a objetos, onde esconde-se parte da implementação do mundo exterior, mas cumprindo seu objetivo escondendo-se detalhes dessa implementação, como uma espécie de caixa preta, onde, não há a necessidade de preocupar-se com o que acontece dentro dela, onde a maioria das linguagens de programação possuem três níveis de acesso, o público, onde todos os

objetos têm acesso, o protegido, onde o acesso é apenas para a instância, isto é, para o objeto e suas subclasses e o método privado, onde o acesso é permitido apenas para o objeto da instância;

- **Classes e Objetos:** Especifica um conjunto de propriedades e comportamento de um molde específico, sendo utilizada para modelar entidades do mundo real e onde todo objeto é uma instância de apenas uma classe e a comunicação entre objetos é realizada por troca de mensagens, Por exemplo ContaEmBanco é uma classe e A conta de uma pessoa específica é uma instância dessa classe;
- **Herança:** É uma das principais características da orientação a objetos, já que permite o reaproveitamento de métodos e atributos diminuindo o tempo de desenvolvimento de uma aplicação;
- **Polimorfismo:** O polimorfismo está diretamente ligado à hereditariedade das classes, este trabalha com a redeclaração de métodos herdados, ou seja, os métodos têm a mesma assinatura (têm o mesmo nome), mas a forma de implementação utilizada os diferem da superclasse (TECHNET MICROSOFT, 2015).

A orientação a objetos hoje, é amplamente utilizada para o desenvolvimento de software, por trazer uma facilidade maior ao desenvolvedor na manutenção dos softwares desenvolvidos, lembrando que este capítulo apresenta apenas os conceitos básicos para melhor contextualização da programação orientada a objetos utilizados para desenvolvimento do trabalho.

MySQL 5.7.16

O *MySQL* é um banco de dados que utiliza o modelo relacional, isto é, representa uma base de dados como uma coleção de relações embasada na teoria dos conjuntos, onde cada tabela é uma relação, é bidimensional, possui um nome único e cada linha corresponde a uma entidade ou relacionamento particular conhecida como atributos. Este foi o modelo e o banco de dados escolhido para utilização no projeto, a versão utilizada é o *MySQL community Edition*. O *MySQL community edition* é a versão gratuita disponibilizada para download livremente. É o banco de dados de código aberto mais popular no mundo, onde uma vasta gama de desenvolvedores e entusiastas de código aberto colaboram para uma constante evolução do *MySQL*, sendo mantido por uma renomada empresa especializada em banco de dados a *Oracle*. O *MySQL* oferece um servidor de banco de dados robusto rápido, multi-usuário e *multi-thread*, onde suporta uma carga pesada de dados, oferecendo todas as

ferramentas necessárias para gerenciamento, backup e segurança que os bancos de dados comerciais oferecem (ORACLE CORPORATION, 2017).

1.4 MySQL WORKBENCH 6.3

O *MySQL Workbench*, é uma ferramenta gráfica para banco de dados *MySQL*, suportando completamente o servidor a partir da versão 5.5, onde é compatível de maneira parcial com versões anteriores do servidor (versões 5.x), não suportando as versões 4.x ou anteriores. A suas funcionalidades abrangem 5 tópicos principais:

- **Desenvolvimento SQL:** Permite criar e gerenciar conexões com banco de dados, realizar consultas *SQL* nas conexões criadas através do editor *SQL* incorporado; (ORACLE CORPORATION, 2017).
- **Modelagem de dados (Design):** Permite criar modelos e esquemas de banco de dados graficamente, permite também a engenharia reversa, onde através de uma base de dados já criadas é possível se obter o modelo conceitual por exemplo, e possui também um editor de tabelas o *Table Editor*, que permite alterações na estrutura das tabelas do banco como índices, colunas, opções, rotinas, visualizações, entre outras sem digitar uma única linha de código *SQL* (ORACLE CORPORATION, 2017).
- **Administração do servidor:** Permite que sejam administradas instâncias do servidor *MySQL*, administrando usuários, executando backup e recuperação, inspecionando dados de auditoria, visualizando a integridade do banco de dados e monitorando o desempenho do servidor *MySQL* (ORACLE CORPORATION, 2017).
- **Migração de dados:** Permite a migração do *Microsoft SQL Server*, *Microsoft Access*, *Sybase ASE*, *SQLite*, *SQL Anywhere*, *PostgreSQL* e outras tabelas *RDBMS*, objetos e dados para o *MySQL*, assim como suporta migração de versões anteriores do *MySQL* para os lançamentos mais recentes.

O *MySQL Workbench* está disponível em duas edições: a *Community Edition* e a *Commercial Edition*. A *Community Edition* que foi a versão utilizada no projeto, está disponível gratuitamente (ORACLE CORPORATION, 2017).

A figura à seguir mostra algumas das telas de trabalho da ferramenta.

Figura 4- MySQL Workbench - Tela modelagem gráfica

Figura 5- Tela Editor SQL

2 PERSISTÊNCIA DE DADOS

A persistência de dados em orientação a objetos consiste na persistência de objetos no banco de dados. As linguagens orientadas a objetos fornecem uma maneira natural e intuitiva de visualizar o processo de design do *software* ao modelar objetos por seus atributos e comportamentos assim como são descritos no mundo real (DEITEL, 2010).

2.1 CONCEITUAÇÃO DO MAPEAMENTO OBJETO RELACIONAL

O mapeamento objeto relacional (MOR) integra os recursos da linguagem de programação orientada a objetos com bancos de dados relacionais como Oracle, DB2, Sybase entre outros (BARRY, 2014).

A técnica de mapeamento objeto relacional permite fazer uma relação dos objetos com os dados que os mesmos representam. Sua utilização vem crescendo muito nos últimos anos, e esse crescimento se deve principalmente pelo fato de muitos desenvolvedores não se sentirem a vontade de escrever código SQL e pela produtividade que esta técnica lhes proporciona. Existem ótimos MOR's como *Hibernate*, *NHibernate*, *Entity Framework*, *Core Data*, etc. (SOUZA, 2011).

Na figura 6 é possível visualizar a estrutura do Mapeamento objeto relacional.

Figura 6 Estrutura ORM - Fonte: Devmedia 2011

Existem muitos benefícios em utilizar o mapeamento objeto relacional (MOR) ao invés de outras técnicas de acesso a dados, pois automatiza a conversão objeto-mesa e mesa-objeto simplificando o desenvolvimento. Este desenvolvimento simplificado leva o produto mais rápido ao mercado com redução de custos de desenvolvimento e manutenção. O MOR requer menos código, em comparação com SQL embutido, possuindo a mesma

funcionalidade, mas com menos código, e por último, e não menos importante, o MOR fornece *cache* transparente de objetos, melhorando assim o desempenho do sistema. Um bom MOR é uma solução altamente otimizada que faz com que uma aplicação seja mais rápida e de fácil manutenção. Uma ferramenta MOR deve ter a capacidade de mapear objetos de negócios de *back-end* de banco de dados via tabelas de algum tipo de mapeamento de metadados. Esta é, fundamentalmente, o núcleo do mapeamento objeto-relacional (MEHTA, 2008).

O método de mapeamento objeto relacional, é uma solução elegante para o problema que aparece quando o sistema precisa armazenar de forma permanente os dados gerados por ele. O MOR tem como função simplesmente converter objetos que estão na memória para dados relacionais e vice-versa (COELHO, 2004).

2.2 INTRODUÇÃO AO JPA E HIBERNATE

O *JPA* é um *framework* que vem ganhando mais espaço no mercado a cada dia que passa. O *JPA* nada mais é do que um conjunto de regras, normas e interfaces para definir um comportamento também conhecido como especificação (OLIVEIRA, [20--?]).

Com o *JPA*, os objetos são *POJO* (*Plain Old Java Objects*), ou seja, não é necessário nada de especial para tornar os objetos persistentes, bastando adicionar algumas anotações nas classes que representam as entidades do sistema e começar a persistir ou consultar objetos. *JPA* é uma especificação e não um produto, para poder trabalhar com *JPA* é necessário a utilização de uma implementação (FARIA; JÚNIOR, 2015).

No caso da *JPA*, a implementação mais famosa e utilizada é o *Hibernate*, mas existem outras no mercado, como *OpenJPA*, *EclipseLink*, *Batoo* entre outras. Cada implementação possui suas vantagens e desvantagens na hora da utilização. O *Hibernate* contém diversas funcionalidades como Engenharia Reversa, Exportação de *Schema* facilitado e anotações próprias para simplificar a utilização da *API* (OLIVEIRA, [20--?]).

O projeto do *Hibernate Object Relational Mapping* (ORM) possui alguns módulos onde o ***Hibernate EntityManager*** é a implementação do *JPA* que encapsula o *Hibernate Core* (FARIA; JÚNIOR, 2015).

Porém, para utilização do *Hibernate*, se faz necessário adicionar diversas dependências para o seu correto funcionamento. A implementação do *JPA* deve satisfazer todas as interfaces determinadas pela especificação. Mesmo uma implementação incompleta pode ser considerada válida. Na versão 3.x do *Hibernate*, caso um desenvolvedor adicione a

anotação *@NamedNativeQuery*, que permite que uma *query* nativa (com a sintaxe do banco de dados) fosse declarada em uma *Entity* (entidade), uma exceção seria lançada pelo *Hibernate* informando que essa funcionalidade ainda não está implementada (OLIVEIRA, [20--?]).

Além das interfaces implementadas do *JPA*, cada implementação pode ter as suas próprias anotações/classes para adicionar mais comportamentos, indo além do que prevê a especificação. O *Hibernate*, por exemplo, tem a anotação *org.hibernate.annotations.ForeignKey*, que consegue determinar o nome de uma chave *foreign key* (chave estrangeira), algo que não existe na *JPA*, e que quando utilizada pode fazer com que sua aplicação não seja mais portátil para outra implementação. Em contrapartida, a vantagem de ter um projeto usando apenas anotações do *JPA* é que ela se torna automaticamente portátil, possibilitando a troca de *Hibernate* para o *EclipseLink*, por exemplo, uma mera questão de alterar o *provider* (uma simples configuração no arquivo de configuração *persistence.xml* (OLIVEIRA, [20--?])).

3 DESENVOLVIMENTO DA APLICAÇÃO

Foram criados, um *software* e um banco de dados para controle eletrônico do planejamento. A metodologia utilizada é descrita conforme as etapas de realização da mesma.

3.1 DOCUMENTO DE REQUISITOS

A partir da definição do conteúdo e da apresentação dos dados da solução computacional, estabeleceu-se o universo de informação, com os conteúdos produzidos na elicitação dos requisitos que o algoritmo deve possuir e especificação do documento destes requisitos.

Os requisitos de software formam a base de onde a qualidade é avaliada, onde os tipos de requisitos são explícitos e implícitos e para que o software possua qualidade deve atender a ambos. A definição da qualidade de software segundo Pressman é: “conformidade a requisitos funcionais e de desempenho explicitamente declarados, a padrões de desenvolvimento claramente documentados e a características implícitas que são esperadas de todo *software* profissionalmente produzido” (Pressman, 2001).

A partir da definição do conteúdo e da apresentação dos dados da solução computacional, estabeleceu-se o universo de informação, com os conteúdos produzidos na

elicitação dos requisitos que o algoritmo deve possuir e especificação do documento destes requisitos. Os requisitos são subdivididos em três partes:

- Requisitos funcionais, ligados as funcionalidades do *software*;
- Requisitos não funcionais, ligados a restrições ou qualidades específicas que o *software* deve possuir;
- Requisitos inversos, onde estabelece condições que nunca devem ocorrer.

A definição de requisitos foi desenvolvida através de um processo de elicitação, que consiste na pesquisa do processo e reuniões com usuários para se levantar as necessidades do sistema, modelagem desses dados e análise posterior a modelagem para validação do documento e assim prosseguir com o desenvolvimento do software. Este processo deve lidar com diferentes pontos de vista, e usar uma combinação de métodos, ferramentas e pessoal.

Atualmente o documento de requisitos encontra-se na versão 2 e encontra-se no apêndice A deste documento.

3.2 DESENVOLVIMENTO DA APLICAÇÃO

Para criação, foi utilizada a plataforma integrada para desenvolvimento de *software Eclipse*, utilizando a linguagem *Java*, diagrama de classes UML que é uma linguagem unificada para modelagem de software, é composta por vários tipos de diagramas que podem ser feitos de acordo com a necessidade durante o projeto de modelagem do software, e um banco de dados MySQL.

Após a finalização do documento de requisitos já descrito acima, foi desenvolvido um diagrama de classes utilizando a *Unified Modeling Language (UML)*, que serve para orientar o desenvolvedor no que diz respeito as classes que serão necessárias no projeto, os métodos que essas classes devem possuir e o relacionamento dessas classes que serve também como orientação na criação do mapeamento dos objetos relacionais e consequente criação do banco de dados e o relacionamento das tabelas. A figura a seguir ilustra o diagrama de classes.

Figura 7 Diagrama de Classes

Após a criação do diagrama de classes é necessário definir qual implementação de persistência será utilizada, no caso deste projeto foi definido que seria o *JPA* que nada mais é que um conjunto de regras, normas e interfaces para definir um determinado comportamento, o que também é conhecido como especificação. Mas para que uma especificação possa ser utilizada é necessário realizar uma implementação, que no caso do *JPA* a mais conhecida é o *Hibernate*, só lembrando que existem outras especificações e outras implementações que podem ser utilizadas com o Java, cada uma com suas vantagens e desvantagens (OLIVEIRA, [20--?]).

Para a utilização do *JPA* e *Hibernate* é necessário verificar se o ambiente de desenvolvimento está apto a utilizar esses recursos, se não estiver é necessário instalar os pacotes de atualização necessários para utilizar os recursos desejados.

Após a definição dos *frameworks* que serão utilizados, cria-se um projeto com o nome desejado para o desenvolvimento do *software*. É criada uma estrutura de pastas para separar cada parte da aplicação também conhecida como pacotes. A figura abaixo exemplifica como ficou a estrutura do projeto que será explicada adiante.

Figura 8 - Estrutura de pacotes do Projeto

O retângulo vermelho da figura mostra o diretório principal que leva o nome do projeto e onde tudo que for criado vai ficar, como pacotes referentes a telas da aplicação, imagens e o arquivo de configuração do projeto.

Abaixo está o retângulo verde mostrando os pacotes dentro do diretório principal, que são nomeados de forma a serem diferenciados, onde todos começam com *com.mestrado* e o nome que representa o pacote. O primeiro é o pacote de nome *DAO*, que significa *data access Object*, responsável pela abstração do acesso aos dados da aplicação, tudo o que o sistema necessitar quanto acesso aos dados como salvar, buscar ou alterar um paciente por exemplo, ele vai “pedir as classes DAO”, sem a necessidade de saber como vai ser executada determinado acesso. Após nota-se a classe *model*, responsável pela criação dos objetos, por exemplo a classe paciente, que representa no ambiente computacional um paciente real com os dados que um paciente deve possuir, onde cada paciente é um objeto no sistema, isso é uma instância de uma classe modelo e logo à seguir nota-se a classe *View*, responsável pelas telas da aplicação, isto é, a parte que aparece para o usuário do sistema, e que faz a “ponte” entre a classe modelo e a classe DAO.

Logo abaixo há um pacote chamado *imagem*, onde estão todas as imagens utilizadas no projeto, e mais abaixo um pacote chamado *META-INF*, onde fica o arquivo de configuração *persistence.xml*, e deve ser criado dentro do diretório principal da aplicação.

O *persistence.xml* é o ponto de partida de qualquer projeto que utilize o *JPA*, pois é nele que constarão as informações de qual banco de dados será utilizado, configurações específicas, permissões como por exemplo como será o acesso ao banco e as tabelas do banco, como por exemplo, se poderá ser criado, criado e atualizado, criado, atualizado e deletado, entidades a serem utilizadas entre outros (OLIVEIRA, [20--?]).

Á seguir segue a figura do *persistence.xml*.

```

1 <?xml version="1.0" encoding="UTF-8"?>
2 <persistence version="2.0"
3   xmlns="http://java.sun.com/xml/ns/persistence" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4   xsi:schemaLocation="http://java.sun.com/xml/ns/persistence http://java.sun.com/xml/ns/persistence/persistence_2_0.xsd">
5
6   <persistence-unit name="mestradoPU" transaction-type="RESOURCE_LOCAL">
7
8     <properties>
9       <property name="javax.persistence.jdbc.url" value="jdbc:mysql://localhost:3306/mestrado-novo"/>
10      <property name="javax.persistence.jdbc.user" value=""/>
11      <property name="javax.persistence.jdbc.password" value=""/>
12      <property name="javax.persistence.jdbc.driver" value="com.mysql.jdbc.Driver"/>
13
14      <!-- validate | update | create | create-drop -->
15      <property name="hibernate.hbm2ddl.auto" value="update"/>
16      <property name="hibernate.dialect" value="org.hibernate.dialect.MySQLDialect"/>
17    </properties>
18  </persistence-unit>
19 </persistence>
20
21 </xml>

```

Figura 9 - *persistence.xml*

O arquivo de persistência define unidade de persistência conhecidas como *persistence units* que neste caso foi definida como *mestradoPU* que é muito importante para o funcionamento correto da aplicação que se nota no retângulo vermelho na figura 9 (FARIA; JÚNIOR, 2015).

Neste projeto, como foram definidos as classes e o relacionamento dessas classes no diagrama de classes, e configurado corretamente o arquivo de persistência, deixamos para o *JPA* e *Hibernate* a criação do banco de dados com suas respectivas tabelas e relacionamentos nos passos que serão detalhados no decorrer deste trabalho.

Será detalhado todo o processo utilizando a classe *Paciente.Class*, já que o processo se repete de maneira muito parecida para as outras classes, o sistema será entregue em arquivo digital anexo para que futuramente possa ser implementado ou atualizado.

Após criado o projeto e configurado o arquivo de persistência, é criado o pacote *com.mestrado.model*, onde estão todas as classes modelos do projeto, depois é criada uma classe java como por exemplo a *Paciente.java* que conterà todos os atributos que um paciente deve possuir como código do paciente, Nome, Data de Nascimento e registro HC, que é o

registro que identifica o paciente no Hospital Das Clínicas de Ribeirão Preto, como mostrado no trecho de código à seguir.

```
package com.mestrado.model;

import java.util.Date;
import java.util.List;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.OneToOne;
import javax.persistence.Table;
import javax.persistence.Temporal;
import javax.persistence.TemporalType;
import javax.persistence.UniqueConstraint;

@Entity
@Table(uniqueConstraints = @UniqueConstraint(columnNames = "registro", name
= "registropaciente_uk"))
public class Paciente {

    private Long codPaciente;
    private String nomePaciente;
    private String registro;
    private Date dtNascimento;

    @OneToOne(mappedBy="Planejamento")
    private List<Planejamento> planejamentos;
```

No trecho de código acima na primeira linha nota-se o pacote onde se encontra a classe, logo abaixo todas as importações que são necessárias para os recursos utilizados pela classe. A anotação *@Entity*, logo abaixo diz para o *JPA/Hibernate* que esta classe será também uma tabela que será criada no banco de dados, já a próxima linha serve para que o *JPA/Hibernate* faça uma verificação se existe outro paciente com o mesmo registro já cadastrado no banco de dados no momento em que o método utilizado para salvar o paciente for chamado, e logo após o início da classe e as declarações de variáveis, que são os atributos do paciente, e que constarão no banco de dados na tabela paciente.

Como pode ser observado no atributo *private List<Planejamento> planejamento*, que é uma coleção da classe *planejamento*, nota-se acima a anotação *@OneToOne(mappedBy="Planejamento")*, que é a anotação que diz para o *JPA/Hibernate* que existe um relacionamento um para muitos com a classe/entidade *planejamento*, onde diz que um paciente pode possuir vários planejamentos, mas que um planejamento terá apenas um paciente, e o implementa automaticamente no banco de dados quando a aplicação é executada pela primeira vez. O caminho inverso também tem que ser feito na classe *Planejamento* onde é feita também as anotações mostrando o relacionamento com a classe

Paciente. Após esta etapa são gerados os *getters* e os *setters*, que são métodos que são utilizados por outras classes para acesso as variáveis/atributos da classe paciente. O trecho de código à seguir demonstra os *getters* e os *setters* da classe Paciente.java.

```
@Id
@GeneratedValue(strategy = GenerationType.IDENTITY)
public Long getCodPaciente() {
    return codPaciente;
}
public void setCodPaciente(Long codPaciente) {
    this.codPaciente = codPaciente;
}
public String getNomePaciente() {
    return nomePaciente;
}
public void setNomePaciente(String nomePaciente) {
    this.nomePaciente = nomePaciente;
}
@Temporal(TemporalType.DATE)
public Date getDtNascimento() {
    return dtNascimento;
}
public void setDtNascimento(Date data) {
    this.dtNascimento = data;
}
public String getregistro() {
    return registro;
}
public void setregistro(String registro) {
    this.registro = registro;
}
public List<Planejamento> getPlanejamentos() {
    return planejamentos;
}
public void setPlanejamentos(List<Planejamento> planejamentos) {
    this.planejamentos = planejamentos;
}
```

Para cada atributo foram criados dois métodos, um com prefixo *set* e outro com prefixo *get*, os métodos com prefixo *set* são utilizados para adicionar um valor para uma variável/atributo, e o método *get* é utilizado para recuperar um valor de uma determinada variável/atributo.

Na primeira linha logo em cima do atributo *getId*, se nota outra anotação,

```
@Id
@GeneratedValue(strategy = GenerationType.IDENTITY)
```

, onde a anotação *@Id* diz que este atributo será a chave primária dessa entidade/classe na tabela do banco de dados, e a anotação *@GeneratedValue(strategy = GenerationType.IDENTITY)* diz que essa chave primária será do tipo auto incremento, isto é, será gerada automaticamente cada vez que um novo paciente for cadastrado. Há uma outra anotação sobre o atributo *getDtNascimento* que é, *@Temporal(TemporalType.DATE)*, que

diz para o *framework* que queremos armazenar apenas a data, pois é possível customizar como o dado será armazenado, como por exemplo, somente o dia, dia e mês, dia mês e ano, data com hora, com hora e minutos entre outros.

Já nesse ponto nota-se a vantagem de se utilizar ferramentas *ORM*, como *JPA/Hibernate*, com apenas algumas anotações e sem digitar uma única linha de código de programação *SQL* que é a linguagem utilizada em banco de dados é possível criar tabelas, gerar chave primária e dizer que essa chave é auto incrementada, poupando trabalho do desenvolvedor e economizando linhas de código e facilitando a manutenção e aumentando a agilidade no desenvolvimento de aplicações.

3.3 BANCO DE DADOS

Para que tudo que foi descrito acima funcione, é necessário que o servidor do banco de dados *MySQL* esteja instalado, e a ferramenta gráfica *workbench*, não é obrigatória, mas facilita na criação do Schema do banco de dados e para tarefas de manutenção referentes ao banco de dados. O modelo conceitual que foi gerado à partir do *workbench* encontra-se no apêndice B.

Primeiramente é necessário criar um schema com o nome do banco de dados que foi especificado no arquivo de persistência. A figura 10 ilustra essa etapa.

Figura 10 - Criando Schema Banco de dados

Clica-se no botão circulado de vermelho acima, após aparece a tela onde na caixa nome especifica o nome do schema que é o nome dado ao banco de dados para que a ferramenta *ORM* possa encontrar o banco onde irá trabalhar e logo após clica-se em *apply*,

feito isso é só olhar na aba *Schemas* à esquerda e já se pode ver o novo schema de banco de dados criado, já é possível prosseguir.

3.3.1 CRIANDO AS TABELAS

As tabelas podem ser criadas de maneira independente, e gerenciadas e acessadas pelo *JPA*, mas demanda conhecimentos mais aprofundados em banco de dados ou como foi utilizado neste trabalho, onde todo o processo de criação e atualização do banco de dados será feito pelo *Hibernate*, graças a propriedade *hibernate.hbm2ddl.auto* com o valor *update* especificada no arquivo *persistence.xml*, então é necessário criar apenas um *EntityManagerFactory*, para que as tabelas mapeadas pela entidade sejam criadas ou atualizadas.

Esse *EntityManagerFactory*, é criado na classe *PacienteDAO* do pacote *com.mestrado.DAO* como o trecho de código à seguir exemplifica.

```
public class PacientesDao {

    private EntityManagerFactory emf = Persistence
        .createEntityManagerFactory("mestradoPU");
}
```

O parâmetro do método *createEntityManagerFactory* deve ser o mesmo nome que informamos no atributo name da tag *persistence-unit*, no arquivo *persistence.xml*.

Ao executar o código a tabela *pacientes* é criada com todos os parâmetros especificados, como mostra a figura 11.

Column Name	Datatype	PK	NN	UQ	B	UN	ZF	AI	G	Default/Expression
codPaciente	BIGINT(20)	☒	☒	☐	☐	☐	☐	☒	☐	
dtNascimento	DATETIME	☐	☐	☐	☐	☐	☐	☐	☐	NULL
nomePaciente	VARCHAR(255)	☐	☐	☐	☐	☐	☐	☐	☐	NULL
registro	VARCHAR(255)	☐	☐	☒	☐	☐	☐	☐	☐	NULL

Figura 11 - Tabela Pacientes

3.3.2 CRIANDO O *EntityManager*

O *EntityManager* é responsável por gerenciar entidades no contexto de persistência, onde através dos métodos desta interface é possível, persistir, realizar buscas e excluir objetos do banco de dados (FARIA; JÚNIOR, 2015).

O trecho de código abaixo exemplifica como o *EntityManager* é criado.

```
private EntityManager em = emf.createEntityManager();
```

3.3.3 PERSISTINDO OBJETOS

Neste capítulo será demonstrado como são inseridos os registros no banco de dados e para isso será usada a classe *PacienteDAO* do pacote *com.mestrado.DAO*, como mostra a figura 12.

```

1 package com.mestrado.Dao;
2
3 import java.util.Date;
4 import java.util.List;
5 import javax.persistence.EntityManager;
6 import javax.persistence.EntityManagerFactory;
7 import javax.persistence.Persistence;
8 import javax.persistence.Query;
9 import javax.swing.JOptionPane;
10
11 import com.mestrado.model.Paciente;
12
13 public class PacientesDao {
14
15     private EntityManagerFactory emf = Persistence
16         .createEntityManagerFactory("mestradoPU");
17     private EntityManager em = emf.createEntityManager();
18
19     public void salvar(String nome, String registro, Date dataNsc)
20         throws Exception {
21         Paciente paciente = new Paciente();
22
23         paciente.setNomePaciente(nome);
24         paciente.setregistro(registro);
25         paciente.setDtNascimento(dataNsc);
26
27         em.getTransaction().begin();
28         em.persist(paciente);
29         em.getTransaction().commit();
30         em.close();
31     }

```

Figura 12 - Método Salvar Paciente.

Na linha 15 da imagem é criado o *EntityManagerFactory*, e na linha 17 o *EntityManager*.

Na linha 19 começa o método salvar onde são passados como referência os dados que deverão ser inseridos no banco de dados.

Na linha 21 é criado um objeto do tipo *Paciente* para que possam ser definidos os valores para os atributos do paciente que se deseja salvar através dos métodos *set* já descrito anteriormente.

Na linha 27 o *EntityManager* inicia uma transação, na linha seguinte chama o método *persist* e passa o objeto paciente com os seus respectivos valores para que possa ser inserido no banco de dados. Na linha 29 é realizado o método *commit* responsável por efetivar a inserção do paciente no banco de dados e finalmente a transação é encerrada na linha 30 com o método *close()*. A figura 13 mostra alguns pacientes inseridos para efeito de teste.

	codPaciente	dtNascimento	nomePaciente	registro
▶	22	1980-02-02 00:00:00	TESTE20	15
	23	1980-02-21 00:00:00	JOSE	1258
	24	1980-02-01 00:00:00	AIRTON SILVA JR	9999A
★	NULL	NULL	NULL	NULL

Figura 13 - Registro Pacientes no Banco de dados

3.3.4 BUSCANDO OBJETOS PELO IDENTIFICADOR

Utilizando como base a mesma classe do capítulo anterior, é demonstrado como é possível recuperar objetos pelo identificador, isto é, pela chave primária, como demonstra a figura 14.

```

35     Paciente original = new Paciente();
36     original = em.find(Paciente.class, paciente.getCodPaciente());
37

```

Figura 14 - Busca por id.

Primeiro é criado um objeto *Paciente* com nome de original, e esse por sua vez, através do método *em.find()*, onde é especificado a classe e também o identificador ou chave primária do paciente que se deseja buscar.

Mais uma vez lembrando que foi realizado inserção e busca no banco de dados sem a necessidade do conhecimento de linguagem SQL, facilitando e agilizando o trabalho do desenvolvedor.

4 RESULTADOS

Foi desenvolvido um *software* para controle eletrônico das etapas de planejamento de pacientes para tratamento radioterápico.

A figura 15 apresenta a tela principal com menus para cadastros e relatórios referentes aos planejamentos.

Figura 15 - Tela Principal com menus

À seguir seguem as figuras das telas de cadastro, onde pode-se inserir novos registros ou alterar algum registro existente.

Figura 16 - Cadastro de Físicos

Cadastro de Médicos

Código:

Nome:

CRM:

Sigla ☐ **Ativo** ☐ **Inativo**

Figura 17 - Cadastro de Médicos

Cadastro Clínica de Origem

Código:

Descrição:

Sigla:

Figura 18 - Cadastro Clínica de origem

The screenshot shows a software window titled "Cadastro Região Anatômica". It features a standard Windows-style title bar with minimize, maximize, and close buttons. The main area contains two input fields: "Código:" with a small rectangular text box, and "Descrição:" with a larger, multi-line text area. Below these fields is a horizontal row of five icons: a blue floppy disk, a yellow pencil on a notepad, a magnifying glass, a yellow broom, and a brown door.

Figura 19 - Cadastro Região Anatômica

The screenshot shows a software window titled "Cadastro de Aparelho". It has a title bar with minimize, maximize, and close buttons. The interface is identical to the previous window, with "Código:" and "Descrição:" input fields and a row of five icons (floppy disk, pencil, magnifying glass, broom, and door) at the bottom.

Figura 20 - Cadastro de Aparelho

The screenshot shows a software window titled "Cadastro sistema de Gerenciamento". It features a title bar with minimize, maximize, and close buttons. The layout is consistent with the other two windows, including "Código:" and "Descrição:" input fields and a row of five icons (floppy disk, pencil, magnifying glass, broom, and door) at the bottom.

Figura 21 - Cadastro sistema de Gerenciamento

A figura 22 mostra a tela principal com pacientes fictícios cadastrados para demonstrar o sistema em funcionamento.

Data	Registro	Nome Paciente	Clinica Origem	Médico	Região Anat.	Data CT	Contorno	Alvo	Qtde Cpos	Técnica	Aparelho	Sist. Gerenc.	Aprovação	Impressão	1ª Ass.	2ª Ass.	Bloco	Data início	Observações	Fila
02/02/2017	111222K	JOAO DAS COUVES	CCP	FLAVIO	LARINGE	02/02/2017	SIM	NAO	5	3D	PRIMUS	LANTIS	NAO	NAO	LEANDRO	EDENYSE	5	15/02/2017	URGENTE	5
10/02/2017	222333A	MIKEY MOUSE	URO	FELIPE	PROSTATA	15/02/2017	SIM	SIM	5	3D	PRIMUS	LANTIS	SIM	SIM					URGENTE	3
15/02/2017	333444B	LUIZ FLORES	ORTOPE...	GUSTAVO	PERNA	17/02/2017	NAO	NAO					NAO	NAO	NAO	NAO				0

Figura 22 - Tela Principal em Funcionamento

Na figura 22 no menu Fila foi implementado um campo que verifica à quantos dias o paciente está em planejamento permitindo que se identifique com maior facilidade o paciente que está levando um tempo maior que o esperado no processo podendo assim melhorar o atendimento.

Foi implementado também um relatório que exibe a quantidade de pacientes planejados em um determinado período. Como mostra figura 23.

Pacientes Planejados - [] [X]

Quantidade de planejamento no período

Data Inicial: Data Final:

TOTAL: **Planejamentos**

Figura 23 - Relatório de planejamentos por Período

5 CONCLUSÃO

O sistema se mostrou satisfatório, já que é um avanço em relação ao atual modelo utilizado no controle de planejamento de pacientes no departamento de radioterapia, onde se utiliza um quadro físico e planilhas eletrônicas separadamente o que pode ocasionar duplicidade e inconsistência nos dados, o sistema utiliza o mapeamento objeto relacional para a persistência dos dados, a persistência de dados é sempre uma preocupação para o desenvolvedores, e a utilização do *JPA/Hibernate* se mostra muito útil ao desenvolvedor de software devido à abstração dos dados que proporciona, não sendo necessário conhecimento de linguagem SQL, apenas conhecer a biblioteca, sua estrutura e funcionamento unido a ferramentas visuais, consegue-se persistir os dados de maneira fácil segura e eficiente, pois não é complicado de ser implementado, claro que é necessário algum tempo para estudar as suas funcionalidades, mas devido sua abstração referente ao SQL se torna uma opção muito interessante para dar agilidade e desempenho no desenvolvimento de aplicativos, já que a redução na quantidade de linhas de código utilizando essa ferramenta é bem significativa em relação à utilização de outras formas de persistência

Outra vantagem é que da maneira que o sistema foi desenvolvido, a manutenção fica mais simples, onde permite que se mude as telas da aplicação sem alterar a base de dados e a base de dados sem interferir nas telas da aplicação. Do ponto de vista de usabilidade, foi desenvolvido da maneira mais próxima possível do que já estava sendo utilizado, o que implica em uma adaptação mais rápida ao sistema, podendo dessa forma contribuir para maior agilidade no planejamento de pacientes. O sistema da maneira que foi desenvolvido, permite que o sistema possa continuar em evolução, e possíveis trabalhos futuros novas funcionalidades possam ser acrescentadas podendo dessa forma contribuir para melhorias ainda maiores no que se refere ao controle de planejamentos de pacientes em radioterapia.

O término dessa dissertação não implica no encerramento do projeto, que pode receber melhorias até a data da defesa desse projeto.

BIBLIOGRAFIA

BARRY, Douglas K. **Object Relational Mapping Articles**. Disponível em:

<<http://www.service-architecture.com/articles/object-relational-mapping/>>. Acesso em: 16 jan. 2017.

COELHO, Camila Arnellas; SARTORELLI, Reinaldo Coelho. **PERSISTÊNCIA DE OBJETOS VIA MAPEAMENTO OBJETO-RELACIONAL**. 2004. 112 f. TCC (Graduação) - Curso de Sistemas de Informação, Faculdade de Computação e Informática da Universidade Mackenzie, São Paulo, 2004. Cap. 3. Disponível em: <http://www.dominiopublico.gov.br/pesquisa/DetalheObraForm.do?select_action=&co_obra=86596>. Acesso em: 10 Março 2017.

COSTA, F. **Celular é o principal meio de acesso a internet no Brasil**. Blog. Disponível em: <<http://saudebusiness.com/celular-e-o-principal-meio-de-acesso-a-internet-no-brasil>>. Acesso em: 7 abr. 2016.

DEITEL, Paul et. al. **Java Como Programar**, 8^o ed, vol. 1. Prentice Hall - 2010.

FARIA, Thiago; NORMANDES JÚNIOR,. **JPA e HIBERNATE**. São Paulo: Algaworks Softwares, Treinamentos e Serviços Ltda., 2015. 176 p

IBM DEVELOPERWORKS (Brasil) (Org.). **Introdução a plataforma**

Eclipse. 2012. Disponível em:

<<https://www.ibm.com/developerworks/br/library/os-eclipse-platform/>>. Acesso em: 28 jun. 2017.

KHAN, F. M. **The Physics of Radiation Therapy**. 3^a ed. Estados Unidos da América: Lippincott USA, 2003. v. 1

MEHTA, Vijay P. **Pro LINQ Object Relational Mapping with C# 2008**: Discover the power that LINQ to SQL and LINQ to Entities can bring to your projects. New York: Apress, 2008. 405 p

OLIVEIRA, Hébert Coelho de. **JPA Eficaz**: As melhores práticas de persistência de dados em Java. São Paulo: Casa do Código, [20--?]. 169 p.

PELIZZON, A. C. DE A. **Newsletter Atuação - A Importância da Radioterapia no Tratamento Oncológico**. Saúde. Disponível em: <<http://www.accamargo.org.br/newsletter-atuacao/a-importancia-da-radioterapia-no-tratamento-oncologico/505/>>. Acesso em: 4 jul. 2016.

PRESSMAN, Roger S.. **Software Engineering**: A practitioner's approach. 5. ed. New York: McGraw-hill Higher Education, 2001. 860 p. (ISBN 0-07-365578-3). Disponível em: <[http://www.resource.mitfiles.com/IT/II year/IV sem/Software Engineering/books/Pressman__Software_Engineering.pdf](http://www.resource.mitfiles.com/IT/II%20year/IV%20sem/Software%20Engineering/books/Pressman__Software_Engineering.pdf)>. Acesso em: 10 jun. 2017.

Radioterapia - INCA. Governamental - Saúde. Disponível em: <http://www.inca.gov.br/conteudo_view.asp?ID=100>. Acesso em: 4 jul. 2016.

SCAFF, L. **Física na Radioterapia - A base analógica em uma era digital**. São Paulo: Editora Projeto Saber, 2010. v. 2

SOUZA, Cadu. **ORM : Object Relational Mapper**. 2011. Disponível em:
<<http://www.devmedia.com.br/orm-object-relational-mapper/19056>>. Acesso em:
05 maio. 2017.

TECHNET MICROSOFT (Brasil) (Org.). **Conceitos de Orientação a**
Objetos. 2015. Disponível em: <<https://social.technet.microsoft.com/wiki/pt-br/contents/articles/9644.conceitos-de-orientacao-a-objetos.aspx>>. Acesso em: 01 jul.
2017.

APÊNDICE A

Controle eletrônico do processo de planejamento de pacientes em Radioterapia

Documento de Requisitos do Sistema

1 – INTRODUÇÃO

O sistema visa o controle do processo de planejamento de pacientes para tratamento de radioterapia, promovendo assim um controle mais rigoroso e servindo para consulta do histórico sobre planejamentos realizados e em andamento através de um sistema para computadores que utilizam sistema operacional *Windows 7* ou superior, *Linux* ou *Mac OS* que utilizam processador dual core ou superior e mínimo 2gb de memória *ram*.

Termo	Descrição
Requisitos Funcionais	Requisitos de software que compõem o sistema, descrevendo ações que o sistema deverá executar quando solicitado.
Requisitos não Funcionais	Requisitos de software que compõem o sistema, descrevendo atributos de qualidade que o sistema deve possuir, ou restrições que ele deve satisfazer.

Tabela 1 - Introdução

2 - REQUISITOS FUNCIONAIS (RF)

2.1 – Controle de acesso

Esta seção apresenta a descrição das funcionalidades de controle de acesso de usuários além das funcionalidades que sistema deve possuir.

RF01 – Cadastro de Físicos

O sistema deve permitir que se possa cadastrar físicos médicos.

O cadastro de Físicos médicos deve conter os campos Nome, registro da associação brasileira de física médica (ABFM) e sigla para utilização no planejamento de pacientes.

Prioridade:	X	Essencial		Importante		Desejável
-------------	---	-----------	--	------------	--	-----------

RF02 – Cadastro de Médicos

O sistema deve permitir que se possa cadastrar médicos, que serão responsáveis pelo paciente naquele planejamento.

O cadastro de médicos deve conter os campos Nome, conselho regional de medicina (CRM) e sigla para utilização no planejamento de pacientes.

Prioridade:	X	Essencial		Importante		Desejável
-------------	---	-----------	--	------------	--	-----------

RF03 – Cadastro de Região Anatômica

O sistema deve permitir que se possa cadastrar regiões anatômicas para identificar a região que será planejada para tratamento.

O cadastro de Região anatômica deve conter os campos código e descrição.

Prioridade:	X	Essencial		Importante		Desejável
-------------	---	-----------	--	------------	--	-----------

RF04 – Cadastro de Aparelho de tratamento

O sistema deve permitir que se possa cadastrar aparelhos de tratamento que serão utilizados no tratamento dos pacientes.

O cadastro de Aparelho deve conter os campos código e descrição.

Prioridade:	X	Essencial		Importante		Desejável
-------------	---	-----------	--	------------	--	-----------

RF05 – Cadastro de Clínica de Origem

O sistema deve permitir que se possa cadastrar clínicas de origem para identificar qual a clínica encaminhou o paciente para radioterapia.

O cadastro de clínicas de origem deve conter os campos código e descrição.

Prioridade:	X	Essencial		Importante		Desejável
-------------	---	-----------	--	------------	--	-----------

RF06 – Cadastro de Sistema de Gerenciamento

O sistema deve permitir que se possa cadastrar sistemas de gerenciamento para que possa ser identificado o sistema utilizado para planejamento do paciente.

O cadastro de sistema de planejamento deve conter os campos código e descrição.

Prioridade:	X	Essencial		Importante		Desejável
-------------	---	-----------	--	------------	--	-----------

RF07 – Cadastro de Pacientes

O sistema deve permitir que o usuário possa cadastrar pacientes que irão ser planejados para tratamento.

O cadastro de pacientes deve conter os campos Nome, Registro e data de nascimento.

Prioridade:	X	Essencial		Importante		Desejável
-------------	---	-----------	--	------------	--	-----------

RF08 – Cadastro de Planejamento

O sistema deve permitir que se possa cadastrar planejamentos de pacientes a serem tratados.

O cadastro de Planejamento deve conter os campos data de cadastro, registro HC do paciente, nome do paciente, clínica de origem, médico responsável, região anatômica, data da tomografia, contorno, alvo, quantidade de campos, técnica a ser utilizada, aparelho que será utilizado, Sistema de Gerenciamento, aprovação, impressão, primeira assinatura, segunda assinatura, quantidade de

blocos, data de início, campo observações, e fila que são os dias que o paciente está aguardando para início do tratamento.

Prioridade:	X	Essencial		Importante		Desejável
-------------	---	-----------	--	------------	--	-----------

RF08 – Alteração de cadastro

O sistema deve permitir que os cadastros possam ter seus dados alterados no sistema.

Prioridade:		Essencial		Importante	X	Desejável
-------------	--	-----------	--	------------	---	-----------

RF07 – Relatórios

O sistema deve permitir que o usuário faça busca de quantidade de pacientes planejados em determinado período.

O sistema deve indicar na tela principal quantos dias o paciente está aguardando para iniciar o tratamento.

O sistema deve permitir que o usuário faça busca de relatório que identifique etapa que levou mais do que 36 horas para ser finalizada.

O sistema deve identificar quantidade de planejamento realizado por médico.

O sistema deve identificar a quantidade de planejamentos liberados por físicos médicos.

Prioridade:		Essencial		Importante	X	Desejável
-------------	--	-----------	--	------------	---	-----------

3 - REQUISITOS NÃO FUNCIONAIS (RNF)

3.1 Requisitos do produto

Esta seção apresenta a descrição dos requisitos do sistema para prover conteúdo para os usuários do sistema.

RNF01 – Segurança

O sistema deve emitir mensagens de erro quando houver erros ou falhas, permitindo o usuário identificar o motivo do erro ou falha ocorrida e realizar outra operação (como reiniciar o sistema, dados inválidos ou fechar o sistema).

Prioridade:	X	Essencial		Importante		Desejável
-------------	---	-----------	--	------------	--	-----------

RNF02 – Usabilidade

O Sistema deve prover o usuário com interface simples e intuitiva facilitando a navegação e utilização do mesmo por parte dos usuários.

O sistema deve permitir que o usuário faça busca de pacientes e seus dados utilizando o registro ou o nome do paciente.

O sistema deve permitir que o usuário faça pesquisas sobre a quantidade de pacientes planejados em um determinado período.

O sistema deve permitir pesquisa sobre o tempo de espera entre o início da fase de planejamento e a liberação do paciente para tratamento.

O sistema também deve permitir a pesquisa sobre o tempo médio das etapas do processo de planejamento.

Prioridade:	X	Essencial		Importante	X	Desejável
-------------	---	-----------	--	------------	---	-----------

RNF04 – Linguagem de Programação adotada

A implementação do sistema deve utilizar a linguagem *Java* que devido sua arquitetura possibilita a utilização do sistema em vários sistemas operacionais e será utilizado a *IDE Eclipse*.

Deve utilizar *JPA* e *Hibernate* para persistência dos dados no banco de dados utilizado.

Prioridade:	X	Essencial		Importante		Desejável
-------------	---	-----------	--	------------	--	-----------

RNF05 – Interface Gráfica

O sistema deve fazer uso, exclusivamente, da língua portuguesa para todo o texto apresentado nos conteúdos disponibilizados pelo sistema que será executado em computadores que utilizam sistema operacional *Windows 7* ou superior, *Linux* ou *Mac OS X* ou superior.

Prioridade:	X	Essencial		Importante		Desejável
-------------	---	-----------	--	------------	--	-----------

RNF06 – Confiabilidade

O sistema deve ter capacidade para recuperar os dados perdidos da última operação que realizou em caso de falha.

No sistema, a tela inicial é exibida maximizada e com barra de menu na parte superior da janela.

Prioridade:	X	Essencial		Importante		Desejável
-------------	---	-----------	--	------------	--	-----------

RNF07 – Eficiência

O sistema deve ter um tempo de resposta de até 10 segundos (podendo variar devido a velocidade de processamento e quantidade de memória disponível).

Prioridade:		Essencial		Importante	X	Desejável
-------------	--	-----------	--	------------	---	-----------

RNF08 – Portabilidade

O sistema pode ser executado em computadores com sistema operacional *Windows 7* ou superior, *Linux* ou *Mac OS X* ou superior devido a portabilidade da linguagem utilizada.

Prioridade:	X	Essencial		Importante		Desejável
-------------	---	-----------	--	------------	--	-----------

RNF09 – Banco de Dados

O sistema deve ser capaz de armazenar os dados em base de dados *MySQL*.

Prioridade:	X	Essencial		Importante		Desejável
-------------	---	-----------	--	------------	--	-----------

Glossário

Termo	Descrição
Windows 7 ou sup.	Sistema operacional utilizado em computadores, microcomputadores e notebooks
Linux	Sistema operacional utilizado em computadores, microcomputadores e notebooks
MySQL	Sistema de banco de amplamente utilizado.
Usuário	Pessoa que manipula e utiliza o sistema.
IDE Eclipse	Software utilizado para desenvolvimento de software em várias linguagens como Java por exemplo.
Java	Linguagem de programação utilizada no desenvolvimento de sistemas computacionais.
JPA/Hibernate	Ferramenta utilizada para persistência de dados.

Tabela 2 - Glossário

APÊNDICE B

MODELO CONCEITUAL DO BANCO DE DADOS

Figura 24 - Modelo Conceitual Banco de Dados