INSTITUTO FEDERAL DE MINAS GERAIS
CAMPUS SÃO JOÃO EVANGELISTA

ANA PAULA DIAS PEREIRA

DESENVOLVIMENTO DE UMA APLICAÇÃO PARA AUXILIAR O PROCESSO DE DESENVOLVIMENTO DE SOFTWARE

SÃO JOÃO EVANGELISTA
2017

ANA PAULA DIAS PEREIRA

DESENVOLVIMENTO DE UMA APLICAÇÃO PARA AUXILIAR O PROCESSO DE DESENVOLVIMENTO DE SOFTWARE

Trabalho de conclusão de curso apresentado ao Instituto Federal de Minas Gerais - *Campus* São João Evangelista como exigência parcial para obtenção do título de Bacharel em Sistemas de Informação.

Orientador: Prof. Me. Bruno de Souza Toledo
Coorientador: Prof. Dr. Wesley Gomes de Almeida
Coorientador: Prof. Me. Edson Batista de Sena

SÃO JOÃO EVANGELISTA

2017

FICHA CATALOGRÁFICA

P426d Pereira, Ana Paula Dias.
2017

Desenvolvimento de uma aplicação para auxiliar o processo de desenvolvimento de software. / Ana Paula Dias Pereira. – 2017.
82f. ; il.

Trabalho de Conclusão de Curso (Bacharelado em Sistemas de Informação) – Instituto Federal de Educação, Ciência e Tecnologia de Minas Gerais – Campus São João Evangelista, 2017.

Orientador: Me. Bruno De Souza Toledo.
Coorientador: Dr. Wesley Gomes de Almeida.
Coorientador: Me. Edson Batista de Sena

1. Engenharia de Software. 2. Métodos Ágeis. 3. Scrum. 4. Protótipo. 5. Aplicação Web. I. Pereira, Ana Paula Dias. II. Instituto Federal de Educação, Ciência e Tecnologia de Minas Gerais – Campus São João Evangelista. III. Título.

CDD 005.1

Elaborada pela Biblioteca Professor Pedro Valério

Instituto Federal de Educação, Ciência e Tecnologia de Minas Gerais
Campus São João Evangelista

Bibliotecária Responsável: Rejane Valéria Santos – CRB-6/2907

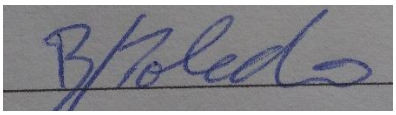
ANA PAULA DIAS PEREIRA

DESENVOLVIMENTO DE UMA APLICAÇÃO PARA AUXILIAR O PROCESSO DE DESENVOLVIMENTO DE SOFTWARE

Trabalho de conclusão de curso apresentado ao Instituto Federal de Minas Gerais - *Campus* São João Evangelista como exigência parcial para obtenção do título de Bacharel em Sistemas de Informação.

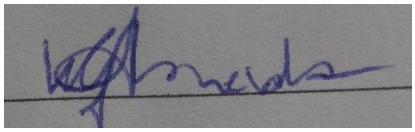
Aprovado em: 08/12/2017

BANCA EXAMINADORA



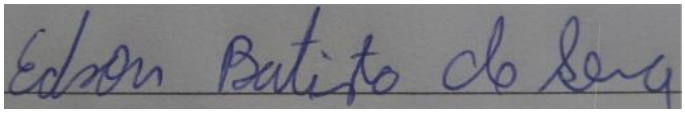
Orientador: Prof. Me. Bruno de Souza Toledo

Instituto Federal de Minas Gerais - *Campus* São João Evangelista



Coorientador: Prof. Dr. Wesley Gomes de Almeida

Instituto Federal de Minas Gerais - *Campus* São João Evangelista



Coorientador: Prof. Me. Edson Batista de Sena

Instituto Federal de Minas Gerais - *Campus* São João Evangelista

SÃO JOÃO EVANGELISTA

2017

AGRADECIMENTOS

Agradeço primeiramente aos meus familiares, em especial à minha mãe, pelo conforto e por toda ajuda concedida nos momentos de dificuldade. Agradeço também aos meus professores por todo conhecimento concedido, pela paciência, atenção e compreensão. Agradeço grandemente aos meus amigos que sempre se fizeram presentes em todos os momentos, me auxiliando em tudo que se fizesse necessário. Finalmente, agradeço a instituição por proporcionar a oportunidade de obtenção do título de Bacharel em Sistemas de Informação.

RESUMO

O *software* é o um dos mais importantes dentre as tecnologias responsáveis pelo crescimento do mercado global. Frente ao rápido crescimento de sua demanda, devido à complexidade dos problemas a serem resolvidos, surge à preocupação por garantir agilidade e economia no processo de desenvolvimento dos produtos de *software*. Estas discussões posicionam a Engenharia de Software como componente substancial neste processo, em que esta conceitua-se na determinação e implantação de conceitos de engenharia, a fim de assegurar economia, confiabilidade e eficiência ao produto final. Os métodos ágeis de desenvolvimento de *software* subsidiam as etapas previstas pela Engenharia de Software e surgiram com o objetivo de criar produtos proficientes dedicando-se menos tempo, passando a serem considerados fundamentais para apropriar a produção de *software* à grande demanda da sociedade. O método ágil *Scrum* se destaca dentre os demais pela ênfase dada ao gerenciamento do projeto e por abranger atividades de monitoramento e *feedback* visando à identificação e a correção de deficiências, tendo como base o planejamento iterativo e incremental. Quando aplicado adequadamente, o *Scrum* pode contribuir grandemente para o sucesso da produção de *software*. Portanto, o presente trabalho visou desenvolver o protótipo de uma aplicação *web* para atuar em pequenos projetos de *software*, auxiliando equipes iniciantes no âmbito do desenvolvimento ágil, colaborando com algumas das fases do processo de desenvolvimento com base na ferramenta *Scrum*. Para isso, a pesquisa se baseou na fundamentação teórica sobre *Scrum* e desenvolvimento de *software*. A metodologia utilizada é caracterizada como descritiva qualitativa, pois os resultados descrevem os dados obtidos por meios de testes que visaram analisar as funcionalidades gerais do sistema, como também, a experiência dos profissionais ao utilizarem o *software*. Os dados obtidos indicam que o *software* segue as normas de usabilidade, sendo interativo e de fácil uso, além de atender como guia prático de *Scrum*, revelando grande potencial para melhorias e expansões.

Palavras-chave: Engenharia de Software. Métodos ágeis. *Scrum*. Protótipo. Aplicação *web*.

ABSTRACT

The software is one of the most important among the technologies responsible for the growing of global market. Against fast growing of his demand, due the complexity of problems to being solved, comes up the preoccupation for ensure the agility and economy in the process from development of products of software. This discussions position the Software Engineering as substantial component in this process, in that this conceptualize in determination and implementation of concepts of engineering, in order to secure economy, reliability and efficiency to the final product. The agile software development subsidize the planned steps for Software Engineering and emerged with the objective of create proficient products dedicating themselves less time, happening to be considered fundamental to appropriate the production of software the great demand of society. The agile Scrum methodology stands out among the others by emphasis given to management of project and for include activities of monitoring and feedback aiming the identification and correction of deficiencies, having has base the planning iterative and incremental. When apply properly, the scrum can contribute greatly for the success of software production. Therefore, the present work aimed develop the prototype of a web application for act in little projects of software, assisting beginner teams in scope of agile development, collaborating with some of the phases of process of development based on the tool scrum. For this, the search was based in theoretical grounds about scrum and software development. The used methodology is characterized as descriptive qualitative, because the results the obtained data by means of tests aimed at analyze the general functionalities of system, as also, the experience of professionals by using the software. The obtained data indicate that the software follow the norms of usability, being interactive and of easy use, besides answer has practice guide of Scrum, revealing big potential for improvements and expansions.

Keywords: Software Engineering. Agile methodology. Scrum. Prototype. Web application.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE SIGLAS

CSS - *Cascading Style Sheet*

DSDM - *Dynamic Software Development Method*

IDE - *Integrated Development Environment*

IEC - *International Electrotechnical Comission*

IFMG-SJE - Instituto Federal de Minas Gerais - *Campus* São João Evangelista

ISO - *International Organization for Standardization*

HTML - *HyperText Markup Language*

HTTP - *HyperText Transfer Protocol*

MVC - *Model-View-Controller*

PHP - *PHP: Hypertext Preprocessor*

RF - Requisito funcional

XP – *Extreming Programing*

SUMÁRIO

1 INTRODUÇÃO

A indústria de produção de *software* representa um dos segmentos mais crescentes, dinâmicos e especializados da computação, tornando-se fator dominante nos diversos setores da economia, tais como: educação, saúde, agronegócio, e tem promovido o desenvolvimento de novas tecnologias, tornando-se o maior e mais influente setor na era industrial.

O *software* é o um dos mais importantes dentre as tecnologias responsáveis pelo crescimento do mercado global. Frente ao rápido crescimento de sua demanda, devido à complexidade dos problemas a serem resolvidos, surge à preocupação por garantir agilidade e economia no processo de desenvolvimento dos produtos de *software*. Estas discussões posicionam a Engenharia de Software como componente substancial neste processo, em que esta conceitua-se na determinação e implantação de conceitos de engenharia, a fim de assegurar economia, confiabilidade e eficiência ao produto final.

A Engenharia de Software abrange um composto de estágios que compreendem métodos, ferramentas e procedimentos. Estes elementos fundamentais proporcionam controle sobre o projeto e base para as equipes de desenvolvimento de *software* dedicadas a modelos de processos. O objetivo da Engenharia de Software é o desenvolvimento de sistemas de *software* cuja relação custo-benefício seja vantajosa (PRESSMAN; MAXIN, 2016).

Os componentes da Engenharia de Software permitem a produção de sistemas com base em diferentes abordagens, de forma que os elementos fundamentais sejam adaptados de maneira conveniente às necessidades de cada método.

Os métodos tradicionais de desenvolvimento são conhecidos por serem orientados a documentação. Foram muito utilizados no passado, em um contexto de desenvolvimento de *software* muito diferente do atual. O custo para realizar alterações e correções era muito alto, uma vez que o acesso aos computadores era limitado e não existiam modernas ferramentas de apoio ao desenvolvimento do *software* como existem atualmente. Portanto, o *software* precisava ser totalmente planejado e documentado antes de ser implementado, o que demandava demasiado custo relacionado ao tempo empregado ao planejamento e tornava inviável que mudanças nos requisitos fossem realizadas ao longo do projeto. Apesar de inflexíveis e rígidas, ainda hoje algumas metodologias tradicionais são utilizadas, como é o caso do modelo Clássico ou Cascata (SOARES, 2004).

O *software* demonstra sua natureza abstrata e intangível, sem restrições por resoluções físicas e sem regimento de processos industriais. Não havendo limitações físicas para o

potencial do *software*, este pode facilmente tornar-se extremamente complexo, consequentemente, bastante complexo para ser produzido. Apesar das técnicas da Engenharia de Software serem amplamente utilizadas, ainda há problemas em produzir *softwares* complexos que atendam às necessidades dos usuários e que sejam produzidos dentro do orçamento e prazo estabelecidos (SOMMERVILLE, 2007).

Os métodos ágeis de desenvolvimento de *software* subsidiam as etapas previstas pela Engenharia de Software e surgiram com o objetivo de criar produtos proficientes dedicando-se menos tempo. O desenvolvimento de *software* baseado em métodos ágeis considera uma abordagem flexível e dinâmica, possibilitando que mudanças nos requisitos do sistema sejam efetuadas a qualquer momento no decorrer do projeto (AMBLER; JEFFRIES, 2002). Segundo Sommerville (2007) as fases do desenvolvimento (concepção, levantamento de requisitos, projeto, implementação, testes e implantação), quando baseadas em metodologias ágeis, são realizadas tendo-se em vista potencializar os resultados, minimizando erros e retrabalho, reduzindo custos e assegurando qualidade. Consequentemente, os processos ágeis adquiriram grande relevância, o que levou a integração com métodos de desenvolvimento mais tradicionais, passando a serem considerados fundamentais para apropriar a produção de *software* à grande demanda da sociedade.

Ao final da década de 90 diversas abordagens que empregavam os conceitos de métodos ágeis emergiram, dentre as quais o *Scrum* se destacou pela ênfase dada ao gerenciamento do projeto. O *Scrum* abrange atividades de monitoramento e *feedback*, normalmente, através de reuniões rápidas e diárias com toda a equipe, visando à identificação e a correção de deficiências ou impedimentos no desenvolvimento dos produtos (BOEHM 2006).

Para Ambler e Jeffries (2002) o *Scrum* é, sem dúvida, um dos métodos ágeis mais utilizados na atualidade, principalmente por permitir a integração com outros métodos com facilidade. Pode ser aplicado ao desenvolvimento de *softwares* como a qualquer ambiente de trabalho. Focado na gestão do projeto, o *Scrum* tem como base o planejamento iterativo e incremental.

Aplicar o método *Scrum* contribui grandemente para o sucesso da produção de produtos de *software*. Dominar bem as suas práticas e ferramentas demonstra ser diferencial para o resultado do projeto. Desse modo, observando a relevância do método ágil *Scrum* bem como a necessidade por manter a sua aplicação prática congruentemente, ressalta-se a insuficiência de tecnologias gratuitas que sejam capazes de auxiliar e assegurar a aplicabilidade deste método no contexto de pequenas implementações e no contexto de

equipes que possuem pouca ou nenhuma experiência em trabalhar sob a dinâmica dos métodos ágeis.

Tendo em vista as informações manifestas sobre Engenharia de Software e o método ágil *Scrum* para a obtenção de resultados positivos no que se refere à qualidade do processo e produto de *software* e as informações acerca da necessidade de tecnologias para apoiar a prática do *Scrum*, a presente pesquisa teve por objetivo geral desenvolver um protótipo de uma aplicação *web* para atuar em pequenos projetos, auxiliando equipes iniciantes no âmbito do desenvolvimento ágil, colaborando com algumas das fases do processo de desenvolvimento com base na ferramenta *Scrum*. A aplicação irá colaborar por meio das seguintes funcionalidades: guia do método *Scrum*, definição da equipe do projeto, criação de listas de requisitos, definição de prioridades, compartilhamento de artefatos entre a equipe e definição de ciclos regulares para o desenvolvimento. Para alcançar o objetivo geral e a finalidade estabelecida nessa pesquisa foram definidos os objetivos específicos: a) Criar um protótipo para contribuir com a aprendizagem sobre o *Scrum*; b) Assegurar a aplicação adequada do método *Scrum* por equipes sem experiência em desenvolvimento ágil; c) Possibilitar flexibilidade e eficiência às etapas do processo de desenvolvimento; d) Incentivar a adoção de métodos ágeis de desenvolvimento *software*.

1.1 ESTRUTURA DA MONOGRAFIA

Este trabalho está organizado em cinco capítulos. O primeiro apresenta uma introdução e os objetivos da pesquisa. O segundo contempla o referencial teórico e apresenta os principais conceitos e abordagens sobre o tema. O terceiro capítulo descreve a metodologia aplicada na realização da pesquisa. O quarto contém os resultados e as discussões do trabalho. E por fim, o quinto e último capítulo destaca as considerações finais.

2 REFERENCIAL TEÓRICO

Este capítulo apresenta informações que fundamentam o estudo, contemplando autores que se posicionam acerca dos conceitos que envolvem o processo de desenvolvimento de *software* e demais tecnologias aplicadas.

2.1 ENGENHARIA DE SOFTWARE

O conceito de Engenharia de Software foi inicialmente proposto em 1968, em uma conferência pautada a deliberar sobre os problemas industriais do *software*, que ficaram conhecidos por "crise do *software*" (SOMMERVILLE, 2007).

A crise do *software* resultava diretamente da introdução de novos recursos de *hardware* de computador, frisando que o desenvolvimento informal de *software* já não era suficiente. A nova demanda implicou em um resultado catastrófico na criação de *software*. Em um curto período de tempo, os recursos de *hardware* expandiram demasiadamente, requisitando o desenvolvimento de produtos mais complexos e de alta qualidade (KOSCIANSKI; SOARES, 2007).

A evolução do *hardware* de computador teve como efeito o aumento da capacidade dos dispositivos, acompanhado por menores custos. Posteriormente, o maior desafio tornou-se o de melhorar a qualidade das soluções implementadas em *software* (PRESSMAN; MAXIN, 2016).

O termo "Engenharia de Software" foi definido tendo em vista evidenciar a necessidade de se incorporar fundamentos teóricos ao desenvolvimento de *software*, como nas demais áreas da engenharia. A Engenharia de Software estabeleceu-se com o objetivo de regulamentar o desenvolvimento de *software* dentro de custos convenientes, visando à alta qualidade. Está relacionada como todos os aspectos da produção, desde os estágios iniciais de especificação do sistema até sua manutenção. Sabe-se atualmente que não existe uma abordagem que seja superior para a Engenharia de Software, uma vez que existe uma ampla diversidade de sistemas de *software* e organizações. Portanto, os modelos de Engenharia de Software podem ser combinados e adaptados a diferentes realidades, levando-se em consideração a complexidade, tamanho e área de aplicação do projeto de *software* (SOMMERVILLE, 2007).

Segundo Pressman e Maxin (2016), o desenvolvimento de *software* depende de três etapas distintas, independentemente do modelo de Engenharia de Software escolhido, complexidade e tamanho do projeto ou área de aplicação:

a) Análise do Sistema: define a atribuição de cada componente de um sistema computacional, e ainda a função a ser desempenhada pelo *software*;
b) Planejamento do projeto: definido o propósito do *software*, riscos, recursos, custos e tarefas são estimados e/ou determinados;
c) Análise de requisitos: possibilita uma determinação mais precisa sobre a função a ser desempenhada pelo *software*.

2.1.1 Desenvolvimento de software

De acordo com Pressman e Maxin (2016), o desenvolvimento de *software* é um processo da Engenharia de Software diretamente relacionado com o *software* real. Este inclui implementação, questões de análise, projeto e implantação funcional.

O desenvolvimento de *software* pode ser dividido nas seguintes partes:

a) Concepção: São efetuadas análises sobre a viabilidade do sistema e como este será construído;
b) Levantamento de requisitos: nesta etapa são identificadas as funcionalidades que sistema deverá apresentar e são projetadas soluções que atendam às necessidades do cliente;
c) Projeto: os resultados do levantamento de requisitos são transformados em uma documentação que possa ser interpretada pela equipe de desenvolvimento;
d) Implementação: Esta fase corresponde à atividade de codificação dos módulos do sistema, que traduz o projeto de *design* em um programa, apresentando a estrutura e funcionalidades descritas;
e) Testes: Para garantir que o sistema irá funcionar devidamente os testes são realizados. É importante comprometer-se com a fase de testes, levando-se em consideração a sua importância para a apuração de erros que venham a prejudicar o funcionamento do sistema;
f) Implantação: o sistema é implantado no ambiente especificado pelo cliente, os usuários são instruídos para utilizá-lo e são disponibilizadas informações que

possam auxiliar o gerenciamento das mudanças ocasionadas pela implantação do sistema.

2.2 O PAPEL DO SOFTWARE

Para Sommerville (2007), efetivamente todos os países ao redor do mundo, nos tempos atuais, dependem de sistemas computacionais complexos. Serviços e produtos indispensáveis na sociedade contam com sistemas computacionais, dedicados a controlar suas atividades (REZENDE, 2005). A indústria encontra-se grandemente automatizada, do mesmo modo que os sistemas financeiros. Consequentemente, a produção e manutenção de *software*, mediante custos convenientes são necessárias para o exercício e crescimento da economia mundial (SOMMERVILLE, 2007).

O *software* transcendeu o *hardware* como solução para diversos sistemas computacionais. Seja utilizado na administração, para monitorar produtos ou habilitar sistemas, o *software* se sobressai, justificado pela oportunidade gerada pelas informações que oferece, situando-se como fator que diferencia produtos ou organizações de seus concorrentes (PRESSMAN; MAXIN, 2016).

Adentro do contexto organizacional, o *software* possui a propriedade de atribuir significado às informações provenientes dos sistemas baseados em computador, por meio da utilização de interfaces que se dedicam a adequar os dispositivos de *hardware* aos objetivos das organizações (SILVA; RIBEIRO; RODRIGUES, 2004).

Sobre a importância do *software*, Sommerville (2007, p. 4) afirma: “Sem *softwares* complexos, não teríamos explorado o espaço, não existiriam a Internet e as modernas telecomunicações e todos os meios de viagem seriam mais perigosos e caros”.

2.3 PROCESSOS DE SOFTWARE

De acordo com Sommerville (2007), processos de *software* são a associação de diversas atividades dedicadas a se obter e manter produtos de *software*.

Segundo Paula Filho (2009), existem organizações que são dotadas de baixa capacitação para produção e que utilizam processos de *software* informais. Estes processos comumente efetuam-se de forma individual, podendo ser passados a outros por meio de

transmissão verbal ou por reprodução. Em contra partida, processos definidos, procedentes de organizações dotadas de alta capacitação para produção de *software*, possuem documentação que contempla todos os seus pormenores, facilitando a preparação dos envolvidos no processo.

Para Sommerville (2007), dentre as atividades exercidas em um processo de *software*, existem as que são fundamentais:

a) Especificação de *software*: são definidas, por clientes e engenheiros, as funcionalidades a serem incumbidas ao *software* e suas restrições;
b) Desenvolvimento de *software*: o *software* é projetado e desenvolvido de acordo com as especificações;
c) Validação de *software*: o *software* é submetido a verificações, destinadas a assegurar a conformidade com as especificações do cliente;
d) Evolução de *software*: o *software* passa por adaptações, visando à satisfação dos requisitos do cliente e do mercado.

2.3.1 Qualidade de Software

Atingir alto nível de qualidade de um produto ou serviço é o objetivo da grande maioria das organizações. Na atualidade, já não é aceitável produtos de baixa qualidade, que não satisfazem as necessidades dos usuários e que apresentam problemas depois de terem sido entregues. Neste sentido, o *software* se assemelha aos demais produtos, sendo necessária a definição da qualidade como objetivo primário em seu planejamento (SOMMERVILLE, 2007).

Para Rincon (2009), pode-se considerar a existência de uma grande dependência das características e serviços oferecidos pelos produtos de *software*, no entanto, parte considerável dos projetos de *software* não atende aos objetivos traçados, em decorrência da falta de processos adequados nas organizações em que são desenvolvidos.

O principal objetivo da Engenharia de Software é garantir a produção de produtos de *software* dotados de qualidade. Contudo, o conceito de qualidade do *software*, quando examinado profundamente, se revela complexo, não podendo ser definido de maneira tão simples (KOSCIANSKI; SOARES, 2007).

Para Crosby (1979) apud (SOMMERVILLE, 2007) a qualidade pode ser verificada mediante a conformidade do produto de *software* com as especificações do mesmo. Essa

definição aplica-se a todos os produtos, mas, é preciso ressaltar que as especificações costumam ser imperfeitas, em razão da dificuldade de determinar especificações para certas características de qualidade, de modo que não apresentem ambiguidade. Além disso, é consideravelmente difícil estruturar uma especificação completa. Portanto, mesmo que o produto esteja em conformidade com as especificações, algumas necessidades dos usuários podem não ter sido consideradas nas especificações.

Deve-se reconhecer os problemas existentes com as especificações e implantar procedimentos para melhorar a qualidade dentro das restrições impostas por uma especificação perfeita. Atributos de *software* como facilidade de manutenção, portabilidade ou eficiência, podem ser considerados como atributos de qualidade fundamentais, uma vez que contribuem para a qualidade percebida do produto de *software*, apesar de não fazerem parte das especificações propriamente (KOSCIANSKI; SOARES, 2007).

A ISO (*International Organization for Standardization*, em português, Organização Internacional de padronização) e a IEC (*International Electrotechnical Comission*, em português, Comissão Internacional Eletrotécnica), órgãos normalizadores de importância internacional reconhecida no seguimento de *software*, se uniram para editar normas internacionais conjuntas. A norma internacional ISO/IEC, estabelece o conceito de qualidade de *software* como sendo o conjunto de características atribuíveis a um produto de *software* capaz de satisfazer necessidades explícitas e implícitas Necessidades explícitas são definidas nos requisitos propostos. Necessidades implícitas são aquelas que, embora não expressas no documento de requisitos, são necessárias para o usuário (WAZLAWICK, 2013).

Contudo, gerenciamento da qualidade de *software*, de acordo com Sommerville (2007), pode ser estruturado em três atividades primordiais:

a) Garantia de qualidade: o estabelecimento de uma estrutura de procedimentos e de padrões organizacionais, que conduzam ao *software* de alta qualidade.
b) Planejamento de qualidade: a seleção de procedimentos e padrões adequados a partir dessa estrutura e a adaptação destes para um projeto específico de *software*.
c) Controle de qualidade: a definição e a aprovação de processos que assegurem que os procedimentos padrões de qualidade de projeto sejam seguidos pela equipe de desenvolvimento de *software*.

2.3.2 Linguagens utilizadas para o desenvolvimento

As linguagens escolhidas para comtemplar à etapa de desenvolvimento da aplicação proposta foram definidas mediante pesquisas realizadas em torno dos recursos *web* mais confiáveis.

HTML é a sigla de *Hypertext Markup Language*, que em português significa Linguagem de Marcação de Hipertexto. Consiste em uma linguagem de marcação amplamente utilizada para a criação de páginas *web*, permitindo a estruturação de documentos que podem ser interpretados pela grande diversidade de máquinas e sistemas da atualidade.

Os documentos HTML são compostos por marcações denominadas *tags* que definem o modo como os elementos do documento são exibidos. Para atribuir estilos aos elementos HTML utiliza-se a linguagem CSS (*Cascading Style Sheet*, em português, folhas de estilo em cascata). A linguagem CSS define os demais aspectos visuais das páginas *web*, como cores, tamanhos, posição e demais atributos de elementos HTML (SILVA, 2008).

É possível facilitar o processo de estruturação da interface de uma aplicação *web* mediante a utilização do *framework* (biblioteca que reúne recursos de um domínio específico) *Bootstrap*. Esta biblioteca é gratuita e de fácil manipulação, além de incluir diversos modelos de *design* baseados em HTML e CSS para tipografia, formulários, botões, tabelas, navegação, modais, e vários outros, assim como componentes em *JavaScript* opcionais (W3SCHOOLS, 2017a).

JavaScript é uma linguagem de programação de *scripts*, interpretada com base em orientação a objetos. Os *scripts* gerados por meio desta linguagem são processados pelo navegador e são capazes de criar efeitos especiais, associados a elementos HTML e propriedades CSS. Utilizar a linguagem *JavaScript* contribui para acrescer qualidade às aplicações *web*, por otimizar a interatividade com os usuários (MACHADO, 2016).

Quando se pretende atribuir agilidade a utilização da linguagem *JavaScript* recomenda-se a manipulação da biblioteca *JQuery*. Além de ser rápida e compacta, a biblioteca *JQuery* fornece diversos recursos convenientes que tornam as tarefas mais inteligíveis e hábeis (W3SCHOOLS, 2017b).

As linguagens HTML e CSS destinam-se a estruturar e definir o visual das páginas *web* e, portanto, não são linguagens de programação. Não é possível programar estruturas de repetição, funções ou declarar variáveis, por exemplo, fazendo uso destas linguagens. Para

este fim são utilizadas as linguagens de programação *web* que são específicas para o desenvolvimento de sites, portais e aplicações *web* de um modo geral.

O PHP (acrônimo recursivo para *"PHP: Hypertext Preprocessor"*, que em português significa Pré-processador de Hipertexto) é uma linguagem de programação *web* livre, capaz de gerar conteúdos dinâmicos. O que difere o PHP das demais linguagens *web* que atuam no lado do cliente é o fato de seu código ser executado no servidor, gerando o HTML que é então enviado para o navegador. Além de muito simples de se implementar, o PHP também possui muitos recursos úteis para o desenvolvimento profissional, motivos pelos quais a linguagem se tornou tão difundida (LOUDON, 2010).

De acordo com Sommerville (2007), no processo de criação de *software*, a modelagem de dados assegura o entendimento dos requisitos do sistema por meio da construção de modelos abstratos baseados em notações UML (*Unified Modeling Language*, em português, Linguagem de Modelagem Unificada). A UML descreve um *software* por meio de diagramas, para compor a documentação estrutural do sistema e para transformar os requisitos de forma a serem entendidos pelos desenvolvedores (GUDWIN, 2010).

Dentre os diagramas empregados no processo de modelagem de sistemas, segundo Sampaio (2007), pode-se mencionar como sendo primordiais:

a) Diagrama de casos de uso: documenta as funcionalidades do sistema considerando o ponto de vista do usuário;
b) Diagrama de sequência: demonstra como as informações fluem entre os objetos do sistema no tempo de execução das operações;
c) Diagrama de atividade: apresenta o fluxo de atividades no interior dos processos e as relações entre as atividades;
d) E diagrama de classes: descreve os diversos tipos de objetos no sistema e o relacionamento entre eles.

2.3.3 Aplicações e Sistemas para o Desenvolvimento

Para o desenvolvimento do *software* proposto considerou-se que não houvessem custos relacionados à aquisição de ferramentas e, portanto, foram utilizadas apenas ferramentas gratuitas.

O desenvolvimento de diagramas referentes a modelagem de dados realizou-se mediante o uso da ferramenta *web Creately*, um *software* livre específico para este fim. Esta

ferramenta é essencialmente voltada para a criação de diagramas e é altamente consolidada devido a sua variedade de recursos disponibilizados *online* (CREATELY, 2017).

Comumente são utilizados pacotes contendo os componentes indispensáveis ao desenvolvimento *web*, normalmente configurados previamente, para facilitar a instalação, utilização e gerenciamento dos mesmos. VertrigoServ é um produto que inclui os programas básicos necessários para se montar um servidor HTTP (*HyperText Transfer Protocol*, em português, Protocolo de Transferência de Hipertexto) com suporte a PHP e MySQL (sistema de gestão de banco de dados). Disponibiliza o servidor *web* Apache (servidor *web* - HTTP), o PHP, o MySQL, o *Zend Optimizer* (ferramenta para otimizar a performance da aplicação) e o PhpMyAdmin (ferramenta para gerir as bases de dados MySQL) (VERTRIGOSERV WAMP SERVER, 2017).

O desenvolvimento de interfaces e funções em aplicações *web* pode ser diretamente solucionado por uma única ferramenta que suporte uma variedade de linguagens de marcação e programação como a IDE (*Integrated Development Environment* - Ambiente de Desenvolvimento Integrado) Sublime Text. O Sublime Text IDE é um ambiente de desenvolvimento integrado gratuito e de código aberto, que oferece para o desenvolvimento de *software* profissional diversas funcionalidades e recursos nativos, além de ser muito simples de ser instalado e utilizado (SUBLIME TEXT, 2017).

Segundo Tableless (2017) o MVC (*Model-view-controller*: Modelo-Visão-Controlador) é um padrão de arquitetura de *software* para implementar interfaces de usuário. Ele divide um determinado sistema de *software* em três partes interligadas, de modo a separar as representações internas de informações das formas em que a informação é apresentada ou aceita pelo usuário.

a) Modelo (*Model*): camada responsável pela leitura e escrita de dados, e também de suas validações;
b) Visão (*View*): compreende a camada de interação com o usuário, ou seja, a camada que realiza a exibição dos dados;
c) Controlador (*Controller*): é o responsável por receber todas as requisições do usuário. Seus métodos chamados *actions* (ações) comandam páginas, controlando qual *model* utilizar e qual *view* será apresentada ao usuário do sistema.

Junto ao Sublime Text IDE foi utilizado o CodeIgniter, que consiste em um *framework* de desenvolvimento de aplicações em PHP, desenvolvido sobre o paradigma da programação Orientada a Objetos e com base no padrão de arquitetura de *software* MVC. Por meio da

manipulação deste *framework* é possível manter o código da implementação bem organizado, elegante e fácil de ser compreendido (CODEIGNITER, 2017).

Para criação das imagens que foram disponibilizadas no sistema foi utilizada a versão de avaliação do *software* de *design* gráfico CorelDRAW *Graphics Suite* 2017. O CorelDRAW Graphics Suite 2017 é o programa de *design* gráfico mais recente oferecido pela Corporação Corel. O *software* disponibiliza um amplo conjunto de recursos avançados e intuitivos para *design* gráfico e demonstra ser muito fácil para iniciantes, dispondo de um *tour* (passeio) de inicialização (CORELDRAW, 2017).

2.4 MODELO DE DESENVOLVIMENTO INCREMENTAL

Em alguns projetos de *software* os requisitos são incialmente bem definidos, pois, evidencia-se a necessidade de prover o fornecimento imediato de um determinado conjunto funcional aos usuários, para que, posteriormente, seja possível melhorar e expandir suas funcionalidades em versões posteriores. Nessas situações, é sugerida a utilização de um modelo de desenvolvimento incremental (PRESSMAN; MAXIN, 2016).

Segundo Mills (1980) apud (SOMMERVILLE, 2007) a abordagem de desenvolvimento incremental conduz a redução do retrabalho durante o processo de desenvolvimento, proporcionando aos clientes maior interação com a produção do sistema de *software*.

Para Pressman e Maxin (2016) este tipo de modelo de processo combina elementos dos fluxos de processos tanto lineares quanto paralelos, ou seja, emprega sequencias lineares de forma escalonada. Cada sequência linear tende a gerar um incremento de *software* entregável e que gere valor para o cliente.

Incialmente é apresentado um esboço ao cliente para que identifique as funcionalidades a serem fornecidas pelo sistema. Definidas as funcionalidades mais importantes, uma série de estágios são estabelecidos, cada estágio compreendendo diferentes compostos de funcionalidades para serem entregues, de forma que as funcionalidades prioritárias sejam entregues primeiramente (SOMMERVILLE, 2007).

O primeiro incremento de uma produção que utiliza um modelo de processo de desenvolvimento incremental compreende o essencial do produto, apresentando as funcionalidades básicas para que o *software* possa ser utilizado pelo cliente. Desse modo,

durante a produção do primeiro incremento deve-se priorizar os recursos primordiais, adiando a implementação de funcionalidades complementares (PRESSMAN; MAXIN, 2016).

Posteriormente ao término do primeiro incremento o cliente coloca em operação e avalia a parte que foi entregue, fornecendo um *feedback* que pode ser utilizado para realizar melhorias. Por conseguinte, o próximo incremento é planejado considerando as modificações do primeiro incremento, caso tenha-se feito necessário, e em conformidade com o *feedback* do cliente. Depois da liberação de cada incremento o processo é repedido até que o produto esteja finalmente concluído (DEVMEDIA, 2017).

2.5 MÉTODOS ÁGEIS

O demasiado crescimento no setor de tecnologia juntamente com a necessidade cada vez maior de sistemas computacionais estimula a evolução das empresas responsáveis pela produção de *software* (SOARES, 2004). O vertiginoso avanço tecnológico, a necessidade por soluções cada vez mais complexas, as intervenções do meio externo, mudanças nos requisitos, entre outros problemas afetam diretamente este setor. Para contrabalancear a existência desses problemas, a Engenharia de Software propõe metodologias de desenvolvimento, que permitem a organização nos processos desde sua especificação até a implantação, assim como fazem os métodos ágeis (SOMMERVILLE, 2007).

De acordo com Sommerville (2007), em meados de 1980 e 1990, a construção de *software* era baseada em um conceito de que, para se obter bons produtos, dever-se-ia aplicar ampla formalidade, mediante planejamento minucioso, uso de métodos de análise e projetos controlados por processos rigorosos. Para Ambler e Jeffries (2002) essa percepção deve-se a preocupação inicial da comunidade de Engenharia de Software em promover sistemas grandes, estáveis e de longa duração.

Contudo, essa abordagem de desenvolvimento gerava grande insatisfação quando aplicada a sistemas de pequenas e médias empresas. As despesas operacionais eram desproporcionais e o tempo empregado no planejamento era maior do que o tempo empregado na implementação e testes do *software*. Conforme os requisitos mudavam, o retrabalho era inevitável, ao passo que a especificação e o projeto precisavam mudar de acordo com o prospecto (SOMMERVILLE, 2007).

Frente à insatisfação gerada por esta abordagem sobrecarregada, um grupo de desenvolvedores propuseram novos métodos que se demonstravam mais ágeis e que se

concentravam, em sua maior parte, no *software* ao invés de se dedicar a documentação. A partir de então criou-se o manifesto ágil que abrangia melhores práticas para o desenvolvimento de *software*. Graças ao Manifesto diversas metodologias e *frameworks* ágeis se uniram e ganharam espaço.

Sobre os métodos ágeis, Sommerville afirma ainda:

> 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. Eles destinam-se a entregar um *software* de trabalho rapidamente aos clientes, que podem então propor novos requisitos e alterações a serem incluídos nas iterações posteriores do sistema (SOMMERVILLE, 2007).

As metodologias ágeis surgiram com o objetivo de focar nas pessoas e não nos processos de desenvolvimento, portanto, enfatizam a comunicação em tempo real. Ressalta-se a atenção por otimizar o processo de desenvolvimento visando garantir a satisfação do cliente por meio de entregas continuas de produtos funcionais (SOARES, 2004).

Segundo Sommerville (2007), dentre os métodos ágeis mais conhecidos e utilizados podem ser referidos: *Extreme Programming* (Programação Extrema, XP), *Scrum*, *Crystal*, Desenvolvimento de *software* Adaptativo, DSDM (*Dynamic Software Development Method*, em português, Método dinâmico de desenvolvimento de *software*) e Desenvolvimento Dirigido a Características (*Feature Driven Development*). O sucesso desses métodos levou a integração com métodos mais tradicionais de desenvolvimento baseados na modelagem do sistema, resultando no conceito de modelagem ágil e instâncias ágeis (AMBLER e JEFFRIES, 2002). Ainda que esses métodos sejam todos baseados no modelo de desenvolvimento incremental, definem diferentes processos para alcançar tal finalidade.

Os diversos métodos ágeis citados integram os princípios do manifesto ágil de diferentes maneiras. Todavia, para comtemplar tais métodos de maneira sumária serão abordados, nas próximas sessões, os dois métodos ágeis mais utilizados: *Extreme Programming* e *Scrum*.

2.6 EXTREME PROGRAMMING

A *Extreme Programming* (XP) é um dos mais conhecidos e utilizados dos métodos ágeis. O nome foi definido por Beck (2000) apud (SOMMERVILLE, 2007), pelo fato de a

abordagem ter sido desenvolvida com intuito de elevar as boas práticas já existentes a níveis reconhecidos como extremos.

De acordo com Ambler e Jeffries (2002), na *Extreme Programming* os requisitos são representados pelas chamadas estórias de usuário, e posteriormente são destrinchados dando origem a um fluxo de tarefas a serem desempenhadas. Os programadores desempenham suas atividades trabalhando em pares e se dedicam a desenvolver testes antes de começar a implementação. Desse modo, quando a implementação é integrada ao sistema, todos os testes devem ser executados com êxito.

Para Sommerville (2007) o XP abrange uma série de práticas que refletem os princípios dos métodos ágeis:

a) O desenvolvimento incremental é sustentado por pequenos e frequentes *releases* (lançamentos) do sistema, havendo um curto intervalo entre eles. Os requisitos são baseados em cenários, utilizados como base para definir as funcionalidades que representam importância para o cliente e que deve ser integradas a um incremente do sistema;
b) A relação com o cliente é estabelecida por meio do envolvimento com a equipe durante o processo de desenvolvimento. O cliente possui um representante que se responsabiliza por definir os testes de aceitação do sistema;
c) O princípio ágil de focar em pessoas no lugar de focar em processos é mantido por meio da técnica de programação em pares, da propriedade coletiva do código da implementação e por meio de um processo de desenvolvimento sustentável que não envolve horas de trabalho excessivamente longas;
d) As mudanças são aceitas por meio de *releases* contínuos para os clientes, do desenvolvimento baseado em implementar os testes com antecedência, por meio da refatoração e da integração contínua de novas funcionalidades;
e) Para alcançar a simplicidade o XP restringe os desenvolvedores a implementar apenas as funcionalidades de necessidade imediata.

No processo XP, os clientes estão intimamente envolvidos na especificação e priorização dos requisitos do sistema, se caracterizando com membros da equipe. Dessa forma, é possível que o cliente discuta os cenários a equipe, para que juntos, desenvolvam um cartão de estória, englobando as necessidades verificadas (BOEHM 2006).

2.6.1 O processo XP

De acordo com Sommerville (2007) a *Extreme Programming* emprega uma abordagem orientada a objetos que envolvem uma série de métodos e regras no contexto de quatro atividades:

a) Planejamento: o planejamento tem início com a atividade de levantamento de requisitos que possibilita que os desenvolvedores compreendam o ambiente de negócio que o produto de *software* irá atuar. Esta atividade resulta em uma série de estórias de usuário que descrevem os resultados esperados, as características e funcionalidades do sistema;
b) Projeto: o XP emprega o princípio de preservar a simplicidade. Portanto define a criação de projetos simples, como guias de implementação para as estórias criadas;
c) Desenvolvimento: Depois que as estórias de usuário foram definidas e a atividade de elaboração do projeto foi realizada, a equipe desenvolve uma série de testes de unidade. Uma vez que os testes tenham sido criados os desenvolvedores podem focar no que deve ser implementado para ser aprovado no teste. O desenvolvimento emprega a abordagem de programação em pares que oferece um mecanismo de resolução de problemas e garantia de qualidade em tempo real;
d) Testes: os testes de unidade devem ser desenvolvidos para serem automatizados, podendo ser repetidos com facilidade. Para que a equipe XP tenha indicadores do progresso e para que sejam identificados os problemas logo no início do projeto, os testes de integração e validação podem ser realizados diariamente.

2.7 SCRUM

De acordo com Ferreira (2005) *Scrum* é um *framework* utilizado desde o início de 1990 para desenvolvimento e gestão de projetos complexos. É empregado não somente ao desenvolvimento de *software* como também em projetos dos demais segmentos. Propõe uma série de atividades e ferramentas que podem ser utilizadas e adaptadas a cada projeto, com o objetivo de garantir a satisfação do cliente diante dos resultados imediatos e da qualidade.

O *Scrum* dispensa documentação excessiva, redirecionando os esforços para o planejamento em conjunto com o cliente, estabelecimento de etapas bem definidas e desenvolvimento disposto a mudanças (SILVA *et al*., 2010).

Os times *Scrum* se associam com papéis, eventos, artefatos e regras. Cada componente do *framework* destina-se a um objetivo específico, sendo, cada um deles, essencial para o uso e sucesso do projeto baseado em *Scrum*. As regras do *framework* constituem as relações e interações entre os eventos, papéis e artefatos (PEREIRA *et al*., 2007).

2.7.1 Fundamentação do Scrum

Segundo Cruz (2006), caracterizando-se como iterativo e incremental, o *Scrum* aborda a gestão de processos através de uma base empírica, considerando que o conhecimento surge por meio de experiências vivenciadas e nas decisões tomadas diante de tais experiências. Isso possibilita que o controle dos riscos seja aperfeiçoado assim como as decisões futuras.

A implementação de controle de processo empírico é baseada em três pilares:

a) Transparência: aspectos significativos do processo devem estar visíveis aos responsáveis pelos resultados. Esta transparência requer pormenores definidos por um padrão comum para que os observadores compartilharem um mesmo entendimento do que está sendo visto.
b) Inspeção: os usuários *Scrum* devem, frequentemente, inspecionar os artefatos e o progresso para detectar variações. Esta inspeção não deve, no entanto, ser tão frequente que atrapalhe a própria execução das tarefas. As inspeções são mais benéficas quando realizadas de forma diligente por inspetores especializados no trabalho a se verificar.
c) Adaptação: o processo *Scrum* deve se adaptar à realidade da organização e ao projeto executado assim como o produto desenvolvido deve ser ajustado às mudanças que surgirem. Dessa forma obtém-se uma versão exclusiva do *Scrum* mais apropriada ao contexto da organização e o produto desenvolvido torna-se mais próximo do que o cliente necessita.

2.7.2 Time Scrum

O *Scrum Team* ou Time *Scrum* é constituído pelo *Product Owner* (Dono do Produto), o *Dev Team* (Time de Desenvolvimento) e o *Scrum Master*. Os times *Scrum* costumam ser auto organizáveis e multifuncionais. Por serem auto organizáveis os membros do *Scrum Team* escolhem a melhor forma para exercer suas atividades, dispensando a necessidade da interferência de um gestor de fora do time. Devido à característica de multifuncionalidade, os times *Scrum* possuem todas as competências cruciais realizar o trabalho necessário (CARVALHO e MELLO, 2009). O *Scrum Team* é projetado para aperfeiçoar a flexibilidade, criatividade e produtividade, entregando produtos de forma iterativa e incremental, maximizando as oportunidades de *feedback* (comentários). Entregas incrementais de um produto garantem que uma versão potencialmente funcional do produto esteja sempre disponível (SCHWABER; SUTHERLAND, 2009).

2.7.3 O Product Owner

Segundo Cruz (2006) o *Product Owner* é o responsável por maximizar o valor do produto e do trabalho do *Dev Team*. Como isso é feito pode variar através das organizações, *Scrum Teams* e indivíduos. O *Product Owner* é a única pessoa responsável por gerenciar o *Product Backlog* (Histórico do produto) devido ao seu conhecimento acerca do produto desejado pelo cliente. De acordo com Carvalho e Mello (2009) o gerenciamento do *Product Backlog* abrange:

- Manifestar com clareza os itens do *Product Backlog*;
- Ordenar os itens do *Product Backlog* de acordo com suas prioridades, ou seja, seu valor para o cliente, a fim de alcançar melhor as metas e missões;
- Garantir o valor do trabalho realizado pelo *Dev Team*;
- Garantir que o *Product Backlog* seja visível, transparente, claro para todos, e mostrar como o *Scrum Team* irá trabalhar;
- E assegurar que o *Dev Team* entenda os itens do *Product Backlog* no nível necessário.

Para Schwaber e Sutherland (2009) o *Product Owner* representa o desejo do cliente sobre as funcionalidades do produto, portanto, para propor uma alteração nas prioridades dos itens de *Backlog* deve-se convencê-lo primeiramente. Para que o *Product Owner* tenha

sucesso no desempenho de suas responsabilidades, toda a organização deve respeitar as suas decisões. Dito isso, o *Dev Team* não possui permissão para agir de acordo com dizeres de terceiros.

2.7.4 Sprint

Sprint é um *time-box* (caixa de tempo) com duração de mês ou menos dedicado a criar uma versão incremental potencialmente utilizável do produto. *Sprints* tem durações coerentes em todo o esforço de desenvolvimento. Uma nova *Sprint* inicia imediatamente após a conclusão da *Sprint* anterior (PEREIRA *et al*., 2007).

Para Carvalho e Mello (2009) as *Sprints* abrangem uma reunião de planejamento da *Sprint*, reuniões diárias, o trabalho de desenvolvimento, uma revisão da *Sprint* e a retrospectiva da *Sprint*.

Cada *Sprint* tem a definição do que é para ser construído, um plano projetado e flexível que irá guiar a construção, o trabalho e o resultado do produto. *Sprints* são limitadas a um mês para garantir que o risco será também limitado ao custo de um mês. Além disso, quando o tempo da *Sprint* é muito longo, a definição do que será construído pode mudar, a complexidade pode aumentar e o risco pode crescer. *Sprints* permitem previsibilidade que garante a inspeção e adaptação do progresso em direção à meta a cada mês (SILVA *et al*., 2010).

De acordo com Schwaber e Sutherland (2009) uma *Sprint* pode ser cancelada antes de seu *time-box* terminar. O *Product Owner* é único com autoridade para cancelar a *Sprint*, ainda que possa ser influenciado pelas partes interessadas, pelo *Dev Team* ou pelo *Scrum Master*. A *Sprint* poderá ser cancelada se o objetivo da *Sprint* se tornar obsoleto. Isto pode ocorrer se a organização mudar sua direção ou se as condições do mercado ou das tecnologias mudarem. Se uma parte do trabalho estiver potencialmente utilizável, o *Product Owner* o considera, estimando novamente os demais itens de *Backlog* incompletos. O cancelamento de *Sprints* consome muitos recursos, considerando a necessidade de todo time Scrum se reagrupar em outra reunião de planejamento da *Sprint* para iniciar outra *Sprint*. Cancelamentos de *Sprints* são muito traumáticos para o Time *Scrum*, portanto, são evitados a todo custo, tornando-se muito incomuns.

2.7.5 O Time de Desenvolvimento

O *Dev Team* ou Time de Desenvolvimento é constituído por profissionais que possuem habilidades e competências técnicas necessárias para realizar o trabalho de entregar uma versão utilizável que incrementa o produto ao final de cada *Sprint*. Somente os integrantes do *Dev Team* se dedicam a criar incrementos. São estruturados e autorizados pela organização para coordenar e gerenciar seu próprio trabalho. Com resultado, observa-se o aperfeiçoamento da eficiência e eficácia do *Dev Team* (SCHWABER; SUTHERLAND, 2009).

O *Dev Team* se caracteriza por ser auto organizado. Dessa forma nem mesmo o *Scrum Master* pode dizer ao *Dev Team* como transformar o *Product Backlog* em incrementos de funcionalidades. O *Dev Team* é também multifuncional, possuindo todas as habilidades necessárias, enquanto equipe, para criar o incremento do produto. Ainda, no *Dev Team* os integrantes podem apresentar habilidades especializadas e área de especialização distinta, mas, a responsabilidade sobre o trabalho realizado no projeto abrange todos igualmente (CARVALHO e MELLO, 2009).

O tamanho ideal do *Dev Team* é o que permite o projeto se manter ágil e abranja as competências necessárias para completar o trabalho (CRUZ, 2006). Times de desenvolvimento muito pequenos, com menos de três integrantes, diminuem a interação e resultam na diminuição do ganho de produtividade. Times de desenvolvimento menores podem encontrar restrições de habilidades durante a *Sprint*, tornando-se incapazes de entregar um incremento potencialmente utilizável. Diante de times de desenvolvimento muito grandes, com mais de nove integrantes, é exigida muita coordenação, gerando muita complexidade para um processo empírico comandar (SCHWABER; SUTHERLAND, 2009).

2.7.6 O Scrum Master

De acordo com Cruz (2006) o *Scrum Master* é responsável por garantir que os conceitos e técnicas do *Scrum* sejam devidamente entendidos e aplicados. É encarregado de auxiliar aqueles que estão fora do *Scrum Team* a entender de que forma podem colaborar com o mesmo. Se dedica a modificar as interações dos agentes externos de forma que estas interações possam potencializar os valores gerados pelo *Scrum Team*.

Schwaber e Sutherland (2009) afirmam que o *Scrum Master* auxilia o *Product Owner* de várias formas:

- Encontrando técnicas para o gerenciamento efetivo do *Product Backlog*;
- Comunicando a visão, objetivo e itens do *Product Backlog* para o *Dev Team*;
- Ensinando ao *Scrum Team* a criar itens do *Product Backlog* de forma clara e concisa;
- Compreendendo a longo-prazo o planejamento do produto no ambiente empírico;
- Compreendendo e praticando a agilidade;
- Facilitando os eventos *Scrum* conforme necessários.

O *Scrum Master* trabalha também com o *Dev Team*. Ele treina o *Dev Team* a ser auto gerenciável e interdisciplinar, estimula a criação de produtos de alto valor, remove impedimentos para o progresso do projeto e desempenha papel de facilitador nos eventos *Scrum* (CARVALHO e MELLO, 2009).

O *Scrum Master* é responsável por liderar e realizar treinamentos na organização diante da adoção do *Scrum*. É sua responsabilidade planejar a implantação do *Scrum*, auxiliando funcionários e partes interessadas a compreender e tornar aplicável o *Scrum* e o desenvolvimento de produtos com base no método empírico. O trabalho do *Scrum Master* gera mudanças que aumentam a produtividade do *Scrum Team* (SCHWABER; SUTHERLAND, 2009).

2.7.7 Eventos Scrum

O *Scrum* possui uma série de eventos prescritos, sendo que cada evento possui uma duração máxima. Quando uma *Sprint* começa, sua duração é determina e não pode ser reduzida ou aumentada. Os demais eventos podem ser encerrados assim que os objetivos tenham sido alcançados garantindo que uma quantidade adequada de tempo seja gasta sem permitir perdas no processo (SILVA *et al*., 2010).

Cada evento *Scrum* representa uma oportunidade inspeção e adaptação. Estes eventos são projetados para permitir transparência e inspeção criteriosa. Deixar de praticar qualquer dos eventos *Scrum* resulta na redução da transparência e da perda de oportunidade de efetuar inspeções e adaptar o que for necessário (PEREIRA *et al*., 2007).

2.7.8 Reunião de Planejamento da Sprint

Durante a reunião de planejamento da *Sprint* é planejado o trabalho a ser realizado. Este planejamento é realizado com a colaboração de todo o Time *Scrum*. O *time-box* da Reunião de planejamento da *Sprint* possui o máximo de oito horas para o planejamento de uma *Sprint* de um mês de duração. Para planejar *Sprints* menores, o tempo da reunião é reduzido. O *Scrum Master* assegura que o evento aconteça que os participantes entendam seu propósito e ensina o *Scrum Team* a manter-se dentro dos limites do *time-box* (PEREIRA *et al*., 2007).

Segundo Schwaber e Sutherland (2009) a reunião de planejamento da *Sprint* abrange as seguintes questões:

a) O que pode ser entregue: O *Dev Team* se dedica para prever as funcionalidades que serão desenvolvidas durante a *Sprint*, enquanto o *Product Owner* discute o objetivo que a *Sprint* deve produzir e os itens de *Product Backlog* que irão compor o objetivo da *Sprint*. Para a reunião é necessário o *Product Backlog*, o mais recente incremento do produto e o desempenho passado do *Dev Team*. Somente o *Dev Team* pode avaliar o que pode ser completado ao longo da próxima *Sprint*. Com base em sua capacidade o *Dev Team* determina a meta da *Sprint*.
b) O que deve ser feito: O *Dev Team* decide como irá construir as funcionalidades definidas e transformá-las em um incremento de produto utilizável. Os itens de *Backlog* do Produto selecionados, junto ao plano de entrega destes itens são chamados de *Sprint Backlog*. O *Dev Team* inicia o desenho do sistema e do trabalho necessário para converter os itens de *Backlog* em um incremento. O trabalho é planejado pelo *Dev Team* para os primeiros dias da *Sprint*. Ao final do planejamento da *Sprint*, o *Dev Team* deve explicar ao *Product Owner* e ao *Scrum Master* como pretende para completar o objetivo da *Sprint* e criar o incremento esperado.

2.7.9 Reunião Diária

A *Daily Scrum* (Reunião Diária) é um evento que dura apenas quinze minutos, para que o *Dev Team* possa sincronizar as atividades e criar um plano para o dia que se inicia. Esta reunião é feita para inspecionar o trabalho desde a última Reunião Diária, e prever o trabalho que deverá ser feito antes da próxima Reunião. O local e horário para a reunião devem ser os

mesmos sempre, para que se evitem confusões e atrasos (PEREIRA *et al*., 2007). Ferreira (2005) estabelece que durante a reunião diária os membros do *Dev Team* devem esclarecer as seguintes questões:

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

O *Dev Team* usa a Reunião Diária para inspecionar o progresso em direção ao objetivo da *Sprint* e para inspecionar se o progresso tende para completar o trabalho do *Sprint Backlog*. Todos os dias, o *Dev Team*, durante a reunião diária, deve discutir como o mesmo pretende trabalhar em conjunto para completar o objetivo da *Sprint*, podendo se reunir imediatamente após a Reunião Diária para discussões detalhadas, ou para adaptar e/ou planejar novamente o restante do trabalho da *Sprint* (SCHWABER; SUTHERLAND, 2009). O *Scrum Master* assegura que o *Dev Team* realize a reunião, mas o *Dev Team* é único responsável por conduzir a Reunião Diária. O *Scrum Master* apenas ensina ao *Dev Team* como manter a Reunião Diária dentro do tempo estimado de quinze minutos, reforçando a regra de que apenas o *Dev Team* participe desta reunião (CRUZ, 2006).

Reuniões Diárias melhoram as comunicações, dispensam a necessidade de outras reuniões, verificam a existência de impedimentos e os removem e propiciam rápidas tomadas de decisão (FERREIRA, 2005).

2.7.10 Revisão da Sprint

A *Sprint Review* (Revisão da *Sprint*) é executada ao final da *Sprint* para inspecionar o incremento e adaptar o *Product Backlog* caso se verifique necessário. Durante a *Sprint Review* o *Scrum Team* e as partes interessadas discutem sobre o que foi realizado durante a *Sprint*. Tendo como base o que foi realizado e as mudanças no *Product Backlog* durante a *Sprint*, os participantes debatem sobre mudanças que podem ser feitas para aperfeiçoar o processo e obtenção de valor (PEREIRA *et al*., 2007).

Esta reunião é informal, destinada a motivar e colher *feedbacks*, oportunizando a colaboração de todo o *Scrum Team* e *Stakeholders* (Partes interessadas). Tem duração máxima de quatro horas de duração para uma *Sprint* de um mês, podendo durar menos tempo se tratando de *Sprints* menores. O *Scrum Master* é responsável por garantir que o evento

aconteça e que os participantes entendam o seu objetivo (SCHWABER; SUTHERLAND, 2009).

Os participantes da *Sprint Review* incluem o *Scrum Team* e os *Stakeholders* chaves, convidados pelo *Product Owner*. O *Product Owner* esclarece quais itens do *Product Backlog* foram implementados e quais itens ainda serão desenvolvidos. O *Dev Team* discute o que ocorreu positivamente durante a *Sprint*, quais problemas cominaram e como estes problemas foram superados. Posteriormente, o *Dev Team* demonstra o trabalho que está pronto para ser entregue e responde as questões sobre o incremento. O *Product Owner* discute o *Product Backlog* e projeta as prováveis datas de conclusão baseado no progresso até o momento, caso seja necessário (PEREIRA *et al*., 2007).

Todos os participantes colaboram opinando sobre o que se fazer em seguida, fornecendo informações importantes para a reunião de planejamento da próxima *Sprint*. É efetuada uma análise de como o mercado ou o uso potencial do produto pode ter mudado e o que é mais importante a se fazer em seguida para que o produto e os processos possam se adaptar. Também é realizada uma análise da linha do tempo, orçamento, potenciais capacidades, e mercado para a próxima versão esperada do produto (SCHWABER; SUTHERLAND, 2009).

O resultado da *Sprint Review* da Sprint é um *Product Backlog* revisado que define o provável *Product Backlog* para a próxima *Sprint*. O *Product Backlog* pode também ser ajustado completamente para atender novas oportunidades (SILVA *et al*., 2010).

2.7.11 Os artefatos do Scrum

No *Scrum* alguns documentos são essenciais para que o trabalho seja realizado com sucesso e o método seja aplicado de forma adequada, visando à obtenção do melhor resultado em potencial. A documentação e os diversos elementos gráficos utilizados ao longo da aplicação do *Scrum* são denominados artefatos e cada um possui uma função específica importante para maximizar a transparência das informações chave (PEREIRA *et al*., 2007).

De acordo com Silva *et al*. (2010), dentre os principais artefatos do *Scrum* podem ser citados:

a) *Backlog*: é uma lista de requisitos da aplicação que será necessária para o seu entendimento e desenvolvimento. O *Backlog* é dividido em duas partes, o *Product Backlog* e o *Sprint Backlog*.

- *Product Backlog*: corresponde a lista de requisitos da aplicação propriamente dita, onde estão contidas todas as informações pertinentes ao projeto e tudo que precisa ser desenvolvido para que a aplicação seja entregue.
- *Sprint Backlog*: se trata de um conjunto de informações necessárias para a produção de um incremento e finalização de uma *Sprint*, criado a partir do *Produc Backlog*.

b) Estórias: se trata de uma descrição clara e objetiva, disposta de forma resumida, da funcionalidade a ser desenvolvida.

c) *Burndown*: corresponde a um gráfico que demonstra a linha de esforço frente aos trabalhos que precisam ser realizados. Os eixos que forma esse gráfico analisam a quantidade de trabalho a ser completado (eixo y) e as datas ou dias de execução (eixo x). O *Burndown*, assim como o *Backlog*, também é divido em duas partes, um gráfico para o Produto e outro para a *Sprint*.

d) *Taskboard*: é uma espécie de quadro onde cada estória é colada, de forma que seja possível verificar quais atividades estão na fila de desenvolvimento, quais foram realizadas e quais estão sendo desenvolvidas.

2.7.12 Transparência dos Artefatos

De acordo com Ferreira (2005) o *Scrum* declara a necessidade por transparência. Decisões para maximizar o valor e controle de riscos são embasadas na percepção da circunstancia dos artefatos. Na medida em que a transparência é ampla, as decisões são embasadas de modo seguro. Uma vez que os artefatos não são inteiramente transparentes, as decisões podem ser imprecisas, valores podem ser reduzidos e riscos podem aumentar.

Segundo Carvalho e Mello (2009) o *Scrum Master* deve trabalhar junto ao *Product Owner*, *Dev Team* e *Stakeholders* para verificar se os artefatos estão convenientemente transparentes. O trabalho do *Scrum Master* é manter juntamente ao Time *Scrum* o crescimento da transparência dos artefatos.

2.8 AS VANTAGENS DO SCRUM

O foco do método ágil *Scrum* está no gerenciamento e desenvolvimento de sistemas complexos dotados de alta qualidade. Os Times *Scrum* são multifuncionais, buscam as habilidades técnicas necessárias para a realização do trabalho, e também são auto organizáveis, a própria equipe descobre a melhor forma de realizar o trabalho (SCHWABER; SUTHERLAND, 2009).

Diversas vantagens são verificadas no método ágil *Scrum*, segundo as definições de Carvalho e Mello (2009):

- Comprometimento da equipe: a equipe participa ativamente da definição das atividades e cronogramas, portanto, o nível de comprometimento e a motivação são maiores;
- Visualização do projeto: todos os membros da equipe possuem uma visão ampla do projeto em tempo real. É possível que qualquer membro do *Scrum Team* tenha acesso ao histórico do produto e utilize suas informações.
- Redução de *bugs* (erros): o *Scrum* prioriza a qualidade e a funcionalidade dos produtos desenvolvidos, portanto, a quantidade de erros e retrabalho é consideravelmente reduzida.
- Atualização de prioridades: o *Scrum* é flexível quanto à definição e alteração de prioridades e da sequência de atividades, admitindo mudanças nos requisitos e ajustes nos processos a qualquer momento.
- Antecipação das funcionalidades com maior valor de negócio: a prioridade de entrega é definida com base nas funcionalidades que proporcionam maior valor ao negócio do cliente.

Considerando as informações sobre os aspectos positivos que tornaram o *Scrum* o mais utilizado dentre as abordagens que empregam os conceitos de gerência ágil de projetos, o referido método foi escolhido para compor o pilar do sistema desenvolvido.

2.9 PROTOTIPAÇÃO DE SOFTWARE

Um protótipo de *software* é uma versão inicial de um sistema utilizada para exibir conceitos, avaliar possibilidades e desvendar os problemas abordados e suas soluções.

Os protótipos de sistemas de *software* possibilitam que os usuários avaliem como o sistema apoia suas atividades. Por meio do uso de protótipos os usuários podem ter novas ideias a respeito dos requisitos do sistema e encontrar falhas (SOMMERVILLE, 2007).

Segundo Pressman e Maxin (2016) existem situações que demandam a prototipação ao se iniciar a análise do projeto de *software*, pois esta conjuga-se como abordagem da Engenharia de Software voltada a atender asseguradamente os requisitos do usuário.

Sobre a definição de requisitos realizada com mais precisão por meio do desenvolvimento de protótipos, Koscianski e Soares (2007) afirmam que o uso de protótipos permitem definir com rigor os requisitos de um *software*, de forma que os recursos da organização possam ser aplicados de maneira conveniente ao objetivo de cumprir com esses requisitos.

2.10 APLICAÇÕES WEB

A célere expansão das aplicações *web*, relacionando as suas finalidades e proporção de suas utilizações, tem resultado em diferentes repercussões no modo de vida das pessoas ao redor do mundo (GINIGE; MURUGESAN, 2001).

Para Winckler e Pimenta (2002), desde o começo da utilização das aplicações *web*, a tecnologia para produção destes *softwares* expande notavelmente, permitindo o desenvolvimento de aplicações cada vez mais complexas.

Atentando-se ao fato de as aplicações *web* caracterizarem uma tecnologia que evoluiu do *software* convencional, evidencia-se a preocupação por aplicar conceitos de Engenharia às aplicações *web*, do mesmo modo das demais, visando à obtenção de sistemas dotados de qualidade. O que difere o modo de produção das aplicações *web* é a necessidade de uma estrutura diferente, considerando o ambiente no qual as aplicações funcionarão, ou seja, levando-se em conta que sua disponibilização e execução acontecerão na *web* (PRESSMAN; MAXIN, 2016).

Winckler e Pimenta (2002) afirmam:

> Processos de desenvolvimento para aplicações de *software web* devem produzir representações para projeto de aspectos de aplicações tradicionais, como estrutura e funcionalidades; e também para aspectos orientados para *Web*, como navegação e apresentação (WINCKLER; PIMENTA, 2002).

De acordo com Ginige e Murugesan (2001), a qualidade de aplicações *web* é avaliada basicamente de acordo com a percepção e conhecimentos de desenvolvedores especialistas da área de desenvolvimento *Web*. Todavia, para atender as particularidades que se referem especificamente ao desenvolvimento *Web* e manter a visão de se priorizar a qualidade, é

necessário considerar a utilização de conceitos da Engenharia *Web*, que se dedica unicamente a isso por meio do manuseio de artefatos próprios.

2.11 INTERFACE

Usabilidade é um conceito utilizado para determinar a qualidade da interação do usuário com uma determinada interface (PREECE, ROGERS e SHARP, 2013). Por meio da priorização da usabilidade durante o desenvolvimento de uma interface é possível contribuir grandemente com o objetivo de se alcançar alta qualidade do sistema de *software*.

Winckler e Pimenta afirmam:

> É necessário reconhecer que a usabilidade está relacionada ao tipo de aplicação em questão, perfil dos usuários, contextos de utilização, etc., que são variáveis. Além disso, tais valores podem se modificar em função do tempo com o crescimento da população de visitantes, mudança dos requisitos e recursos da aplicação e mesmo atualização da tecnologia (WINCKLER; PIMENTA, 2002).

Conforme as definições de Garrett (2003) a qualidade da interface está diretamente associada com alguns parâmetros:

a) Facilidade de aprendizado;
b) Facilidade de lembrar como realizar uma tarefa após algum tempo;
c) Rapidez no desenvolvimento de tarefas;
d) Baixa ocorrência de erros;
e) Satisfação do usuário.

É essencial considerar a integração de fatores humanos no processo de desenvolvimento de interfaces, a fim de se reduzir a complexidade e melhorar a usabilidade. Estes fatores humanos estão relacionados ao comportamento do usuário e funcionam como base para a adaptação das interfaces às suas necessidades (MORAES; ROSA, 2012).

2.12 DESIGN DE INTERAÇÃO

O objetivo do *design* de interação é a otimização da relação do usuário com sistemas computacionais. Segundo Preece, Rogers e Sharp (2013) o *design* de interação corresponde ao procedimento de criar melhores experiências de trabalho, comunicação e interação, baseando-se em informações acerca do comportamento dos usuários.

A aplicação do *design* de interação exerce melhorias na interação homem-computador, visando à criação de produtos e serviços dotados de maior usabilidade, sob o conceito de *design* centrado no usuário. É considerado como essencial por todas as disciplinas, campos e abordagens que se dedicam a projetar sistemas computacionais para pessoas (PREECE, ROGERS e SHARP, 2013).

2.13 TRABALHOS CORRELACIONADOS

Tendo em vista evidenciar a importância do tema abordado, foram pesquisados e analisados trabalhos semelhantes na área de criação de sistemas, Engenharia de Software e métricas ágeis. Constatou-se a existência de um *software* denominado FireScrum, que se mostrou semelhante à aplicação a proposta neste trabalho.

O Acadêmico Eric de Oliveira Cavalcanti apresentou em 2009 a sua dissertação de mestrado com o título “FireScrum: Ferramenta de apoio à gestão de projetos utilizando Scrum”, com o objetivo desenvolver uma ferramenta para apoiar o gerenciamento de projetos utilizando o *Scrum* como base, suprindo necessidades como apoio a equipes remotas, geração automática de gráficos, obtenção de métricas e históricos (CAVALCANTI, 2009).

O sistema FireScrum e o *software* proposto neste trabalho possuem algumas semelhanças que podem ser verificadas nas funcionalidades de ambos e no foco ao método ágil *Scrum*. No entanto, o protótipo de *software* proposto neste trabalho, além de apresentar recursos para a prática do método *Scrum*, pretende assegurar a efetividade deste método por meio do fornecimento de instruções acerca dos pilares, papéis, eventos, artefatos e dinâmica do *Scrum*. O sistema proposto é uma ferramenta de guia e apoio da prática do *Scrum*, com o objetivo de estimular os princípios ágeis no cenário de equipes inexperientes.

3 METODOLOGIA

Este capítulo descreve os métodos de pesquisa adotados, como a natureza da pesquisa, com a identificação do caráter da pesquisa realizada, os instrumentos, os materiais e procedimentos usados, a população e amostra trabalhada e o tratamento de dados por meio dos instrumentos utilizados.

3.1 NATUREZA DA PESQUISA

A metodologia aplicada no desenvolvimento deste estudo detém caráter descritivo, tendo como objetivo primordial a descrição das características de um determinado fenômeno por meio do estabelecimento de relações entre variáveis (GIL, 2008). Neste tipo de pesquisa realiza-se o estudo, a análise, o registro e a interpretação dos fatos do meio físico sem a interferência do pesquisador. A pesquisa descritiva pode ser comparada a um estudo de caso, no qual os dados pertinentes são coletados e realizadas as análises das relações entre as variáveis e são determinados os efeitos resultantes (PEROVANO, 2014). Neste estudo foi realizada uma análise do protótipo desenvolvido, com base nos dados provenientes da fase de testes, obtidos por meio da aplicação de um questionário aos envolvidos.

O caráter da pesquisa é qualitativo, que, segundo Diehl (2004), descreve a complexidade de determinado problema, sendo necessário para compreender e classificar os processos dinâmicos vividos nos grupos, contribuir no processo de mudança, possibilitando o entendimento das mais variadas particularidades dos indivíduos.

3.2 POPULAÇÃO E AMOSTRA

Este estudo abrange como população profissionais da área de tecnologia do Instituto Federal de Minas Gerais - *Campus* São João Evangelista (IFMG-SJE) e na amostra como campo de observação, a participação de um profissional da área de Engenharia de Software e três professores do curso que trabalham na área de desenvolvimento de *software* para contribuir com a pesquisa por meio da avaliação dos requisitos do *software*.

3.3 INSTRUMENTOS UTILIZADOS

Considerando o objetivo deste estudo, que consiste em desenvolver um protótipo para auxiliar o processo de desenvolvimento de projetos de *software* de pequeno porte e equipes de desenvolvimento iniciantes, aplicando o método *Scrum*, foram efetuadas pesquisas, mediante uma fundamentação teórica, em busca de informações acerca do tema abordado, que substanciassem a importância do mesmo.

Para a obtenção dos dados importantes ao desenvolvimento da aplicação proposta, verificou-se a necessidade de uma análise e levantamento de requisitos, que se sucedeu por meio da colaboração do professor da área de Engenharia de Software, que verificou as funcionalidades que seriam importantes que protótipo demonstrasse.

Após a análise dos requisitos do sistema houve a modelagem dos dados por meia elaboração de diagramas UML, utilizando-se da ferramenta *Creately* versão *web*.

A análise dos requisitos manteve o princípio ágil que sugere simplicidade tendo em vista desenvolver apenas as funcionalidades essenciais para verificação da viabilidade da proposta de *software*. Ao final da análise e levantamento dos requisitos definiu-se que o protótipo deveria atender aos seguintes requisitos funcionais (RFs):

- [RF01]: O sistema deve apresentar um manual do *Scrum*, contemplando os conceitos mais importantes do método, bem como a dinâmica envolvida em sua aplicação;
- [RF02]: O sistema deve permitir que o indivíduo com o papel de *Product Owner* realize o registro e *login* no ambiente do projeto;
- [RF03]: O sistema deve permitir que o indivíduo com o papel de *Scrum Master* realize o registro e *login* no ambiente do projeto;
- [RF04]: O sistema deve permitir que o indivíduo com o papel de integrante do *Dev Team* realize o registro e *login* no ambiente do projeto;
- [RF05]: O sistema deve permitir que o indivíduo com o papel de *Product Owner* altere suas credenciais;
- [RF06]: O sistema deve permitir que o indivíduo com o papel de *Scrum Master* altere suas credenciais;
- [RF07]: O sistema deve permitir que o indivíduo com o papel de desenvolvedor altere suas credenciais;
- [RF08]: O sistema deve permitir que o indivíduo com o papel de *Product Owner* mantenha o *Product Backlog* e publique *Sprints*;
- [RF09]: O sistema deve permitir que o indivíduo com o papel de *Scrum Master* publique avisos;
- [RF10]: O sistema deve permitir que os diversos artefatos sejam visualizados pelos demais integrantes do Time *Scrum*.

Na etapa do processo de desenvolvimento referente à implementação da aplicação, o *layout* do sistema foi desenhado com base nas normas regentes de desenvolvimento de

páginas *web*. Essas normas são especificações na W3C *(World Wide Web Consortium)*, que desenvolve normas técnicas e orientações através de um processo projetado para maximizar a consenso sobre as recomendações, garantindo qualidades técnicas e editoriais, além de alcançar apoio da comunidade de desenvolvedores, do consórcio e do público em geral (W3C BRASIL, 2017).

Para a fase de implementação foram utilizados os *softwares*:

- VertrigoServ versão 2.46: utilizado para a realização dos testes e visualização das etapas físicas do projeto de *software*, através da instalação dos componentes Apache, PHP 5, MySQL 4, PhpMyAdmin e Zend Optimizer;
- Sublime Text 3: para o desenvolvimento da interface e programação das funcionalidades da aplicação. A IDE Sublime Text oferece recursos apropriados para desenvolvimento profissional de *software*, significativos para a construção do protótipo em questão.
- CodeIgniter versão 3.1.6: utilizado para a aplicação do padrão de desenvolvimento MVC, com o objetivo de desenvolver a aplicação de forma que seu código adquira elegância e organização;
- CorelDRAW Graphics Suite 2017: para criação das imagens disponibilizadas no *software*.

Posterior à fase de implementação foi aplicado um questionário aos que participaram da fase de testes, para que estes avaliassem o comportamento do *software*, sua usabilidade, desempenho e aplicabilidade. O questionário foi elaborado com a maioria das perguntas fechadas para a obtenção de resultados uniformes, utilizados para a realização da análise qualitativa.

3.4 MÉTODOS E PROCEDIMENTOS

Para dar início ao processo de prototipação foram levantados os requisitos do sistema e foram projetadas soluções para atender as funcionalidades requeridas.

A análise dos requisitos efetuou-se através de reuniões e conversações com o professor da área de Engenharia de Software do IFMG-SJE, para que fossem identificadas as funcionalidades que o sistema deveria apresentar tendo em vista atender as necessidades verificadas.

Realizada a análise dos requisitos a estrutura do sistema precisou ser projetada da melhor forma a atender as necessidades identificadas. Para garantir a compreensão dos requisitos do sistema realizou-se a modelagem dos dados, por meio de diagramas que empregam notações UML, utilizados posteriormente para integrar a documentação do protótipo. Identificou-se a utilidade de se elaborar diagramas de casos de uso e diagramas de classes.

Posteriormente, foi desenvolvida a interface do *software*, considerando os conceitos de usabilidade abordados nesta pesquisa. Nesta etapa, a interface do sistema foi desenvolvida aplicando-se a linguagem de marcação HTML, para compor a estrutura, e a linguagem de folhas de estilo CSS, para a formatação dos elementos que integram a aparência das páginas. Visando potencializar esta etapa, decidiu-se pela utilização da biblioteca *Bootstrap* em alguns elementos da interface, objetivando agilidade e eficiência.

Para atribuir efeitos especiais aos elementos das páginas e proporcionar maior interatividade com os usuários do sistema, foi utilizada, em conjunto com o HTML e o CSS, a linguagem *JavaScript*, bem como a biblioteca *JQuery*.

A proposta do protótipo prevê a execução de cálculos a partir de dados de projetos de *software* cadastrados pelo usuário, para a produção do relatório de orçamento de *software*. Para a realização dos cálculos e demais conteúdo dinâmico foi utilizada a linguagem de programação *web* PHP, que se caracteriza como uma linguagem interpretada madura e consistente.

Após o término do desenvolvimento do protótipo teve início os testes funcionais e de usabilidade. Os testes consistiram em uma análise dinâmica do sistema, que, de acordo com Sommerville (2007), verifica se o projeto de *software* atende os requisitos estabelecidos e se comporta da maneira esperada. Esta fase contou com a participação dos professores do curso Bacharelado em Sistemas de Informação, que efetuaram a avaliação do produto final por meio do questionário aplicado.

3.5 TRATAMENTO DOS DADOS

Baseando-se nos dados provenientes dos testes do sistema, elaborou-se um relatório contendo informações fornecidas pelos participantes da fase de testes, acerca da usabilidade, desempenho e aplicabilidade do sistema desenvolvido. Os dados acerca do *software* foram

obtidos por meio da aplicação de um questionário para que se pudesse ser realizada a análise qualitativa dos resultados.

3.6 PROTOTIPAÇÃO

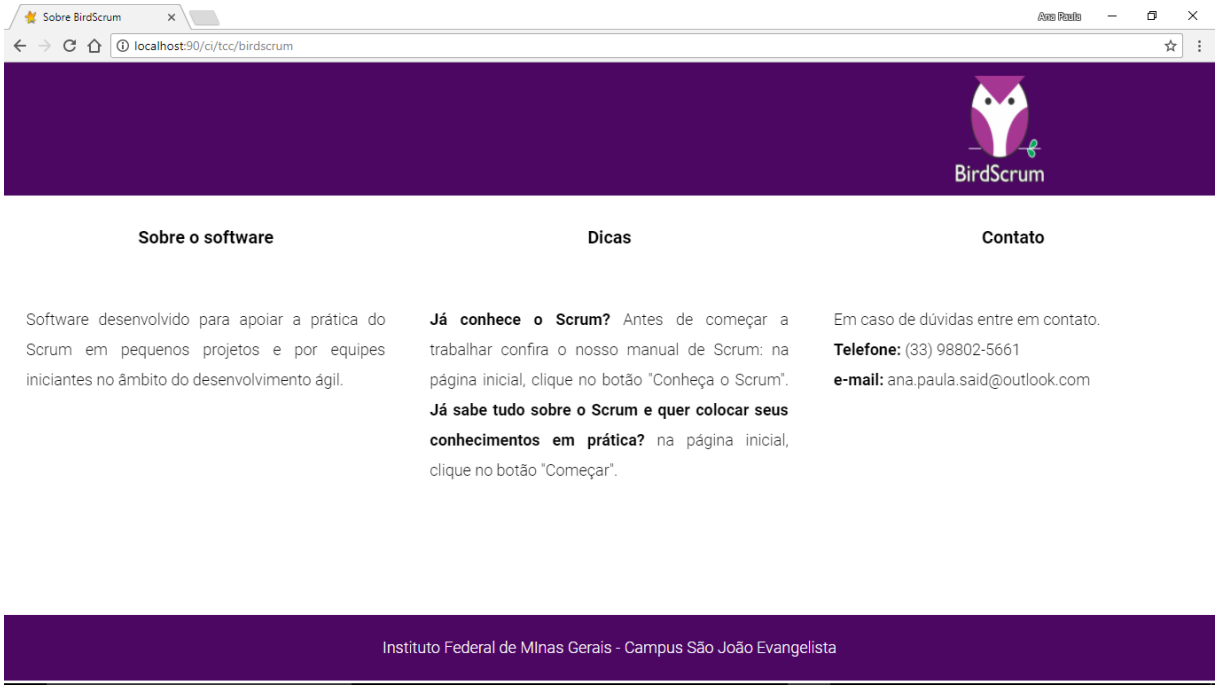
A Figura 1 apresenta a tela inicial do sistema, contendo a logomarca e três botões com distintas funções: “Sobre BirdScrum”, “Conheça o Scrum” e “Começar”.

Figura 1 - Tela inicial do sistema

Fonte: Elaborada pelo autor.

Ao ser clicado, o botão “Sobre BirdScrum” apresenta informações sobre o *software*, dicas e informações de contato, conforme mostrado na Figura 2.

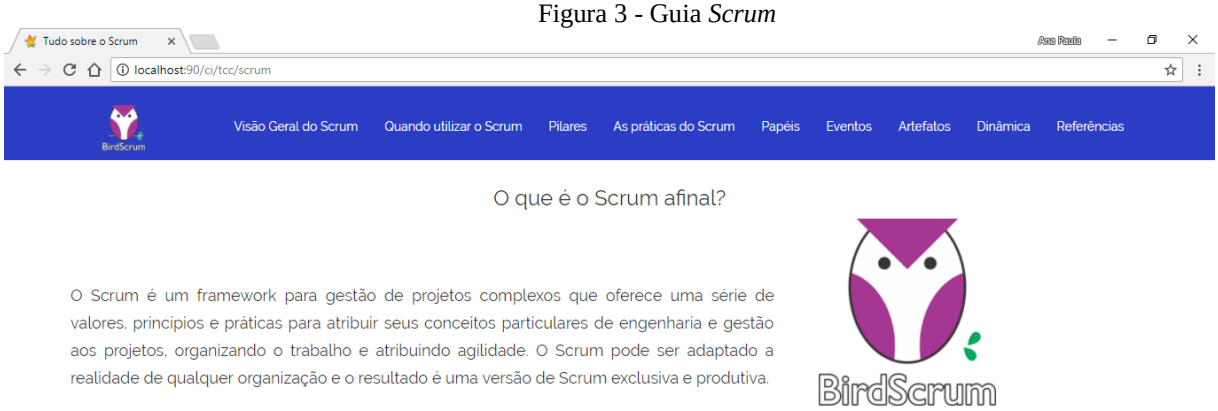
Figura 2 - Sobre BirdScrum



Fonte: Elaborada pelo autor.

O botão “Conheça o Scrum”, ao ser clicado, direciona para uma página que apresenta um guia do *Scrum*, com os conceitos mais importantes sobre o método (Figura 3).

Figura 3 - Guia *Scrum*

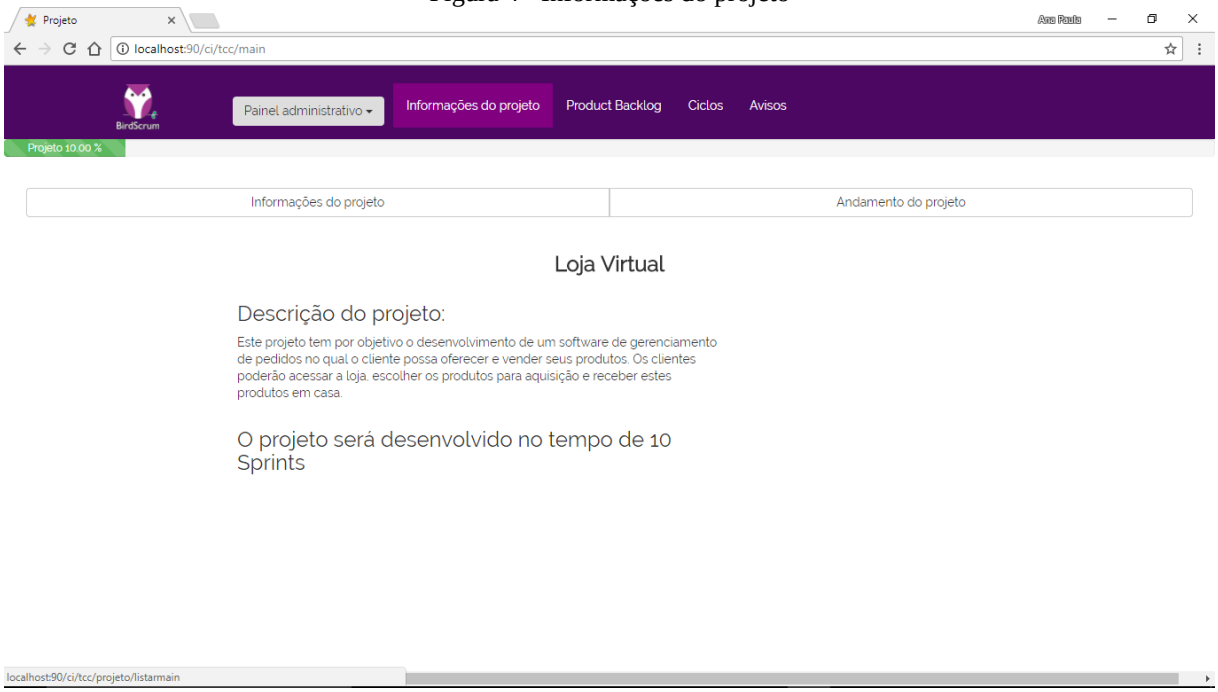


Fonte: Elaborada pelo autor.

O botão “Começar” direciona para o ambiente de prática do *Scrum*, que apresenta um menu com os itens: “Informações do projeto”, “Product Backlog”, “Ciclos” e “Avisos”. O

item de menu “Informações do projeto” mostra uma barra de progresso que demonstra a porcentagem do projeto que está concluída. Além disso, existem dois outros botões: “Informações do projeto” e “andamento do projeto”. O botão “Informações do projeto” apresenta informações como: nome do projeto, descrição do projeto e total de *Sprints*, conforme a Figura 4.

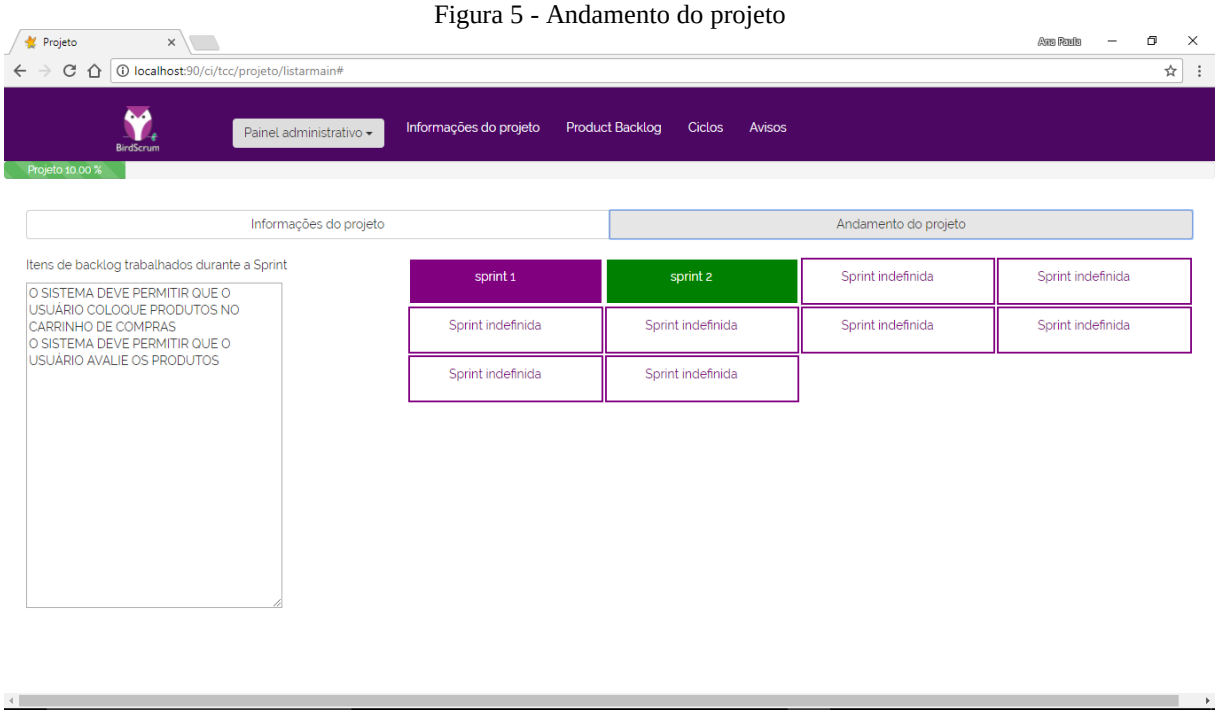
Figura 4 - Informações do projeto



Fonte: Elaborada pelo autor.

Ao ser clicado, o botão “Andamento do projeto” apresenta as *Sprints* encerradas, a *Sprint* em andamento e as *Sprints* indefinidas. Ao clicar sobre as *Sprints* encerradas (lilás), e sobre a *Sprint* em andamento (verde), são mostrados os itens do *Product Backlog* trabalhados durante a *Sprint* selecionada (Figura 5).

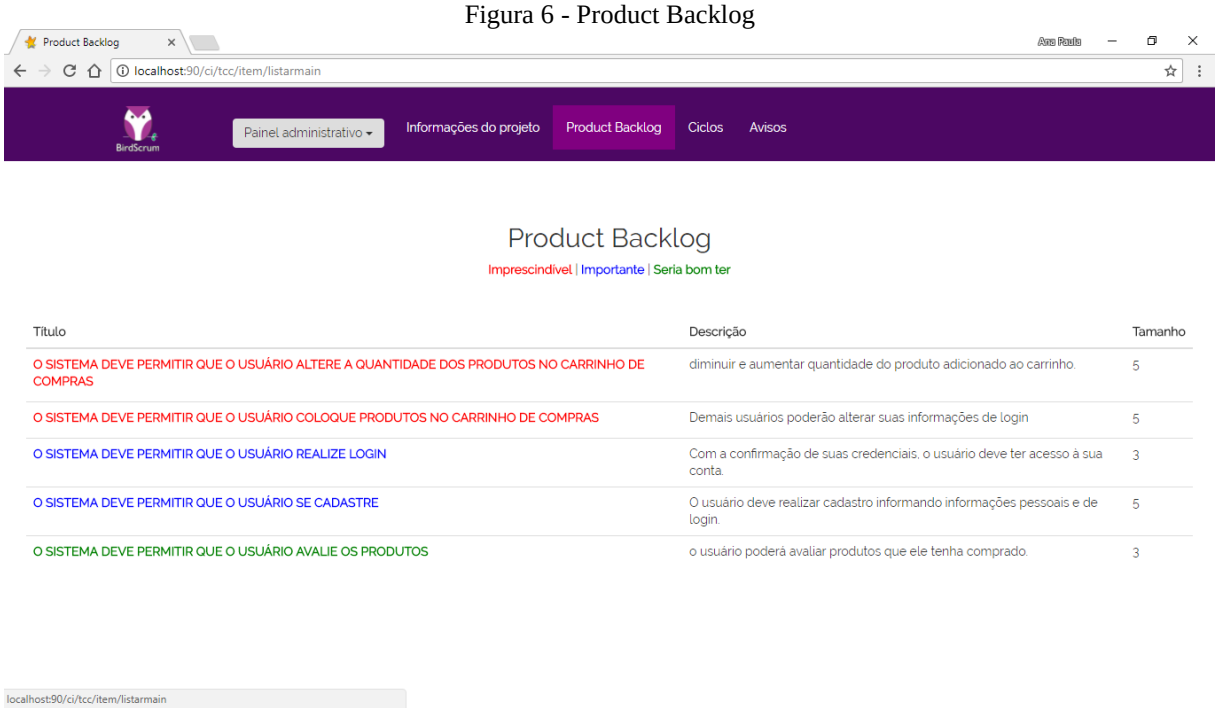
Figura 5 - Andamento do projeto



Fonte: Elaborada pelo autor.

Quando selecionado, o item de menu “Product Backlog” apresenta uma lista organizada de acordo com a prioridade de cada item: o item denominado como “imprescindível” fica no topo da lista (vermelho), o item denominado como “importante” fica na região intermediária (azul) e o item denominado como “Seria bom ter” fica na região inferior da lista (verde), como apresentado na Figura 6.

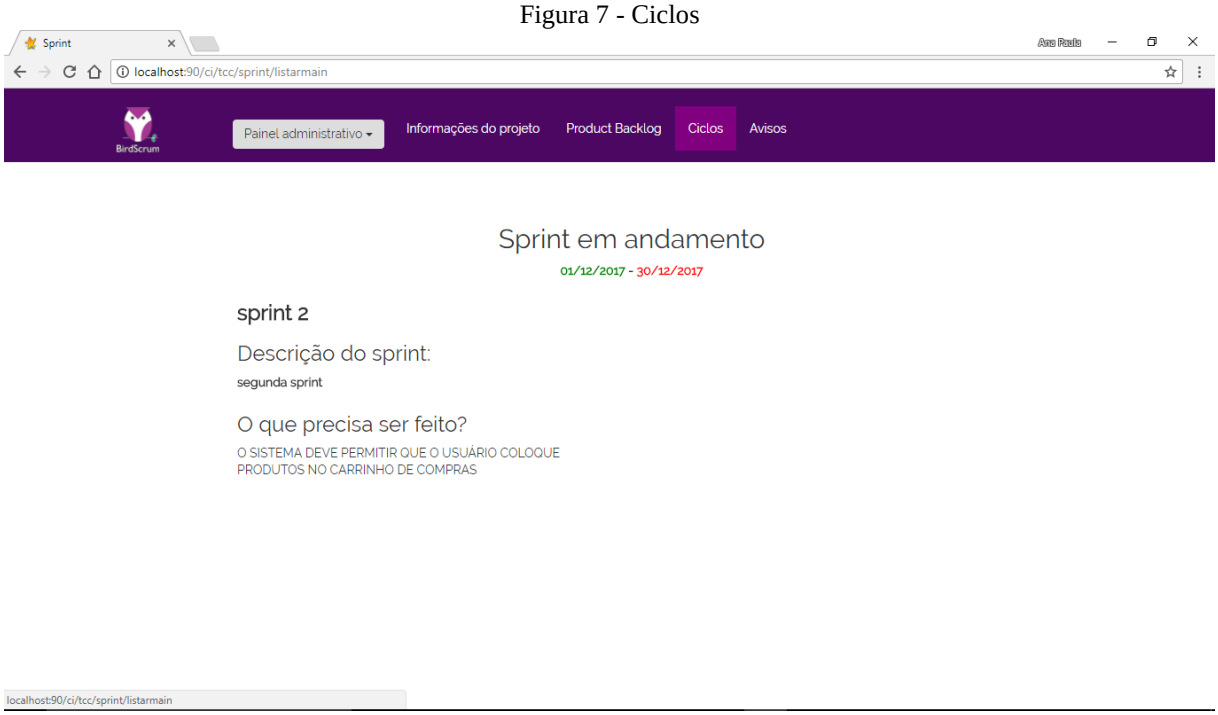
Figura 6 - Product Backlog



Fonte: Elaborada pelo autor.

O item de menu “Ciclos” apresenta informações sobre a *Sprint* em andamento: data de início, data de término, nome, descrição e metas, como mostra a Figura 7.

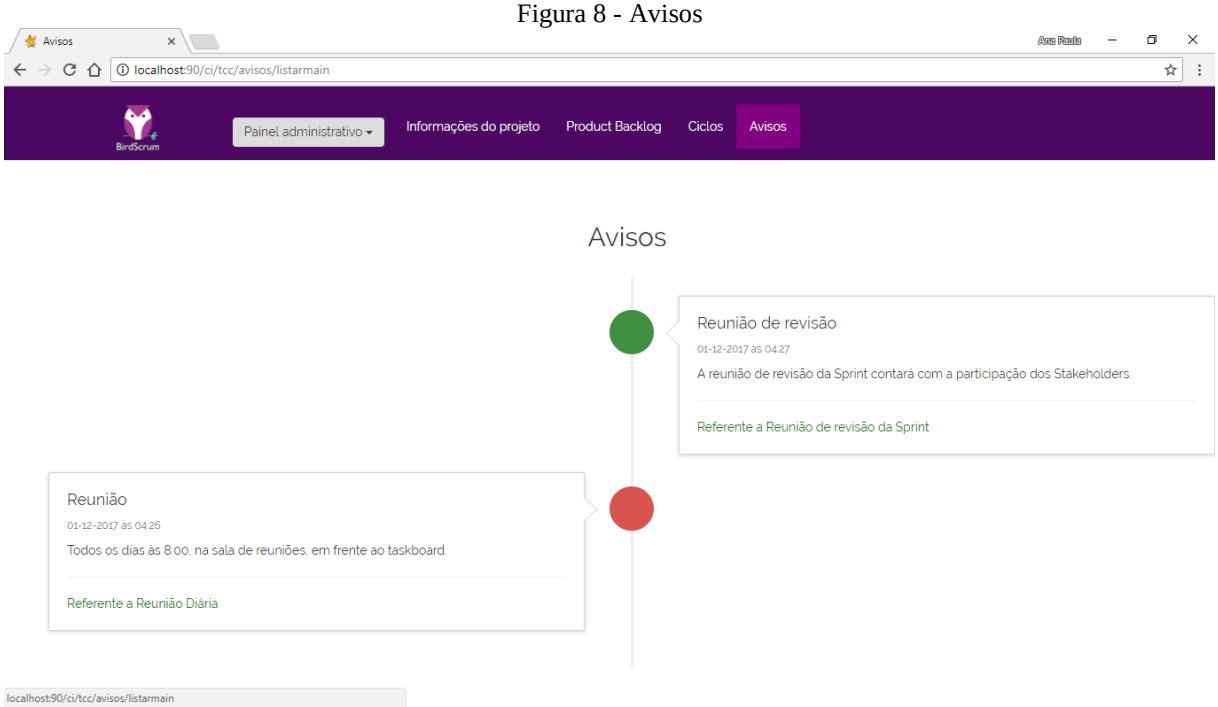
Figura 7 - Ciclos



Fonte: Elaborada pelo autor.

O item de menu “Avisos”, quando clicado, lista os avisos publicados pelo *Scrum Master* (Figura 8).

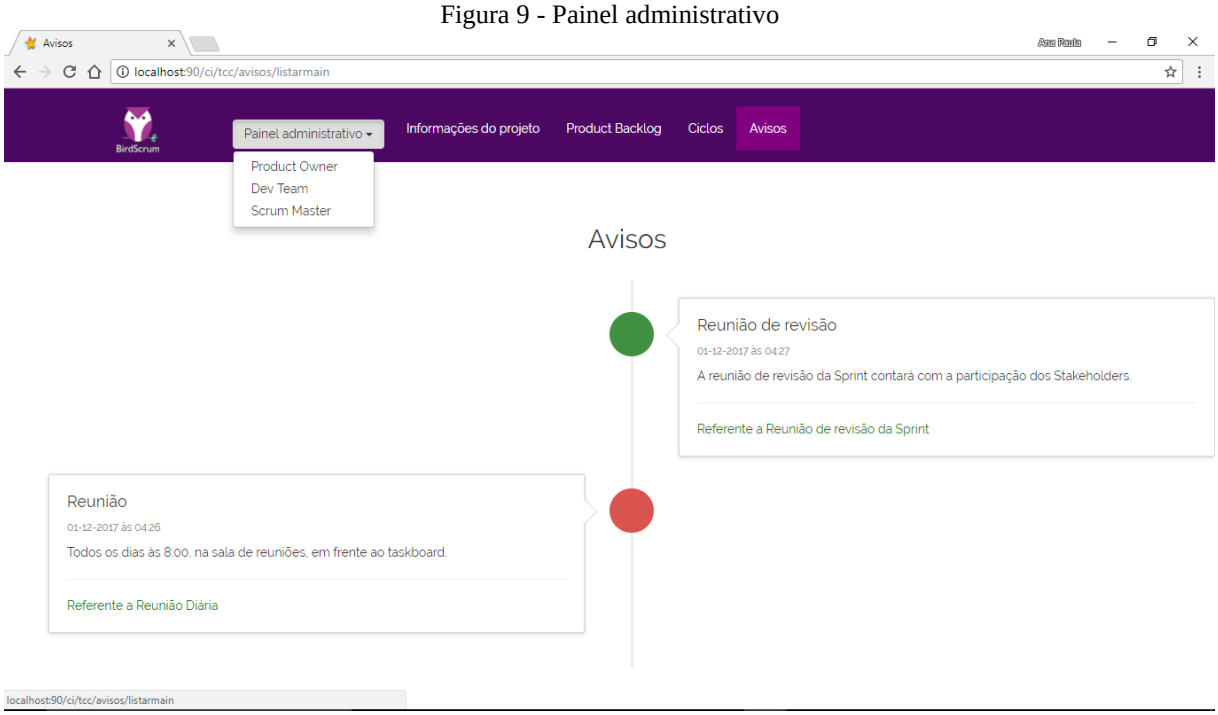
Figura 8 - Avisos



Fonte: Elaborada pelo autor.

O item de menu “Painel administrativo”, direciona para os ambientes de trabalho dos demais papéis: “Prodcut Owner”, “Dev Team” e “Scrum Master” (Figura 9).

Figura 9 - Painel administrativo



Fonte: Elaborada pelo autor.

A seguir serão contemplados detalhadamente os pormenores dos demais papés.

3.6.1 Product Owner

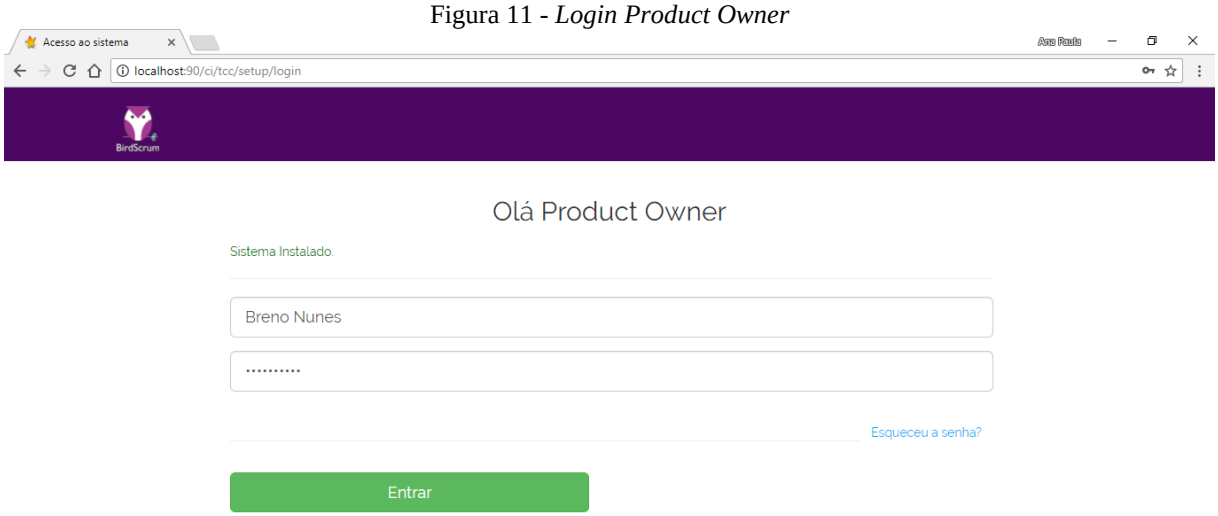
O sub-item de menu “Product Owner” direciona para a página de registro, caso não haja cadastro como *Product Owner*. Para efetuar o registro é necessário informar: nome de usuário, e-mail, senha e confirmação da senha. Após inserir todas as informações corretamente é necessário clicar no botão “Registre-se” (Figura 10).

Figura 10 - Registro *Product Owner*

Fonte: Elaborada pelo autor.

Efetuado o registro, o sistema direciona para a página de *login*, onde é necessário informar o nome de usuário e a senha (Figura 11).

Figura 11 - *Login Product Owner*

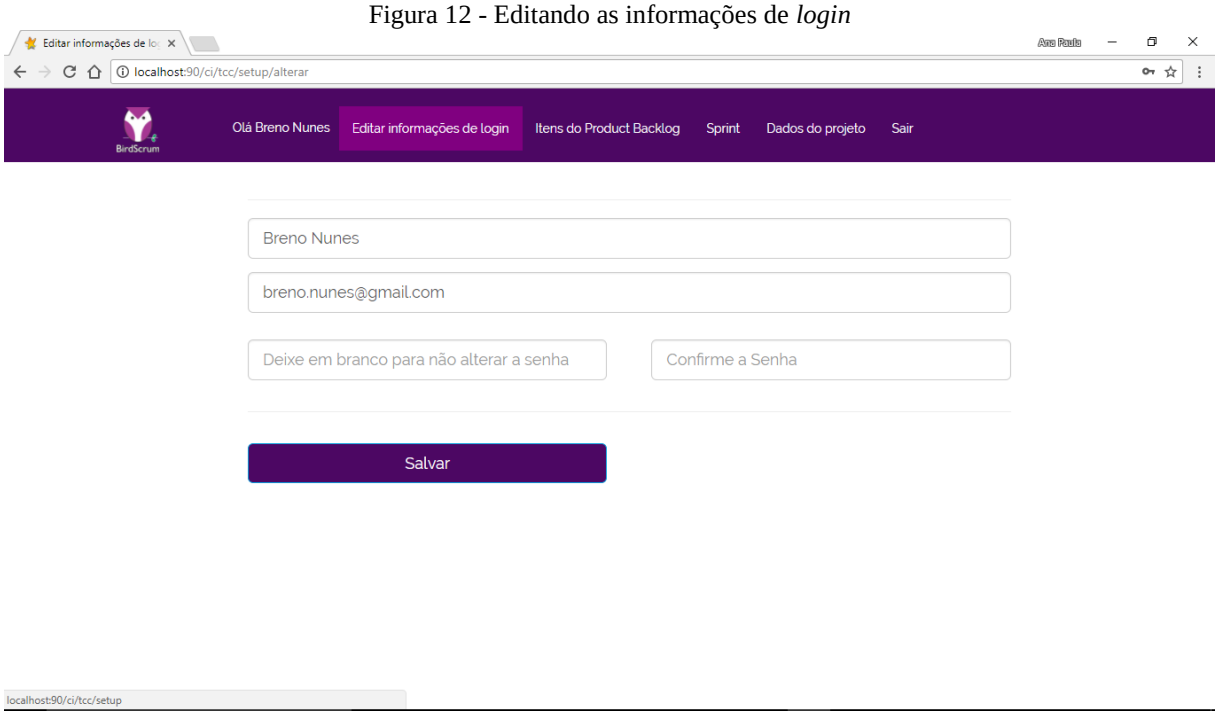


Fonte: Elaborada pelo autor.

Após efetuar *login*, o *Product Owner* tem acesso ao seu ambiente de trabalho que apresenta um menu com os itens: “Editar informações”, “Itens do Product Backlog”, “Sprint”,

"Dados do projeto" e "Sair". O item de menu "Editar informações de login" permite que o *Product Owner* edite suas credenciais, conforme a Figura 12.

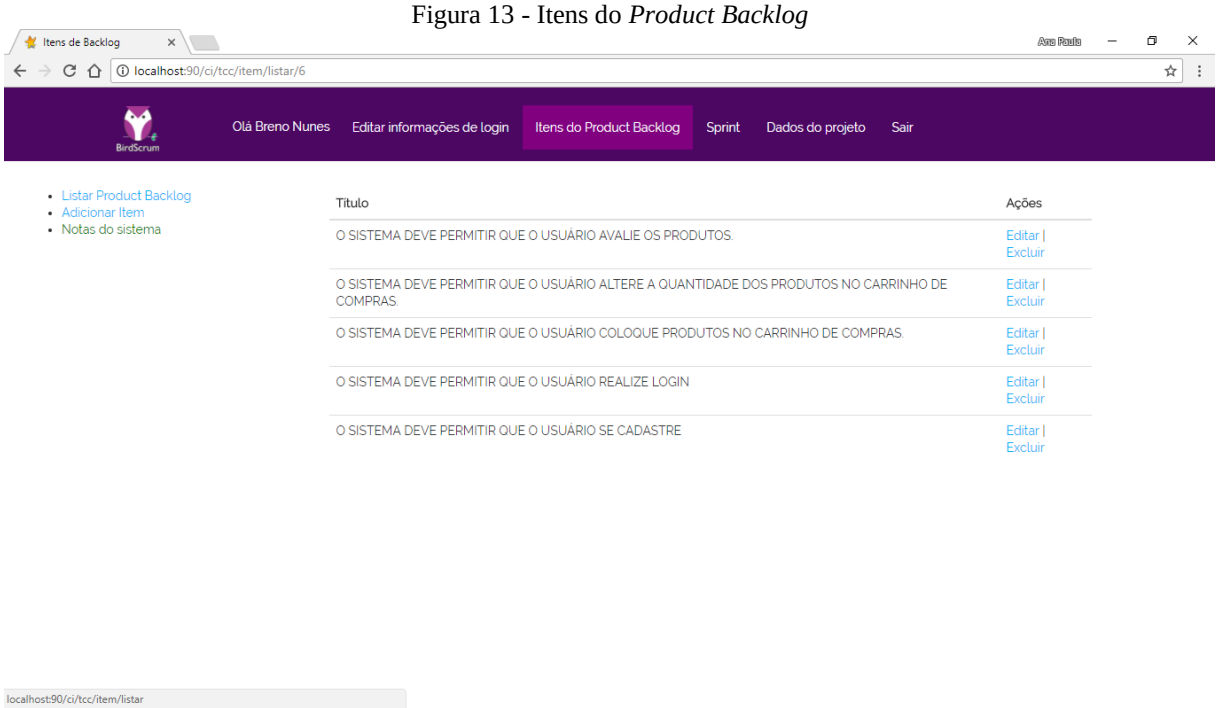
Figura 12 - Editando as informações de *login*



Fonte: Elaborada pelo autor.

No item de menu "Itens do Product Backlog" o *Product Owner* pode listar o *Product Backlog* (no *link* "Listar Product Backlog"), Adicionar itens (no *link* "Adicionar Item") e verificar as mensagens do sistema ("Notas do sistema"). Os itens listados apresentam as opções "Editar" e "Excluir" (Figura 13).

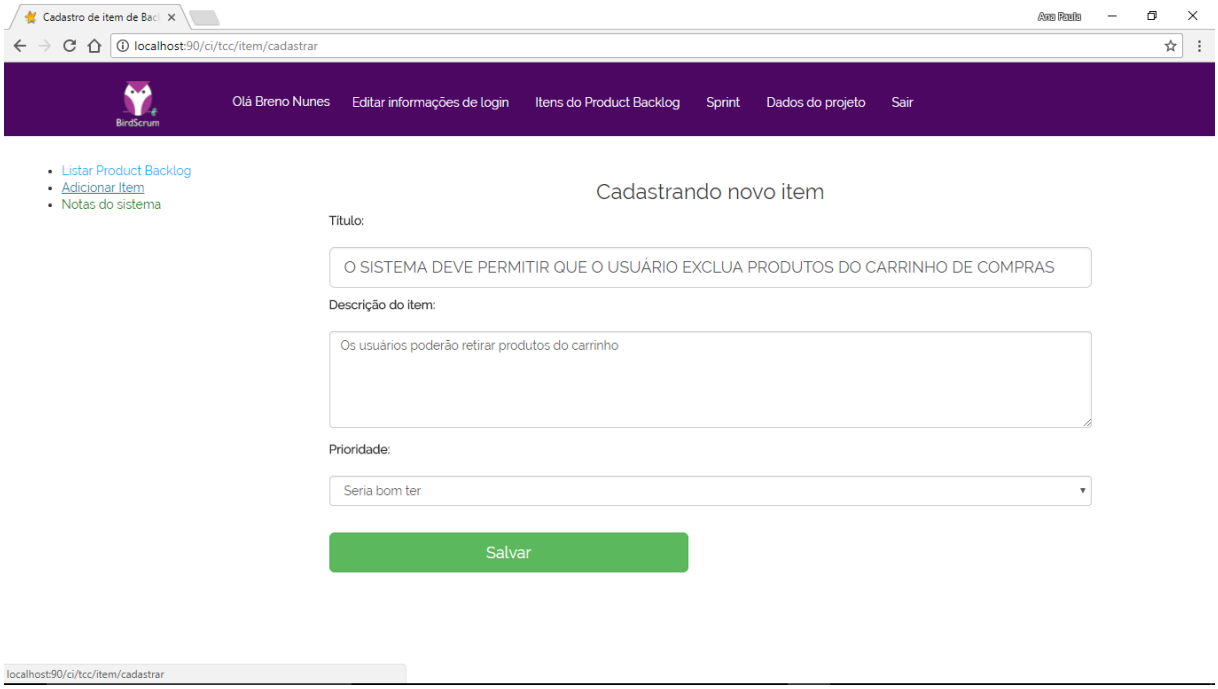
Figura 13 - Itens do *Product Backlog*



Fonte: Elaborada pelo autor.

A opção “Adicionar Item” possibilita que o *Product Owner* adicione novos itens ao *Product Backlog* mediante algumas informações: título, descrição do item e prioridade, como mostra a Figura 14.

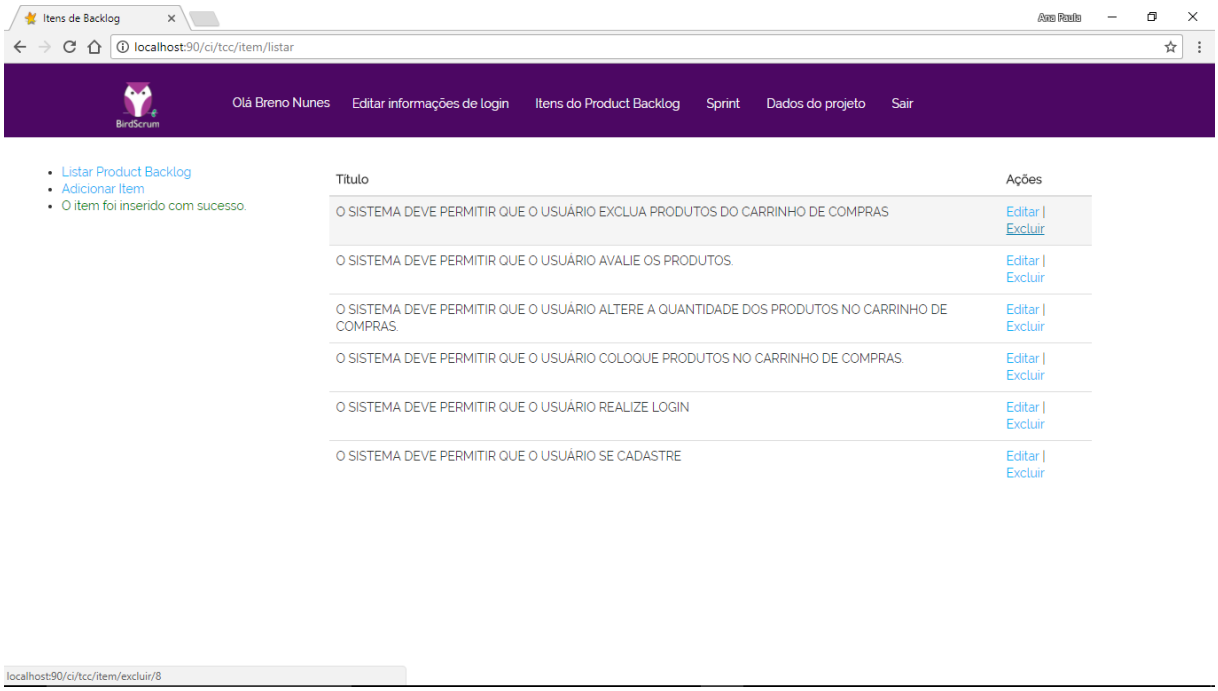
Figura 14 - Adicionando novo item no *Product Backlog*



Fonte: Elaborada pelo autor.

Quando o botão “Salvar” é clicado, o item é salvo e listado como os demais. O sistema apresenta uma mensagem informando que o item foi inserido com sucesso, conforme a Figura 15.

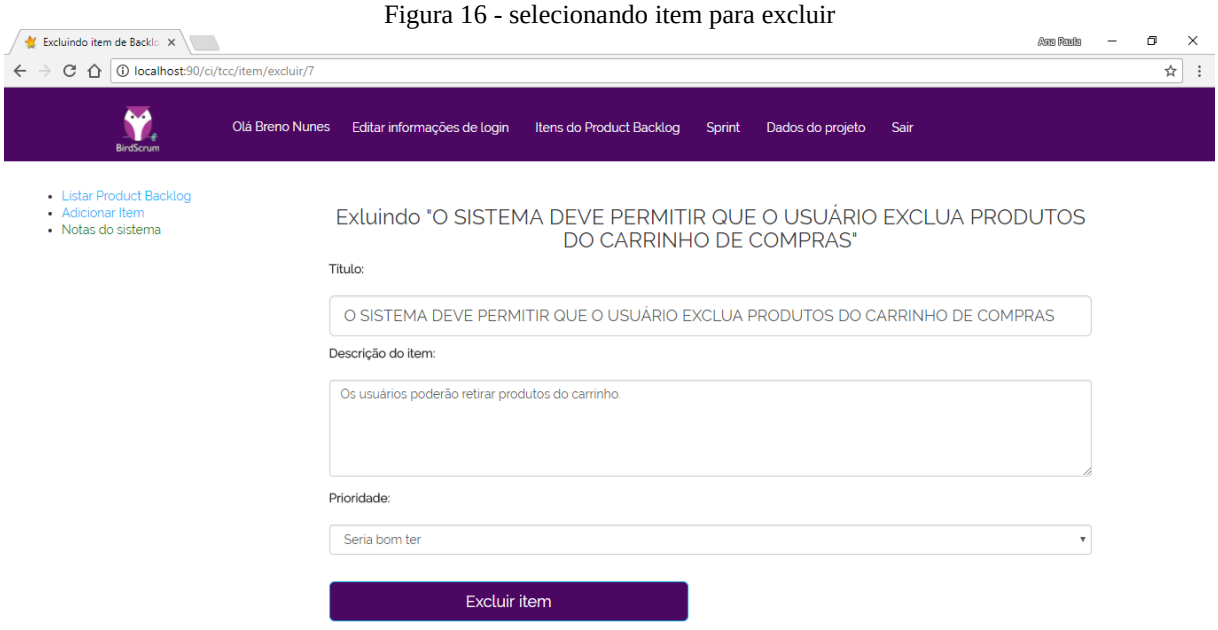
Figura 15 - Salvando novo item



Fonte: Elaborada pelo autor.

A opção “Excluir”, presente em todos os itens listados, possibilita que o item selecionado seja excluído (Figura 16).

Figura 16 - selecionando item para excluir



Fonte: Elaborada pelo autor.

Quando clicado, o botão “Excluir item” exclui o item selecionado e lista os restantes. O sistema apresenta uma mensagem informando que o item foi excluído (Figura 17).

Figura 17- Excluindo item

Fonte: Elaborada pelo autor.

A opção “Editar”, que possibilita alterar as informações do item selecionado (Figura 18).

Figura 18 - Editando item

Fonte: Elaborada pelo autor.

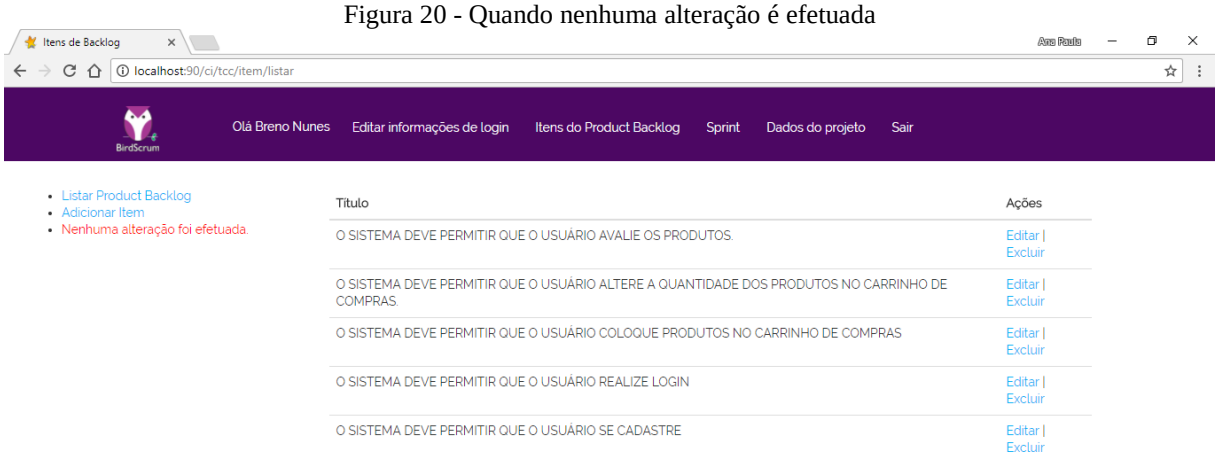
O botão “Salvar alterações” grava as mudanças realizadas e lista todos os itens cadastrados. O sistema apresenta uma mensagem informando que o item foi atualizado

(Figura 19). Caso nenhuma informação tenha sido alterada, o sistema apresenta uma mensagem informando que nenhuma alteração foi efetuada (Figura 20).

Figura 19 - Salvando alterações

Fonte: Elaborada pelo autor.

Figura 20 - Quando nenhuma alteração é efetuada

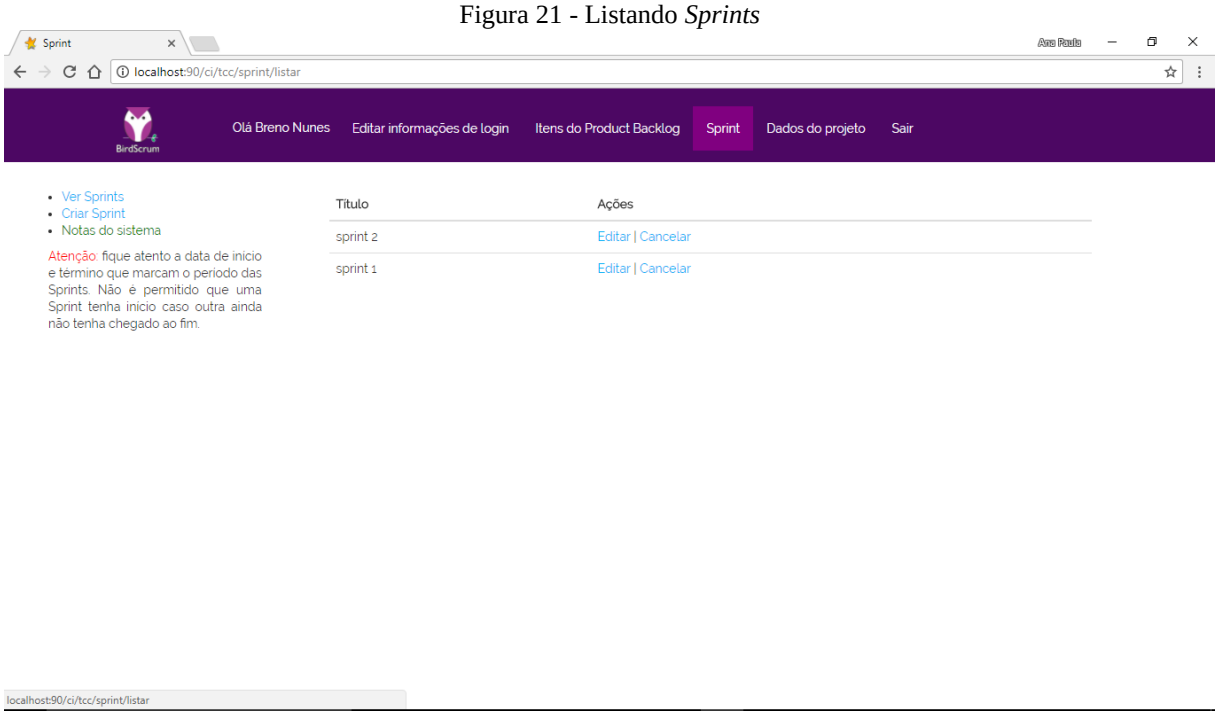


Fonte: Elaborada pelo autor.

No item de menu “Sprint” o *Product Owner* pode listar as *Sprints* cadastradas (no *link* “Ver Sprints”), Adicionar nova *Sprint* (no *link* “Criar Sprint”) e verificar as mensagens do

sistema (“Notas do sistema”). As *Sprints* listadas apresentam as opções de “Editar” e “Cancelar” (Figura 21).

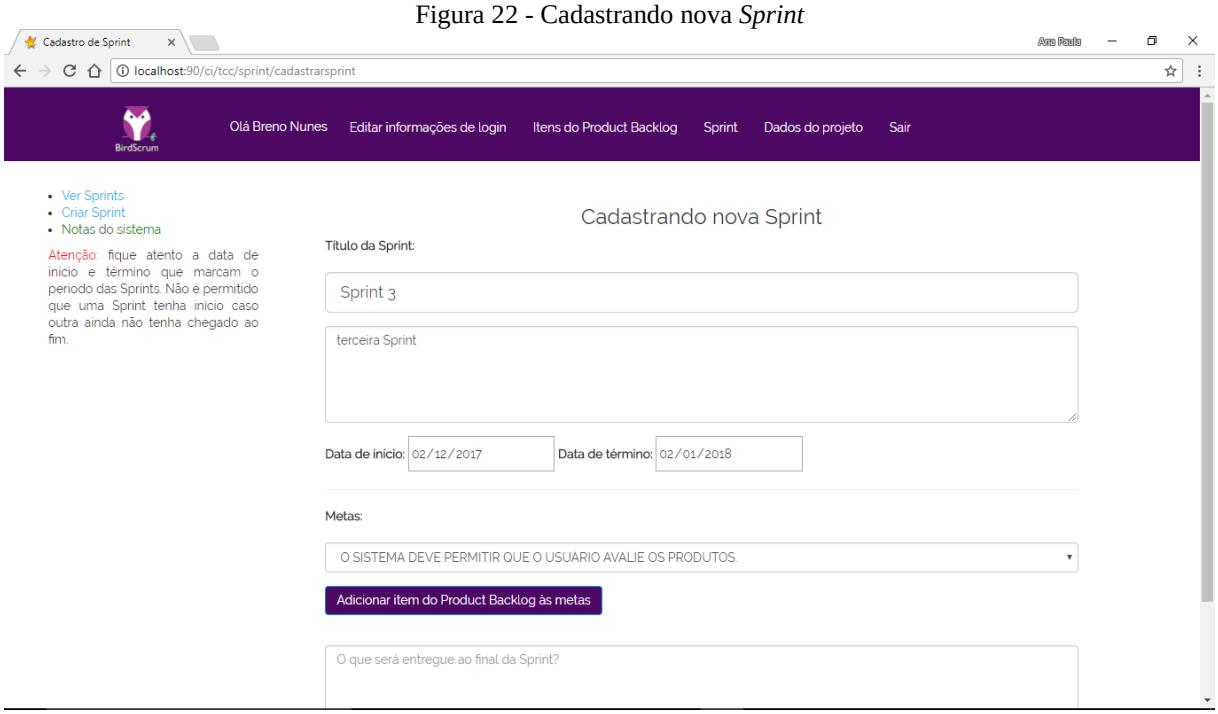
Figura 21 - Listando *Sprints*



Fonte: Elaborada pelo autor.

A opção “Criar Sprint” possibilita que o *Product Owner* crie uma nova *Sprint*, mediante algumas informações: título da *Sprint*, descrição da *Sprint*, data de início, data de término e metas (Figura 22).

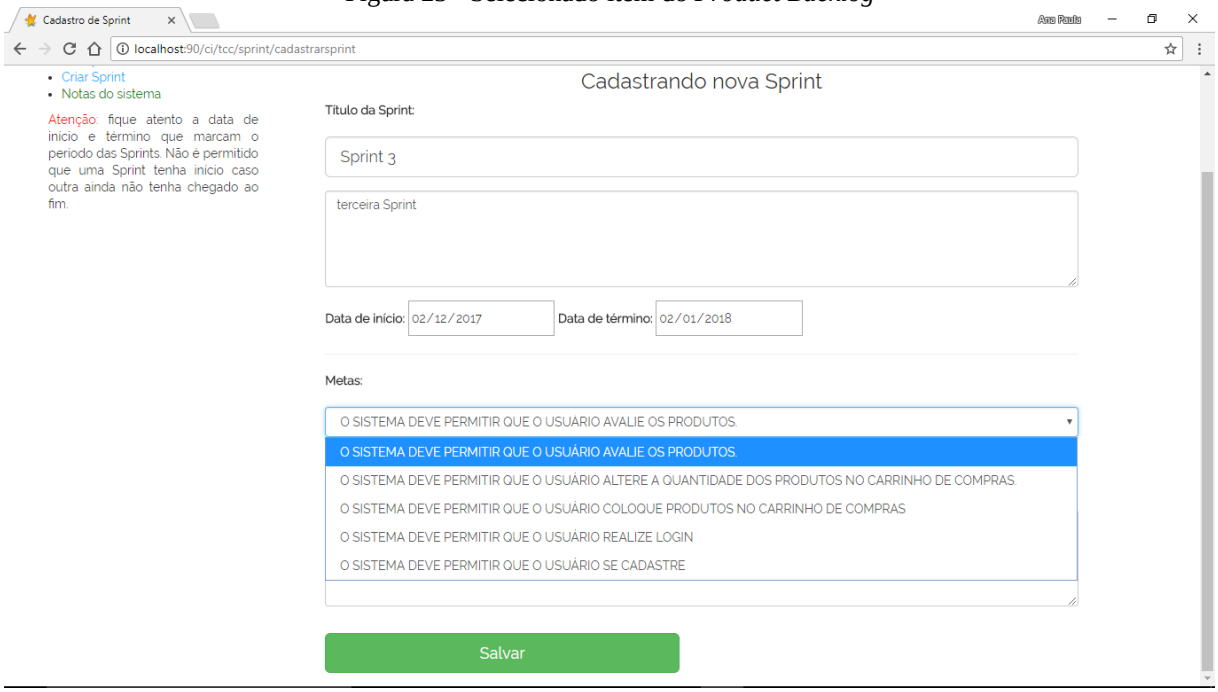
Figura 22 - Cadastrando nova *Sprint*



Fonte: Elaborada pelo autor.

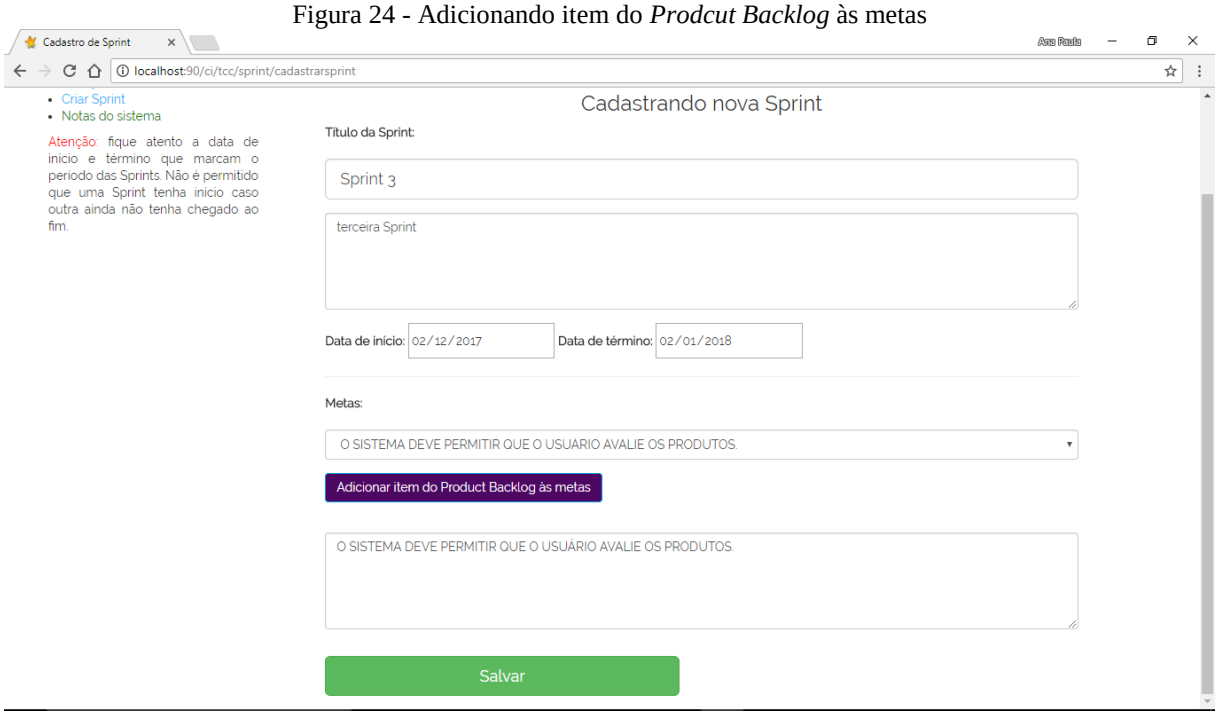
Para adicionar itens do *Product Backlog* às metas basta selecionar o item na lista apresentada e clicar em “Adicionar item do Product Backlog às metas”. O item é então adicionado ao campo “O que será entregue ao final da Sprint?” (Figuras 23 e 24).

Figura 23 - Selecionado item do *Product Backlog*



Fonte: Elaborada pelo autor.

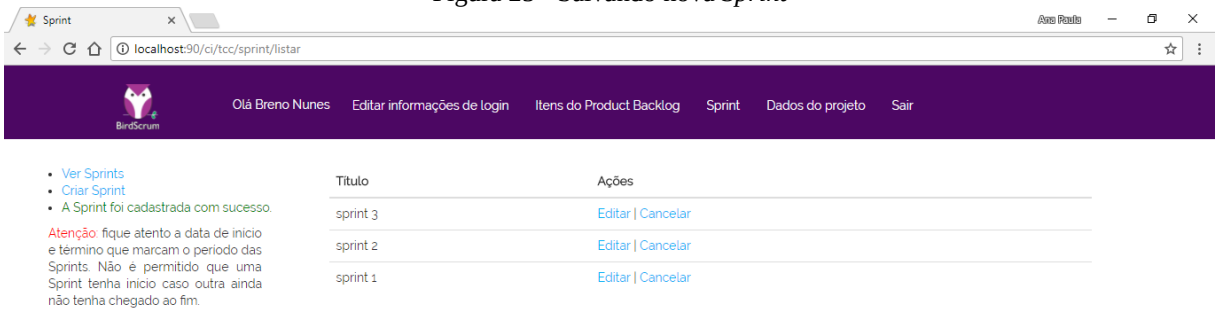
Figura 24 - Adicionando item do *Prodcut Backlog* às metas



Fonte: Elaborada pelo autor.

Quando o botão “Salvar” é clicado, a nova *Sprint* é cadastrada e listada como as demais. O sistema apresenta uma mensagem informando que a *Sprint* foi cadastrada com sucesso (Figura 25).

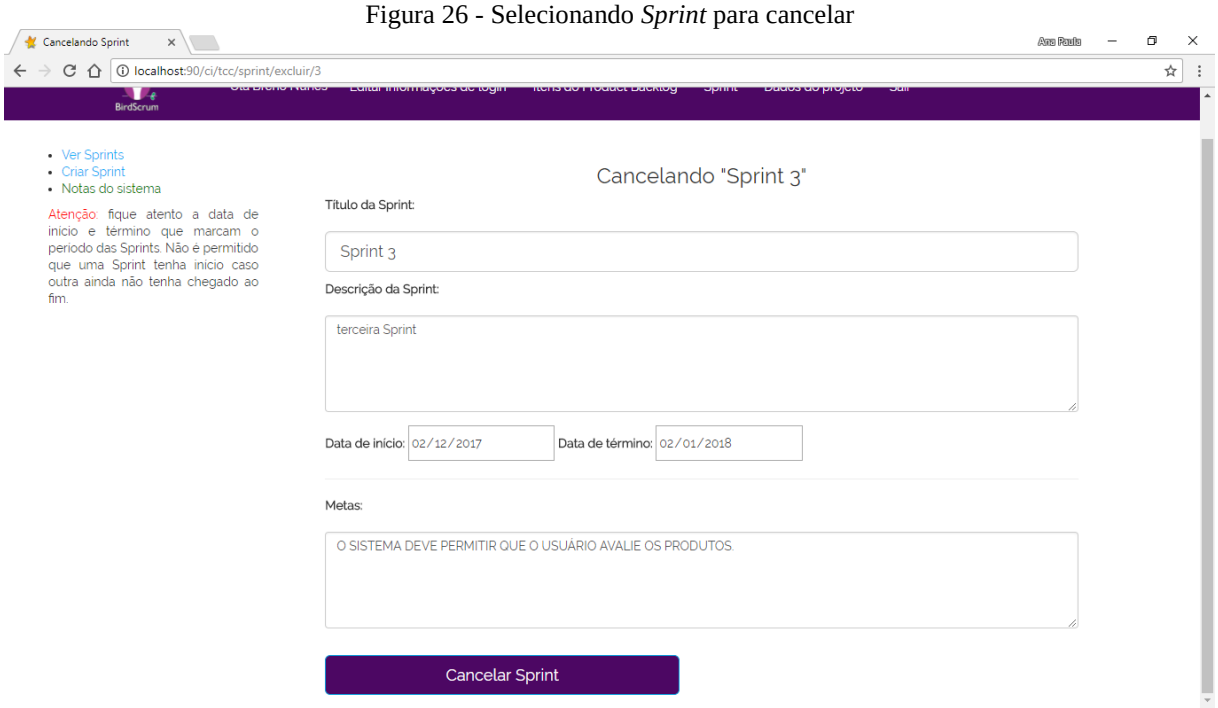
Figura 25 - Salvando nova *Sprint*



Fonte: Elaborada pelo autor.

A opção “Cancelar”, presente em todas as *Sprints* listadas, possibilita excluir a *Sprint* selecionada (Figura 26).

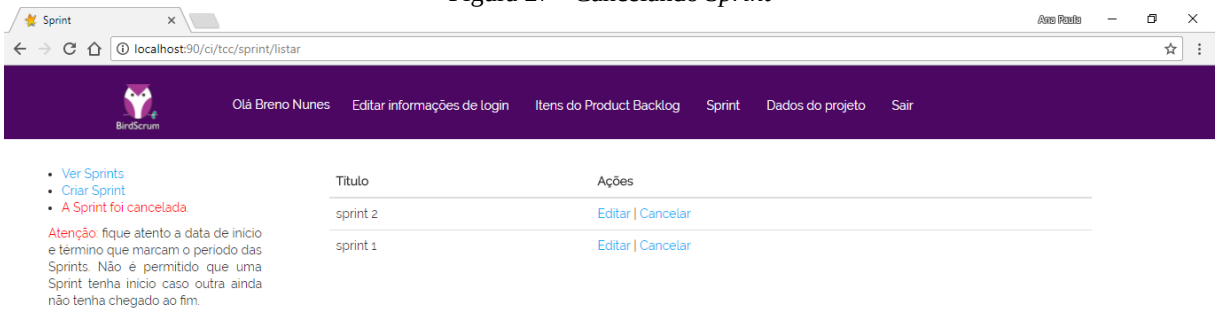
Figura 26 - Selecionando *Sprint* para cancelar



Fonte: Elaborada pelo autor.

O botão “Cancelar Sprint”, quando clicado, exclui a *Sprint* selecionada e lista as demais. O sistema apresenta uma mensagem informando que a *Sprint* foi cancelada (Figura 27).

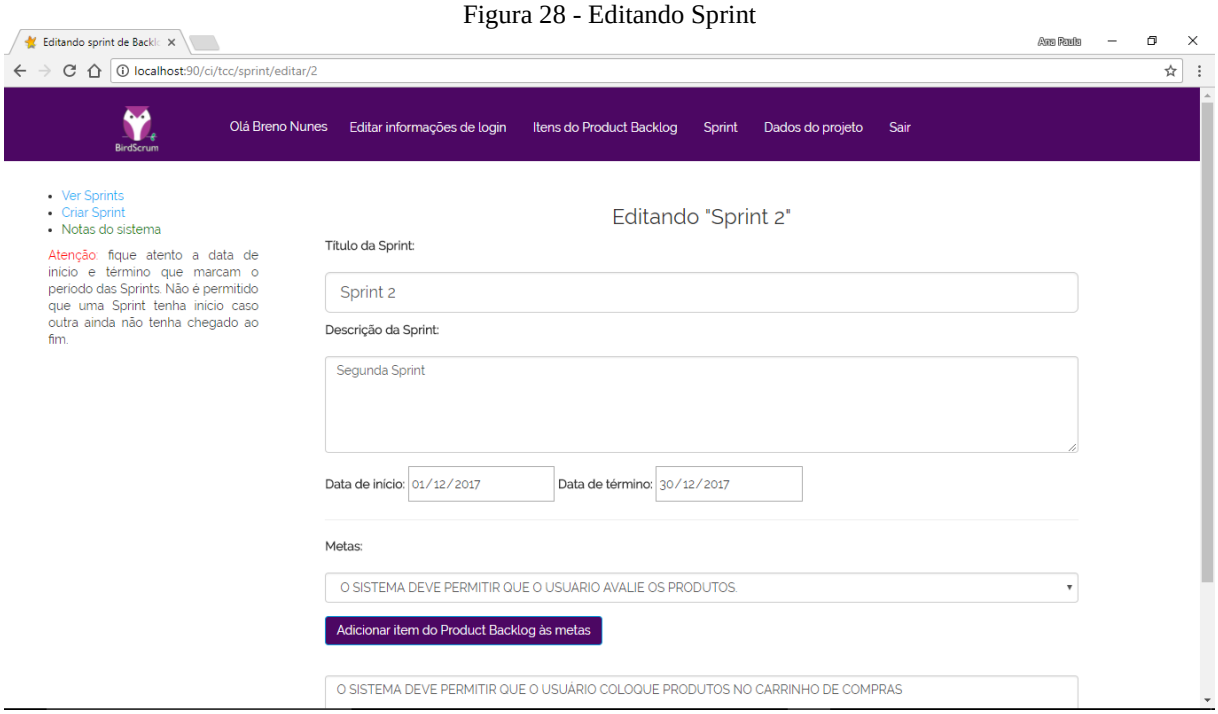
Figura 27 - Cancelando *Sprint*



Fonte: Elaborada pelo autor.

A opção “Editar”, presente em todas as *Sprints* listadas, possibilita alterar as informações da *Sprint* selecionada (Figura 28).

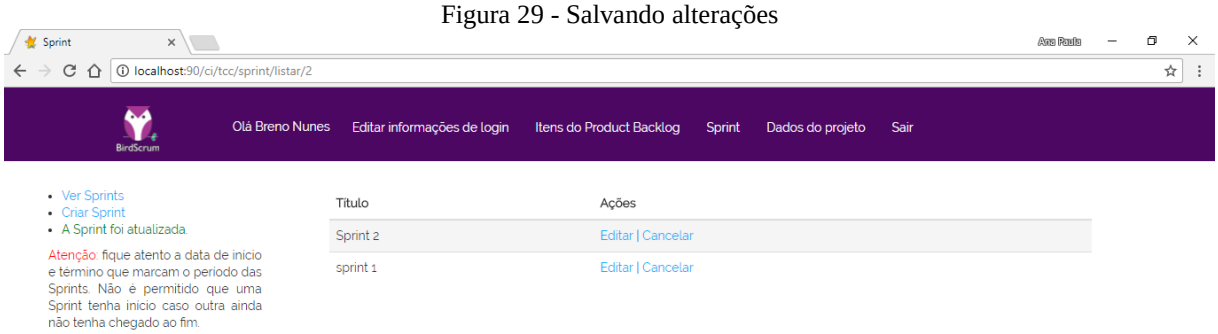
Figura 28 - Editando Sprint



Fonte: Elaborada pelo autor.

Quando o botão “Salvar alterações” é clicado as informações são gravadas e as *Sprints* são listadas. O sistema apresenta uma mensagem informando que a Sprint foi atualizada (Figura 29).

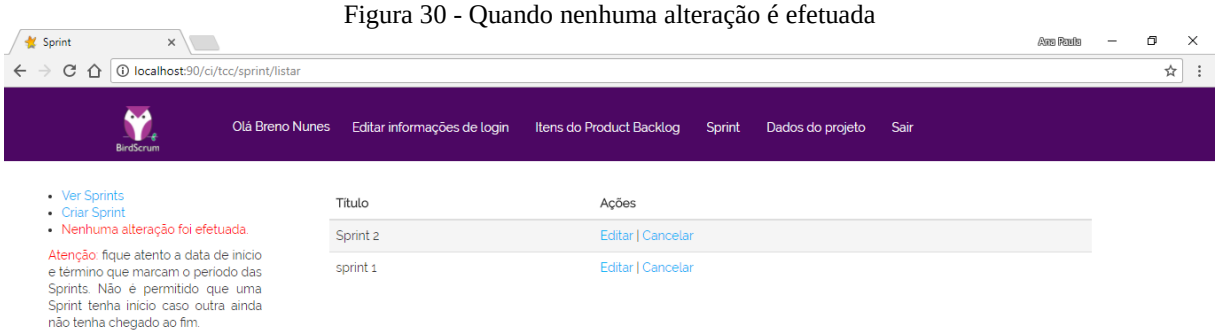
Figura 29 - Salvando alterações



Fonte: Elaborada pelo autor.

Caso nenhuma informação tenha sido alterada o sistema emite uma mensagem informando que nenhuma alteração foi efetuada (Figura 30).

Figura 30 - Quando nenhuma alteração é efetuada



Fonte: Elaborada pelo autor.

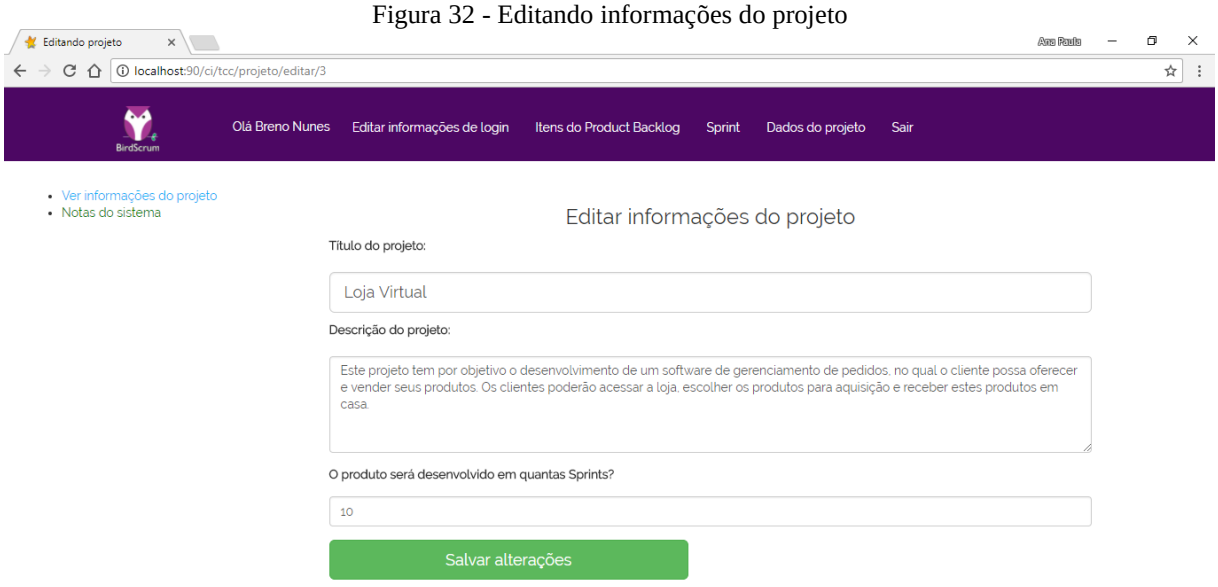
O item de menu “Dados do projeto” apresenta as informações do projeto: título do projeto, descrição e total de *Sprints*, conforme apresentado na Figura 31.

Figura 31 - Informações do projeto

Fonte: Elaborada pelo autor.

O botão “Editar informações” possibilita que o *Product Owner* altere as informações do projeto (Figura 29).

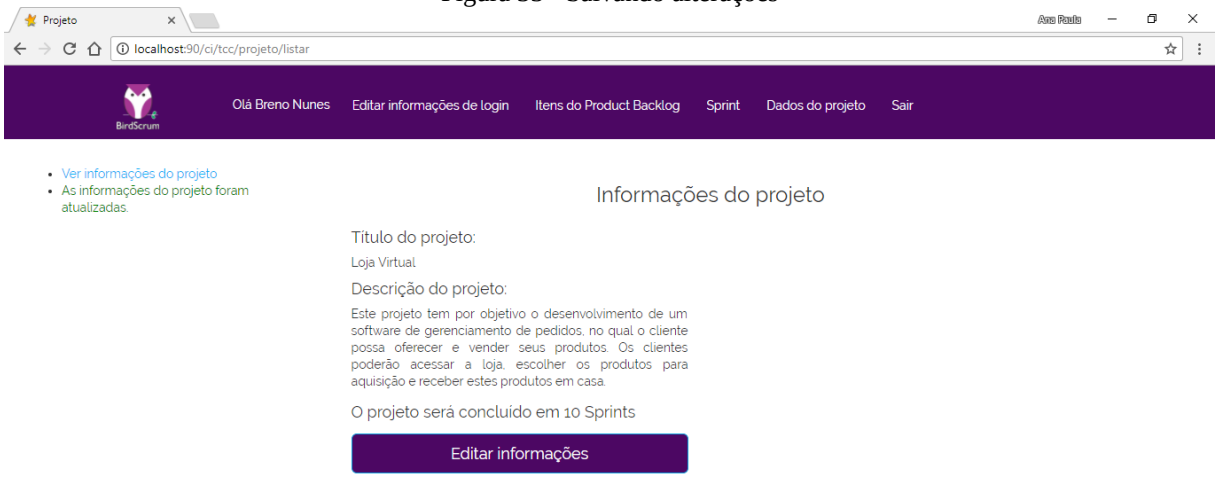
Figura 32 - Editando informações do projeto



Fonte: Elaborada pelo autor.

O botão “Salvar alterações”, quando clicado, grava as alterações realizadas e apresenta as informações do projeto atualizadas. O sistema apresenta uma mensagem informando que as informações do projeto foram atualizadas (Figura 33).

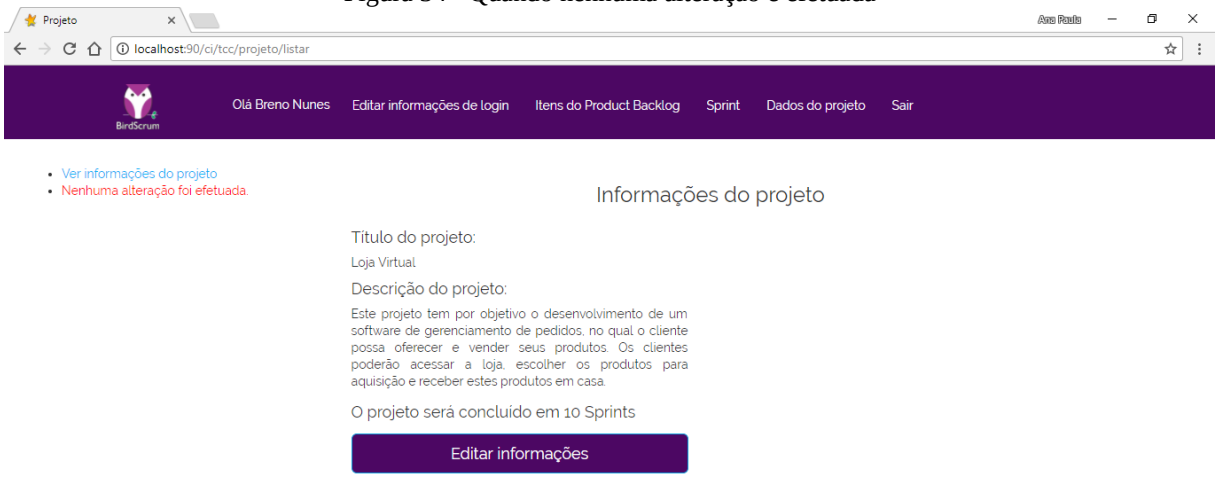
Figura 33 - Salvando alterações



Fonte: Elaborada pelo autor.

Caso nenhuma informação tenha sido alterada, o sistema emite uma mensagem informando que nenhuma alteração foi efetuada (Figura 34).

Figura 34 - Quando nenhuma alteração é efetuada



Fonte: Elaborada pelo autor.

O item de menu “Sair” permite que o *Product Owner* realize *logout* (sair).

3.6.2 Dev Team

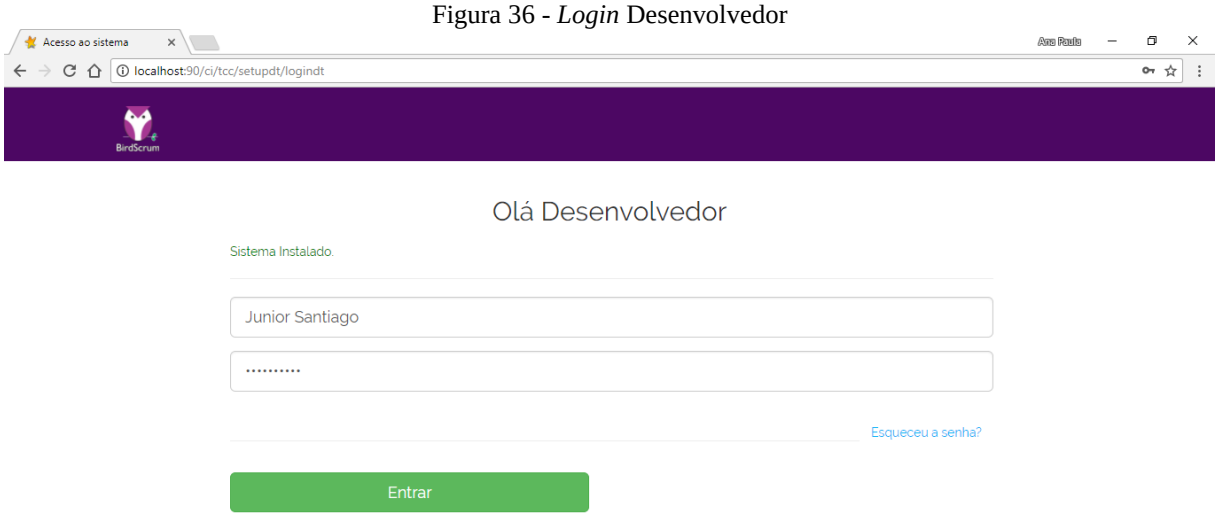
O sub-item de menu “Dev Team” direciona para a página de registro, caso não haja cadastro como Desenvolvedor. Para efetuar o registro é necessário informar: nome de usuário, e-mail, senha e confirmação da senha. Após inserir todas as informações corretamente é necessário clicar no botão “Registre-se” (Figura 35).

Figura 35 - Registro do Desenvolvedor

Fonte: Elaborada pelo autor.

Efetuado o registro, o sistema direciona para a página de *login*, onde é necessário informar o nome de usuário e a senha (Figura 36).

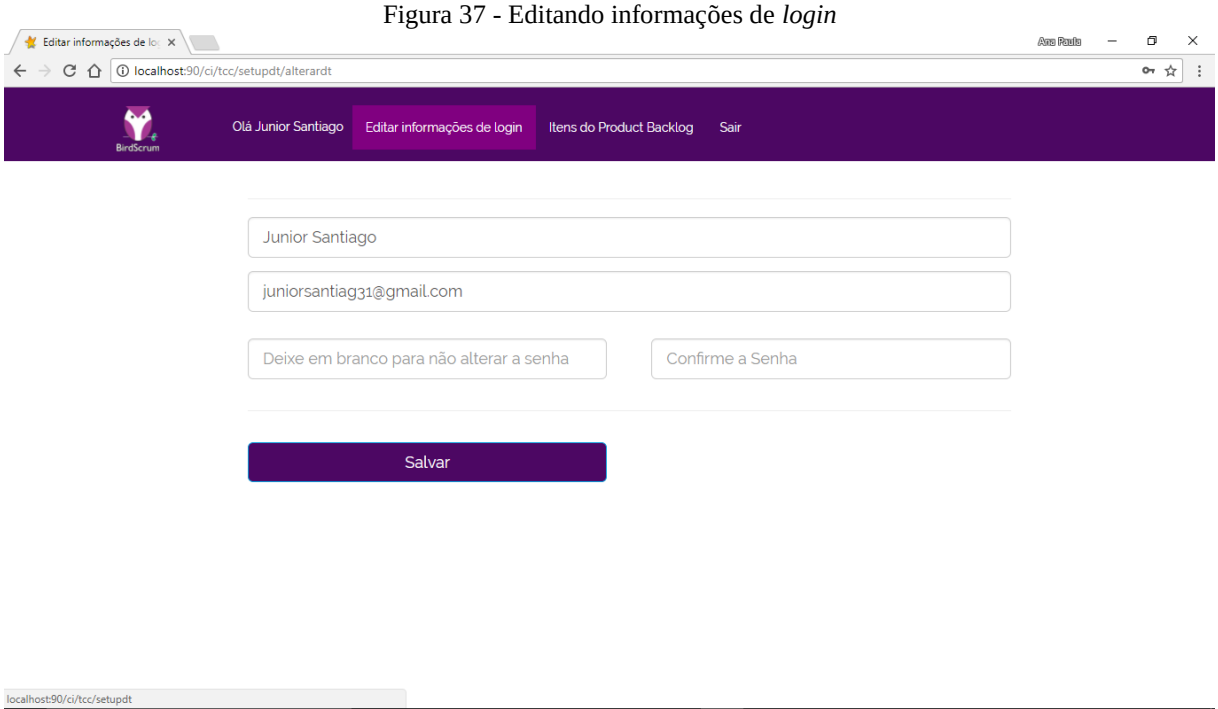
Figura 36 - *Login* Desenvolvedor



Fonte: Elaborada pelo autor.

Após efetuar *login*, o Desenvolvedor tem acesso ao seu ambiente de trabalho, que apresenta um menu com os itens: “Editar informações”, “Itens do Product Backlog” e “Sair”. O item de menu “Editar informações de login” permite que o Desenvolvedor altere suas credenciais, como mostra a Figura 37.

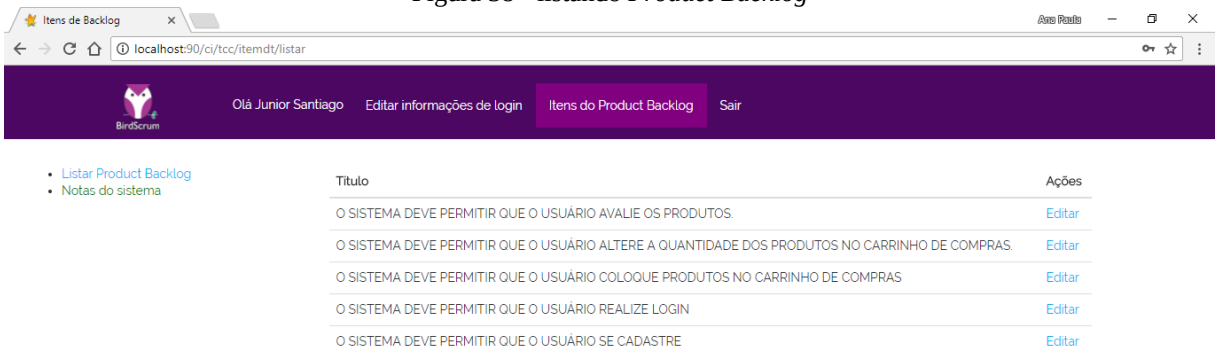
Figura 37 - Editando informações de *login*



Fonte: Elaborada pelo autor.

No item de menu “Itens do Product Backlog” o Desenvolvedor pode listar o *Product Backlog* (no *link* “Listar Product Backlog”) e verificar as mensagens do sistema (“Notas do sistema”). Os itens listados apresentam a opção de “Editar” (Figura 38).

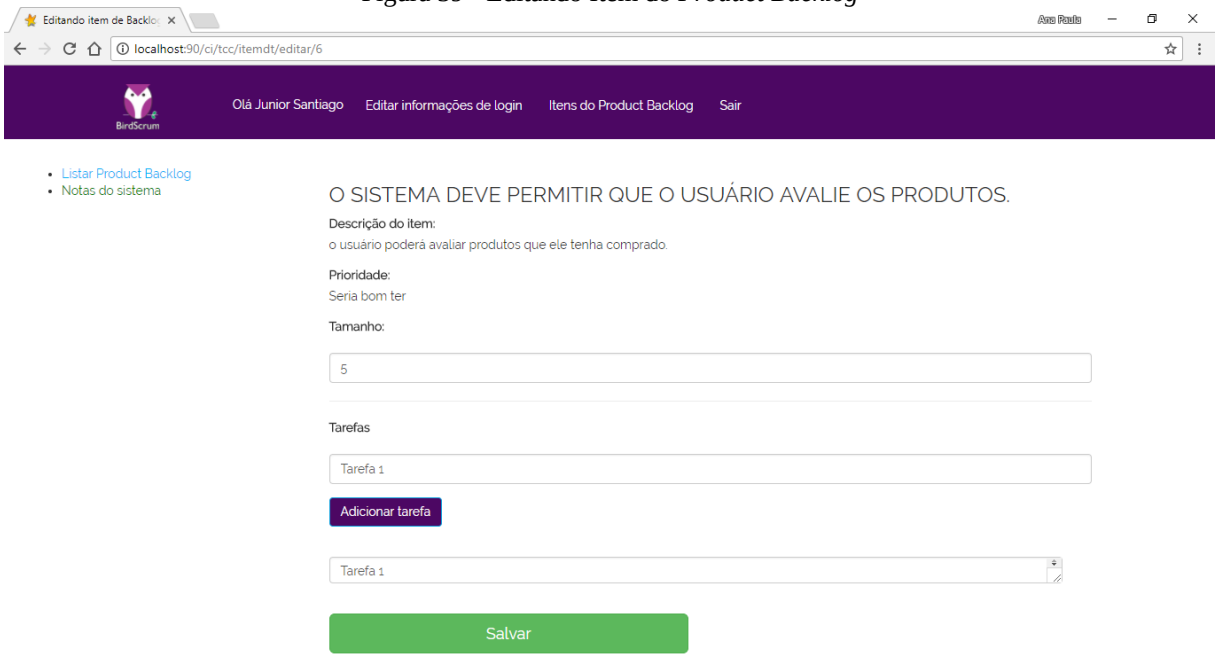
Figura 38 - listando *Product Backlog*



Fonte: elaborado pelo autor.

A opção “Editar”, possibilita inserir ou alterar informações como: tamanho e tarefas, conforme apresentado na Figura 39.

Figura 39 - Editando Item do *Product Backlog*



Fonte: Elaborada pelo autor.

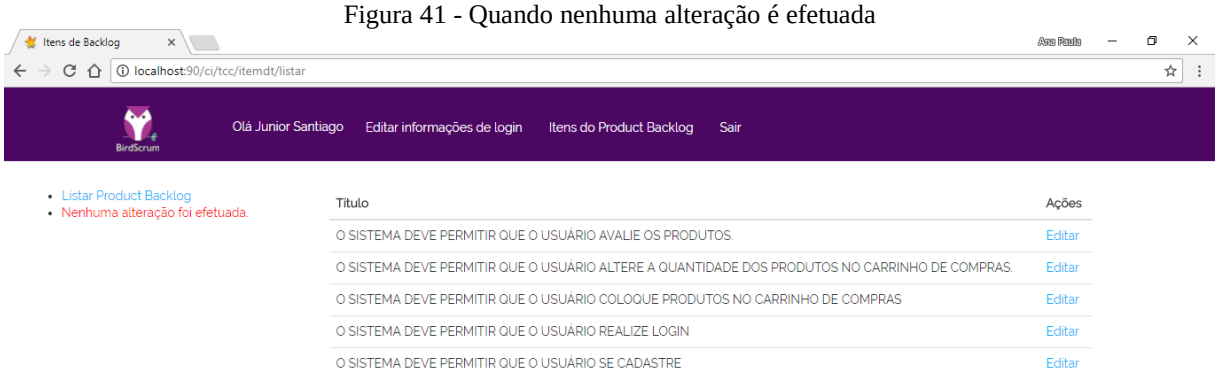
O botão salvar, quando clicado, registra as informações inseridas e lista todos os itens. O sistema apresenta uma mensagem informando que o item foi atualizado (Figura 40).

Figura 40 - Salvando alterações

Fonte: Elaborada pelo autor.

Caso nenhuma informação tenha sido alterada o sistema apresenta uma mensagem informando que nenhuma alteração foi efetuada (Figura 41).

Figura 41 - Quando nenhuma alteração é efetuada



Fonte: Elaborada pelo autor.

O item de menu “Sair” permite que o Desenvolvedor realize *logout* (sair).

3.6.3 Scrum Master

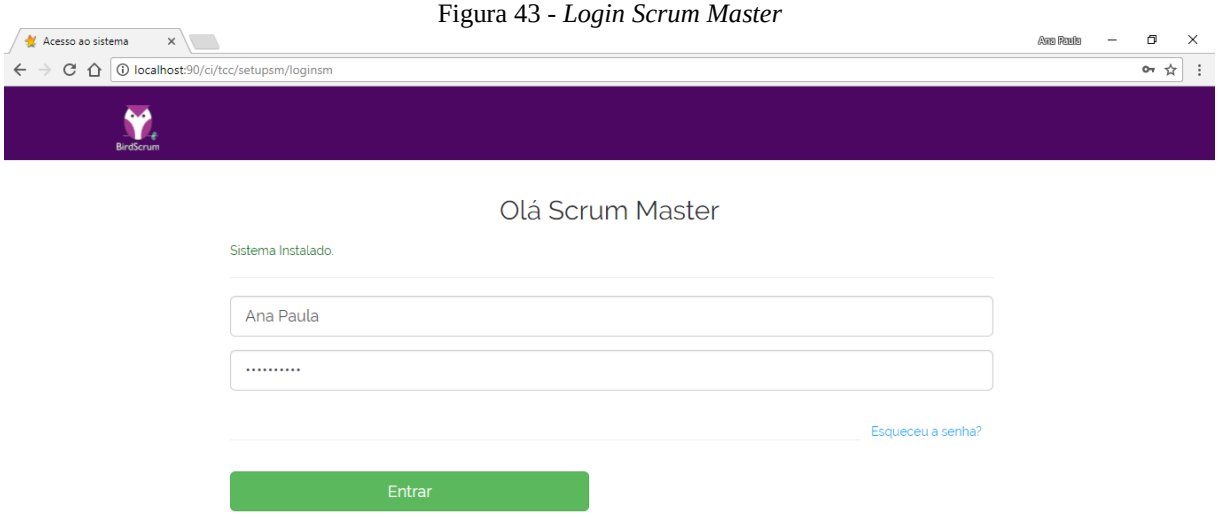
O sub-item de menu “Scrum Master” direciona para a página de registro, caso não haja cadastro como *Scrum Master* (Figura 42). Para efetuar o registro é necessário informar: nome de usuário, e-mail, senha e confirmação da senha. Após inserir todas as informações corretamente é necessário clicar no botão “Registre-se”.

Figura 42 - Registro *Scrum Master*

Fonte: Elaborada pelo autor.

Efetuado o registro, o sistema direciona para a página de *login*, onde é necessário informar o nome de usuário e a senha (Figura 43).

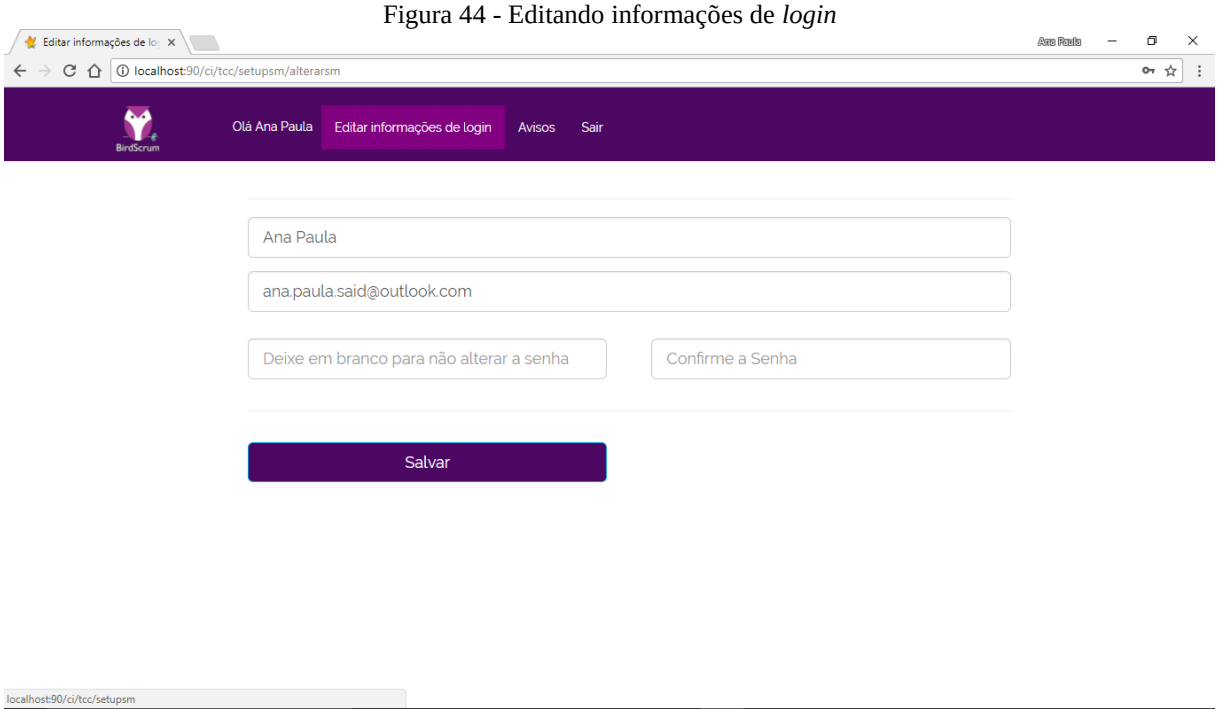
Figura 43 - *Login Scrum Master*



Fonte: Elaborada pelo autor.

Após efetuar *login*, o *Scrum Master* tem acesso ao seu ambiente de trabalho, que apresenta um menu com os itens: “Editar informações”, “Avisos” e “Sair”. O item de menu “Editar informações de login” permite que o *Scrum Master* altere suas credenciais, conforme apresenta a Figura 44.

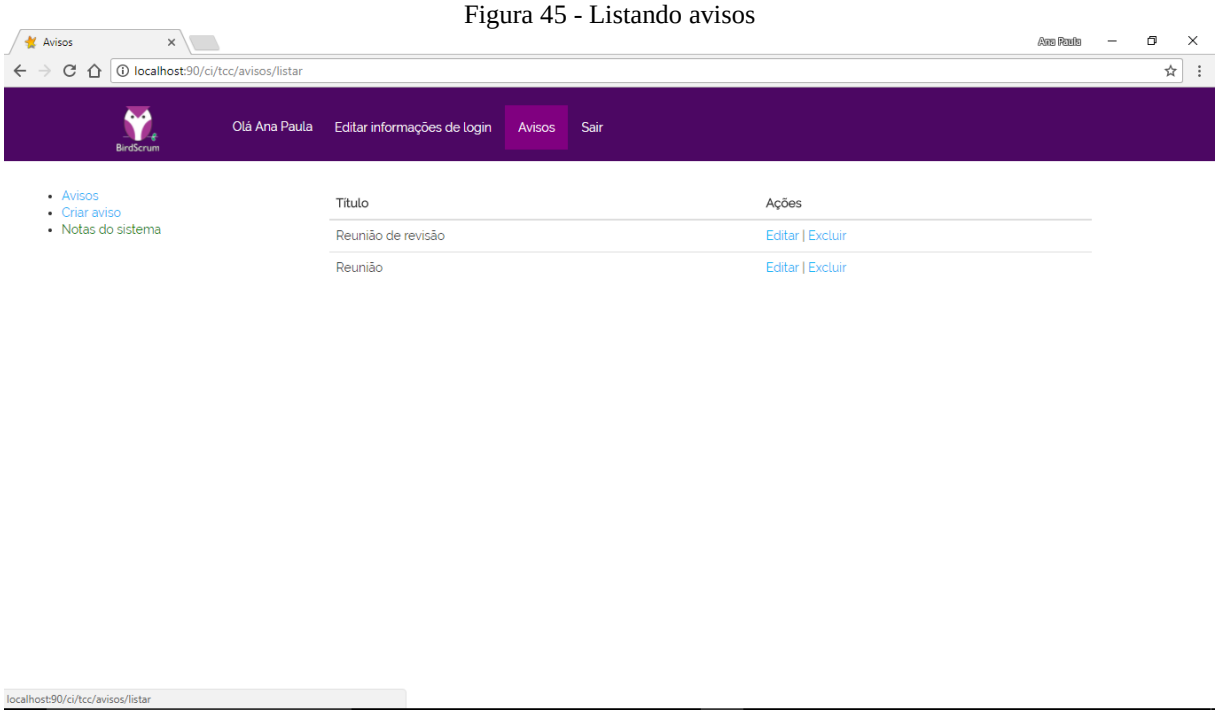
Figura 44 - Editando informações de *login*



Fonte: Elaborada pelo autor.

No item de menu “Avisos” o *Scrum Master* pode listar os avisos cadastrados (no *link* “Avisos”), criar novos avisos (no *link* “Criar Aviso”) e verificar as mensagens do sistema (“Notas do sistema”). Os avisos listados apresentam as opções de “Editar” e “Excluir” (Figura 45).

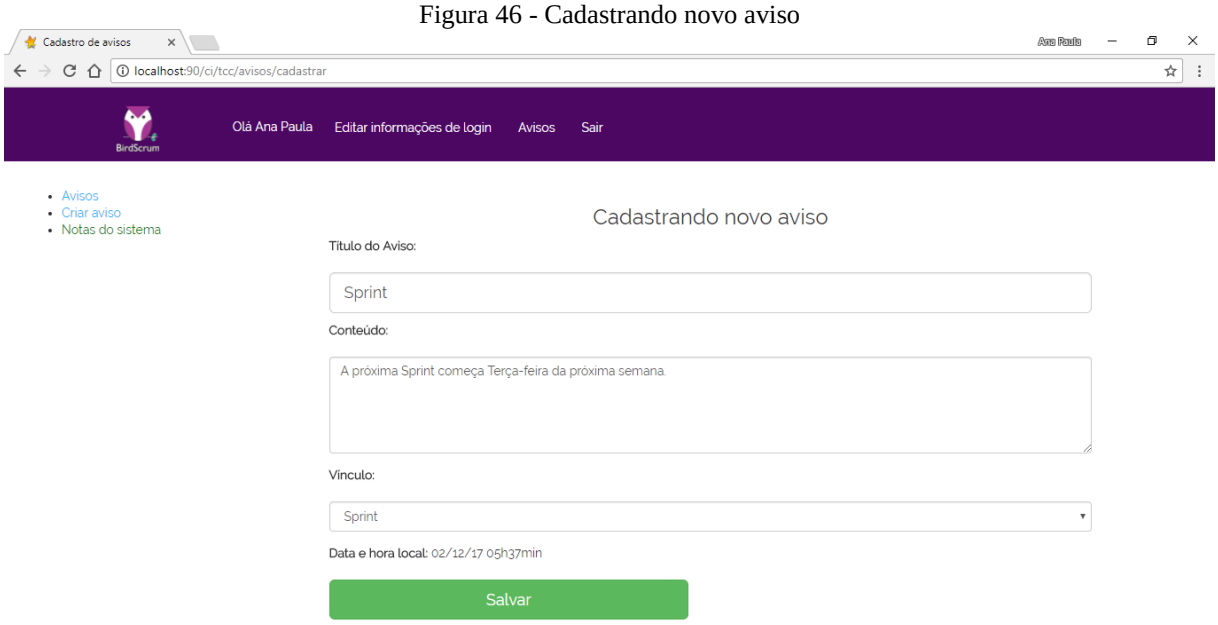
Figura 45 - Listando avisos



Fonte: Elaborada pelo autor.

Ao clicar em “Criar aviso” é possível cadastrar um novo aviso, informando: título do aviso, conteúdo e vínculo (Figura 46).

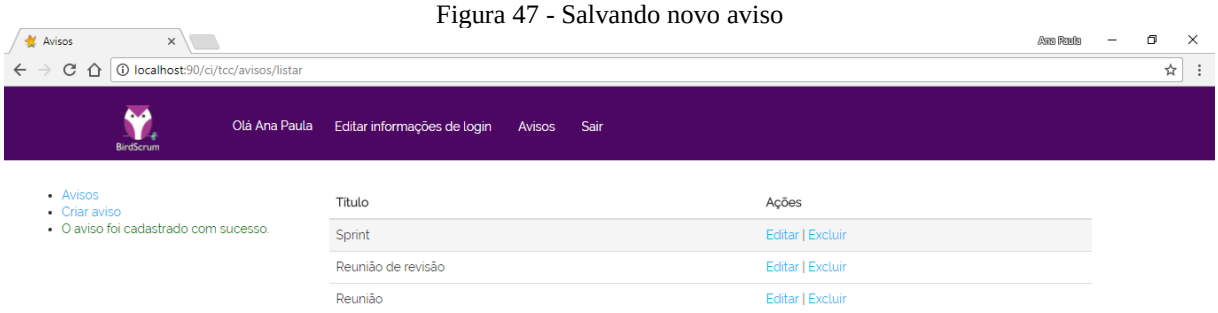
Figura 46 - Cadastrando novo aviso



Fonte: Elaborada pelo autor.

O botão “Salvar”, ao ser clicado, cadastra e lista o novo aviso com os demais. O sistema apresenta uma mensagem informando que o aviso foi cadastrado com sucesso (Figura 47).

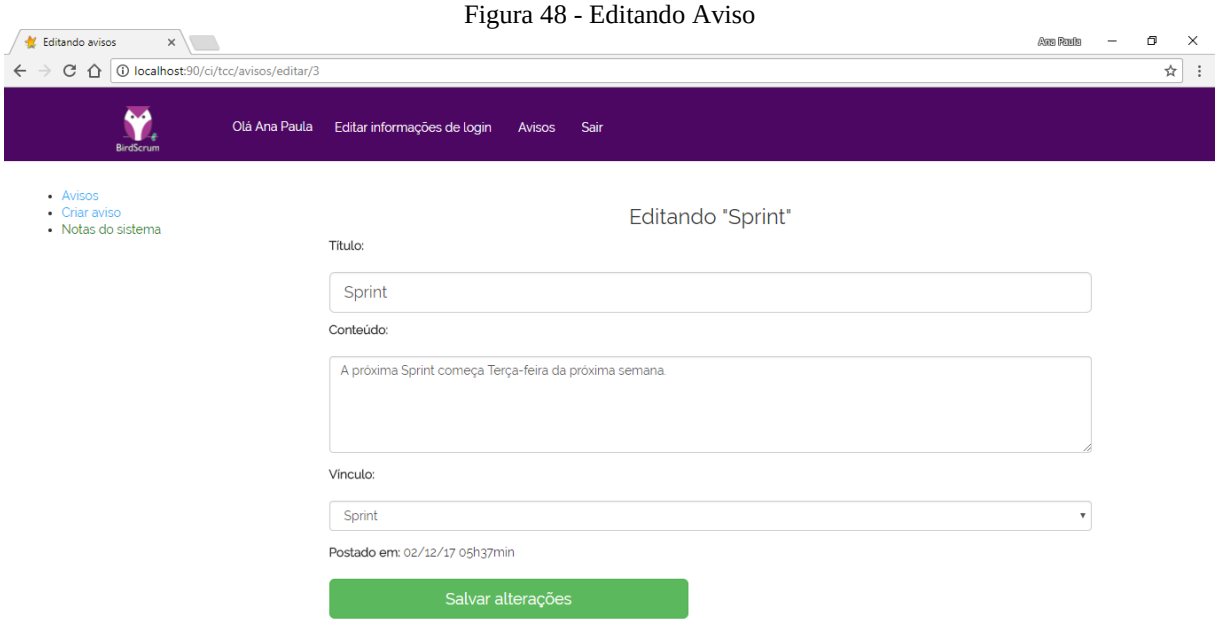
Figura 47 - Salvando novo aviso



Fonte: Elaborada pelo autor.

A opção “Editar”, presente nos avisos listados, permite que as informações do aviso selecionado sejam alteradas (Figura 48).

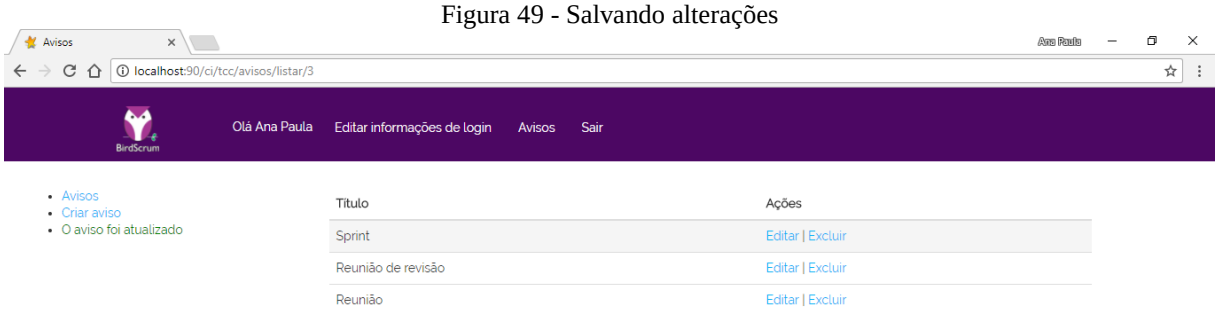
Figura 48 - Editando Aviso



Fonte: Elaborada pelo autor.

O botão “Salvar alterações”, ao ser clicado, registra as alterações e lista todos os avisos. O sistema apresenta uma mensagem informando que o aviso foi atualizado, como mostra a Figura 49.

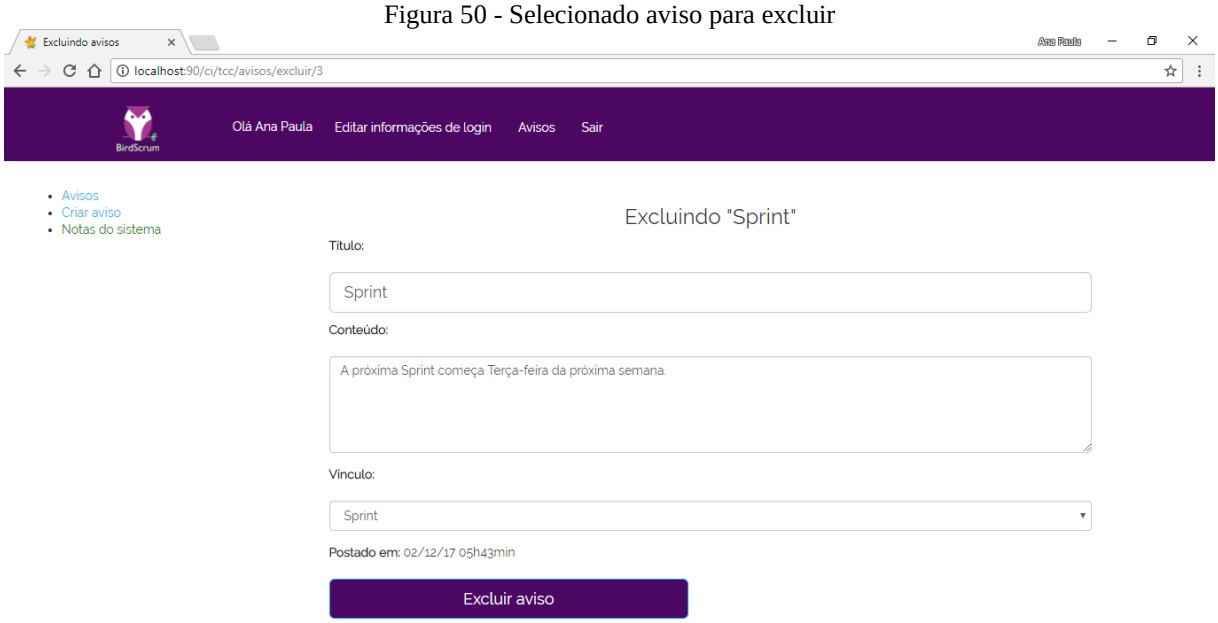
Figura 49 - Salvando alterações



Fonte: Elaborada pelo autor.

A opção “Excluir”, presente nos avisos listados, permite que o aviso selecionado seja excluído (Figura 50).

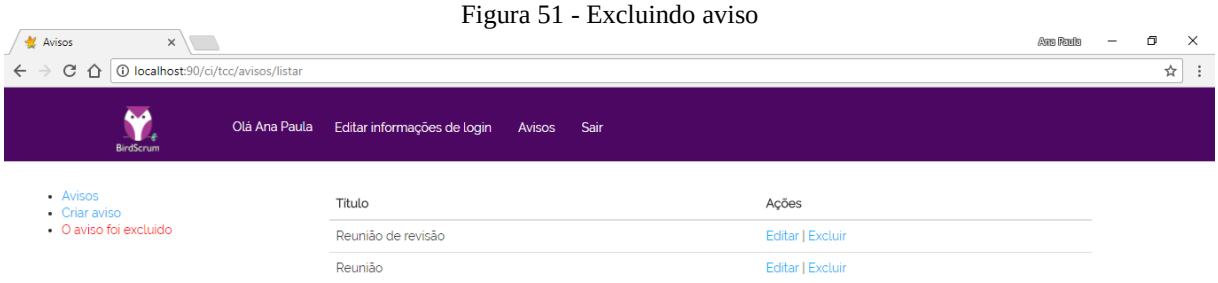
Figura 50 - Selecionado aviso para excluir



Fonte: Elaborada pelo autor.

O botão “Excluir aviso”, quando clicado, exclui o aviso e lista os demais. O sistema apresenta uma mensagem informando que o aviso foi excluído (Figura 51).

Figura 51 - Excluindo aviso



Fonte: Elaborada pelo autor.

O item de menu “Sair” permite que o Desenvolvedor realize *logout* (sair).

4 RESULTADOS E DISCUSSÃO

Este capítulo manifesta os resultados dos dados obtidos por meio do questionário aplicado aos envolvidos na fase de testes. A análise utilizada na pesquisa qualitativa, para tratar os dados adquiridos, detém caráter descritivo, considerando o objetivo de qualificar a experiência dos indivíduos perante a utilização do *software*.

4.1 ANÁLISE QUALITATIVA DO SOFTWARE

A Tabela 1 apresenta o questionário aplicado aos professores do curso Bacharelado em Sistemas de Informação que participaram da fase de testes. O questionário combinou perguntas abertas e fechadas, totalizando doze questões fechadas e apenas uma questão aberta. O questionário aplicado visou analisar os quesitos: usabilidade (questões de 1 a 4), desempenho (questões 5 e 6) e aplicabilidade (questões de 7 a 12).

Tabela 1 - Questionário para Análise do Software

Questão 1	É fácil aprender a manipular a aplicação?
Questão 2	O vocabulário utilizado é adequado?
Questão 3	O uso de ilustrações e ícones auxilia o aprendizado?
Questão 4	Os recursos estão dispostos de maneira adequada para serem utilizados?
Questão 5	O sistema apresentou falhas com frequência?
Questão 6	O tempo de resposta é satisfatório?
Questão 7	O sistema facilitou o aprendizado sobre os conceitos e práticas do Scrum?
Questão 8	O sistema favoreceu o aprendizado sobre o planejamento de projeto baseado em Scrum de maneira simplificada?
Questão 9	O sistema contribui para a formação do Time Scrum de maneira adequada?
Questão 10	O sistema contribui para tomada de decisão participativa?
Questão 11	O sistema contribui para estabelecimento de metas?
Questão 12	O sistema contribui para acompanhamento do andamento do projeto?
Questão 13	Descreva sua sugestão para melhoria e/ou expansão do sistema.

Fonte: Elaborado pelo autor.

A análise do *software* tendeu grandemente para a verificação da aplicabilidade, em razão do objetivo deste trabalho, consistente no desenvolvimento de protótipo. De acordo com

Sommerville (2007), o protótipo possibilita que os usuários avaliem como o sistema apoia suas atividades. Por meio do uso e avaliação de um protótipo os usuários podem ter novas ideias a respeito dos requisitos do sistema e encontrar falhas nos mesmos. As questões referentes a aplicabilidade foram baseadas no questionário do trabalho de mestrado "Gerenciamento ágil de projetos: proposta e avaliação de método para gestão de escopo e tempo" de Edivandro Carlos Conforto da Universidade de São Paulo, Escola de Engenharia de São Carlos.

A usabilidade compreendeu grande importância no desenvolvimento do protótipo, tendo como base as definições de Preece, Rogers e Sharp (2013).

O objetivo da questão final do questionário (questões 13) incluiu avaliar possibilidades para expansão e melhoria, desvendando os problemas abordados e suas soluções.

Nas questões fechadas foram utilizadas escalas de resposta do tipo *Likert* de cinco pontos: 5 - Concordo plenamente; 4 - Concordo parcialmente; 3 - Nem concordo nem discordo; 2 - Discordo parcialmente; 1 - Discordo totalmente.

A escala *Likert* é geralmente dividida em quatro ou cinco categorias ordinais (ALEXANDRE *et al*., 2003). Este instrumento foi escolhido para mensurar a experiência dos respondentes mediante a utilização do *software* e para favorecer o tratamento das informações adquiridas por meio do questionário.

Para a realização dos testes, foram selecionados três professores do curso Bacharelado em Sistemas e um técnico administrativo do IFMG-SJE para realizarem a análise do *software*, respondendo às perguntas do questionário aplicado. Verificou-se nos resultados que a maioria dos respondentes considera que o *software* possui grande usabilidade, sendo que alguns alegaram ainda que o sistema apresenta uma interface muito elegante. Também verificou-se em todas as respostas relacionadas ao desempenho, que o *software* não apresentou falhas e possui tempo de resposta satisfatório.

Sobre a aplicabilidade, todos os respondentes afirmaram que o *software* facilita o aprendizado sobre conceitos e práticas do *Scrum* e favorece o planejamento do projeto baseado em *Scrum* de maneira simplificada. A maioria dos respondentes concordou que o sistema contribui para a formação do Time Scrum de forma adequada e todos os respondentes consideraram que o sistema favorece a tomada de decisão participativa, favorece o estabelecimento de metas e o acompanhamento do projeto.

5. CONSIDERAÇÕES FINAIS

O *software* encontra-se finalizado, apresentando todas as funcionalidades previstas. Todos os objetivos e etapas propostas nesse trabalho foram concluídos, entretanto, observou-se que o produto desenvolvido possui grande potencial, podendo ser enriquecido com diversas funcionalidades do método ágil *Scrum*.

O *software* teve boa aceitação por parte dos envolvidos na pesquisa, que o avaliaram positivamente nos quesitos usabilidade, desempenho e aplicabilidade. O foco deste trabalho se manteve em expor os conceitos fundamentais do *Scrum* e incluir a prática de algumas atividades primordiais para que os resultados da avaliação do protótipo desenvolvido pudesse viabilizar a definição assegurada dos requisitos. A realização dos testes e a aplicação do questionário proporcionaram a coleta de comentários sobre os requisitos do *software*, revelando diversas soluções convenientes ao problema abordado.

Como proposta de trabalho futuro, o *software* pode comtemplar conceitos mais aprofundados sobre a dinâmica do *Scrum* e proporcionar ambientes de trabalho integrais para o exercício dos diversos papéis definidos pelo método. O método *Scrum* abrange muitas ferramentas para auxiliar os processos que podem ser integradas ao sistema, tornando o produto ainda mais rico. Além disso, verificou-se a viabilidade de possibilitar que o sistema permita o gerenciamento de múltiplos projetos.

REFERÊNCIAS

ALEXANDRE, J. W. C. et al. **Análise do número de categorias da escala de Likert aplicada à gestão pela qualidade total através da teoria da resposta ao item**. In: ENCONTRO NACIONAL DE ENGENHARIA DE PRODUÇÃO, 23. 2003. Ouro Preto. **Anais...** Ouro Preto: [s.l.], 2003.

AMBLER, S. W.; JEFFRIES, R. **Agile Modeling**. Nova York: John Wiley & Sons, 2002.

CREATALY. **About Creately , How it all Began and Meet the Team**. 2016. Disponível em: < https://creately.com/about-us>. Acesso em: 03 set. 2017.

BOEHM, B. **A View of 20th and 21st Century Software Engineering**. ‖ ICSE06.Shanghai, China: ACM, 2006.

CARVALHO, B. V.; MELLO, C. H. P. **Revisão, análise e classificação da literatura sobre o método de desenvolvimento de produtos ágil Scrum**. In: SIMPÓSIO DE ADMINISTRAÇÃO DA PRODUÇÃO, LOGÍSTICA E OPERAÇÕES INTERNACIONAIS – SIMPOI, 12. 2009, São Paulo. Anais... São Paulo, 2009.

CAVALCANTI, E. O. **Firescrum**: ferramenta de apoio à gestão de projetos utilizando Scrum. 2009. Disponível em: <https://pt.scribd.com/document/36064003/firescrum-ferramenta-de-apoio-a-gestao-de-projetos-utilizando-scrum#>. Acesso em: 11 set. 2017.

CODEIGNITER. **CodeIgniter Rocks.** 2017. Disponível em: < https://codeigniter.com/>. Acesso em: 21 set. 2017.

CORELDRAW. **CorelDRAW Graphics Suite 2017.** 2017. Disponível em:<https://www.coreldraw.com/br/product/software-de-design-grafico/>. Acesso em: 21 set. 2017.

CRUZ, R. S. L. **Metodologia Scrum**. 2006. Disponível em: http://www.scrum-master.com. Acesso em: 01 ago. 2017.

DEVMEDIA. **Modelo Incremental.** 2017. Disponível em: <http://www.devmedia.com.br/introducao-aos-processos-de-software-e-o-modelo-incremental-e-evolucionario/29839>. Acesso em: 22 set. 2017.

DIEHL, A. A. **Pesquisa em ciências sociais aplicadas:** métodos e técnicas. São Paulo:

Prentice Hall, 2004.

FERREIRA, D.; COSTA, F.; ALONSO, F.; ALVES, P.; NUNES, T. **Scrum:** Um Modelo Ágil para Gestão de Projetos de Software. 2005. Disponível em: <http://paginas.fe.up.pt/~aaguiar/es/artigos%20finais/es_final_19.pdf>. Acesso em: 15 set. 2017.

GARRETT, J. **The elements of user experience:** user-centered design for web. New York: New Riders, 2003.

GINIGE, A.; MURUGESAN, S. **Web Engineering:** an Introduction. IEEE Multimedia, vol. 8, Issue: 1, 2001, 14-18 p.

GUDWIN, R. R. **Introdução à Linguagem UML.** 2010. Disponível em: . Acesso em: 17 maio 2016.

KOSCIANSKI, A.; SOARES, M. S. **Qualidade de Software**. 2. ed. São Paulo: Novatec, 2007.

LOUDON, K. **Desenvolvimento de grandes aplicações Web.** São Paulo: Novatec, 2010.

PAULA FILHO, W. P. **Engenharia de Software:** Fundamentos, Métodos e Padrões. 3. ed. Rio de Janeiro: LTC, 2009.

PEREIRA, P.; TORREÃO, P.; MAÇAL, A. S. **Entendendo Scrum para Gerenciar Projetos de Forma Ágil**. (2007). In.: Mundo PM.

PEROVANO, D. G. **Manual de metodologia científica para a segurança pública e defesa social**. Curitiba: Juruá, 2014.

PREECE, J.; ROGERS, Y.; SHARP, H. **Design de interação:** além da interação humano-computador. 3. ed. Porto Alegre: Bookman, 2013.

PRESSMAN, R. S.; MAXIN, B. R. **Engenharia de Software:** uma abordagem profissional. 8. ed. São Paulo: McGraw Hill Brasil, 2016.

MACHADO, H. **Os 4 pilares da Programação Orientada a Objetos**. 2016. Disponível em: <http://www.devmedia.com.br/os-4-pilares-da-programacao-orientada-a-objetos/9264>. Acesso em: 08 set. 2017.

MORAES, A.; ROSA, G. **Avaliação e projeto no design de interface**s. Teresópolis: 2AB, 2012.

REZENDE, D. A. **Engenharia de software e sistemas de informação**. São Paulo: Brasport, 2005.

RINCON, A. M. **Qualidade de Software**. XI Encontro de Estudantes de Informática do Tocantins, p. 75-86, 2009.

SCHWABER, K.; SUTHERLAND, J. **Um guia definitivo para o Scrum:** As regras do jogo. 2009. 19 p. Disponível em: <https://www.scrumguides.org/docs/scrumguide/v1/Scrum-Guide-Portuguese-BR.pdf>. Acesso em: 02 set. 2017.

SILVA, A.; RIBEIRO, A.; RODRIGUES, L. **Sistemas de Informação na Administração Pública.** Rio de Janeiro: Revan, 2004.

SILVA, M. A. C.; FILHO, H. R.; SILVA, H. F. N. **Análise do BA durante o Processo Scrum**. (2010). XVII Simpósio de Engenharia de Produção. Bauru – SP, Novembro.

SILVA, M. S. **Criando sites com HTML:** sites de alta qualidade com HTML e CSS. São Paulo: Novatec, 2008.

SOARES, M. S. **Comparação entre metodologias Ágeis e tradicionais para o desenvolvimento de software**. INFOCOMP Journal of Computer Science, v. 3, n. 2, p. 8-13, 2004.

SOMMERVILLE, I. **Engenharia de Software**. 8. ed. São Paulo: Pearson Addison, 2007.
SUBLIME TEXT. **A sophisticated text editor for code, markup and prose**. 2017. Disponível em: <https://www.sublimetext.com>. Acesso em: 01 ago. 2017.

TABLELESS. **MVC – Afinal, é o quê?** 2017. Disponível em: <https://tableless.com.br/mvc-afinal-e-o-que/>. Acesso em: 01 jun. 2017.

VERTRIGOSERV WAMP SERVER. **Project information**. 2017. Disponível em:

<http://vertrigo.sourceforge.net/>. Acesso em: 01 ago. 2017.

W3C BRASIL. **Padrões Web**. 2017. Disponível em: <http//www.w3c.br/Padroes>. Acesso em: 05 ago. 2017.

W3SCHOOLS. **Bootstrap (3) Tutorial**. 2017a. Disponível em: <https://www.w3schools.com/bootstrap/default.asp>. Acesso em: 08 maio 2017.

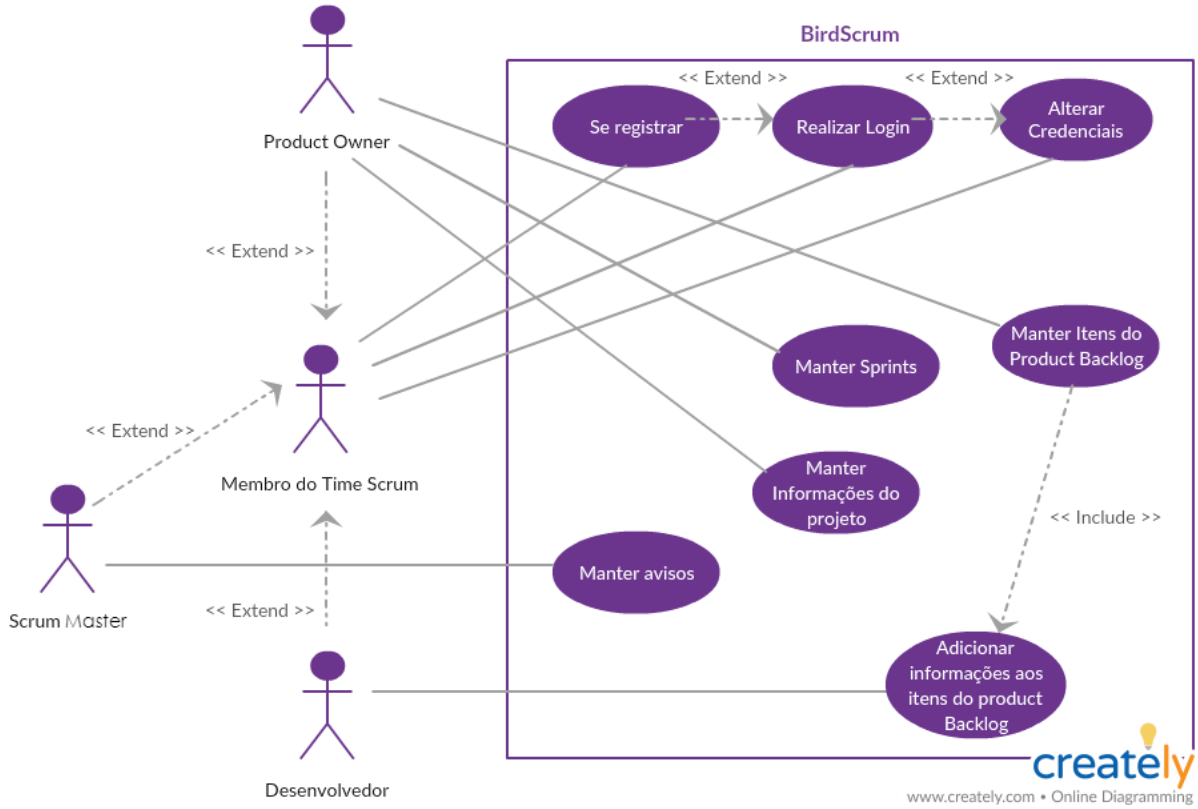
_____. **jQuery Introduction**. 2017b. Disponível em: <http:// www.w3schools.com/jquery/jquery_intro.asp>. Acesso em: 08 maio 2017.

WAZLAWICK, R. **Engenharia de Software:** Conceitos e Práticas. Elsevier Brasil, 2013.

WINCKLER, M.; PIMENTA, M. S. **Avaliação de usabilidade de sites web.** Escola de Informática da SBC SUL (ERI 2002). Porto Alegre: Sociedade Brasileira de Computação (SBC), v. 1, p. 85-137, 2002.

APÊNDICES

APÊNDICE A – Diagrama de caso de uso



APÊNDICE B – Diagrama de classes