UNIVERSIDADE FEDERAL DOS VALES DO JEQUITINHONHA E MUCURI
Bacharelado em Sistemas de Informação
Natália Caroline Ferreira Leal

SISTEMA DE PRESCRIÇÃO E CONTROLE DO TREINAMENTO INTERVALADO DE ALTA INTENSIDADE BASEADO NO BEEP TEST

Diamantina
2018

Natália Caroline Ferreira Leal

SISTEMA DE PRESCRIÇÃO E CONTROLE DO TREINAMENTO INTERVALADO DE ALTA INTENSIDADE BASEADO NO BEEP TEST

Trabalho de Conclusão de Curso apresentado ao curso de graduação em Sistemas de Informação, como parte dos requisitos exigidos para a obtenção do título de Bacharel em Sistemas de Informação.

Orientador: Alessandro Vivas Andrade

Diamantina
2018

Natália Caroline Ferreira Leal

SISTEMA DE PRESCRIÇÃO E CONTROLE DO TREINAMENTO INTERVALADO DE ALTA INTENSIDADE BASEADO NO BEEP TEST

Trabalho de Conclusão de Curso apresentado ao curso de graduação em Sistemas de Informação, como parte dos requisitos exigidos para a obtenção do título de Bacharel em Sistemas de Informação.

Orientador: Alessandro Vivas Andrade

Data de aprovação 03/08/18.

Dr. Alessandro Vivas Andrade
Faculdade de Ciências Exatas - UFVJM

Dr. Fernando Joaquim Gripp Lopes
Faculdade de Ciências Biológicas e da Saúde - UFVJM

Dra. Luciana Pereira de Assis
Faculdade de Ciências Exatas - UFVJM

Diamantina
2018

Dedico este trabalho àqueles que lutaram incondicionalmente para torná-lo possível:
Valdete, Damasceno e Marcus.

AGRADECIMENTOS

Primeiramente, agradeço a Deus. Confiar em Seus planos me trouxe até aqui.

Aos meus pais: Valdete e Damasceno, que acreditaram mais que todos na minha competência e que já sabiam, de alguma forma, desde o começo que eu era capaz. Aos meus irmãos: Marcus, Wellington e Tony, por serem meus melhores amigos e maiores exemplos de força, perseverança e bondade.

A Nícollas, que foi meu porto seguro em todos os momentos. Que incentivou quando faltou motivação e trouxe harmonia quando faltou equilíbrio. Que vibrou e comemorou todas as vitórias como se fossem dele. E são.

Aos bons amigos com quem a Universidade me presenteou: Nanda, Mayra, Liliane, Thalita, Lucão, Arthur , Izumi, Adolfo, Deco, Gabriel, Luiz. E àqueles de longa data: Maraísa, Mariana, Catarine, Maiara. Obrigada pelo apoio e por tornar essa caminhada tão agradável.

Aos professores do curso de Sistemas de Informação desta Universidade, que se importaram o suficiente para compartilhar de seu conhecimento científico e humano com todos os alunos que têm sede pelo saber. Em especial, ao meu orientador Alessandro Vivas, por ter me conduzido até o final com grande dedicação e zelo.

Ao professor Fernando Gripp, do departamento de Educação Física da UFVJM, por investir seu tempo e seus conhecimentos neste empreendimento e por acreditar no sucesso deste.

RESUMO

O objetivo deste trabalho é apresentar as etapas do desenvolvimento de um sistema de suporte à prescrição e controle do treinamento intervalado de alta intensidade (HIIT). A aplicação tem duas finalidades principais: ministrar o teste progressivo de corrida de vai-e-vem (Beep Test), para medir a aptidão física do avaliado, e determinar o treinamento individualizado, de acordo com o resultado obtido no teste. O processo de criação se baseou na metodologia ágil de desenvolvimento Extreme Programming, focando na comunicação constante com o cliente e no desenvolvimento incremental para alcançar um produto final de maior qualidade. O sistema resultante trata-se de um aporte significativo para o ambiente acadêmico da UFVJM. Ao automatizar a aplicação do teste intervalado e a prescrição do treinamento, o mesmo contribuirá para o desenvolvimento de pesquisas importantes relacionadas ao HITT. Além do mais, a ferramenta também possibilita a análise aprofundada de informações pertinentes à evolução de cada avaliado.

Palavras-chave: Desenvolvimento de Software. Extreme Programming. Metodologia Ágil. Beet Test. Treinamento Intervalado.

ABSTRACT

The purpose of this work is to present the development stages of a system to support and control High-Intensity Interval Training (HIIT). The software has two main goals: to provide the shuttle run test (Beep Test), in order to measure the evaluated's fitness, and to prescribe an individualized training, according to the Beep Test's result. The creation process was based on the agile methodology Extreme Programming, focusing on stable communication between client and developer, and on incremental development to achieve a high quality product. The resulting system is a significant contribution to the academic environment at UFVJM. By automating both the Beep Test and the HIIT, it will contribute to the development of relevant researches related to this subject. Besides, this tool also enables deep analyses of the information pertinent to the evolution of each evaluated.

Keywords: Software Development. Extreme Programming. Agile Methodology. Beep Test. Interval Training.

LISTA DE ILUSTRAÇÕES

Figura 1 – Camadas da Engenharia de Software 14
Figura 2 – O Modelo em Cascata 17
Figura 3 – O Modelo Incremental 18
Figura 4 – O Modelo Espiral 19
Figura 5 – O Processo da Extreme Programming 22
Figura 6 – Página Inicial 30
Figura 7 – Cadastro de Equipe 31
Figura 8 – Cadastro de Avaliado 31
Figura 9 – Novo Teste 32
Figura 10 – Teste Individual 33
Figura 11 – Teste de Equipe 34
Figura 12 – Abrir Teste 34
Figura 13 – Gerar Treinamento 35
Figura 14 – Criar Treinamento 36
Figura 15 – Abrir Treinamento 36
Figura 16 – Treinamento 37
Figura 17 – Relatórios do Avaliado 38
Figura 18 – Editar Avaliado 38
Figura 19 – Relatórios da Equipe 39
Figura 20 – Editar Equipe 39
Figura 21 – Detalhes do Avaliado 40
Figura 22 – Gráficos 41

LISTA DE ABREVIATURAS E SIGLAS

ACID - Atomicidade Consistência Isolamento Durabilidade

API - Application Programming Interfaces

CRUD - Create Read Update Delete

CSS - Cascading Style Sheets

ES - Engenharia de Software

FXML - Java FX Extensible Markup Language

HIIT - High-Intensity Interval Training

IDE - Integrated Development Environment

JDK - Java Development Kit

JRE - Java Runtime Environment

JVM – Java Virtual Machine

MVC - Modelo Visão Controlador

SQL - Structured Query Language

UFVJM - Universidade Federal dos Vales do Jequitinhonha e Mucuri

XP - Extreme Programming

SUMÁRIO

1 INTRODUÇÃO 11
1.1 Objetivo 12
1.1.1 Objetivo Geral 12
1.1.2 Objetivos Específicos 12

2 FUNDAMENTAÇÃO TEÓRICA 13
2.1 Software 13
2.2 Engenharia de Software 13
2.3 Processo de Software 14
2.4 Modelos de Processos de Software 15
2.4.1 Modelos Prescritivos 16
2.4.1.1 Modelo em Cascata 16
2.4.1.2 Modelo Incremental 17
2.4.1.3 Prototipagem 18
2.4.1.4 Espiral 18
2.4.1.5 Desenvolvimento Baseado em Componentes 19
2.4.2 Métodos Ágeis 20
2.4.2.1 Extreme Programming 20
2.4.2.2 SCRUM 22
2.5 Padrão de Arquitetura 23
2.6 Tecnologias Utilizadas 23
2.6.1 Linguagem de Programação 23
2.6.1.1 Java 23
2.6.1.2 JavaFX 24
2.6.2 Banco de Dados 25
2.6.2.1 SQLite 25
2.6.3 Ambiente de Desenvolvimento 25
2.6.3.1 NetBeans IDE 25
2.6.3.2 JavaFX Scene Builder 25
2.6.3.3 JFoenix 26

3 METODOLOGIA 27
3.1 Planejamento 27
3.2 Projeto 27
3.3 Codificação 28
3.4 Testes 28

4 SISTEMA DESENVOLVIDO 30
4.1 Menu Principal 30
4.2 Cadastro de Equipes e Cadastro de Avaliados 30
4.3 Novo Teste 32
4.4 Teste Individual 32
4.5 Teste de Equipe 33
4.6 Abrir Teste 33
4.7 Gerar Treinamento 33
4.8 Criar Novo Treinamento 35
4.9 Abrir Treinamento 36
4.10 Treinamento 37
4.11 Relatórios 37
4.11.1 Relatórios do Avaliado 37
4.11.2 Relatórios da Equipe 39
4.11.3 Detalhes do Avaliado 40
4.12 Gráficos 40

5 CONSIDERAÇÕES FINAIS 42

REFERÊNCIAS 44

Appendices 44

A DIAGRAMA DE CASOS DE USO DO SISTEMA 46

B DIAGRAMA DE ATIVIDADES: NOVO TESTE 47

C DIAGRAMA DE ATIVIDADES: NOVO TREINAMENTO 48

1 INTRODUÇÃO

O método conhecido como Treinamento Intervalado de Alta Intensidade (HIIT - High-Intensity Interval Training) é amplamente utilizado por treinadores com o objetivo de mensurar o condicionamento físico de seus avaliados e elaborar séries de exercícios personalizados e adequados a cada indivíduo. Segundo Gibala *et al.* (2012), HIIT descreve exercícios físicos caracterizados por intervalos breves e intermitentes de atividade intensa, interrompidos por períodos de descanso ou de exercícios de baixa intensidade.

Gibala *et al.* (2012) diz que o HIIT é infinitamente variável, parte desta grande diversidade surge devido aos diferentes protocolos de avaliação da capacidade aeróbica que podem ser utilizados para a composição do treinamento. Para o desenvolvimento do sistema apresentado neste trabalho, baseou-se no protocolo do Beep Test.

O protocolo tem como objetivo estimar o consumo máximo de oxigênio do avaliado. Para isso, ele deve correr em uma pista de 20 metros delimitada por cones com intensidade progressiva controlada por sinais sonoros. A cada sinal sonoro emitido pelo sistema de som o avaliado deve atingir um dos cones que delimitam o início ou o fim da pista. A velocidade inicial e de 8,5 km/h, sendo incrementada progressivamente em 0,5 km/h a cada estágio. O teste é encerrado pelo avaliador quando o avaliado não consegue alcançar um dos cones pela segunda vez consecutiva antes do respectivo beep sonoro (LOPES, 2017).

Para se desenvolver um treinamento baseado em HIIT, no entanto, é necessário fazer uso de equipamentos que auxiliem na aplicação do teste e do próprio treinamento, uma vez que é preciso acompanhar a quantidade de voltas concluídas, o nível de dificuldade e o tempo gasto para integralizar cada uma dessas voltas e níveis.

Diante desta necessidade, diversos aplicativos de gerência de Beep Test foram disponibilizados no mercado. Contudo, foi constatado que nenhum possui todas as funcionalidades esperadas por treinadores. Eles buscam um sistema que possa administrar o teste de forma integral e ainda oferecer a possibilidade de controlar o treinamento que pode ser gerado a partir de seu resultado. Em consequência, julgou-se necessário que fosse desenvolvido um sistema que suprisse essa carência dos demais softwares existentes.

O sistema desenvolvido deve possibilitar a prescrição e controle do treinamento intervalado. Para isso, ele deve ser capaz de aplicar o teste de aptidão, seja ele individual ou para uma equipe de avaliados previamente cadastrados. Deve também gerar um treinamento baseando-se no resultado obtido no teste ou ainda permitir que seja criado um treinamento personalizado com valores definidos pelo treinador. Além disso, o sistema deve cadastrar equipes e avaliados em um banco de dados. Ao término de cada teste, será possível salvar ou descartar seu resultado, a critério do treinador. Este resultado será relacionado aos respectivos avaliado e equipe.

Adicionalmente, o software oferecerá a opção de visualizar o desempenho individual

ou para uma equipe na forma de relatórios e gráficos. Ele consiste em um aplicativo para desktop e será compatível com diferentes Sistemas Operacionais.

Este sistema não fará conexão com a Internet e não possibilitará o compartilhamento das informações salvas em seu banco de dados com outras máquinas. Além do mais, o mesmo não será disponibilizado para diferentes dispositivos, sendo o foco principal deste trabalho que ele seja desenvolvido para plataformas desktop.

1.1 Objetivo

1.1.1 Objetivo Geral

Professores do departamento de Educação Física da Universidade Federal dos Vales do Jequitinhonha e Mucuri conduzem um projeto nomeado: Efeitos do Treinamento Intervalado De Alta Intensidade Na Saúde De Indivíduos Sedentários: Uma Proposta De Treinamento Físico Para A Polícia Militar De Minas Gerais.

Buscando uma forma mais eficiente de coleta de dados, eles propõem um software capaz de gerenciar de maneira integral os testes de aptidão baseados no Beep Test e que também possibilite conceber e conduzir rotinas de treinamento personalizadas para cada avaliado que realizar o teste, além de gerar relatórios de acompanhamento.

1.1.2 Objetivos Específicos

Os objetivos específicos do software desenvolvido neste trabalho são:

- Conduzir a aplicação de testes de aptidão individual ou de equipe;
- Gerar um treinamento a partir do resultado obtido em testes;
- Criar treinamento personalizado com valores definidos manualmente;
- Cadastrar avaliados e equipes no banco de dados do sistema;
- Salvar o resultado dos testes de aptidão no banco de dados;
- Salvar treinamentos gerados no banco de dados;
- Gerar relatórios e gráficos de acompanhamento de desempenho.

2 FUNDAMENTAÇÃO TEÓRICA

Neste capítulo são apresentados todos os conceitos necessários ao entendimento da evolução deste trabalho. Dentre eles, destacam-se as definições fundamentais da Engenharia de Software (ES), desde seus primórdios até o manifesto ágil. Uma forma de compreender as escolhas que concernem ao modelo de processo de software escolhido para o desenvolvimento deste projeto.

São especificadas também as tecnologias utilizadas para preparação do ambiente de desenvolvimento, bem como a Linguagem de Programação, o Sistema Gerenciador de Bancos de Dados e o Padrão de Arquitetura utilizado.

2.1 Software

Para Sommerville (2011), a associação do termo *software* a programas de computador é uma visão muito restritiva. O autor defende que software não é apenas um programa, mas também todos os dados de documentação e configuração associados, necessários para que o programa opere corretamente.

Seguindo este raciocínio, para este projeto, é importante visualizar o fato de que o aplicativo desenvolvido constitui apenas um dos artefatos resultantes do processo de software descrito neste documento. O software produzido, como um todo, é integrado pelos documentos e relatórios gerados, bem como por toda a configuração necessária ao funcionamento do aplicativo.

Neste contexto, é interessante ressaltar também que cada software se encaixa em uma categoria. Pressman (2007) aponta que hoje podemos identificar sete amplas categorias, que se dividem em: Software de Sistemas, Software de Aplicação, Software Científico e de Engenharia, Software Embutido, Software para Linhas de Produtos, Aplicações da Web e Software para Inteligência Artificial.

O software desenvolvido está enquadrado na categoria de Software de Aplicação. Pressman (2007) explica que este tipo consiste de programas isolados que resolvem uma necessidade específica. Ele ainda diz que aplicações nessa área processam dados comerciais ou técnicos de um modo que facilita as operações ou tomada de decisões técnicas.

A criação de um software, no entanto, não é uma tarefa simples ou superficial. Para se chegar a um produto de qualidade, é preciso estudar as técnicas e metodologias pré-estabelecidas e observar quais padrões desenvolvidos ao longo do tempo mais se adéquam à necessidade de cada projeto.

2.2 Engenharia de Software

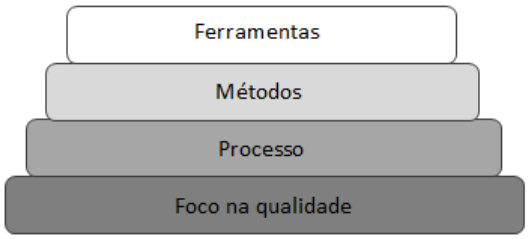
A engenharia de software, segundo Pressman (2007), abrange um processo, um conjunto de métodos e um leque de ferramentas que possibilitam aos profissionais desenvolverem

software de altíssima qualidade. A área busca a padronização das formas de se construir um software.

Quando se fala em padronização, entende-se que, ao buscar uma forma estandardizada de se fazer algo, tais padrões funcionem como pilares para a construção de um produto de qualidade. Este é o foco da ES. Enquanto executa todos os passos previstos em um processo de software adequado ao projeto, utilizando métodos e ferramentas inerentes a esse processo, espera-se que a qualidade seja uma consequência.

Pressman (2007) identifica a ES como uma tecnologia em camadas, como mostrado na figura 1, cada uma definida da seguinte maneira:

Figura 1 – Camadas da Engenharia de Software



Fonte: Pressman (2007). Adaptado.

- A base é o *foco na qualidade*. Todo o processo deve se apoiar num compromisso organizacional com a qualidade.
- O alicerce é a camada de *processo*. Os processos de software formam a base para o controle gerencial de projetos de software e estabelecem o contexto no qual os métodos são aplicados, os produtos de trabalho são produzidos, os marcos são estabelecidos, a qualidade é assegurada e as modificações são adequadamente geridas.
- Os *métodos* de ES fornecem a técnica de "como fazer" para construir softwares. Eles abrangem um amplo conjunto de tarefas e incluem atividades de modelagem e outras técnicas descritivas.
- As *ferramentas* de ES fornecem apoio automatizado ou semi-automatizado para o processo e para os métodos.

2.3 Processo de Software

Sommerville (2011) fala que a engenharia procura selecionar o método mais apropriado para um conjunto de circunstâncias. Esse método pode ser tanto uma abordagem sistemática e organizada, quanto uma abordagem mais criativa e menos formal. A eficácia do método escolhido vai depender das características ligadas ao problema a ser solucionado. Entretanto, existem

atividades fundamentais do processo que se aplicam à maioria dos tipos de projetos de software. Elas são chamadas de *atividades de arcabouço*.

As atividades de arcabouço são a base dos chamados modelos de processos de software, vistos mais adiante, e estão presentes na concepção de todos os tipos de softwares, sejam estes simples ou complexos. Um *arcabouço de processo genérico* é apresentado por Pressman (2007) na forma de cinco etapas:

- *Comunicação*: Envolve comunicação e colaboração com o cliente e outros interessados e abrange o levantamento de requisitos e outras atividades relacionadas.
- *Planejamento*: Estabelece um plano para o trabalho, descreve tarefas técnicas a ser conduzidas, os riscos prováveis, os recursos que serão necessários, os produtos de trabalho a ser produzidos e um cronograma do trabalho.
- *Modelagem*: Inclui a criação de modelos que permitam ao desenvolvedor e ao cliente, entender melhor os requisitos do software e o projeto que vai satisfazer a esses requisitos.
- *Construção*: Combina geração de código e os testes necessários para revelar erros no código.
- *Implantação*: Entrega do software ao cliente, que avalia o produto e fornece feedback com base na avaliação.

Pressman (2007) ainda destaca que além das genéricas, algumas atividades de suporte, conhecidas como atividades guarda-chuva, estão presentes durante todo o processo de software. Dentro de cada atividade genérica estão contidas as tarefas que definem detalhadamente o que deve ser feito. Estas tarefas podem ser modificadas, de forma a se adaptar às particularidades do software a ser desenvolvido e também da equipe e do ambiente no qual ele será desenvolvido.

Diferentes tipos de sistemas necessitam de diferentes processos de desenvolvimento. Consequentemente, atividades genéricas podem ser organizadas de diferentes maneiras e descritas em níveis diferentes de detalhes, para diferentes tipos de software. Porém, o uso de um processo de software inadequado pode reduzir a qualidade ou a utilidade do produto de software a ser desenvolvido e/ou aumentar os custos do desenvolvimento (SOMMERVILLE, 2011).

2.4 Modelos de Processos de Software

Sommerville (2011) define um processo de software como um conjunto de atividades que leva à produção de um produto de software. Como citado anteriormente, algumas atividades genéricas de processos podem ser aplicadas ao desenvolvimento de praticamente qualquer tipo de software.

Embora o arcabouço de processo genérico apresente um bom guia para o desenvolvimento de software, a escolha de quais atividades devem ser incluídas no trabalho e como

essas atividades funcionam em conjunto ainda é uma tarefa crítica. Se para cada projeto fosse preciso construir um processo de software completamente novo, as empresas perderiam um tempo enorme com o planejamento do processo e com o treinamento de equipes.

Para contornar estes obstáculos, existem representações abstratas e pré-determinadas de um processo de software, os chamados *modelos de processos de software*. Cada modelo apresenta uma solução básica para questões de ES, com diferentes enfoques. Para Sommerville (2011), eles são uma descrição simplificada do processo de software. Incluem as atividades, que fazem parte do processo de software, os produtos de software e os papéis das pessoas envolvidas na ES.

No entanto, ainda que funcionem como facilitadores, os modelos não podem ser percebidos como uma solução absoluta. Pressman (2007) salienta que a aplicação inteligente de qualquer modelo de processo de software deve reconhecer que a adaptação, seja ela ao projeto, à equipe e à cultura organizacional, é essencial para o sucesso. Sommerville (2011) também ressalta que eles podem ser considerados frameworks de processo que podem ser ampliadas e adaptadas para criar processos mais específicos de ES. O autor ainda diz que tais modelos não são mutuamente exclusivos e, na prática, são frequentemente combinados. A seguir, veremos alguns dos modelos mais comuns.

2.4.1 Modelos Prescritivos

Os modelos prescritivos de processos de software têm esse nome porque prescrevem um conjunto de elementos de processo e também um fluxo de trabalho. Quando os modelos prescritivos são aplicados, o objetivo é melhorar a qualidade do sistema para tornar os projetos mais gerenciáveis, as datas de entrega e os custos mais previsíveis e para guiar equipes de engenheiros de software à medida que eles realizam o trabalho necessário para construir um sistema (PRESSMAN, 2007).

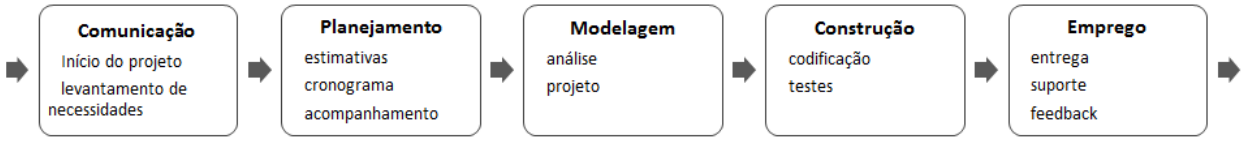
Também chamados de modelos clássicos, eles são reconhecidos por tratar a produção de software de maneira mais formal, proporcionando um maior controle sobre as atividades de processo. Valorizam a geração de documentação e dão grande ênfase ao estabelecimento de etapas bem descritas, divididas e previsíveis.

2.4.1.1 Modelo em Cascata

Proposto originalmente por Royce (1970), este paradigma sugere um fluxo linear do processo de software, onde cada etapa é executada e finalizada antes que a próxima se inicie. Também é possível iterar para as etapas anteriores sempre que necessário. Os estágios do modelo são apresentados na figura 2.

O modelo em cascata é adequado para o desenvolvimento de softwares em que já se tem um entendimento suficiente acerca de todos requisitos, que devem ser bem definidos e estáveis. Além disso, ele requer o planejamento impecável, já na fase inicial, de cada detalhe que concerne todo o processo.

Figura 2 – O Modelo em Cascata



Fonte: Pressman (2007).

Apesar de ser o paradigma mais antigo da ES, ele apresenta problemas devido à sua estrutura rígida. Em primeiro lugar, observa-se que em ambientes reais de desenvolvimento, é improvável que todas as particularidades possam ser previstas na fase inicial. Os requisitos levantados, por exemplo, podem sofrer modificações para atender a pedidos do cliente ou por necessidades relacionadas ao projeto. Além do mais, devido aos custos de produção e aprovação de documentos, as iterações são onerosas e envolvem um retrabalho significativo (SOMMERVILLE, 2011).

Por outro lado, as modificações podem causar confusão à medida que a equipe de projeto prossegue (PRESSMAN, 2007). Durante a fase final do ciclo de vida (operação e manutenção), o software é colocado em uso. Erros e omissões nos requisitos originais de software são descobertos. Os erros de programação e projeto emergem e a necessidade de novas funcionalidades é identificada. Essas mudanças podem implicar repetição de estágios anteriores do processo (SOMMERVILLE, 2011).

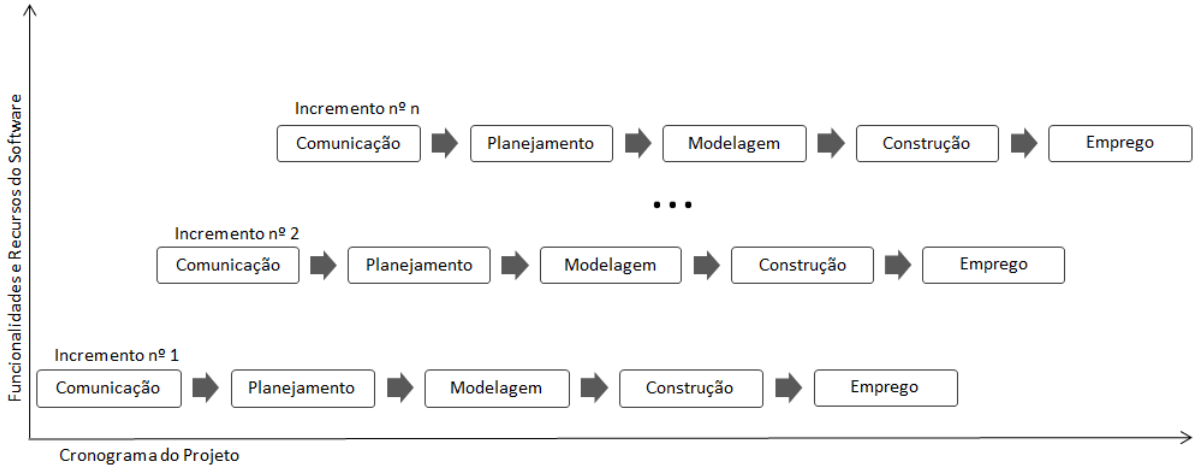
2.4.1.2 Modelo Incremental

O modelo incremental é uma espécie de evolução do modelo em cascata. Nele, o modelo em cascata é utilizado iterativamente para o desenvolvimento de pequenos incrementos de software. Cada incremento consiste de uma versão executável do sistema com uma nova funcionalidade. O modelo é representado na figura 3.

Utilizando o modelo em cascata de forma iterativa, os problemas identificados anteriormente são consideravelmente reduzidos. Para o primeiro incremento do software (chamado de núcleo do produto), os requisitos básicos de maior prioridade são implementados e apresentados ao cliente. A partir daí novos requisitos podem surgir e ser incrementados aos últimos em cada iteração, enquanto o produto evolui até sua versão final.

Entretanto, de acordo com Sommerville (2011), existem problemas com a entrega incremental. O autor destaca que, como os incrementos devem ser relativamente pequenos, pode ser difícil mapear os requisitos do cliente em incrementos de tamanho adequado. Além disso, o fato de que os requisitos não são definidos detalhadamente até que um incremento seja implementado, pode dificultar a identificação dos recursos comuns exigidos por todos os incrementos.

Figura 3 – O Modelo Incremental



Fonte: Pressman (2007). Adaptado.

2.4.1.3 Prototipagem

O paradigma de prototipagem propõe a criação rápida e barata de modelos que representem razoavelmente as funcionalidades e a interface com o usuário do que deveria ser o software real. Tais modelos podem ser uma amostra executável do software ou uma representação gráfica dos requisitos.

Um protótipo é uma versão inicial de um sistema de software usado para demonstrar conceitos, experimentar opções de projeto e, geralmente, conhecer mais sobre o problema e suas possíveis soluções (SOMMERVILLE, 2011).

A criação de protótipos apresenta diversas vantagens. Para Pressman (2007), ela auxilia o engenheiro de software e o cliente a entenderem melhor o que deve ser construído quando os requisitos estão confusos. Serve também como um mecanismo para identificação dos requisitos do software. Sommerville (2011) aponta também que um protótipo pode ajudar a encontrar os pontos fortes e fracos no software, além de permitir verificar a viabilidade de um projeto proposto.

É importante ressaltar que todos os protótipos desenvolvidos devem ser completamente descartados, uma questão de controle de qualidade. Como são implementados de forma rápida, os programadores geralmente não se importam com questões técnicas de qualidade de código, segurança do software, reusabilidade, entre outros fatores. A utilização de um protótipo como um incremento de software pode levar a altos custos de correção de erro ou de reprojeto futuramente.

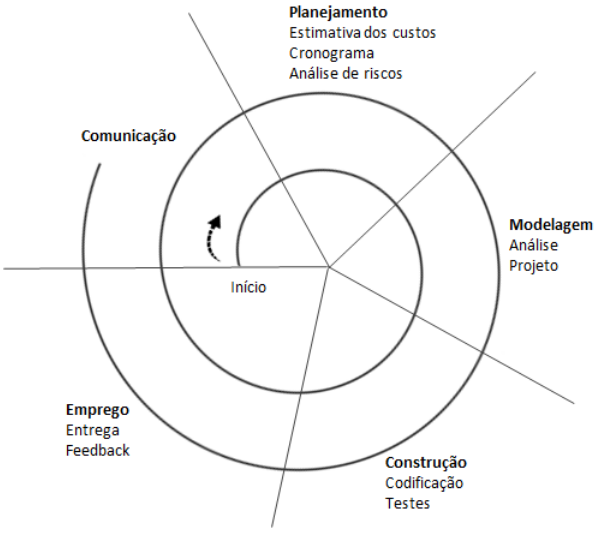
2.4.1.4 Espiral

O modelo em espiral foi proposto por Boehm (1988). O autor apontou duas características principais do modelo: a primeira é uma abordagem cíclica, para aumentar incremental-

mente o grau de definição e implementação de um sistema enquanto diminui seu grau de risco. A outra é um conjunto de marcos de ancoragem, para garantir o comprometimento dos interessados com soluções exequíveis e mutuamente satisfatórias para o sistema.

A ideia deste modelo é reunir as características da prototipagem e do modelo em cascata e aplicá-las iterativamente ao mesmo tempo em que uma análise de riscos é feita. A cada iteração, o produto é evoluído e os riscos são analisados. Dessa forma, é possível conhecer e lidar com quaisquer ameaças gradativamente, antes que elas se transformem em um problema grande e custoso.

Figura 4 – O Modelo Espiral



Fonte: Pressman (2007).

Exemplificando o uso do modelo espiral em um processo de software, Pressman (2007) diz que o primeiro circuito em torno da espiral poderia resultar no desenvolvimento da especificação de um produto; passagens subsequentes em torno da espiral poderiam ser usadas para desenvolver um protótipo e depois, progressivamente, versões mais sofisticadas do software. Cada passagem pela região de planejamento resulta em ajustes do plano do projeto. A figura 4 ilustra o modelo espiral típico.

2.4.1.5 Desenvolvimento Baseado em Componentes

Este modelo de processo utiliza componentes pré-fabricados para integrar o software a ser desenvolvido. Cada componente implementa uma funcionalidade completa, que será incorporada ao sistema a partir de sua interface bem definida. Segundo Pressman (2007), o modelo incorpora muitas das características do modelo espiral e demanda uma abordagem iterativa para a criação de softwares.

Sommerville (2011) explica o desenvolvimento baseado em componentes com base em estágios: após a especificação de requisitos, é feita uma busca pelos componentes que podem

ser usados. Depois, os requisitos são modificados para refletir os componentes disponíveis. Em seguida, é feito o projeto de sistema com o reuso, onde o framework do sistema é projetado. Por fim, o software que não pode ser adquirido externamente é desenvolvido e os componentes são integrados para criar o novo sistema.

Entre as vantagens deste modelo, estão a redução da quantidade de código produzido; a diminuição dos riscos, uma vez que componentes reutilizáveis já foram testados; e a entrega mais rápida do software pronto. Por outro lado, pode ser que os requisitos iniciais não possam ser atendidos simplesmente por não existir componentes que os implementem.

2.4.2 Métodos Ágeis

Os métodos ágeis de desenvolvimento de software surgiram como uma alternativa às dificuldades dos modelos convencionais. Em 2001, Kent Beck e outros desenvolvedores, membros da chamada Aliança Ágil, criaram o Manifesto para o Desenvolvimento Ágil de Software. Nele, foram apresentados uma série de princípios da metodologia, ressaltando seus pontos essenciais. Dentre eles se destacam: envolvimento com o cliente, entrega incremental, foco em pessoas ao invés de processos, aceitação de mudanças e simplicidade.

Geralmente, os métodos ágeis contam com uma abordagem iterativa para especificação, desenvolvimento e entrega de software, e foram criados principalmente para apoiar o desenvolvimento de aplicações de negócios nas quais os requisitos de sistema mudam rapidamente durante o processo de desenvolvimento (SOMMERVILLE, 2011).

Entretanto, alguns problemas podem ser percebidos a partir dos princípios da metodologia. Sommerville (2011) aponta alguns: primeiramente, o envolvimento do cliente torna todo o processo dependente da disposição e do conhecimento que este cliente tem acerca do software a ser criado. A aceitação de mudanças também pode ser um gargalo, uma vez que implementação de mudanças pode ser difícil e ter um alto custo, dependendo de quão adiantado está o desenvolvimento. Por fim, a manutenção da simplicidade do software pode envolver trabalho extra.

DeMarco e Boehm (2002) defendem que uma abordagem híbrida na qual os métodos ágeis incorporam algumas técnicas de desenvolvimento baseado em planos pode ser a melhor solução.

2.4.2.1 Extreme Programming

Para Sommerville (2011), o Extreme Programming (XP) pode ser a metodologia ágil de desenvolvimento mais conhecida. O autor explica que seu nome foi cunhado por Beck (1999) e denota o significado da abordagem, que busca incentivar boas práticas a níveis ‘extremos’. Sua proposta é o desenvolvimento incremental, com pequenas entregas.

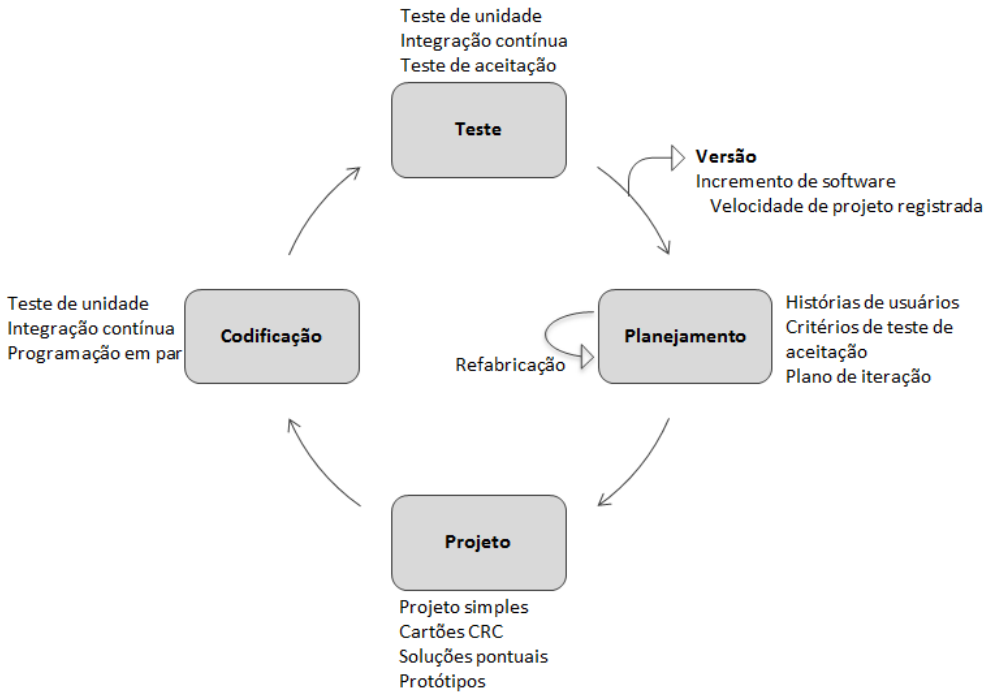
As práticas do XP, como colocado por Sommerville (2011), envolvem:

- *Planejamento Incremental*: Os requisitos são especificados pelo cliente como cenários, que são transformados em um 'cartão de história', contendo as suas necessidades. O cliente então aponta os de maior prioridade e os desenvolvedores os transformam em 'tarefas'. Caso surjam mudanças, novos cartões de histórias são desenvolvidos.
- *Pequenas Entregas*: Um conjunto de funcionalidades prioritárias são desenvolvidas primeiro. Depois disso, releases frequentes adicionam funcionalidades incrementalmente ao sistema.
- *Projeto Simples*: O projeto é feito apenas para atender aos requisitos atuais.
- *Desenvolvimento Orientado por Testes*: Testes unitários são escritos para uma nova funcionalidade antes que esta seja implementada.
- *Refatoração*: Desenvolvedores reescrevem o código continuamente para torna-lo simples e fácil de manter.
- *Programação em Pares*: Desenvolvedores trabalham em pares, um verificando o trabalho do outro e fornecendo apoio para realizar um bom trabalho.
- *Propriedade Coletiva*: Devido à programação em pares, não são formadas ilhas de conhecimento, todos os desenvolvedores têm conhecimento de todo o código.
- *Integração Contínua*: Depois de desenvolver uma nova tarefa, esta é integrada ao sistema como um todo e os testes unitários do sistema são realizados.
- *Ritmo Sustentável*: Horas extras são evitadas, pois, no médio prazo, há uma redução na qualidade do código e na produtividade.
- *Cliente on-site*: Um representante do cliente trabalha como um membro da equipe de desenvolvimento e deve estar disponível em tempo integral.

As maiores vantagens do XP estão relacionadas à sua capacidade de promover um maior entendimento dos requisitos. Isso se dá por meio do desenvolvimento orientado por testes, uma vez que, ao escrever testes para uma determinada tarefa, o programador pode visualizar melhor o que precisa ser feito para que os testes sejam bem-sucedidos. As pequenas entregas, o projeto simples e a programação em pares também contribuem para a redução de erros no projeto.

Por outro lado, o método também apresenta desvantagens. Ao promover o planejamento incremental, deixando o projeto aberto a mudanças não previstas, pode causar a degradação da estrutura do software, fazendo com que novas mudanças tornem-se cada vez mais difíceis de implementar (SOMMERVILLE, 2011).

Figura 5 – O Processo da Extreme Programming



Fonte: Pressman (2007).

2.4.2.2 SCRUM

O modelo Scrum foi desenvolvido por Jeff Sutherland e por sua equipe no início da década de 1990 e foi aprimorado posteriormente por Schwaber e Beedle (2002). Tal qual o manifesto ágil em si, o Scrum prioriza equipes pequenas, processo adaptável a mudanças e pequenos e frequentes incrementos de software.

De acordo com Pressman (2007), as atividades básicas de requisitos, análise, projeto, evolução e entrega são realizadas com o auxílio de um conjunto de padrões de processo de software que se mostram efetivos para projetos com prazos apertados, requisitos mutantes e criticalidade de negócio.

O autor explica cada padrão da seguinte forma:

- *Pendência*: Uma lista priorizada de requisitos ou características do projeto que fornecem maior valor de negócio para o cliente. Itens podem ser adicionados a qualquer momento
- *Sprints*: Consiste de unidades de trabalho que são necessárias para satisfazer a um requisito definido na pendência que precisa ser cumprido em um intervalo de tempo definido.
- *Reuniões Scrum*: São reuniões curtas (normalmente de 15 minutos) feitas diariamente pela equipe. Têm foco em três perguntas:
  - O que você fez desde a última reunião de equipe?
  - Que obstáculos você está encontrando?

  - O que você planeja realizar até a próxima reunião?

  Um líder, chamado de *Scrum Master* lidera a reunião e avalia as respostas de cada pessoa.

- *Demos*: Entrega o incremento de software ao cliente de modo que a funcionalidade implementada possa ser demonstrada e avaliada pelo cliente. Uma demo talvez não contenha toda a funcionalidade planejada, mas sim as funções que podem ser entregues dentro do intervalo de tempo estabelecido.

2.5 Padrão de Arquitetura

Segundo Deitel e Deitel (2010), os padrões de arquitetura incentivam o baixo acoplamento entre subsistemas. Tendo em vista a produção de código mais organizado, a redução da complexidade e ainda a possibilidade de reutilização, julgou-se necessário aplicar um padrão de arquitetura que auxiliasse no cumprimento de tais objetivos. O padrão escolhido foi o MVC: Modelo-Visão-Controlador.

Para Sommerville (2011), a maior motivação por trás da abordagem MVC é o desejo de separar o código que cria e manipula os dados do código que apresenta estes dados. Além disso, outra vantagem do padrão citada por Deitel e Deitel (2010) é a possibilidade de se modificar componentes individualmente, sem que isso reflita nos demais.

Baseando-se no que foi explicado por Sommerville (2011) sobre o framework MVC, no componente Modelo encontram-se as classes responsáveis pela gerência de dados persistentes e pelas regras de negócio. No componente Visão, estão todos os arquivos FXML (XML com alguns elementos próprios do JavaFX), que contêm o código gerado pela ferramenta de criação de interface com o usuário JavaFX Scene Builder. Por fim, no componente Controlador, as interações do usuário são gerenciadas e passadas para a Visão e o Modelo.

2.6 Tecnologias Utilizadas

2.6.1 Linguagem de Programação

2.6.1.1 Java

O Java é uma linguagem de programação orientada a objetos que surgiu como resultado de uma pesquisa corporativa interna da Sun Microsystems em 1991. Em 2010, a empresa foi adquirida pela Oracle, que continuou a dar suporte à linguagem.

O maior diferencial do Java, é o fato de ser multiplataforma. Isso significa que, independente do sistema operacional e das configurações de hardware de um computador ou dispositivo, um software desenvolvido em Java funcionará sem problemas. Isso é possível graças à Máquina Virtual Java (*JVM – Java Virtual Machine*).

De acordo com Deitel e Deitel (2010), a JVM é um aplicativo de software que simula um computador, mas oculta o sistema operacional e o hardware subjacentes dos programas que

interagem com ela. Se a mesma máquina virtual é implementada em muitas plataformas de computador, os aplicativos escritos para ela podem ser utilizados em todas essas plataformas. Para utilizar a JVM, basta instalar o Ambiente de Execução Java (*JRE - Java Runtime Environment*) no computador.

Além da portabilidade, outro benefício da linguagem diz respeito à grande quantidade de classes e métodos previamente implementados nas bibliotecas de classe Java, também conhecidas como Java APIs (Application Programming Interfaces, ou Interfaces de Programação de Aplicativos). Estas APIs facilitam o processo de desenvolvimento de software ao oferecer soluções prontas para diversos problemas triviais e recorrentes. Uma API de grande utilidade para este projeto é a API de interface gráfica JavaFX, que será explicada a seguir.

2.6.1.2 JavaFX

O JavaFX, a princípio, era um projeto de um desenvolvedor chamado Chris Oliver, mas foi adquirido pela Sun Microsystems. Inicialmente, se chamava JavaFX Script e sua primeira versão foi anunciada em 2007. Em 2010, a empresa decidiu descontinuar a linguagem. Entretanto, após a aquisição da Sun Microsystems pela Oracle, uma versão 2.0 foi preparada e lançada em 2011.

Como descrito pela Oracle (2014), o JavaFX é um conjunto de pacotes gráficos e de mídia que permite aos desenvolvedores projetar, criar, testar, depurar e implementar aplicações de *rich client* que operam de forma consistente em diversas plataformas.

Um dos maiores benefícios do JavaFX é o fato de ser uma biblioteca escrita como uma Java API. Dessa forma, o código produzido em uma aplicação JavaFX é, na prática, escrito em Java. Isso significa que, de acordo com a Oracle (2014), é possível fazer uso de todas Java APIs disponíveis para a linguagem. Além disso, qualquer programador Java pode construir aplicações JavaFX.

Outros privilégios apontados pela Oracle (2014) incluem a possibilidade de customizar a interface com o usuário com folhas de estilo (CSS - Cascading Style Sheets), separando a implementação da aparência e estilo. A organização do código também apresenta mudanças positivas, pois é possível usar a linguagem de script FXML para desenvolver os aspectos da interface, enquanto Java é empregado apenas para a lógica da aplicação. FXML consiste em um XML (Extensible Markup Language) com elementos do JavaFX

Segundo Lowe (2015), o JavaFX também se destaca de várias formas sobre o Swing, que foi a primeira API de interfaces gráficas do Java. Além do CSS, outras capacidades importantes são: a grande variedade de efeitos visuais que podem ser adicionados aos elementos de interface, os efeitos de animação, os gráficos, os objetos tridimensionais e até a possibilidade de lidar com dispositivos *touchscreen*. De fato, acredita-se que o Swing tende a se tornar obsoleto e que o JavaFX será seu substituto.

Assim como o Java, qualquer aplicação desenvolvida em JavaFX é multiplataforma e funciona em qualquer ambiente com o JRE instalado.

2.6.2 Banco de Dados

2.6.2.1 SQLite

O SQLite é uma biblioteca que implementa um banco de dados SQL transacional autônomo. Seu projeto foi iniciado em 2000, é de código aberto e conta com uma equipe profissional de desenvolvedores e de suporte. As palavras usadas por seus desenvolvedores para descrevê-lo são: pequeno, rápido e confiável.

O mecanismo de banco de dados do SQLite é incorporado, ou seja, ele não faz uso de servidores externos. Pelo contrário, os dados são lidos e gravados em arquivos comuns de disco. De acordo com sua documentação, as ações de leitura e escrita de pequenas quantidades de dados em um disco rígido são aproximadamente 35% mais rápidas do que um Sistema de Arquivos local. As transações também asseguram as propriedades ACID (Atomicidade, Consistência, Isolamento e Durabilidade) (SQLITE, 2017).

Por ser um banco de dados incorporado, o SQLite funciona muito bem para aplicações desktop que guardam informações em disco. Existem muitos benefícios para esta abordagem, incluindo melhor desempenho das aplicações, redução de custos e complexidade e maior confiabilidade (SQLITE, 2017).

2.6.3 Ambiente de Desenvolvimento

2.6.3.1 NetBeans IDE

O NetBeans é um ambiente de desenvolvimento integrado (IDE - Integrated Development Environment) de código aberto, desenvolvido pela Sun Microsystems e adquirido posteriormente pela Oracle. A ferramenta oferece diferentes plug-ins de suporte à construção de softwares em diversas linguagens de programação, tais como: Java Standard Edition, Java Enterprise Edition, HTML 5, JavaScript, PHP, C e C++.

Para desenvolvimento Java, é necessário que o computador tenha instalado o Pacote de Desenvolvimento Java (*JDK - Java Development Kit*). De acordo com a Oracle (2017), o JDK consiste de um ambiente de desenvolvimento para criação de aplicativos, applets e componentes. Inclui ferramentas úteis para desenvolver e testar programas escritos na linguagem de programação Java e executados na plataforma Java. Inclui também ferramentas JRE. Os plug-ins disponíveis no NetBeans fornecem o pacote JDK, dispensando a necessidade de buscar por uma cópia externa do mesmo.

2.6.3.2 JavaFX Scene Builder

O JavaFX Scene Builder é uma ferramenta de desenvolvimento que possibilita a criação de interfaces com o usuário sem que seja necessário escrever código. A ferramenta foi criada especificamente para trabalhar em conjunto com a API JavaFX e pode ser integrada com a IDE NetBeans.

À medida em que a interface é desenhada “arrastando e soltando” componentes na área de trabalho, o FXML do layout é gerado automaticamente. Este código pode ser incorporado em uma IDE onde o desenvolvedor pode adicionar a lógica de negócio.

Como exposto pela Oracle (2014), além das funções de arrastar e soltar componentes, a ferramenta também permite modificar as propriedades desses componentes, adicionar estilo com um documento CSS ou no próprio FXML e ainda integrar o código resultante com o código da aplicação.

2.6.3.3 JFoenix

JFoenix é uma biblioteca Java de código aberto, que implementa o sistema de design da Google, Material Design (JFOENIX, 2017). A biblioteca oferece uma série de componentes JavaFX estilizados de maneira harmoniosa e elegante. Os mesmos podem ser acrescentados ao projeto via código ou por meio do Scene Builder.

3 METODOLOGIA

O processo de construção deste produto se baseou na metodologia XP, com mais foco na implementação dos incrementos do que na documentação formal dos mesmos. Entretanto, para um melhor entendimento de todo o processo de construção do software, fez-se uso também de alguns dos artefatos característicos dos modelos prescritivos.

Seguindo as especificações do XP, o desenvolvimento foi dividido entre as etapas de Planejamento, Projeto, Codificação e Testes. Antes de tudo, foi realizado o Planejamento de todo o software e, posteriormente, para cada funcionalidade a ser implementada, foram cumpridas as demais etapas, com documentação sucinta para cada uma delas.

3.1 Planejamento

O Planejamento do sistema foi dividido em quatro momentos: levantamento de requisitos, investigação de softwares similares disponíveis no mercado, estudo da viabilidade e organização das etapas de implementação. O levantamento de requisitos envolveu entrevistas com o cliente, onde foi formalizada a relação de todos os requisitos funcionais solicitados.

A investigação de softwares existentes se deu por meio de pesquisa e exploração dos mesmos. Ela buscou facilitar a compreensão plena do que foi solicitado, uma vez que forneceu uma visão abrangente do propósito do software em questão.

Com a relação de requisitos em mãos, o próximo passo foi o estudo da viabilidade de se implementar todas as funcionalidades com a ferramenta sugerida, a API JavaFX. Neste estágio do planejamento, procurou-se compreender a ferramenta, descobrindo todas as suas possibilidades, bem como suas limitações. A finalidade aqui foi a familiarização com o JavaFX e a busca por funcionalidades que fossem úteis ao propósito do projeto.

Por fim, as etapas de implementação foram organizadas. Foram elaborados os Diagramas de Casos de Uso (Apêndice A) e de Atividades (Apêndices B e C) para fornecer uma visão mais abrangente do que deveria ser implementado e da forma como o sistema deveria se comportar.

3.2 Projeto

Na metodologia XP, a fase de projeto pode acontecer continuamente durante todo o ciclo de desenvolvimento. Para este aplicativo, executou-se esta etapa inicialmente para o sistema como um todo e, depois, antes da implementação de cada um dos requisitos descritos. O projeto preliminar foi dividido em quatro momentos: projeto de arquitetura do sistema, modelagem de interfaces com o usuário, modelagem do banco de dados e, após a produção de cada incremento, refatoração.

A decisão acerca da arquitetura do sistema foi fortemente influenciada pela API utilizada em seu desenvolvimento. O JavaFX apresenta elementos comuns aos componentes do modelo MVC. O Modelo representa as classes responsáveis pela gerência de dados persistentes e pelas regras de negócio. O componente Visão reúne os arquivos chamados de FXML, que são gerados pela ferramenta de criação de interface com o usuário, o SceneBuilder. Por fim, cada FXML tem o seu Controler equivalente, onde as interações do usuário com o sistema são tratadas e transmitidas entre Visão e Modelo.

Como dito anteriormente, a cada iteração do desenvolvimento foram realizadas tarefas de projeto, codificação e testes. Sendo assim, as interfaces com o usuário foram modeladas na medida em que fossem ser construídas. Para cada tela a ser implementada, primeiro foi feita a modelagem, para só então codificar e testar.

Em relação ao banco de dados, constatou-se que, pelo fato de o sistema desenvolvido ser de pequeno porte, o mesmo exigiria poucos acessos ao banco e teria um número relativamente pequeno de dados persistentes. A ferramenta SQLite foi escolhida para fazer o gerenciamento do banco devido à sua capacidade de administrar dados locais de forma simples e eficaz.

Ainda como uma tarefa de projeto, após cada iteração do ciclo de desenvolvimento, foi realizada a refatoração do código. Refatoração é o processo de retornar a um código que funciona perfeitamente para realizar melhorias em sua estrutura. Um detalhe importante é que ela não efetua mudanças na forma como o código se comporta, apenas o simplifica. Seu maior benefício é permitir que as funcionalidades sejam implementadas sem que haja preocupação com a elegância do código, uma vez que este será reformulado com o objetivo de atender a esta necessidade.

3.3 Codificação

A próxima etapa do desenvolvimento foi a Codificação. Contrário ao proposto pela metodologia XP, para este software não foi utilizada a técnica de desenvolvimento orientado por testes, onde primeiro se escreve os testes para só então implementar. A codificação ocorreu de forma tradicional.

À medida em que foram feitos os projetos detalhados dos componentes e funções de cada tela, avançou-se para a etapa de codificação de cada uma delas. Nesta etapa, foi produzido apenas o código meramente necessário para cumprir os requisitos especificados no projeto. A preocupação com sua clareza e concisão ficou a cargo da etapa de Projeto, com a refatoração, que acontece depois de concluídos os testes. Com o código finalizado e compilando corretamente, inicia-se a fase de testes.

3.4 Testes

A Regra 10 de Myers (2004) foi apresentada pelo autor em seu livro “The Art of Software Testing”. Nela, o autor afirma que o custo de correção de defeitos em um software,

seja esse custo financeiro, de tempo ou mesmo relacionado a quaisquer outros recursos, tende a aumentar em dez vezes para cada estágio de evolução do processo de desenvolvimento. Com o intuito de diminuir tais custos, foram efetuados testes em cada iteração do ciclo de desenvolvimento, seguindo a abordagem evolucionária de teste.

Na abordagem evolucionária de planejamento de testes, um módulo (ou unidade) de uma aplicação é desenvolvido, testado e corrigido, e então é adicionado um pequeno pedaço que inclua uma nova funcionalidade. As duas unidades devem então ser testadas como um componente integrado, aumentando a complexidade à medida em que se prossegue (MICROSOFT, 2017).

Ao fim do projeto, foi feito o Teste de Sistema para uma verificação integral da aplicação. Nele, foi apurado se todos os requisitos identificados previamente foram implementados de forma correta e se funcionam devidamente em conjunto.

4 SISTEMA DESENVOLVIDO

Nesta seção serão apresentadas as principais telas do sistema, sua aparência e suas funcionalidades.

4.1 Menu Principal

O aplicativo é iniciado a partir da Tela Menu Principal (Figura 6), disponível também no menu Arquivo: Página Inicial. Esta tela funciona como um guia rápido, onde estão todas as principais funcionalidades do software para acesso direto. A organização de todas as opções em um só lugar contribui para que o usuário tenha um melhor entendimento do escopo do aplicativo, sem a necessidade de navegar por todos os menus. O que pode ser feito é apresentado de forma mais visual e intuitiva.

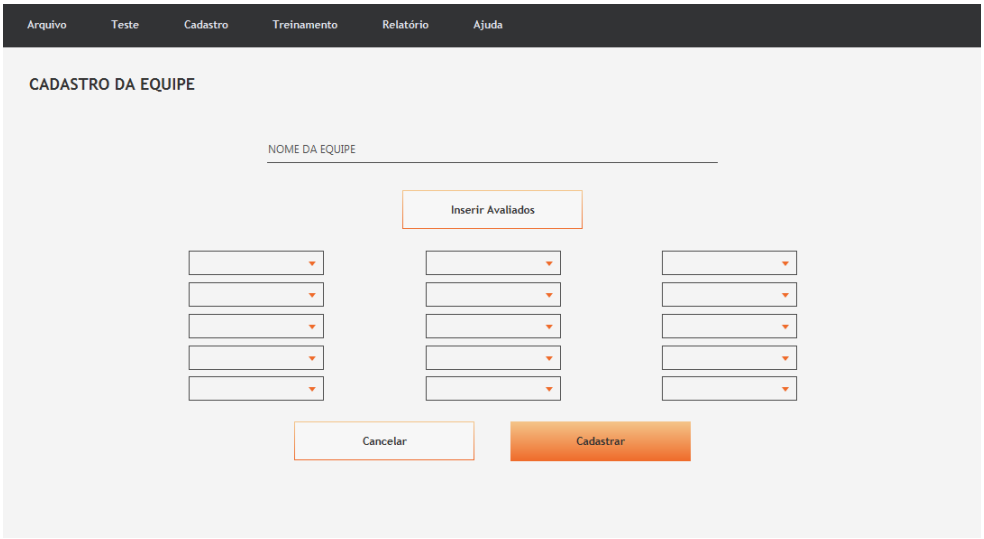
Figura 6 – Página Inicial

Fonte: Própria (2018).

4.2 Cadastro de Equipes e Cadastro de Avaliados

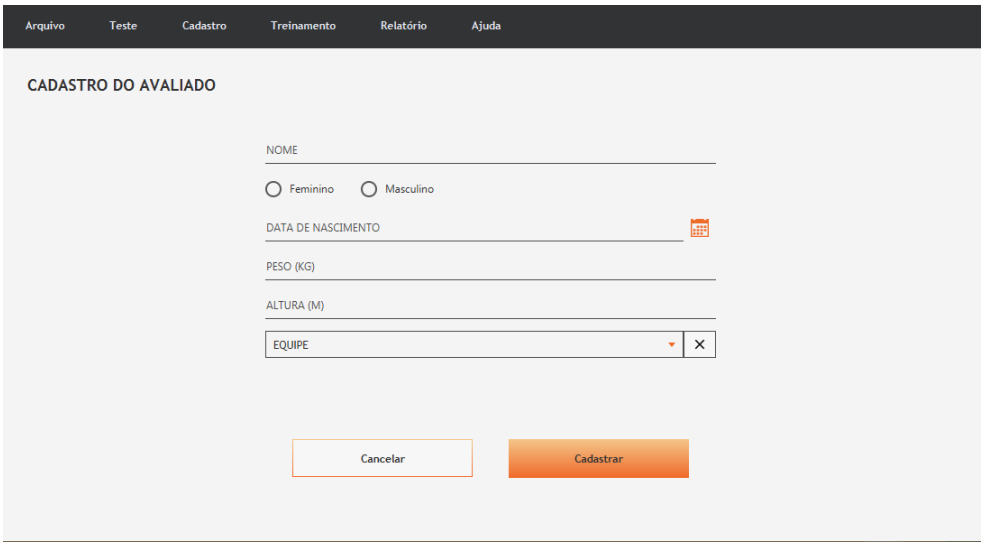
Em Cadastro de Equipes (Figura 7), pode-se criar uma nova equipe e acrescentar avaliados na mesma, desde que já estejam cadastrados no sistema. Por sua vez, na Tela de Cadastro de Avaliados (Figura 8), é possível salvar no Banco de Dados do aplicativo os indivíduos que participarão dos testes intervalados e, possivelmente, dos treinamentos. Aqui também está disponível a opção de adicionar este novo avaliado a uma equipe pré-cadastrada ou ainda criar uma nova equipe e inseri-lo ao mesmo tempo em que se realiza o cadastro desta equipe. Por fim, é permitido também adicionar o avaliado sem relacioná-lo a uma equipe. As telas estão disponíveis pelo menu Cadastro, Nova Equipe ou Novo Avaliado.

Figura 7 – Cadastro de Equipe



Fonte: Própria (2018).

Figura 8 – Cadastro de Avaliado



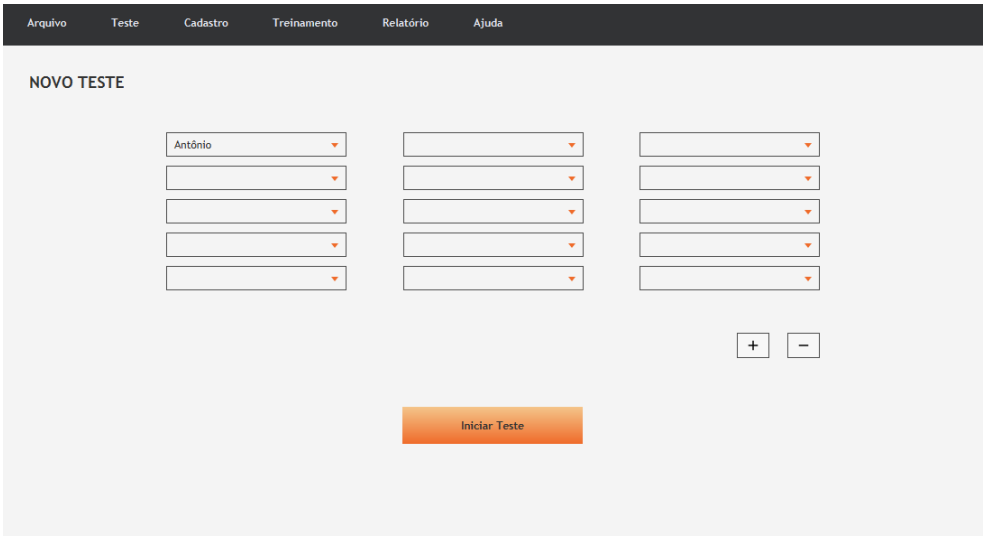
Fonte: Própria (2018).

4.3 Novo Teste

A partir do momento em que se tem um ou mais avaliados cadastrados, é possível aplicar um teste individual ou de equipe, começando pela Tela Criar Novo Teste (Figura 9). Nesta tela, a escolha do avaliado ou avaliados a participarem do teste é feita por meio do menu suspenso contendo todos os nomes registrados na base de dados.

Caso queira aplicar um teste individual, basta que o usuário escolha o avaliado e clique em “Iniciar Teste”. Se o teste for em equipe, deve-se clicar no botão correspondente a adicionar avaliado (+) para possibilitar a escolha de cada indivíduo a ser testado. No máximo quinze avaliados podem ser inscritos para um teste de equipe. Ao clicar no botão remover avaliado (-), serão apagados um a um os menus suspensos, até que sobre apenas um, o número mínimo de avaliados requeridos para se aplicar o teste. A tela Criar Novo Teste está disponível no menu Teste, Novo Teste.

Figura 9 – Novo Teste

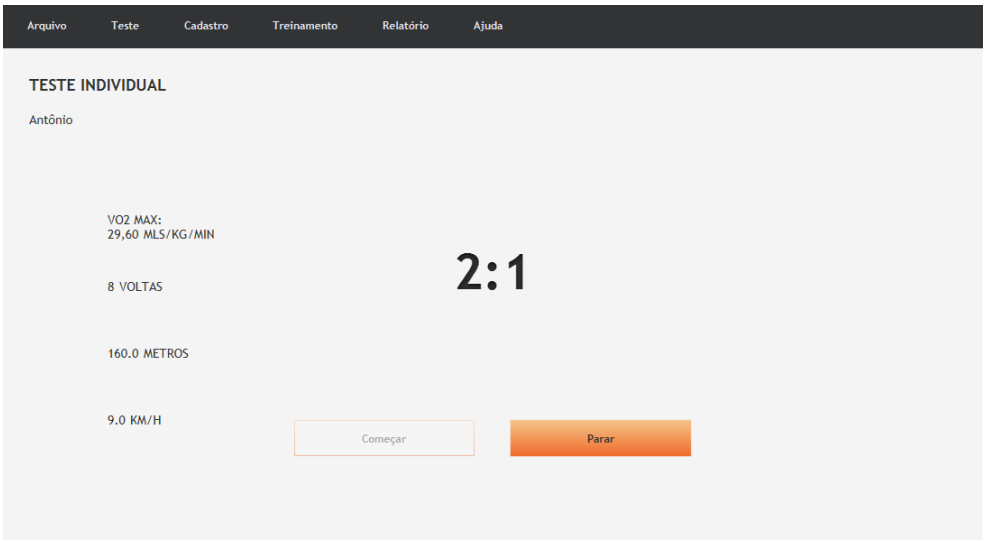


Fonte: Própria (2018).

4.4 Teste Individual

Para Teste Individual, ou seja, quando se escolhe somente um avaliado, o aplicativo abrirá a Tela de Teste Individual (Figura 10). Nela, o usuário poderá iniciar o teste a qualquer momento clicando no botão “Começar”. À esquerda da tela, são apresentados o nome do avaliado testado, o valor do VO2 Max correspondente, o número de voltas, a quantidade de metros já percorridos e a velocidade média do avaliado em cada momento. O nível e a volta atual do avaliado são exibidos no indicador no centro da tela. O botão “Parar” finaliza o teste e dispara uma caixa de diálogo com as opções de “Salvar” o resultado no Banco de Dados ou “Descartar” o mesmo.

Figura 10 – Teste Individual



Fonte: Própria (2018).

4.5 Teste de Equipe

Em contrapartida, no caso de o usuário escolher dois ou mais avaliados, o aplicativo o direcionará para a Tela de Teste em Equipe (Figura 11). Similar à Tela de Teste Individual, a principal diferença aqui é que é possível finalizar o teste de cada membro da equipe separadamente clicando no botão correspondente, enquanto o teste continua para os demais. O VO2 Max referente a cada indivíduo aparece dentro do seu respectivo botão de parar. Os demais valores: número de voltas, quantidade de metros e velocidade média, aparecem à esquerda da tela. O botão “Começar” inicia o teste para toda a equipe e o botão “Parar Todos” o finaliza para todos ao mesmo tempo. Ao finalizar o teste para cada um dos avaliados, pode-se escolher entre “Salvar” ou “Descartar” os resultados.

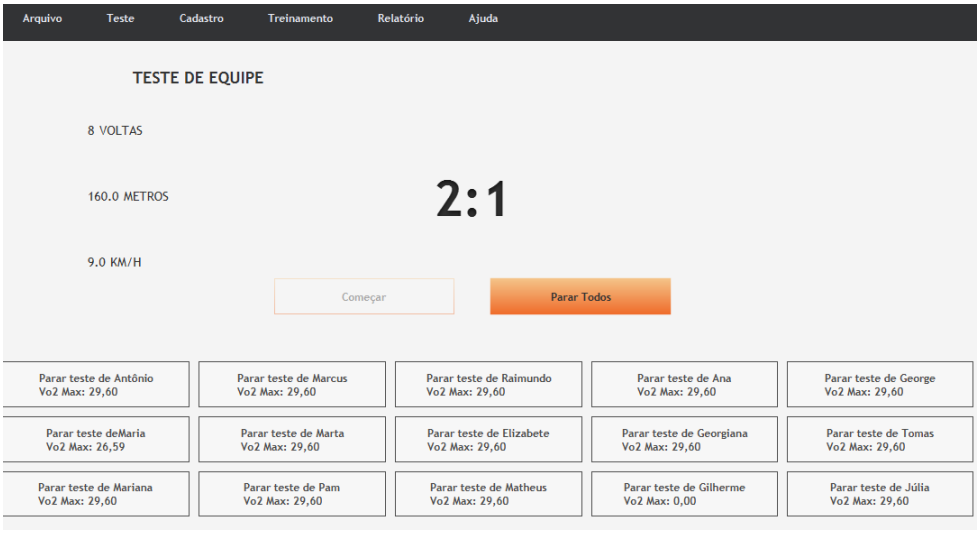
4.6 Abrir Teste

A Tela Abrir Teste (Figura 12) permite que o usuário visualize os resultados de todos os testes já ministrados. Para isso, basta escolher o avaliado de interesse no menu suspenso. Caso o teste não seja mais relevante, o usuário pode apagá-lo clicando em “Excluir”. Para ver mais informações dos avaliados cadastrados, o botão “Relatório” leva à Tela de Relatórios do sistema. A Tela Abrir Teste pode ser acessada pelo menu Teste, Abrir Teste.

4.7 Gerar Treinamento

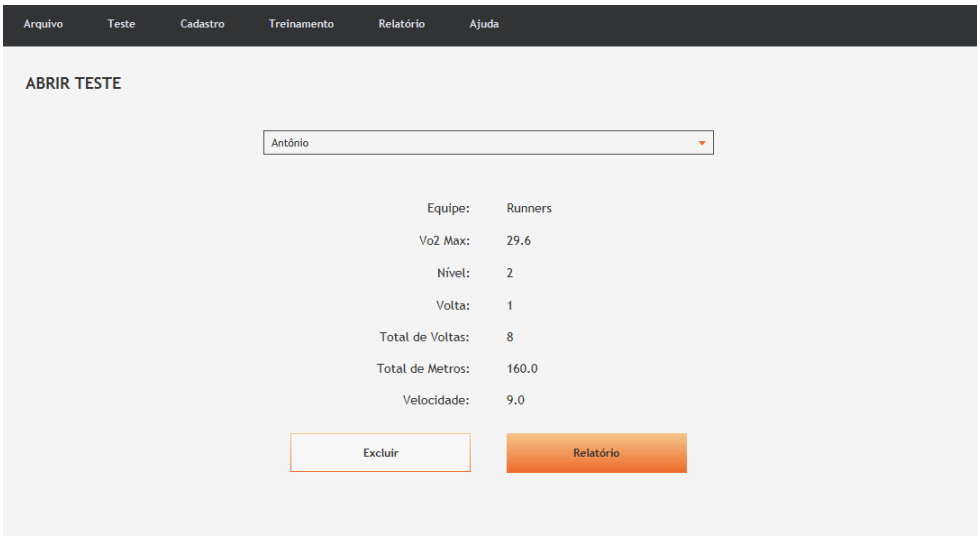
Em Gerar Treinamento (Figura 13), é possível estabelecer um treinamento com base nos resultados de teste de cada avaliado cadastrado. Para tanto, basta definir o avaliado no menu suspenso “Teste Individual” ou fazer a busca pelos testes de determinada equipe.

Figura 11 – Teste de Equipe



Fonte: Própria (2018).

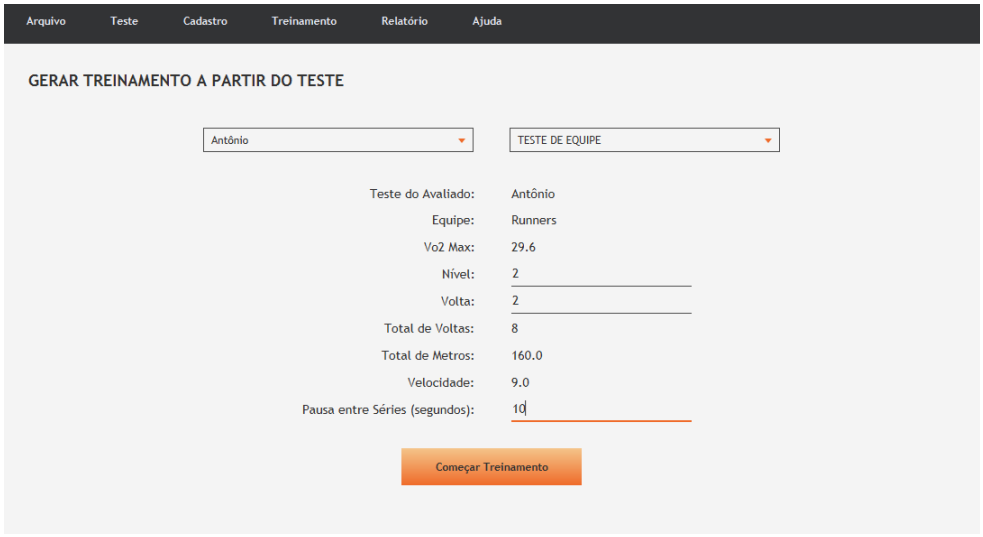
Figura 12 – Abrir Teste



Fonte: Própria (2018).

Se pelo menos um avaliado da equipe possuir um teste salvo, a mesma estará disponível em “Teste de Equipe”. Dessa forma, pode-se escolher a equipe e iterar entre seus testes com os botões correspondentes (< e >). Com o teste selecionado, é preciso preencher os campos Nível, Volta e Pausa entre Séries antes de clicar em “Começar Treinamento” para salvar as alterações e ser direcionado à Tela de Treinamento.

Figura 13 – Gerar Treinamento



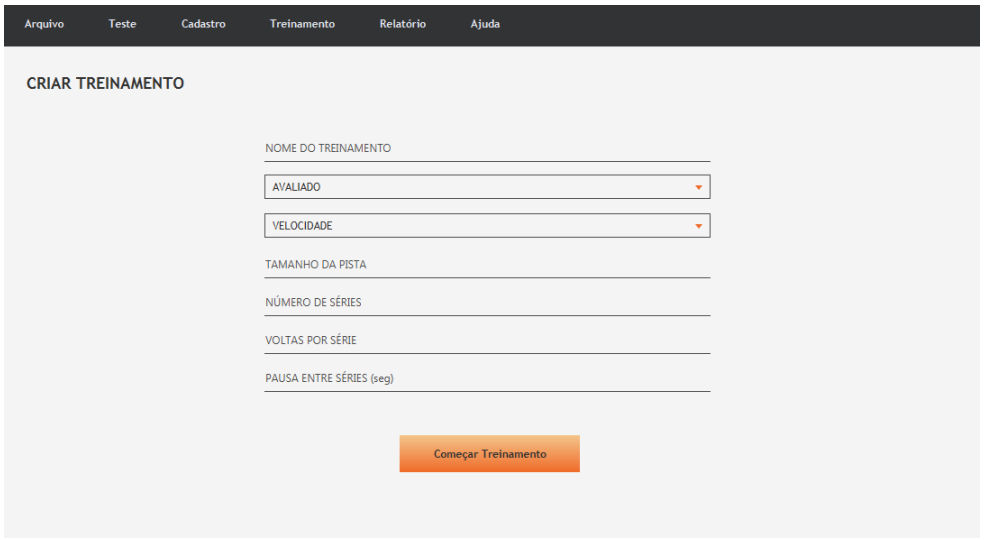
Fonte: Própria (2018).

4.8 Criar Novo Treinamento

Na Tela Criar Novo Treinamento (Figura 14), o usuário pode compor um treinamento para qualquer avaliado cadastrado, mesmo que este ainda não tenha feito um teste. Esta opção possibilita uma maior flexibilidade quanto à escolha do treinamento dos avaliados, podendo preencher com os valores que julgar sensatos para treinar cada indivíduo.

Nesta tela, é possível inserir um nome para o treinamento, escolher o avaliado dentre os cadastrados, a velocidade média, o tamanho da pista, o número de séries, o número de voltas por série e a o tempo de pausa entre cada série. A velocidade do treinamento é baseada nos valores estabelecidos pelo modelo de Léger para Beep Test e varia entre 8,5 a 18,5. Para salvar e iniciar o treinamento, com todos os campos preenchidos, basta clicar em “Começar Treinamento”. As telas Gerar Treinamento e Criar Novo Treinamento estão disponíveis no menu Treinamento, Novo Treinamento.

Figura 14 – Criar Treinamento

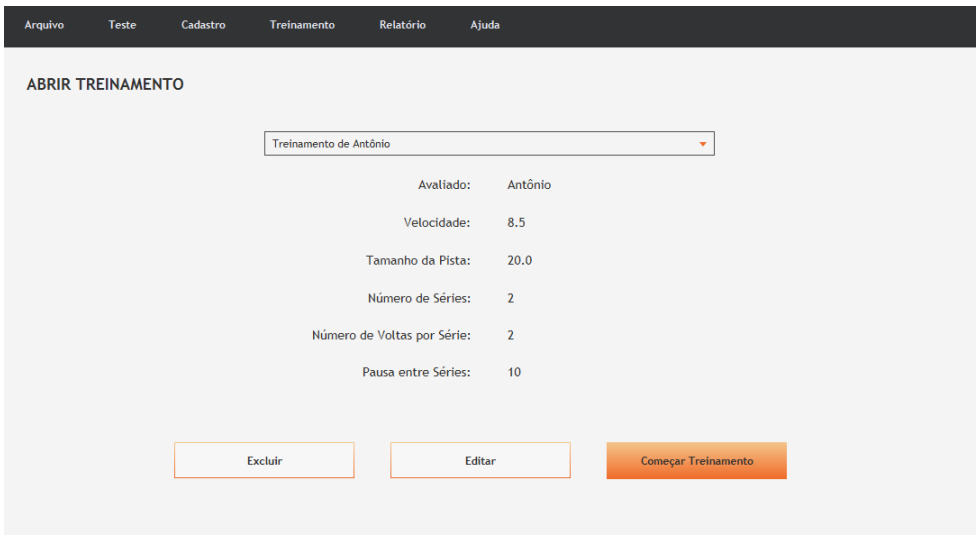


Fonte: Própria (2018).

4.9 Abrir Treinamento

Na Tela de Abrir Treinamento (Figura 15), é possível editar, excluir ou iniciar um treinamento já existente. Ao selecionar seu nome no menu suspenso, as informações relacionadas serão exibidas e será possível alterá-las. Para guardar as mudanças no Banco de Dados, basta clicar em “Editar”e, depois de fazer as devidas alterações, em “Salvar”. Para apagar o treinamento em questão, deve-se pressionar “Excluir Treinamento”. Clicando em “Começar Treinamento”, o usuário será direcionado para a Tela Treinamento. A tela está disponível no menu Treinamento, Abrir Treinamento.

Figura 15 – Abrir Treinamento

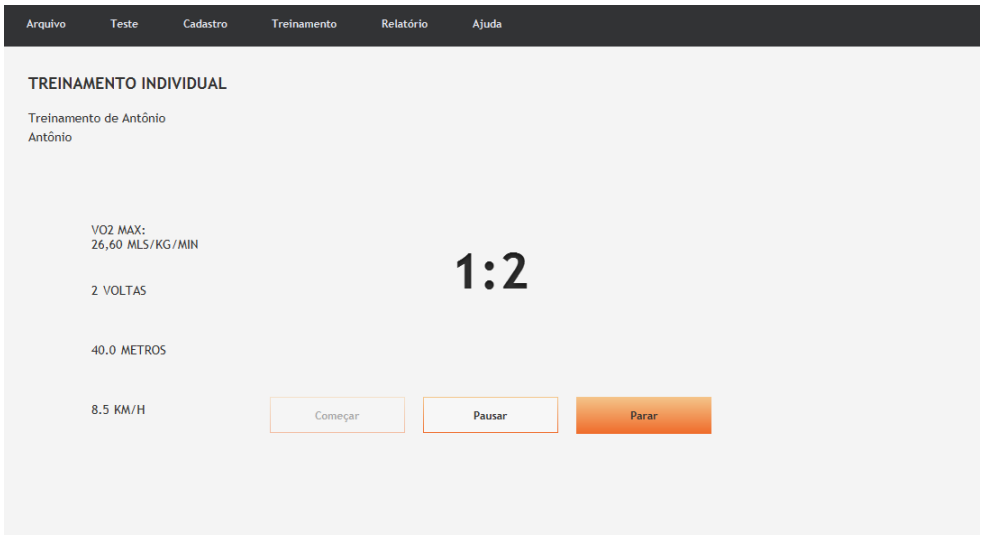


Fonte: Própria (2018).

4.10 Treinamento

Quando a opção “Começar Treinamento” é acionada a partir das telas Criar Novo Treinamento, Gerar Treinamento ou Abrir Treinamento, o aplicativo será direcionado para a Tela de Treinamento (Figura 16). Seguindo o modelo das Telas de Teste, à esquerda estão dispostas as informações relacionadas: nome do treinamento, nome do avaliado, VO2 Max, número total de voltas, distância percorrida e velocidade média. O nível e a volta atual do avaliado são exibidos no indicador no centro da tela. Para iniciar, basta clicar em “Começar”. O botão “Pausar” suspende o treinamento no ponto corrente e o mesmo pode ser retomado clicando novamente no botão “Começar”. “Parar” finaliza o treinamento.

Figura 16 – Treinamento



Fonte: Própria (2018).

4.11 Relatórios

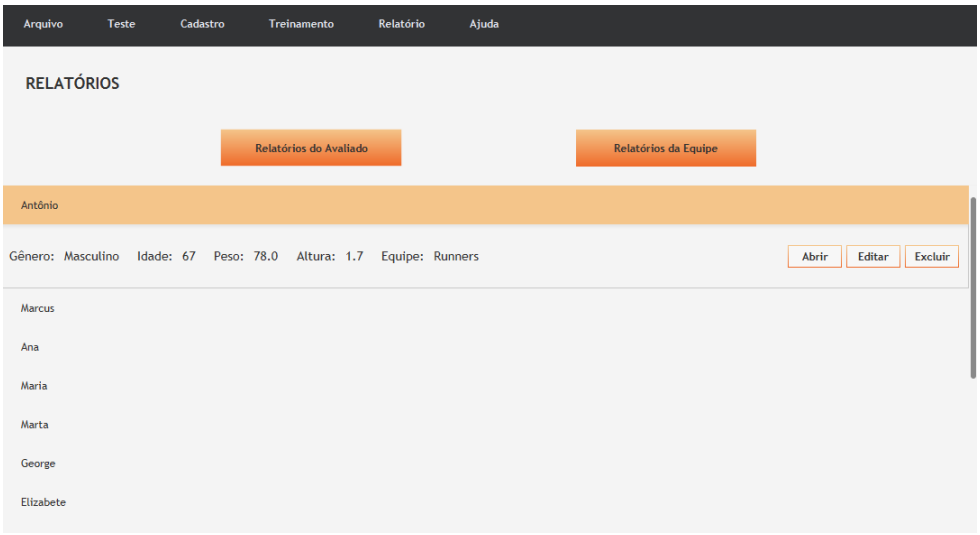
4.11.1 Relatórios do Avaliado

A Tela de Relatórios oferece a possibilidade de gerenciar de forma integral todos os avaliados e equipes cadastrados.

Clicando em “Relatórios do Avaliado” (Figura 17), serão exibidos menus expansíveis para cada avaliado. Ao abrir o menu, o usuário poderá visualizar seus dados básicos de cadastro e os botões de ação relacionados ao mesmo: “Abrir”, “Editar” e “Excluir”.

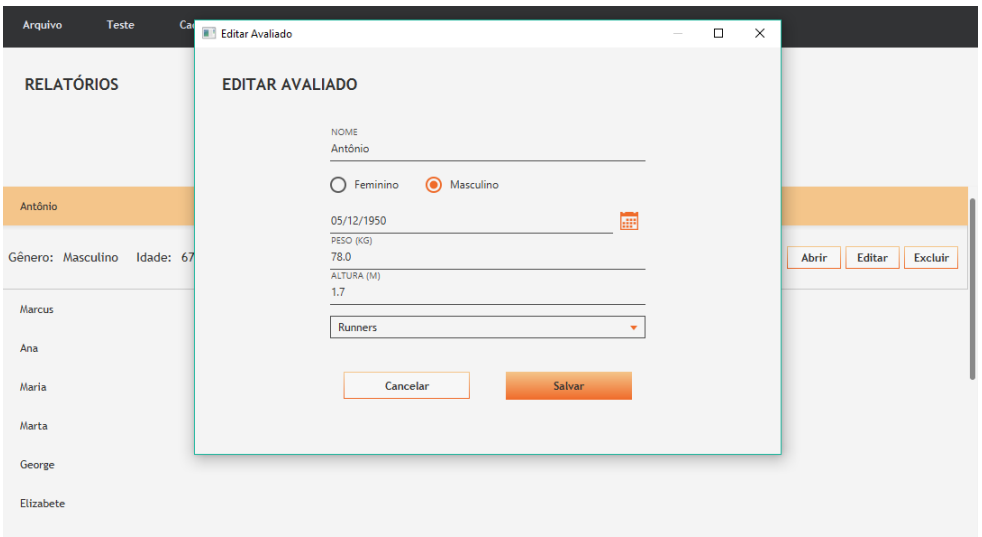
O botão “Editar” abre um modal para edição dos dados de cadastro do avaliado (Figura 18). Nele, é possível alterar nome, gênero, data de nascimento, peso, altura e equipe do avaliado em questão.

Figura 17 – Relatórios do Avaliado



Fonte: Própria (2018).

Figura 18 – Editar Avaliado

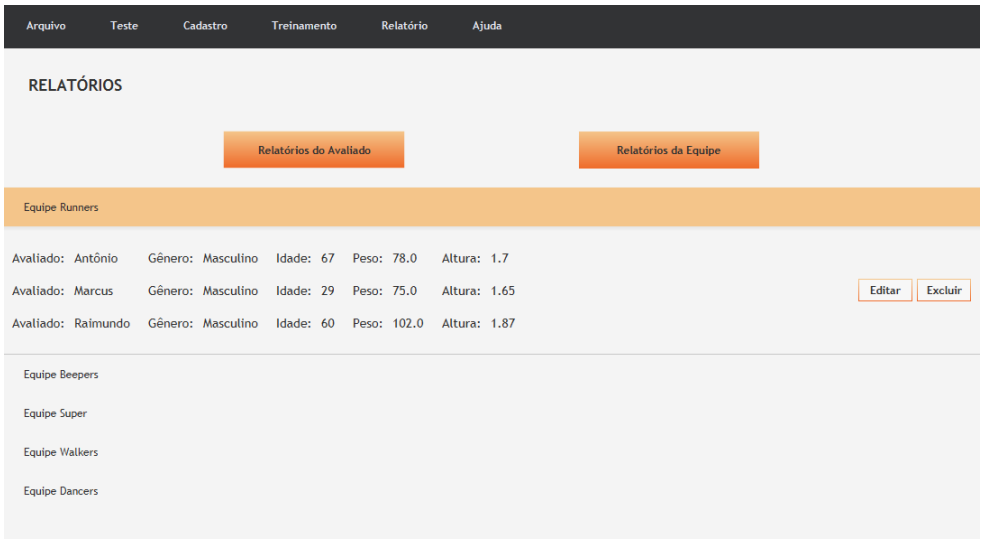


Fonte: Própria (2018).

4.11.2 Relatórios da Equipe

Ainda na Tela de Relatórios, ao clicar no botão “Relatórios da Equipe” (Figura 19), serão dispostos menus expansíveis para cada equipe cadastrada. Ao abrir o menu, tornam-se visíveis os dados básicos de cada avaliado daquela equipe, bem como os botões de ação relacionados a ela: “Editar”e “Excluir”

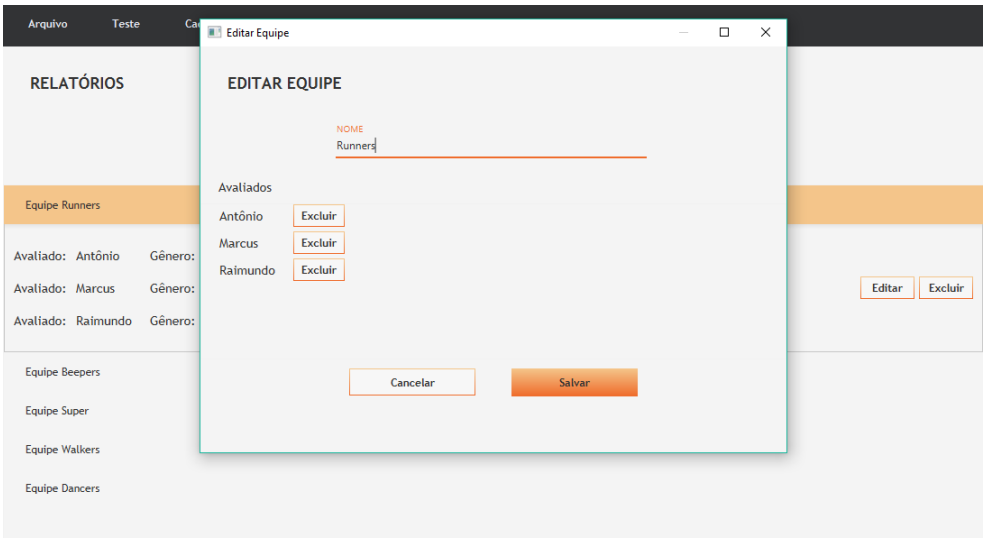
Figura 19 – Relatórios da Equipe



Fonte: Própria (2018).

Da mesma forma que nos Relatórios do Avaliado, o botão “Editar” abre um modal onde é pode-se modificar os dados relacionados à equipe (Figura 20). Aqui é possível mudar o nome da equipe e excluir os avaliados que não pertençam mais à mesma.

Figura 20 – Editar Equipe

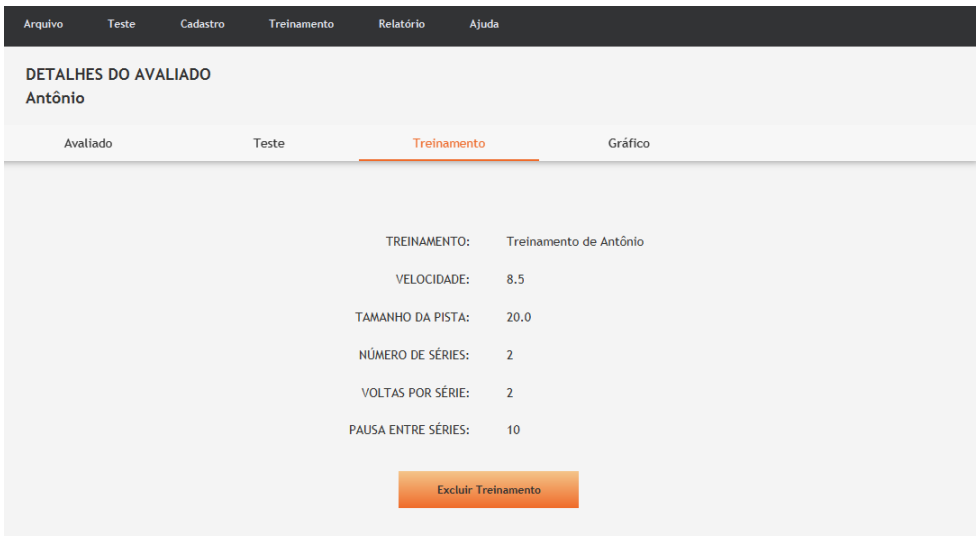


Fonte: Própria (2018).

4.11.3 Detalhes do Avaliado

Ao clicar no botão “Abrir” dentro do menu de um avaliado, o usuário será direcionado para a Tela de Detalhes do Avaliado (Figura 21). Nesta tela, estão dispostas todas as informações pertinentes àquele avaliado, organizadas em abas. Na aba Avaliado, estão os dados de cadastro. Teste e Treinamento exibem os detalhes dos mesmos e permitem excluir, se houver, ou criar um novo teste ou treinamento, caso não haja. Por fim, a aba Gráfico exibe a evolução de todos os testes feitos pelo avaliado, representados em três fatores diferente: VO2 Max, velocidade e distância Percorrida. Nesta aba o usuário também tem a opção de excluir todo o histórico de testes do avaliado.

Figura 21 – Detalhes do Avaliado



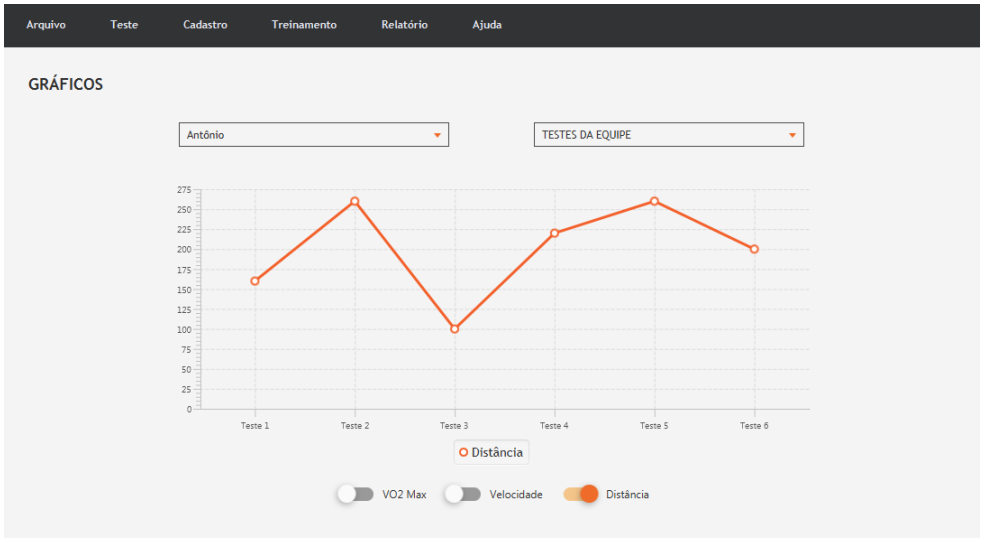
Fonte: Própria (2018).

Nos dois tipos de relatório, o botão “Excluir” remove todos os registros relacionados àquele avaliado ou equipe. No caso do avaliado, é deletado também qualquer teste ou treinamento criados para ele.

4.12 Gráficos

Na Tela de Gráficos (Figura 22), estão dispostas as informações relativas aos testes aplicados, possibilitando uma análise mais aprofundada da evolução de avaliados e equipes. Para visualizar, o usuário deve escolher entre “Testes do Avaliado” e “Testes da Equipe” quais informações deseja exibir. Assim como na tela Detalhes do Avaliado, é preciso selecionar também o fator de comparação, que pode ser: VO2 Max, velocidade ou distância percorrida. Na opção Testes do Avaliado, é possível ver o histórico de testes de cada avaliado cadastrado. Já na opção Testes da Equipe, o usuário poderá comparar os resultados de teste de todos os avaliados de determinada equipe.

Figura 22 – Gráficos



Fonte: Própria (2018).

5 CONSIDERAÇÕES FINAIS

O desenvolvimento do sistema proposto teve como motivação principal a necessidade de um sistema capaz de gerenciar integralmente a aplicação do Beep Test e a prescrição do treinamento intervalado. Entretanto, muito mais do que apenas o software finalizado, os resultados obtidos vão além. Este trabalho proporcionou a ampliação dos conhecimentos acerca de modelos de processo de software e padrões de arquitetura, bem como das tecnologias utilizadas.

No que diz respeito ao processo de desenvolvimento de software, é imprescindível que se faça um estudo prévio das metodologias que mais se adaptam à necessidade e à realidade do projeto. A metodologia ágil Extreme Programming comprovou ser um importante instrumento para alcançar o nível esperado de qualidade.

O contato frequente com o cliente possibilitou um entendimento muito mais profundo de suas necessidades e de suas aspirações para o sistema. Além disso, impediu diversas vezes que houvesse retrabalho, pois impossibilitou que o desenvolvimento de funcionalidades mal compreendidas fosse levado adiante.

A entrega frequente de incrementos de software também agiu como um controle de qualidade periódico. A cada entrega foi possível visualizar como as diferentes partes do sistema se integram para formar o todo e se cada uma delas realmente fazia sentido inserida no escopo maior. Da mesma forma, o incremento assegura que cada desvio do objetivo possa ser identificado em estágios iniciais.

O padrão de arquitetura escolhido contribuiu para o projeto de forma efetiva ao incentivar a criação de código reutilizável. Esta prática otimizou o tempo de desenvolvimento, na mesma medida em que promoveu as boas práticas de codificação, tornando o código limpo, de fácil compreensão e alta manutenibilidade.

Como abordado no capítulo 3, a cada iteração do ciclo de desenvolvimento foram realizados testes de cada funcionalidade implementada. Essa prática contribuiu para que a evolução do software ocorresse de maneira controlada, sem que pendências fossem postergadas. Ao final do ciclo, o teste de sistema foi executado para assegurar que todas as expectativas relacionadas à qualidade e à efetividade do produto foram alcançadas.

Por meio destes testes, chegou-se a algumas conclusões acerca do software desenvolvido. A primeira é de que o objetivo geral do trabalho foi alcançado. Em relação aos objetivos específicos, foi possível implementar todas as funcionalidades principais previstas inicialmente para a aplicação. No entanto, a princípio havia a proposta de gerar a versão em áudio dos treinamentos, que se revelou como um grande desafio. A decisão final acerca desta utilidade foi de que a mesma deveria ser retirada do escopo da pesquisa e considerada para trabalhos futuros.

As operações básicas de interação com a base de dados foram realizadas com êxito pela aplicação. O sistema é capaz de se comunicar com o banco de dados a partir das quatro operações básicas: criação, leitura, atualização e exclusão (CRUD - Create, Read, Update, Delete).

Os testes realizados mostraram que todas as informações armazenadas podem ser inteiramente administradas por meio das diferentes telas do sistema.

As ferramentas adotadas para a realização deste empreendimento se mostraram eficazes para a resolução do problema em questão. Desde o sistema gerenciador de banco de dados, até a biblioteca gráfica e a linguagem de programação escolhidas, todas as ferramentas ofereceram soluções efetivas para se produzir cada uma das funcionalidades pretendidas.

Ao final do trabalho, conclui-se que foi possível desenvolver o software proposto com êxito. Como almejado, o resultado é um instrumento facilitador na aplicação do teste, na elaboração do treinamento e na análise dos resultados. Espera-se que o mesmo contribua de maneira positiva para o progresso de pesquisas relacionadas ao HITT produzidas na UFVJM e em outras instituições.

REFERÊNCIAS

BECK, K. **Extreme Programming Explained: Embrace Change**. [S.l.: s.n.], 1999. ISSN 20161641. ISBN 0201616416.

BOEHM, B. W. SPIRAL MODEL OF SOFTWARE DEVELOPMENT AND ENHANCEMENT. **Computer**, 1988. ISSN 00189162.

DEITEL, H.; DEITEL, P. **Java - Como Programar**. [s.n.], 2010. 1176 p. ISBN 8576055635. Disponível em: ⟨http://books.google.com/books?id=xWMVRAAACAAJ{&}pgi⟩.

DEMARCO, T.; BOEHM, B. The Agile Methods Fray. **Computer**, 2002. ISSN 00189162.

GIBALA, M. J.; LITTLE, J. P.; MACDONALD, M. J.; HAWLEY, J. A. Physiological adaptations to low-volume, high-intensity interval training in health and disease. **Journal of Physiology**, v. 590, n. 5, p. 1077–1084, 2012. ISSN 00223751.

JFOENIX. **JFoenix**. 2017. Disponível em: ⟨http://www.jfoenix.com/⟩.

LOPES, F. J. G. **Comparação dos Efeitos do Treino e do Destreino de Dois Programas de Corrida, Intervalado de Alta Intensidade e Contínuo, sobre Parâmetros de Risco Cardio-metabólico em Homens com Excesso de Peso**. Tese (Doutorado) — Universidade Federal dos Vales do Jequitinhonha e Mucuri, 2017.

LOWE, D. **JavaFX For Dummies**. [S.l.]: John Wiley & Sons, Inc, 2015. ISBN 9781118385340.

MICROSOFT. **Testing Overview**. 2017. Disponível em: ⟨https://msdn.microsoft.com/en-us/library/aa292191(v=vs.71).aspx⟩.

MYERS, G. **The Art of Software Testing, Second edition**. [S.l.: s.n.], 2004. ISSN 0960-0833. ISBN 0-471-46912-2.

ORACLE. **JavaFX Overview (Release 8)**. 2014. Disponível em: ⟨http://docs.oracle.com/javase/8/javafx/get-started-tutorial/jfx-overview.htm{\#}JFXST⟩.

ORACLE. **Java SE Development Kit 8**. 2017. Disponível em: ⟨http://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html⟩.

PRESSMAN, R. S. **Engenharia de Software - Uma Abordagem Profissional**. [S.l.: s.n.], 2007. ISSN 85860457. ISBN 8586804576.

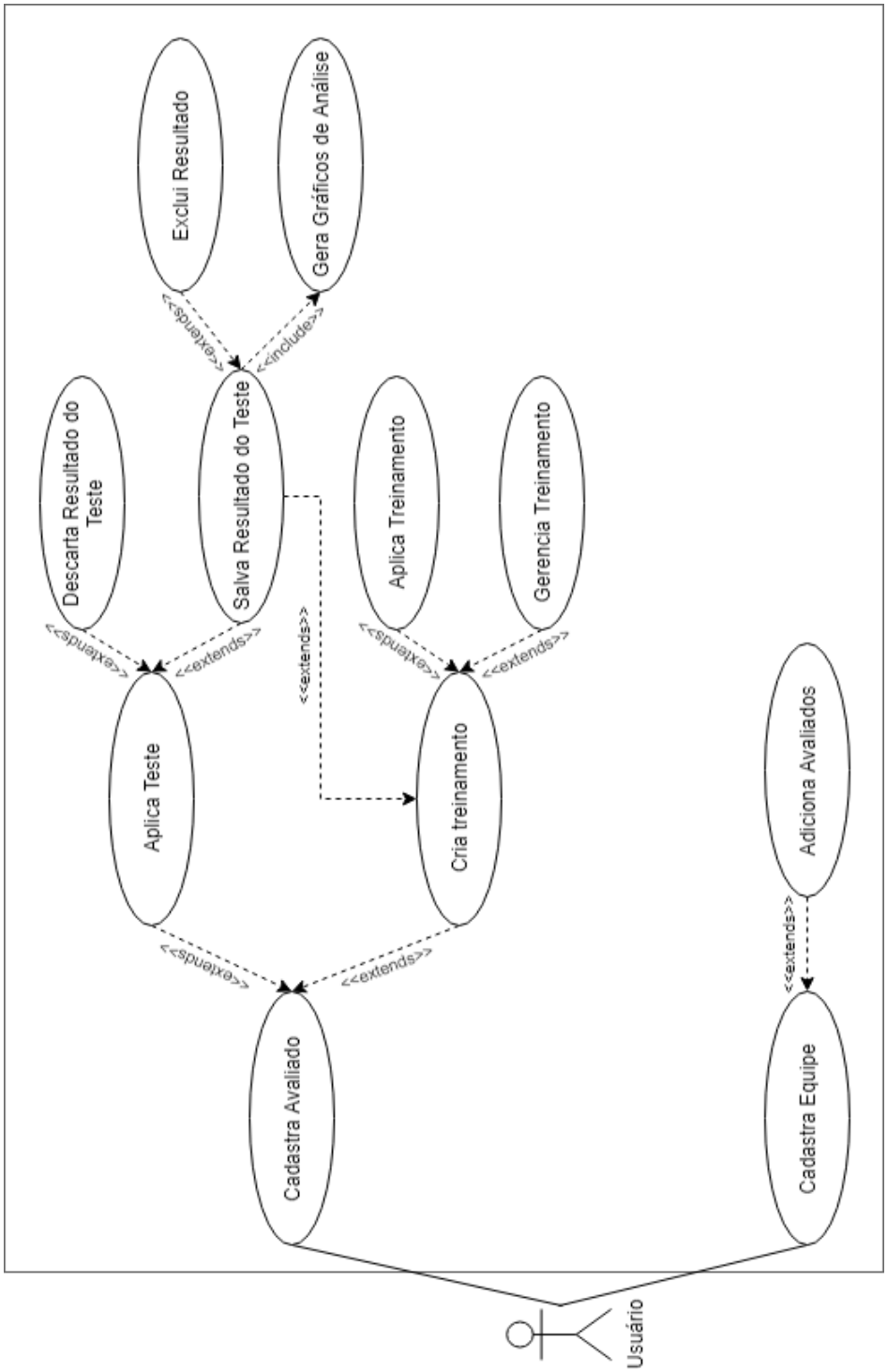
ROYCE, W. Managing the development of large software systems. In: **Proceedings of IEEE WESCON'70**. [S.l.: s.n.], 1970.

SCHWABER, K.; BEEDLE, M. **Agile Software Development with Scrum**. [S.l.: s.n.], 2002. ISSN 00189162. ISBN 0130676349.

SOMMERVILLE, I. **Engenharia de Software**. [S.l.: s.n.], 2011. ISBN 8579361087.

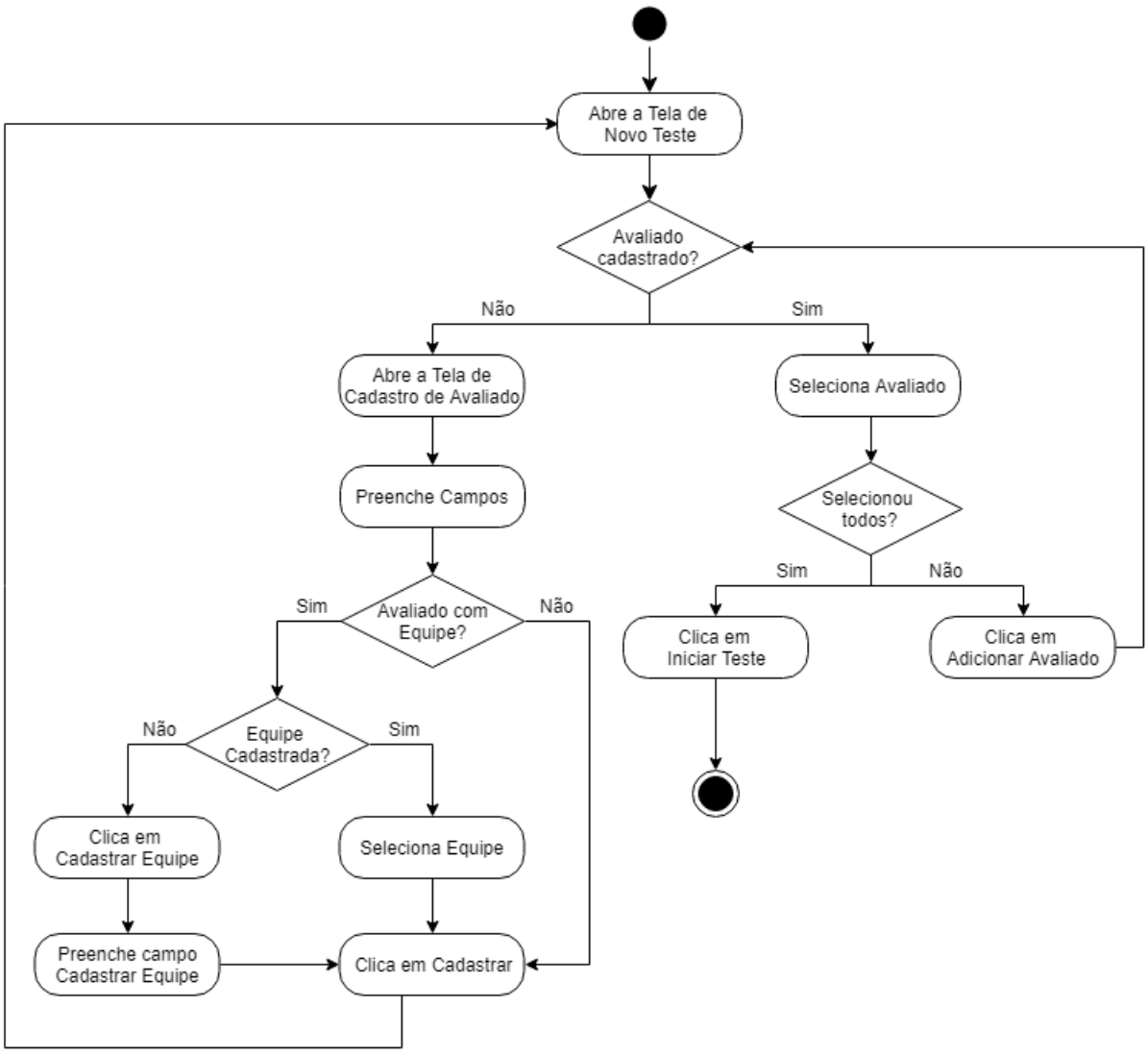
SQLITE. **SQLite Documentation**. 2017. Disponível em: ⟨https://www.sqlite.org/docs.html⟩.

Apêndice A – DIAGRAMA DE CASOS DE USO DO SISTEMA



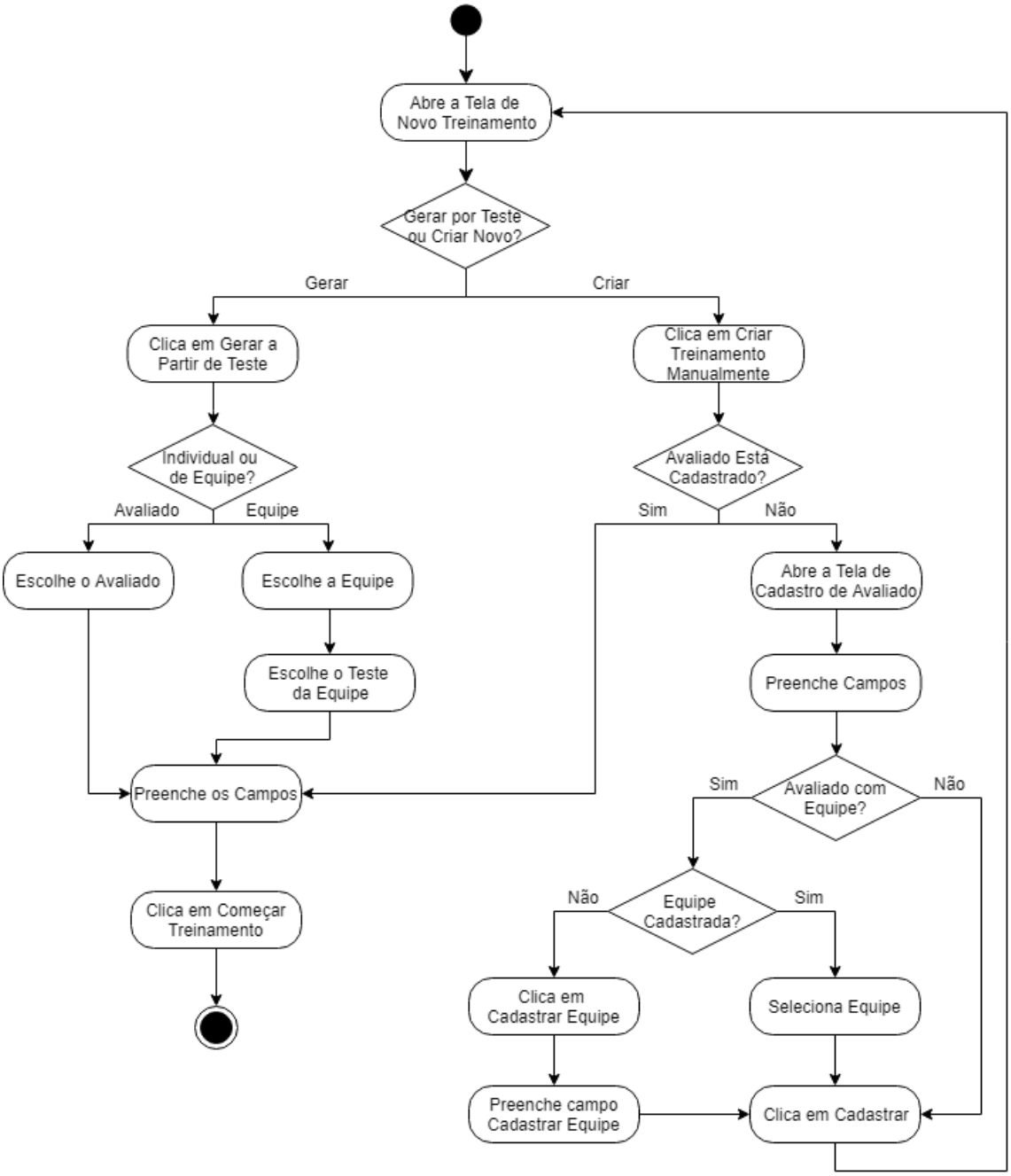
Fonte: Própria (2018).

Apêndice B – DIAGRAMA DE ATIVIDADES: NOVO TESTE



Fonte: Própria (2018).

Apêndice C – DIAGRAMA DE ATIVIDADES: NOVO TREINAMENTO



Fonte: Própria (2018).