UNIVERSIDADE ESTADUAL PAULISTA
"JÚLIO DE MESQUITA FILHO"
Campus de São José do Rio Preto

Felipe de Lima Pimenta

Abordagem para gerenciamento ágil de mudanças nos requisitos

São José do Rio Preto
2021

Felipe de Lima Pimenta

Abordagem para gerenciamento ágil de mudanças nos requisitos

Monografia apresentada ao Departamento de Ciências de Computação e Estatística do Instituto de Biociências, Letras e Ciências Exatas da Universidade Estadual Paulista "Júlio de Mesquita Filho", como parte dos requisitos necessários para a aprovação na disciplina Trabalho de Conclusão de Curso.

Orientadora:

Profa. Dra. Rogéria Cristiane Gratão de Souza

São José do Rio Preto
2021

P644a

Pimenta, Felipe de Lima
Abordagem para gerenciamento ágil de mudanças nos requisitos / Felipe de Lima Pimenta. -- São José do Rio Preto, 2021
61 p. : il., tabs.

Trabalho de conclusão de curso (Bacharelado - Ciência da Computação) - Universidade Estadual Paulista (Unesp), Instituto de Biociências Letras e Ciências Exatas, São José do Rio Preto
Orientadora: Rogéria Cristiane Gratão de Souza

1. Desenvolvimento ágil de software. 2. Scrum (Desenvolvimento de software). 3. Documentação. I. Título.

Sistema de geração automática de fichas catalográficas da Unesp. Biblioteca do Instituto de Biociências Letras e Ciências Exatas, São José do Rio Preto. Dados fornecidos pelo autor(a).

Essa ficha não pode ser modificada.

Felipe de Lima Pimenta

Abordagem para gerenciamento ágil de mudanças nos requisitos

Monografia apresentada ao Departamento de Ciências de Computação e Estatística do Instituto de Biociências, Letras e Ciências Exatas da Universidade Estadual Paulista "Júlio de Mesquita Filho", como parte dos requisitos necessários para a aprovação na disciplina Trabalho de Conclusão de Curso.

Banca Avaliadora:

Profa. Dra. Rogéria Cristiane Gratão de Souza
Orientadora

Profa. Dra. Carina Alexandra Rondini

Prof. Dr. Leandro Alves Neves

São José do Rio Preto
2021

À minha família e amigos.

"What we know is a drop. What we do not know... is an ocean."

- Adam

Agradecimentos

Agradeço primeiramente aos meus amigos e familiares que me apoiaram durante toda minha jornada acadêmica.

Agradeço aos meus pais por todo o apoio pessoal e financeiro durante os vários anos de graduação.

Agradeço também à minha orientadora Profa. Dra. Rogéria Cristiane Gratão de Souza, que aceitou me orientar mesmo depois de muito tempo enrolando e não deixou de me ajudar em nenhum momento, e tornou a realização deste trabalho possível.

Agradeço aos meus amigos Leonardo Longo, Thiago Leal, Luis Felipe, Diego Lacerda, Guilherme Gervaes, Matheus Cassarotti e outros, que ouviram incansavelmente minhas reclamações e me ouviram e ajudaram de diversas formas na conclusão deste trabalho. Agradeço também à minha amiga Natália de Grande, pela ajuda nos momentos que o inglês sumiu da minha cabeça, além da revisão do abstract.

Finalmente, agradeço à empresa PCA pela oportunidade de estágio que me proporcionou conhecimentos técnicos que permitiram o desenvolvimento do protótipo deste trabalho.

Resumo

Mudanças nos requisitos de projetos ocorrem no mundo continuamente e, portanto, o desenvolvimento e entrega rápidos se tornaram essenciais. Os métodos ágeis surgem então para atender a realidade que abordagens tradicionais da Engenharia de Software tinham restrições para contemplar, tais como a entrega rápida e constante. O Scrum, a metodologia ágil mais usada atualmente, tem como um dos seus princípios a aceitação de mudanças durante o desenvolvimento. Devido à iteração rápida das metodologias ágeis, existe pouca ou nenhuma documentação do sistema e das mudanças que ocorrem nos requisitos, o que pode afetar o tempo e o orçamento do projeto em razão da falta de controle sobre o que está sendo realizado. Assim, o presente trabalho propõe uma abordagem para ajudar as equipes de desenvolvimento de software no gerenciamento das mudanças nos requisitos em um ambiente ágil Scrum. Para tanto, desenvolveu-se o protótipo *History Teller*, que permitiu avaliar a eficácia da abordagem proposta. Participaram da avaliação 22 pessoas, entre elas 13 desenvolvedores e nove gerentes de projeto com experiência no mercado de trabalho. Os resultados obtidos evidenciam que a abordagem apresentada auxilia no rastreio das mudanças nos requisitos por meio da manutenção do histórico das mesmas e pode ser útil em projetos que usam a metodologia ágil Scrum, contribuindo para um acompanhamento efetivo das mudanças no projeto.

Palavras-chave: Gerenciamento ágil, Mudanças nos requisitos, Documentação das mudanças.

Abstract

Changes in projects requirements continuously occur worldwide; therefore, rapid development and delivery have become essential. As a result, agile methodologies arise to compensate for the restrictions of the traditional Software Engineering approaches through fast and constant delivery. Currently, Scrum is the most used agile methodology, and one of its principles is the acceptance of changes during development. Because of the rapid iteration in such methodologies, there is little or no documentation involving the system and changes in the requirements, which may affect project time and budget due to a lack of control over the process. Accordingly, our study proposes an approach to help software development teams to manage the changes in the requirements in a Scrum agile environment. For this purpose, we developed the "History Teller" prototype that allowed us to evaluate the effectiveness of the proposed approach. This evaluation consisted of 22 individuals, including 13 developers and nine project managers with experience in the job market. The participants performed previously defined tasks and filled an online form to gather the answers. The results revealed that our approach helps track the requirements changes and may be useful in projects that use the Scrum agile methodology, contributing to effectively monitoring changes in the project.

Keywords: Agile managment, requirements change, change documentation.

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Abreviaturas e Siglas

FDD: *Feature Driven Development*

GMR: Gerência de Mudança de Requisitos

IBM: *International Business Machines Corporation*

XP: *Extreme Programming*

Capítulo 1

Introdução

O mundo passa por mudança constante e rápida de ambiente. Empresas trabalham em resposta à oportunidades novas, à condições que mudam continuamente e aos produtos de seus concorrentes. O desenvolvimento e a entrega rápidos se tornaram algo essencial. Tais mudanças, assim como regulações governamentais, demandas de mercado e ambiente de trabalho, influenciam os requisitos dos projetos (NURMULIANI; ZOWGHI; POWELL, 2004; AKBAR et al., 2018).

Até então, os processos usados para o desenvolvimento de sistemas de software por meio de abordagens tradicionais da Engenharia de Software, não atendiam plenamente essa realidade. Assim, as metodologias ágeis surgiram com o intuito de lidar com essa rápida e imprevisível evolução do mercado e seus requisitos (KUMAR; BHATIA, 2012).

Os métodos ágeis, no geral, propõem diferentes processos baseados no desenvolvimento e entrega incrementais para atingir seu objetivo. Porém, todos eles seguem um grupo de princípios formalmente definidos em 2001 pelo Manifesto Ágil: envolvimento do cliente, entregas incrementais, foco nas pessoas e não no processo, aceite às mudanças e manutenção da simplicidade (AGILE, 2001).

Dentre os diversos métodos ágeis, o mais usado é o Scrum (STAVRU, 2014) que, como outras metodologias ágeis, aceita mudanças durante qualquer estágio do pro-

cesso. Essas mudanças, porém, podem afetar o orçamento e o tempo esperado para conclusão do projeto em desenvolvimento (ALSALEMI; YEOH, 2015), caso não sejam bem gerenciadas.A principal diferença do Scrum em relação à outras metodologias ágeis é o foco no gerenciamento do processo, e não do desenvolvimento (AGILE, 2001).

Neste contexto, apesar de ser um *framework* adaptável, o Scrum não possui uma diretriz de como gerenciar as mudanças que surgem nos requisitos. A partir desta constatação, foi possível estabelecer duas hipóteses para nortear o desenvolvimento do presente trabalho:

- **H1:** o histórico de mudanças efetuadas nos requisitos de um projeto, desenvolvido com a metodologia Scrum, contribui para a rastreabilidade adequada de tais requisitos;
- **H2:** o conhecimento sobre quais foram as mudanças realizadas nos requisitos ajuda na tomada de decisões durante o desenvolvimento do projeto.

1.1 Justificativa e motivação

Mudanças nos requisitos do projeto de um sistema são inevitáveis e previstas pelo Manifesto Ágil, que sugere que o sistema seja projetado de forma que acomode mudanças no futuro (AMBLER, 2005; SOMMERVILLE, 2007). No Scrum, *sprints* são iterações do ciclo de desenvolvimento de software (SHARMA; SARKAR; GUPTA, 2012) e histórias de usuário são artefatos usados no desenvolvimento de software e descrevem necessidades do usuário, ou seja, os requisitos do projeto (ANANJEVA; PERSSON; BRUUN, 2020; CHOPADE; DHAVASE, 2017). Após várias *sprints*, muitas das histórias de usuários não estão mais relacionadas umas com as outras, são contradizentes entre si, ou modificam definições iniciais, em decorrência de mudanças sofridas (URBIETA et al., 2020).

Em muitos casos, desenvolvedores assumem que o código-fonte é uma melhor fonte de informação que outros documentos (NIAZI et al., 2013), o que torna necessário entender, em projetos Scrum, qual parte do código está relacionado com qual história de usuário. Não somente, quando um membro da equipe necessita entender um requisito, depender de um outro membro da equipe pode não ser a melhor das opções: equipes mudam e perspectivas ou vocabulários diferem, dificultando o entendimento da evolução dos requisitos entre membros da equipe (URBIETA et al., 2020) e dos *stakeholders*, principalmente em locais geograficamente diferentes (JAYATILLEKE; LAI, 2018), (AL-ZAIDI; QURESHI, 2017).

É necessária, então, uma forma para a equipe gerenciar esses requisitos e suas mudanças. Três trabalhos (AKBAR et al., 2017; BHATTI et al., 2010; INCE, 1995 apud AKBAR et al., 2019) introduzem modelos para Gerência de Mudança de Requisitos (GMR) com o intuito de auxiliar empresas desenvolvedoras de software no controle de mudanças de requisitos, visto que a maior razão do cancelamento ou atraso de projetos são requisitos incompletos ou com mudanças constantes (SAHER; BAHAROM; GHAZALI, 2017). Os modelos propostos, porém, foram sugeridos não considerando as metodologias ágeis, pois os mesmos são processos relativamente longos e demorados, por consequência de seus vários passos.

A principal motivação para o desenvolvimento deste trabalho, então, se deve à identificação da carência de trabalhos relacionados à proposição de ferramentas e métodos para gerência e documentação de mudanças de requisitos em processos ágeis e, mais especificamente, no Scrum. Atacando essa demanda, vislumbra-se a possibilidade de contribuir com a comunidade científica, por meio de uma proposta voltada para a abordagem ágil que visa a documentação de mudanças nos requisitos em um ambiente de desenvolvimento Scrum.

1.2 Objetivo

O objetivo principal deste trabalho é contribuir para o processo de desenvolvimento e documentação ágil, por meio do gerenciamento das mudanças nos requisitos desenvolvidos com a metodologia Scrum. Para isso, tem-se dois objetivos específicos:

- Desenvolver mecanismos de apoio para que membros da equipe do desenvolvimento de um projeto, que utilizam como metodologia de desenvolvimento o Scrum, possam documentar e manter o histórico de mudanças nos requisitos dos projetos de forma simplificada, por meio da definição de template, e garantindo o acompanhamento do histórico de mudanças ocorridas;

- Desenvolver um protótipo de sistema, denominado *History Teller*, para apoiar a documentação e visualização das mudanças de requisitos de forma ágil, com o intuito de auxiliar na tomada de decisões relacionadas ao processo de desenvolvimento de projetos Scrum.

1.3 Metodologia

A realização deste trabalho foi composta por cinco etapas.

Na etapa 1 tem-se o levantamento bibliográfico sobre os tópicos relacionados ao trabalho, envolvendo conceitos como metodologias ágeis, como elas lidam com mudanças, requisitos e documentação, com foco no Scrum, explicando como esta metodologia trabalha com requisitos no projeto e suas mudanças ao longo do tempo.

Na etapa 2 tem-se a identificação de requisitos necessários para a definição dos mecanismos de apoio à documentação e gerenciamento ágil de requisitos e suas mudanças, embasando o desenvolvimento do protótipo.

Na etapa 3 tem-se o estabelecimento da arquitetura a partir dos requisitos definidos na etapa anterior, priorizando a facilidade de manutenção no futuro.

Em seguida, na etapa 4, tem-se a definição dos recursos computacionais necessários para a implementação do protótipo, priorizando recursos gratuitos.

Após a implementação, tem-se na etapa 5, a avaliação do protótipo por profissionais convidados, com o intuito de avaliar a usabilidade, a adequação das funções e as possíveis melhorias e mudanças almejadas para evidenciar a contribuição do trabalho.

1.4 Organização da Monografia

Nesta seção, apresenta-se os demais capítulos que compõem a presente monografia de Trabalho de Conclusão de Curso.

No Capítulo 2 tem-se o levantamento bibliográfico, proporcionando um embasamento teórico sólido para a compreensão do trabalho, bem como a análise de trabalhos relacionados.

No Capítulo 3 mostra-se a arquitetura estabelecida para o projeto, sua descrição e os recursos computacionais usados na implementação do protótipo. Finalmente, é apresentada a descrição das funções implementadas.

No Capítulo 4 apresenta-se o perfil dos avaliadores do protótipo, o processo usado para avaliar, as tarefas analisadas, resultados obtidos e análise dos mesmos.

Finalmente, no Capítulo 5 discute-se os resultados obtidos, suas contribuições e propostas de trabalhos futuros.

Capítulo 2

Revisão bibliográfica

Neste capítulo são apresentados conceitos relacionados à metodologia ágil e gerência de mudanças, além de evidenciar carências existentes, com base na literatura atual da área, que motivaram o desenvolvimento do presente trabalho. Nas seções 2.1, 2.2 e 2.3, apresenta-se o que são metodologias ágeis e sua importância, o que é o Scrum e como ele lida com histórias de usuário e sua documentação. Então, na seção 2.4, é discutido o gerenciamento de mudanças no processo de desenvolvimento de *software*. Em seguida, na seção 2.5, são apresentados trabalhos relacionados ao gerenciamento ágil de mudanças bem como ferramentas disponíveis no mercado para auxílio nesse gerenciamento, de forma a evidenciar as suas limitações que justificam a necessidade de novos estudos capazes de oferecer contribuições como as apresentadas neste trabalho.

2.1 Metodologias ágeis

O desenvolvimento de software surgiu como uma atividade indisciplinada, em que o software era escrito quase sem nenhum plano e o projeto era feito a partir de decisões tomadas a curto prazo. Esta prática deixou de funcionar quando sistemas se tornaram maiores e mais complexos, o que dificultava a adição de novas funções e a correção de

problemas (AWAD, 2005).

Para ajudar a solucionar este problema, surgiram as metodologias tradicionais, tais como o Modelo Cascata e o Modelo Espiral. Nas metodologias tradicionais, o trabalho começa com a realização da elicitação e documentação dos requisitos, seguido do desenvolvimento de alto nível do sistema, assim como o projeto da arquitetura.

Alguns praticantes acreditavam que este processo era frustrante e trazia dificuldades, mesmo quando as mudanças eram poucas, pois tal processo dependia de uma sequência de passos longos e burocráticos. A partir disso, surgiram as metodologias ágeis (AWAD, 2005).

As metologias ágeis buscam fornecer uma melhor abordagem ao processo de desenvolvimento de *software*, considerando a demanda do mercado atual em que os requisitos sofrem constantes mudanças e é necessário entregar um produto funcional o mais rápido possível, por meio de iterações e testes contínuos. Outros objetivos dos métodos ágeis incluem diminuir o tempo de desenvolvimento do projeto e reduzir o número de problemas técnicos (CHO, 2008). Em 2001, o Manifesto Ágil definiu princípios básicos que deram origem à diferentes metologias ágeis, aos quais cada uma das metodologias aderem em diferentes níveis (DINGSØYR et al., 2012). Os princípios são:

- Indivíduos e interações mais que processos e ferramentas;
- *Software* em funcionamento mais que documentação abrangente;
- Colaboração com o cliente mais que negociação de contratos;
- Responder a mudanças mais que seguir um plano.

Existem diferentes metodologias ágeis disseminadas no mercado, tais como *Extreme Programming* (XP), Crystal Methodologies, Feature Driven Development (FDD), Scrum. Destas, o Scrum é a mais usada (SRIVASTAVA; BHARDWAJ; SARASWAT,

2017; ANGUELOV, 2019; PEREIRA; RUSSO, 2018), sendo, portanto, adotada no presente trabalho.

2.2 Scrum

O Scrum é uma metodologia ágil para desenvolvimento de sistemas, projetada para acelerar o desenvolvimento, definir uma cultura focada em performance, fornecer suporte à criação de valor por *stakeholders* e melhorar o desenvolvimento pessoal e qualidade de vida (SRIVASTAVA; BHARDWAJ; SARASWAT, 2017; SUTHERLAND et al., 2007). Suas características incluem entregáveis e cronograma flexíveis, times pequenos, revisões frequentes e colaboração (SCHWABER, 1997).

O nome Scrum tem origem no esporte Rugby, que possui uma jogada com mesmo nome, que consiste de uma formação cerrada (*tight formation*) entre partes de cada time com o objetivo de recuperar a bola (CERVONE, 2011). Além do nome, a metodologia Scrum compartilha outras características com o esporte Rugby (SCHWABER, 1997):

- O objetivo é mover a bola para a frente. No caso do metodologia, o objetivo é melhorar o produto desenvolvido;
- O esporte surgiu e evoluiu a partir da quebra de regras do futebol. No caso da metodologia, o Scrum, assim como outras metodologias ágeis, surgiram a partir da quebra de regras das metodologias tradicionais;
- O jogo não acaba até que condições do ambiente ditem o mesmo. Na metodologia, regras e necessidades do mercado ditam a continuidade do processo.

A metodologia Scrum consiste em três componentes: papéis, cerimônias e artefatos (CHO, 2008). Existem três papéis distintos no Scrum: O *Product Owner*, responsável pelo financiamento e criação dos requisitos gerais do projeto; o time, responsável

pela implementação das funções descritas nos requisitos; e o *Scrum Master*, responsável por garantir que os valores, práticas e regras do Scrum estão sendo cumpridos.

Sobre cerimônias, o Scrum possui as *daily scrums*, *sprint review*, *sprint planning* e *sprint retrospective*. Já com relação aos artefados, o Scrum possui o *product backlog* e o *sprint backlog* (CHO, 2008). Para explicar o significado das cerimônias e artefatos, o processo seguido pelo Scrum e esquematizado na Figura 2.1 é descrito.

Tal processo começa com a criação de um *product backlog* inicial, ou seja, uma lista de requisitos que definem um produto. De acordo com a prioridade estabelecida (CHO, 2008), o *product backlog* é dividido em *sprint backlogs*, ou seja, cada *sprint backlog* representa um subconjunto de requisitos que será trabalhado em uma *sprint* que pode durar de uma semana até a um mês, a depender da equipe. O *product backlog* é atualizado e refinado ao longo do ciclo de vida do desenvolvimento de software, conforme surjam as mudanças de requisitos, qualquer que seja a razão (RUBIN, 2012).

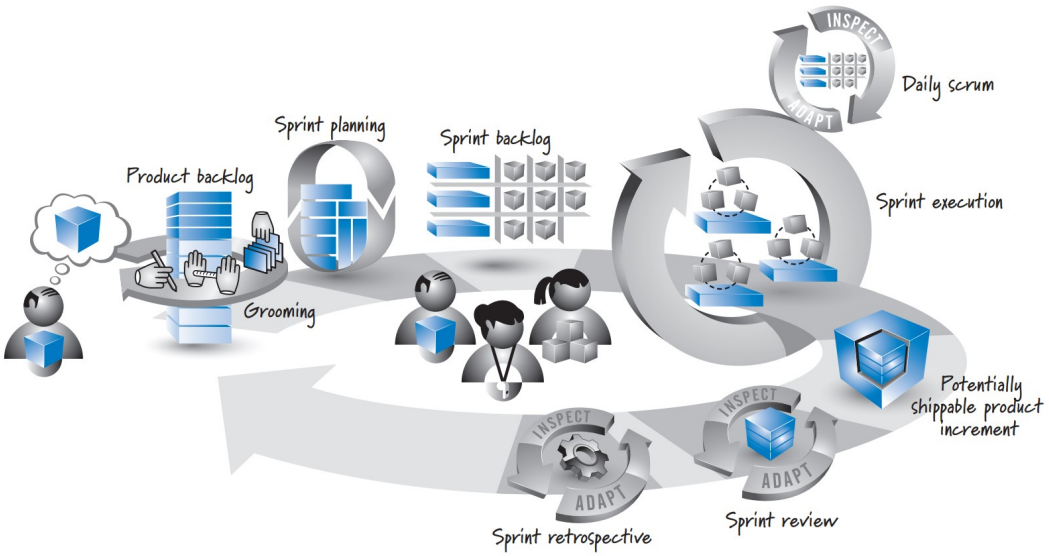
Cada *sprint* é definida com o objetivo de, no final dela, finalizar um produto potencialmente entregável, sendo dividida em quatro passos: planejamento, desenvolvimento/testes, revisão e retrospectiva (SRIVASTAVA; BHARDWAJ; SARASWAT, 2017).

O planejamento de uma *sprint* (*sprint planning*) começa com a criação de um *sprint backlog*, em que são descritas tarefas a serem cumpridas para que a equipe possa projetar, desenvolver, integrar e entregar um subconjunto de funções do *product backlog* (RUBIN, 2012)

O desenvolvimento/testes é o período em que a equipe realiza as tarefas do *sprint backlog*. Todo dia a equipe se reune para discutir o trabalho realizado, tirar dúvidas e sincronizar o trabalho entre si, o que é chamado de *daily scrum* (RUBIN, 2012).

No final da *sprint*, ocorrem os dois últimos passos: revisão e retrospectiva. Na revisão (*sprint review*), os *stakeholders* e time Scrum analisam o produto desenvolvido. Na retrospectiva (*sprint retrospective*), o time Scrum analisa o processo scrum

Figura 2.1: Framework scrum.



Fonte: Rubin (2012)

utilizado para criar o produto e, caso necessário, são realizadas adaptações no processo ou no *sprint backlog*. É neste passo que quaisquer mudanças são introduzidas (AWAD, 2005). Então, o ciclo da *sprint* recomeça (RUBIN, 2012), até que o sistema seja considerado legado.

O Scrum, porém, possui problemas e desafios. Como outros métodos ágeis, o Scrum reduz significamente a documentação, o que pode causar dificuldades de entendimento do sistema quando o time responsável pela manutenção futura do sistema é composto por pessoas que não participaram do time de desenvolvimento geral. Com isso, tem-se um aumento de tempo necessário para que os novos membros da equipe entendam o código fonte correspondente.

Outro desafio é a comunicação, visto que times nem sempre conversam entre si, o que pode gerar códigos duplicados, por exemplo. Outro desafio inclui as cerimônias, que por vezes podem demorar mais do que o necessário, ou são dificeis de agendar, devido aos diferentes horários de trabalho da equipe (CHO, 2008), problema exacerbado quando membros das equipes estão em fusos horários diferentes (ALI; LAI, 2016).

2.3 Histórias de usuário

Histórias de usuário são uma forma de documentar o *product backlog*, e podem ser escritas em vários níveis de detalhe, conforme necessário. Utiliza-se cartões, físicos ou virtuais, para escrever as histórias de usuário. Um template possível é escrever, para um determinado papel exercido pelo usuário, qual o objetivo desejado e qual a razão para se atingir esse objetivo (COHN, 2004 apud RUBIN, 2012)), como é possível observar na Figura 2.2. A organização das informações descritas em um cartão podem variar; por exemplo Elallaoui, Nafil e Touahni (2015) sugere indicar diretamente os seguintes campos a serem preenchidos: Quem, O Que, Para Que.

Figura 2.2: Exemplo ilustrativo de cartão de histórias de usuário.

Fonte: Rubin (2012)

Após a definição inicial no cartão, conversas constantes com os *stakeholders* acontecem de forma a suplementar as informações do cartão, permitindo um melhor entendimento do requisito. A partir das conversas, os cartões podem ser complementados com informações novas.

Finalmente, existe um passo chamado de Confirmação, geralmente escrito na parte de trás do cartão, a confirmação contém informações para que a equipe tenha uma maior noção se o objetivo foi alcançado ou não, e auxilia no processo de testes do sistema (RUBIN, 2012).

As histórias de usuário podem apresentar diferentes tamanhos e níveis de precisão.

Histórias curtas e específicas são úteis no momento do desenvolvimento durante uma *sprint*, para um melhor entendimento do requisito pelo desenvolvedor. Enquanto isso, histórias de usuário mais abstratas e menos detalhadas, em menor quantidade, são importantes para planejamento de lançamentos, priorização de recursos para próximas *sprints* ou descrição do produto em um nível executivo (RUBIN, 2012).

2.4 Gerenciamento de mudanças

A gerência de mudança de requisitos (GMR) é importante, visto que mudanças nos requisitos são inevitáveis e afetam o planejamento de tempo, qualidade e custos de um projeto (SHAFIQ et al., 2018; JAYATILLEKE; LAI, 2013; JAYATILLEKE; LAI, 2018). Em um contexto geral de desenvolvimento de *software*, o processo de mudança de requisitos não é fácil, visto que quaisquer alterações precisam ser informadas e entendidas pelos membros da equipe (SHAFIQ et al., 2018).

No contexto do Scrum, mudanças não são aceitas durante uma *sprint*, depois que seu objetivo foi definido. Isso parece contradizer o conceito ágil de que mudanças são bem vindas no processo de desenvolvimento, sejam elas originadas da demanda de mercado, necessidade do consumidor, mudanças nos conhecimentos dos membros da equipe, visão do projeto e a solução em si (ALBUQUERQUE et al., 2020; JAYATILLEKE; LAI, 2018). Contudo, deve-se observar que a introdução de mudanças no meio de uma *sprint* possui consequências de tempo e dinheiro, visto que elas atrapalham o que já foi planejado para aquela *sprint* (RUBIN, 2012).

Como não há controle de quando essas mudanças nos requisitos surgirão e nem de quão importante elas são, torna-se necessário a existência do gerenciamento de mudanças dentro da equipe em todas as etapas de um processo de desenvolvimento, afim de minimizar os impactos dessas mudanças no planejamento do desenvolvimento do produto.

Ao considerar a etapa de desenvolvimento de código, por exemplo, o controle de

mudanças e histórico é importante, visto que erros acontecem e problemas podem surgir a partir destes erros. O controle de versionamento, neste caso, é essencial durante o desenvolvimento, de forma que seja possível revisar para buscar origens de erros e reverter para versões funcionais de um código. Uma das ferramentas mais populares entre desenvolvedores para controle de versionamento é o Git (BLISCHAK; DAVENPORT; WILSON, 2016), uma ferramenta *open source*, ou seja, de código aberto.

2.5 Trabalhos relacionados

Jayatilleke e Lai (2018) faz uma revisão sistemática sobre GMR na literatura, em um total de 184 trabalhos, envolvendo tanto métodos tradicionais quanto métodos ágeis. No contexto ágil foram identificados diversos processos utilizados para gerenciar mudanças de requisitos, tais como comunicação face a face, interação com o cliente, retrospectivas, histórias de usuário, entre outros. Destes, as histórias de usuário criadas para o *product backlog* são a única forma de documentação, o que pode trazer problemas em casos de lapsos de comunicação, uma vez que a identificação e análise de mudanças ocorrem em todas as etapas do processo ágil. O trabalho também identificou que os métodos ágeis funcionam melhor em equipes ou empresas menores, em que existe uma maior comunicação entre diferentes níveis organizacionais.

Albuquerque et al. (2020) faz um estudo de mapeamento com 21 artigos do processo de GMR ágil, e cita também Jayatilleke e Lai (2018) como uma de suas principais referências no trabalho. Nele, identifica-se que o processo de GMR ágil possui três passos: identificação da mudança, análise da mudança e estimativa de custo e esforço. Além disso, identificou-se quais as práticas realizadas em cada um desses passos. Nota-se que nenhuma dessas práticas envolve documentação, visto que metodologias ágeis, tais como o Scrum, propõem pouca documentação.

O trabalho Alsalemi e Yeoh (2015) realizou um questionário com 89 participantes e observou em seus resultados que as maiores razões para mudanças de requisitos

são correção de defeitos, requisitos faltantes e melhorias de funcionalidade. Essas mudanças de requisitos, em aproximadamente 56% das vezes, são registradas como um novo requisito e 36% como uma nova versão de um requisito existente. Além disso, quando ocorrem mudanças, em cerca de 63% dos casos, um novo *product backlog* é gerado e, em 28% dos casos, cria-se uma nova versão. Por fim, as respostas obtidas com o trabalho mostram que a rastreabilidade dos requisitos, apesar de gerar um maior gasto de tempo e dinheiro, traz benefícios especialmente no controle da complexidade do projeto e satisfação dos *stakeholders*.

Complementando Alsalemi e Yeoh (2015), Inayat et al. (2015) mostra que práticas ágeis de engenharia de requisitos resolvem alguns dos desafios existentes na gerência de requisitos. Por outro lado, estas práticas ágeis geram novos desafios que afetam o balanço entre agilidade e estabilidade. Alguns destes desafios são a falta de abordagens para lidar com requisitos não funcionais e a documentação mínima. O desafio da existência de documentação miníma é grande, visto que a comunicação da equipe e *stakeholders* torna-se limitada quando se considera espaços geográficos diferentes, ou um escritório que não acomode todos os membros (DANEVA et al., 2013 apud INAYAT et al., 2015).

Urbieta et al. (2020) propõe um método para consolidar o conhecimento espalhado entre diferentes histórias de usuário em um léxico, como uma forma de rastrear informações e transferir conhecimentos sobre os requisitos com maior facilidade, por meio de um glossário. Seus resultados mostraram que houve uma diminuição do tempo utilizado para entender um determinado requisito. Contudo, não foi observada mudança na capacidade ou precisão de entendimento do requisito.

Considerando os trabalhos mencionados, foi realizada uma análise de ferramentas de apoio à documentação e mudança de requisitos existentes no mercado com base nos critérios definidos a seguir. As ferramentas analisadas são:

- *VersionOne*[1], desenvolvida pela *Digital.ai*, é uma ferramenta *web* que possui versão gratuita para testes por 30 dias;
- *Azure DevOps*[2], desenvolvida pela *Microsoft*, é uma ferramenta web gratuita para equipes de até 5 membros com projetos ilimitados;
- *ScrumDesk*[3], desenvolvida pela empresa de mesmo nome, é uma ferramenta web gratuita para até 4 usuários ou para contextos acadêmicos;
- *Jira + Confluence*[4], desenvolvida pela Atlassian, é uma ferramenta web gratuita para até 10 usuários com limites na sua funcionalidade, e possui versões pagas com mais funcionalidades;
- *Jazz - IBM Engineering Workflow Management*[5], desenvolvida pela IBM, é uma ferramenta web com período de testes de dois meses e possui versões pagas para ambientes profissionais e corporativos.

As ferramentas foram testadas e avaliadas utilizando os seguintes critérios:

- Critério 1 - Configuração: A ferramenta é fácil de configurar para uso?
- Critério 2 - Facilidade de uso: Quão fácil foi utilizar os diferentes recursos disponibilizados pela ferramenta, tais como adicionar requisitos, visualizar o histórico, controlar as *sprints*, entre outros?
- Critério 3 - Integração com Git: É possível configurar a ferramenta para realizar integração com o Git?
- Critério 4 - Histórico de mudanças nas histórias de usuário: É possível rastrear facilmente o histórico de mudanças efetuadas?

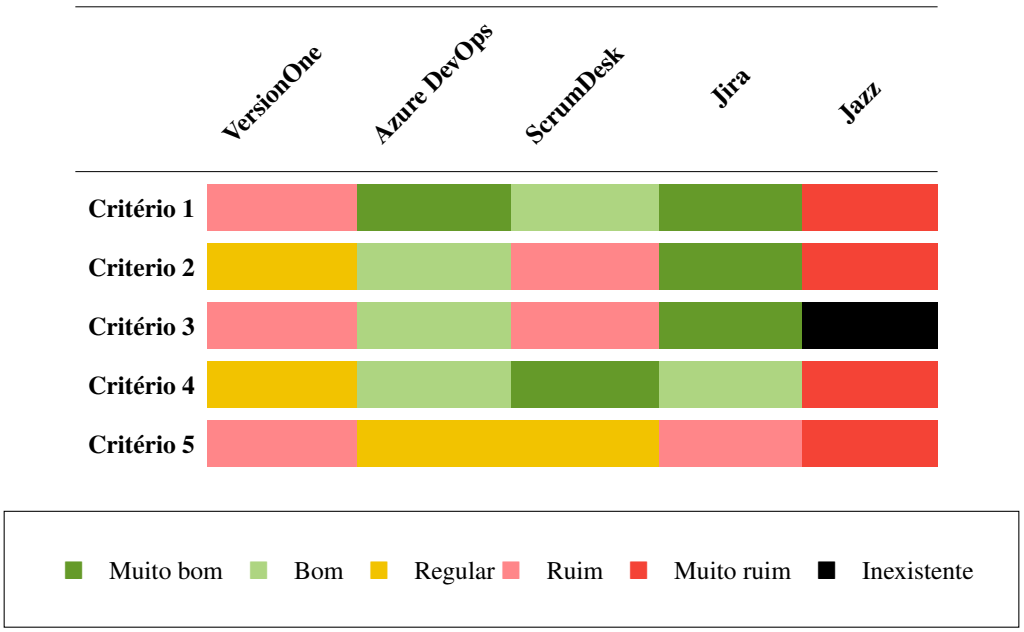
[1] https://www.collab.net/products/versionone
[2] https://azure.microsoft.com/en-us/services/devops/
[3] https://www.scrumdesk.com/
[4] https://www.atlassian.com/software/jira
[5] https://www.ibm.com/products/requirements-management-doors-next

- Critério 5 - Relação entre histórias de usuário e os diferentes tipos de mudanças: É possível identificar quais tipos de mudanças são mais comuns nas histórias de usuário?

Para cada critério, cada ferramenta recebeu uma avaliação de “Muito bom”, “bom”, “regular”, “ruim”, “muito ruim” e “inexistente”. Os resultados são apresentados na Tabela 2.1.

Dentre as ferramentas, o *Jira* foi a ferramenta com o melhor desempenho na avaliação devido à sua integração com o Git, sua facilidade de uso e configuração simples. Por outro lado, o *Jazz* obteve o pior resultado entre as ferramentas avaliadas visto que, baseado na avaliação realizada, não foi possível encontrar a funcionalidade de integração com o Git, é de difícil uso e entendimento, além de possuir uma configuração inicial complexa e lenta e ser a ferramenta mais cara entre as avaliadas. Assim, é possível evidenciar a importância de desenvolver novas propostas que visem sanar as carências observadas, especialmente relacionadas à gerência de mudanças requisitos, com o intuito de garantir sua rastreabilidade e a identificação dos motivos que justificam tal mudança, como realizado no presente trabalho.

Tabela 2.1: Avaliação das ferramentas de apoio à documentação de requisitos de projeto.



Capítulo 3

Características do protótipo

Neste capítulo estão descritas a arquitetura (Seção 3.1) e as funções (Seção 3.2) do protótipo desenvolvido a partir da identificação de carências em ferramentas atuais voltadas para documentação e gerência de mudanças em requisitos.

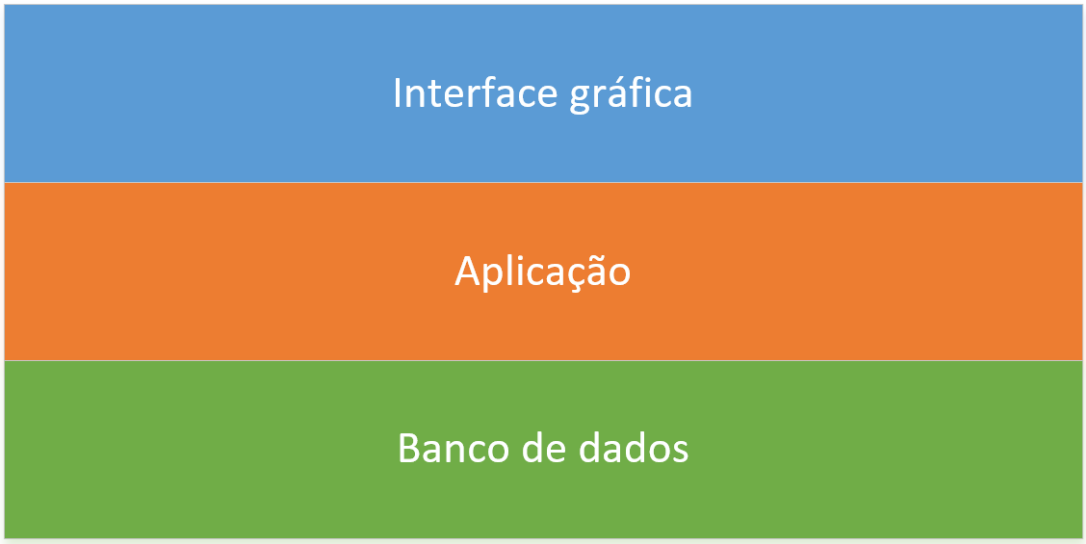
3.1 Arquitetura

A estruturação da arquitetura do protótipo foi feita de acordo com o Modelo em Camadas (GASSER; ALMEIDA, 2017), em que foram utilizadas três camadas: Interface de Usuário, Aplicação e Banco de Dados. Neste modelo de arquitetura, cada camada se comunica com as camadas adjacentes, de forma que as inferiores forneçam serviço às camadas superiores. Sua maior vantagem é a sua independência: modificações em uma camada não interferem profundamente nas outras, de forma a tornar o desenvolvimento e a manutenção mais fáceis. Na Figura 3.1 é apresentada a ilustração visual do modelo proposto.

A camada de Interface de Usuário permite a interação do usuário com o sistema, por meio de menus, botões, campos de entrada e seleção. A camada intermediária, que contém a Aplicação, é responsável pelas regras de negócio necessárias para auxiliar na documentação e acompanhamento das mudanças nos requisitos de um sistema. E,

por fim, a camada inferior de Banco de Dados é responsável por armazenar o conteúdo utilizado pelo sistema.

Figura 3.1: Visão geral simplificada da arquitetura do *History Teller*.



Fonte: Elaborado pelo autor

3.2 Funções do *History Teller*

Para a criação do protótipo, denominado *History Teller*, foram utilizados apenas *softwares* gratuitos, tais como o Visual Studio Community e Visual Studio Code para escrita do código, React como *framework* JavaScript para construção da interface, linguagem C# com o *Entity Framework* para escrita da aplicação e conexão com o banco de dados, *SQL Server* como banco de dados e o *Git* para controle de versão.

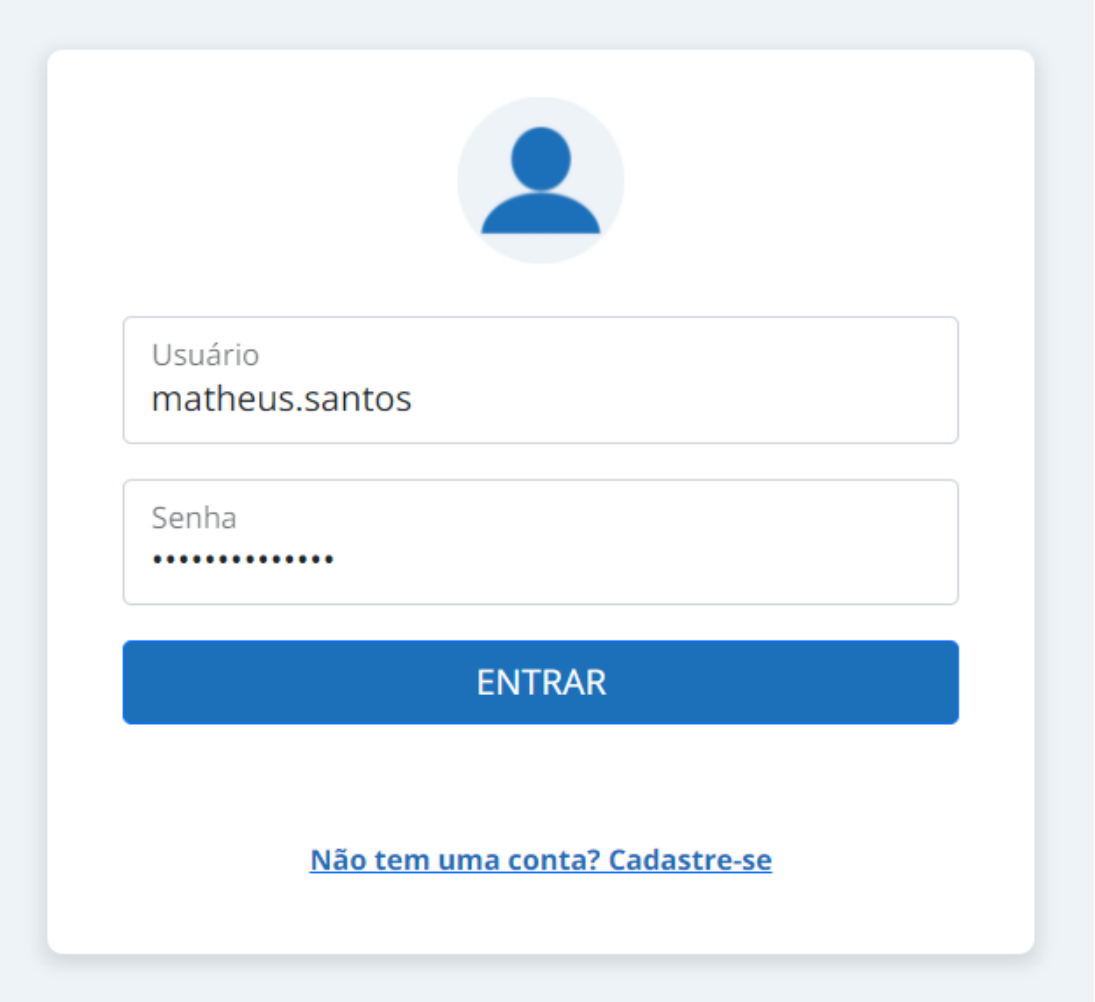
Nas subseções a seguir são descritas as funções contempladas pelo *History Teller*.

3.2.1 Acesso ao sistema

O acesso ao sistema é feito por meio de uma tela de *login*, exemplificada na Figura 3.2, sendo considerados dois tipos de usuários: colaboradores e *Product Owners*. Caso o *Product Owner* não possua uma conta, o seu registro pode ser feito pelo link “Não

tem uma conta? Cadastre-se".

Figura 3.2: Ilustração da tela de login.



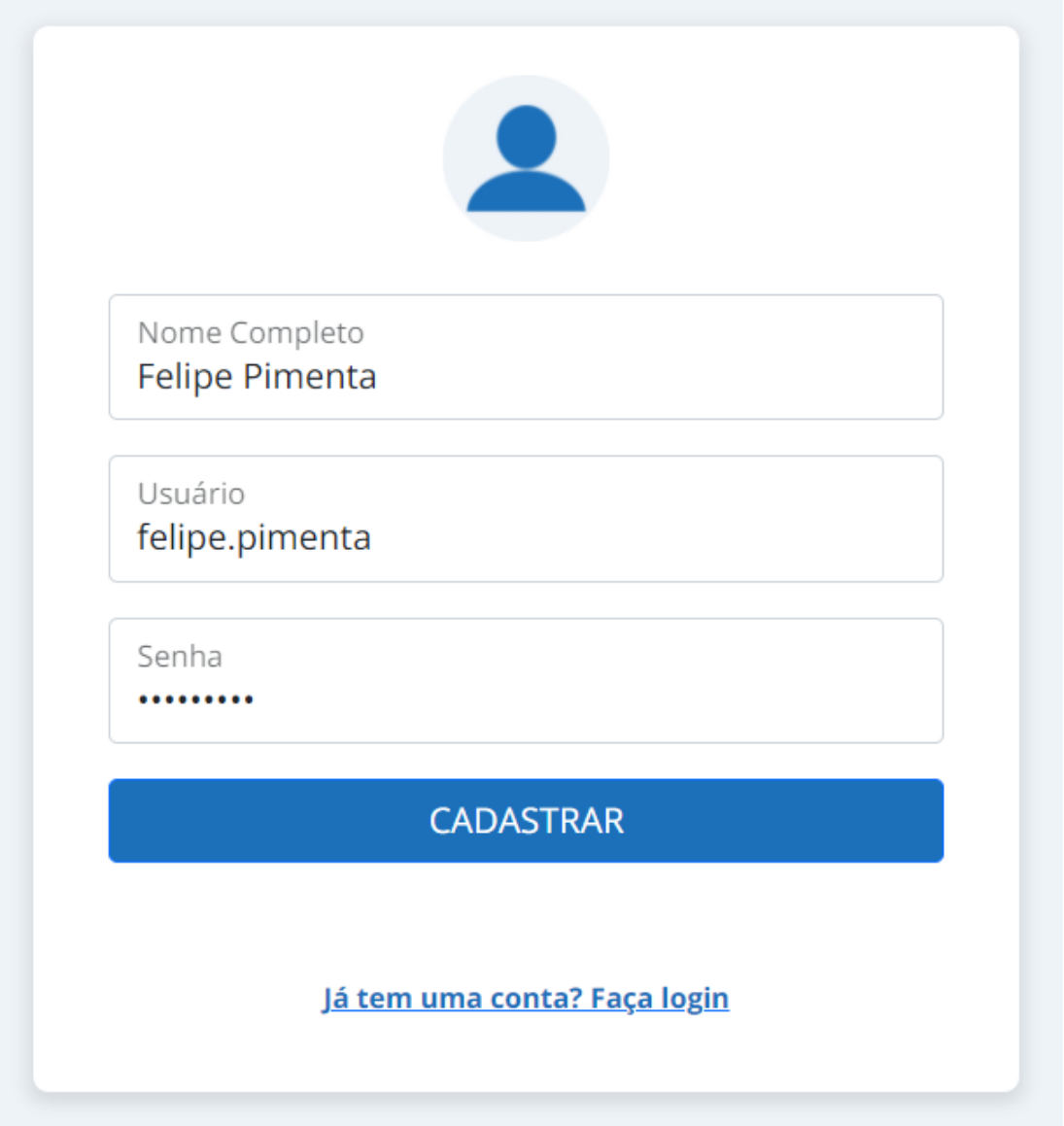
Fonte: Elaborado pelo autor

Na tela de cadastro, observada na Figura 3.3, é permitido que um Product Owner cadastre-se informando nome, usuário e senha.

3.2.2 Criação e edição de projetos e cadastro de colaboradores

Na tela inicial do projeto, apresentada na Figura 3.4, o usuário pode ver todos os projetos que criou, no caso do *Product Owner*, ou dos quais possui histórias de usuário associadas a ele, no caso do colaborador. Selecionando qualquer um dos projetos, o sistema mostra mais detalhes sobre ele no lado direito da tela, além de um botão para

Figura 3.3: Ilustração da tela para cadastro de conta.



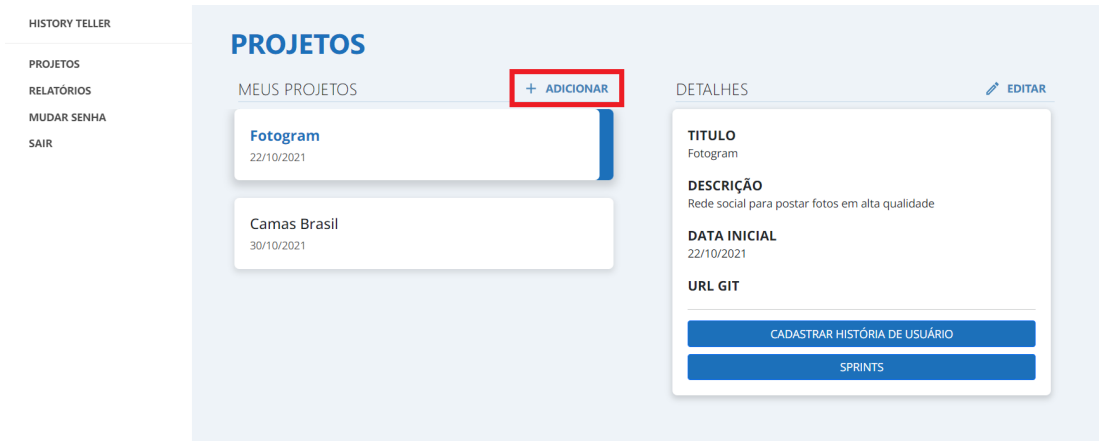
Fonte: Elaborado pelo autor

cadastrar novas histórias de usuário e um botão para abrir suas *sprints*.

Nesta interface, caso o usuário logado seja um *Product Owner*, por meio do botão "Adicionar", é possivel criar novos projetos no sistema. O usuário deve definir um título para o projeto, uma breve descrição, uma data de início e, opcionalmente, um link para um repositório do *Git* onde se encontra a implementação do projeto, como pode ser visto na Figura 3.5.

Já no lado direito, ainda na Figura 3.5, é possível que o *Product Owner*, a partir de

Figura 3.4: Ilustração da tela inicial do protótipo.



Fonte: Elaborado pelo autor

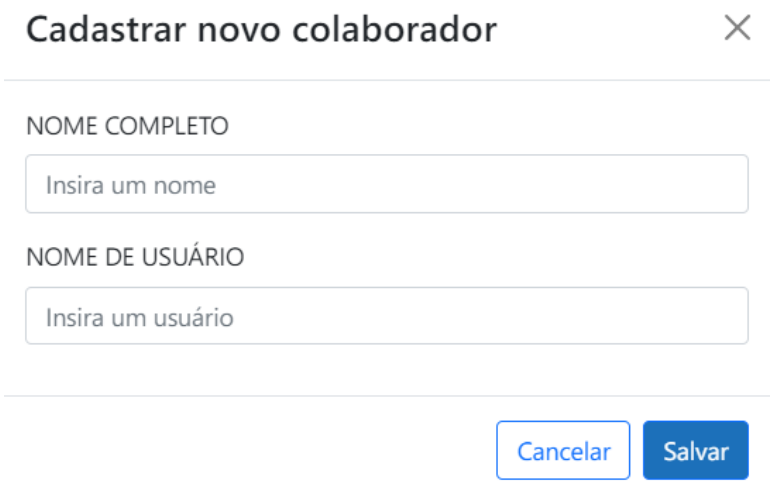
Figura 3.5: Ilustração da tela de cadastro de projetos.

Fonte: Elaborado pelo autor

uma lista suspensa, selecione os colaboradores que participarão do projeto e, caso deseje, cadastre novos colaboradores, pressionando o botão correspondente a esta opção. Durante o cadastro, deverá fornecer um nome e um usuário para o novo colaborador, conforme mostrado na Figura 3.6. O sistema gerará uma primeira senha aleatória para o colaborador, sendo responsabilidade do *Product Owner* informar tal colaborador os

dados para seu primeiro acesso. Tão logo tal acesso seja efetivado, será solicitado ao colaborador que troque sua senha.

Figura 3.6: Ilustração da tela de cadastro de novos colaboradores.



Fonte: Elaborado pelo autor

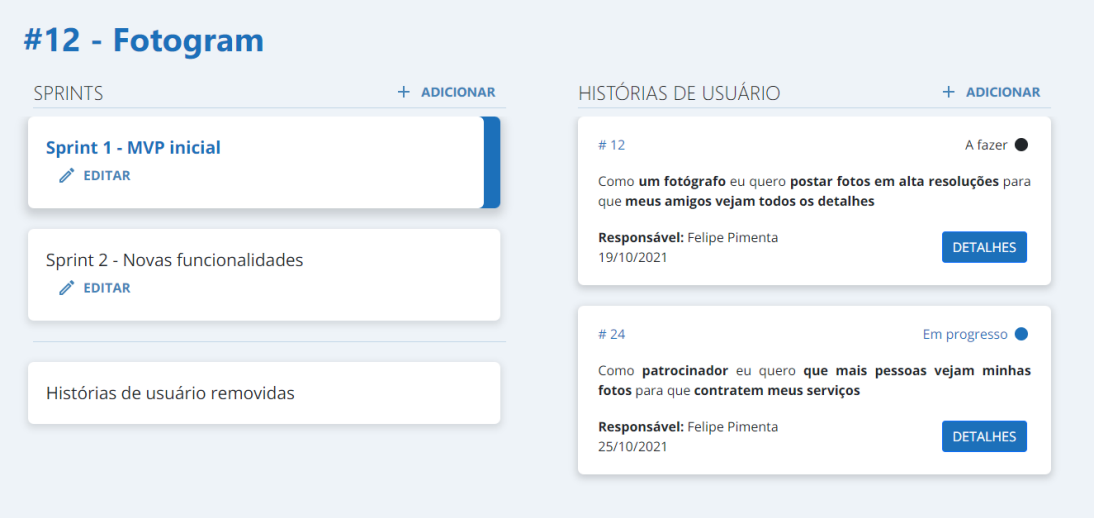
Cabe observar que, após o cadastro de um projeto, o usuário *Product Owner* também poderá editar todas as informações do mesmo, cadastrar novos colaboradores e desassociar antigos, caso deseje. Ao desassociar colaboradores de um projeto, o sistema verifica se ele possui alguma história de usuário naquele projeto com status "Em Progresso". Caso sim, impede a desassociação, uma vez que será preciso, neste caso, excluir a respectiva história de usuário previamente, mudar seu status para "Finalizado" ou, ainda, alterar o responsável por ela de forma a viabilizar a exclusão de tal colaborador.

3.2.3 Lista de *sprints* e histórias de usuário

Ao clicar no botão "SPRINTS" de um projeto, a partir da tela inicial (Figura 3.4), uma nova tela é exibida com a lista de *sprints* daquele projeto na coluna à esquerda. Ao selecionar uma *sprint*, o sistema exibirá as histórias de usuário associadas àquela *sprint* em cartões em uma coluna no lado direito da tela, como pode ser visto na Figura

3.7. Cada cartão possui como informação a sua descrição (seguindo o template Quem, O Que, Para Que), o seu identificador, seu status, a data de criação, o colaborador responsável e um botão para mais detalhes.

Figura 3.7: Ilustração da tela de lista de *sprints* e histórias de usuário de um determinado projeto.



Fonte: Elaborado pelo autor

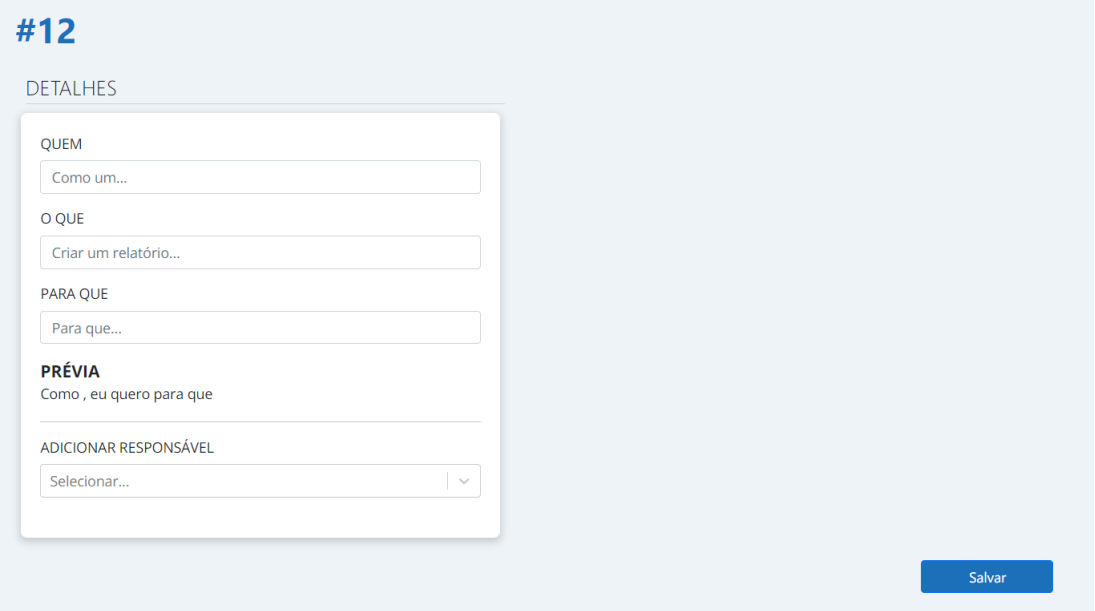
Caso o usuário logado seja um *Product Owner*, o usuário possuirá acesso a mais funcionalidades: na coluna da esquerda, das *sprints*, o botão "Adicionar" levará à tela de cadastro de novas *sprints* e, na coluna direita, ao cadastro de novas histórias de usuário.

3.2.4 Cadastro e edição de histórias de usuário

Ao adicionar uma nova história de usuário, conforme apresentado na Figura 3.8, o usuário deve preencher os campos que formam a descrição da história de usuário: Quem, O Que, e Para Que. Além disso, deve selecionar o colaborador que será responsável por aquela história de usuário.

Histórias de usuário previamente cadastradas podem ser modificadas. Para tanto, conforme apresentado na Figura 3.9, observa-se que o lado esquerdo da tela de edição é semelhante ao da tela de cadastro (Figura 3.8), com a adição da informação do

Figura 3.8: Ilustração da tela de cadastro de história de usuário.



Fonte: Elaborado pelo autor

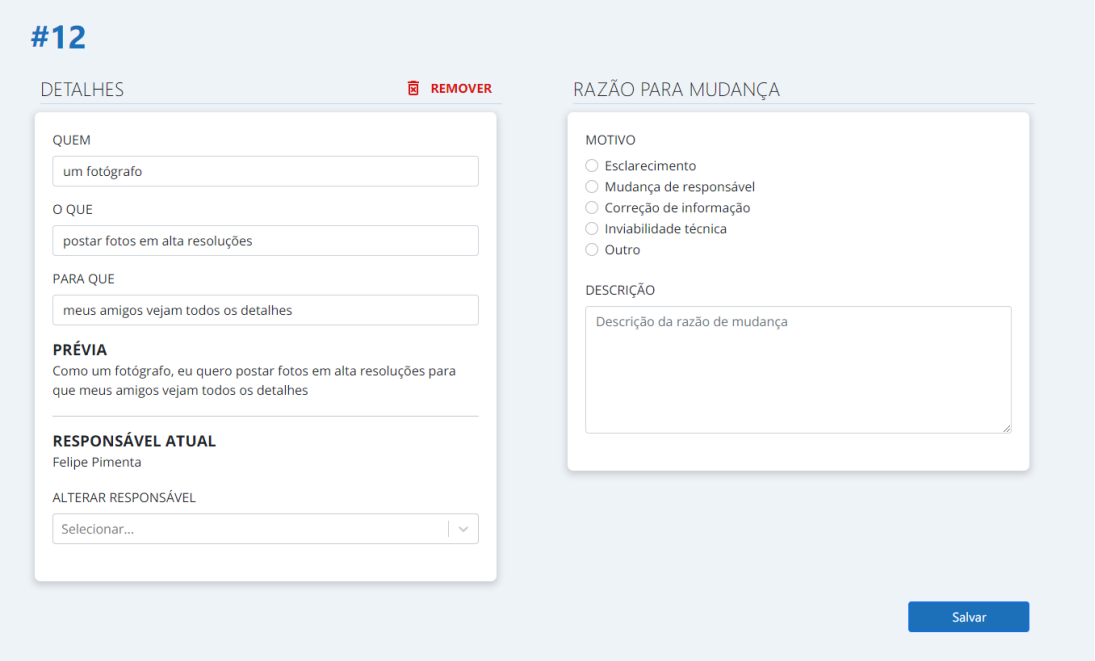
responsável atual da história de usuário. No lado direito, o usuário deve fornecer o motivo daquela edição na história de usuário (Esclarecimento, Mudança de responsável, Correção de informação, Inviabilidade técnica e outros), além de um campo de texto para explicar brevemente a mudança.

Além disso, caso deseje, o usuário também poderá remover a história de usuário. Quando a história de usuário é marcada como excluída, ela entra em uma lista de histórias de usuário removidas, para fins de documentação e histórico.

3.2.5 Visualização de mudanças nas histórias de usuário e mudança de status

A tela de detalhes de uma história de usuário, mostrada na Figura 3.10, é também dividida em duas colunas. No lado direito, são exibidas as mudanças que aquela história de usuário passou. O sistema exibe, ordenado do mais novo ao mais antigo, a data da mudança, o motivo selecionado e a descrição inserida por quem modificou a história de usuário. Desta forma, o colaborador pode visualizar o histórico das mudanças e

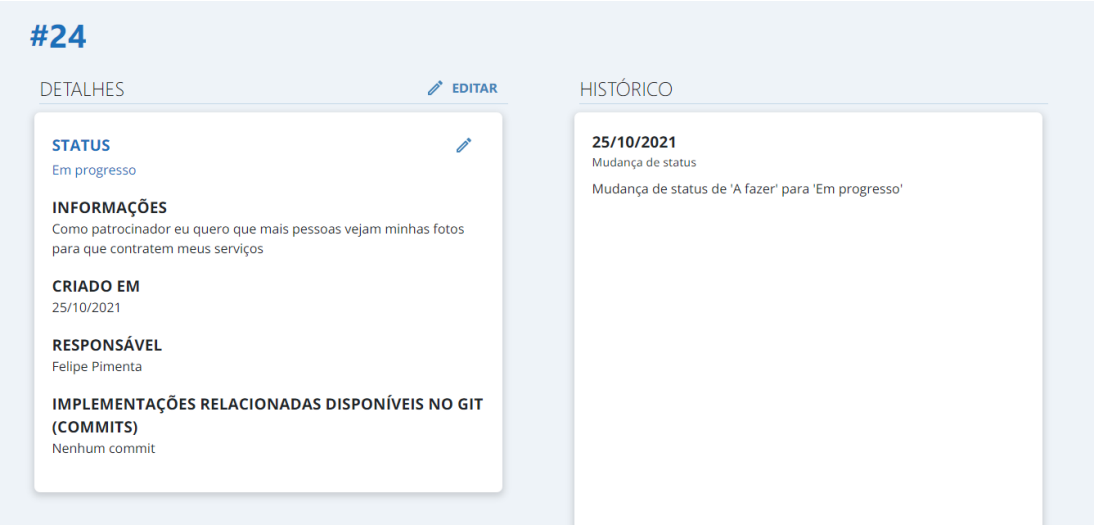
Figura 3.9: Ilustração da tela de edição de história de usuário.



Fonte: Elaborado pelo autor

usar essa informação durante a fase de desenvolvimento do projeto.

Figura 3.10: Ilustração da tela de detalhes de história de usuário, mudanças na definição e mudança de status.



Fonte: Elaborado pelo autor

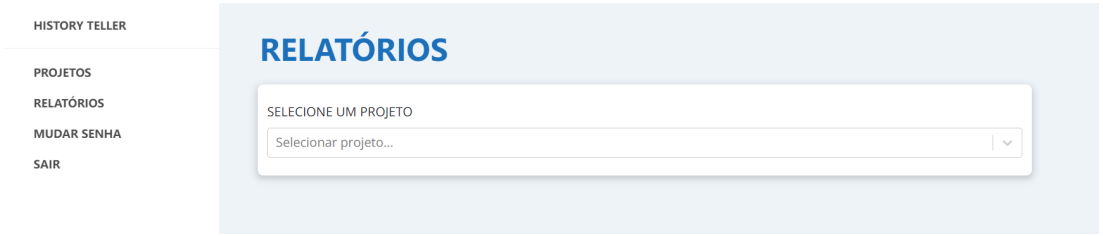
No lado esquerdo da tela, ainda na Figura 3.10, o usuário pode alterar o status da história de usuário entre “A fazer”, “Em Progresso” e “Finalizado”.

Além disso, observa-se que são listadas as implementações relacionadas a tal história de usuário que estão disponíveis no *Git* para o projeto correspondente. Isso é possível porque cada história de usuário pode ser associada, por meio do seu identificador, a um *commit*, uma vez que, a partir do link cadastrado no *Git*, o protótipo relaciona as histórias de usuário às suas implementações. Com isso, tais implementações podem ser acessadas por meio dos links dos *commits* listados, uma vez que o protótipo abre uma nova janela no Git correspondente à respectiva implementação.

3.2.6 Relatórios de projeto

Por meio da opção de menu “Relatórios”, disponibilizada na tela inicial do protótipo (Figura 3.4) apenas aos usuários do tipo *Product Owner*, é possível gerar relatórios relacionados à mudanças em histórias de usuário, selecionando o projeto desejado, conforme mostrado na Figura 3.11.

Figura 3.11: Ilustração da tela de escolha de projeto para geração de relatórios.

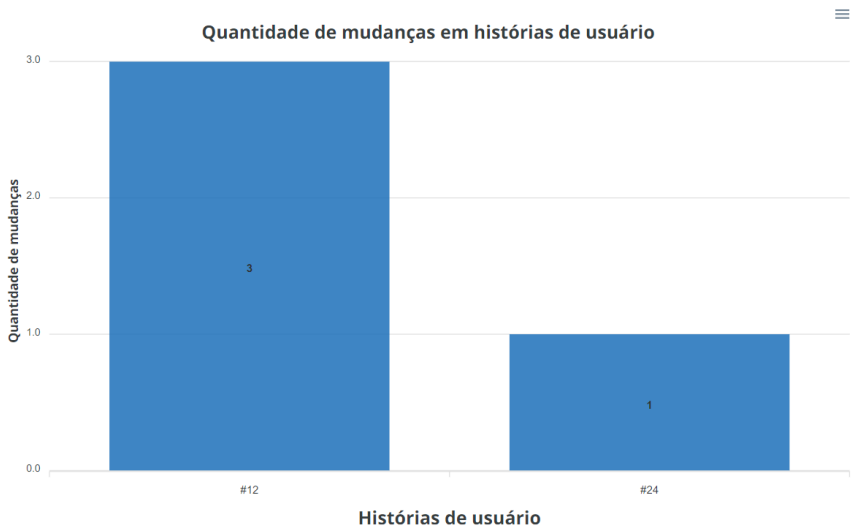


Fonte: Elaborado pelo autor

O primeiro relatório gerado, exemplificado na Figura 3.12, é um gráfico da relação história de usuário com a quantidade de mudanças que cada história de usuário sofreu. Por meio deste gráfico, o *Product Owner* pode analisar quais histórias de usuário passaram por uma quantidade alta de mudanças e, então, trabalhar com a equipe para minimizar os impactos causados por estas mudanças e evitar que ocorram em futuros projetos.

O segundo relatório gerado, ilustrado na Figura 3.13, é um gráfico que apresenta a quantidade de mudanças ocorridas por motivo de mudança. Desta forma, o *Product*

Figura 3.12: Exemplo de relatório gerado pelo *History Teller* de quantidade de mudanças em cada história de usuário em um dado projeto.

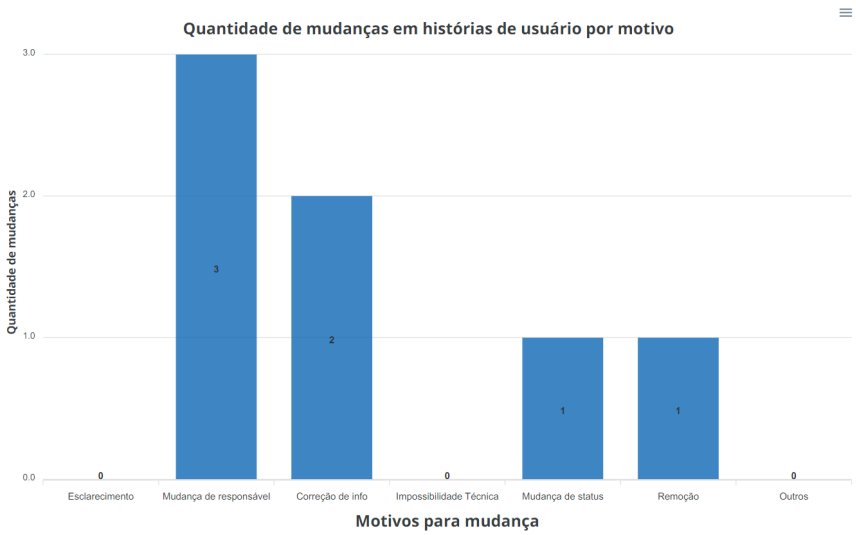


Fonte: Elaborado pelo autor

Owner pode identificar um padrão (caso exista) nas mudanças nas histórias de usuário do projeto e atacar a razão delas durante o processo de gerenciamento do projeto em questão, também com o intuito de minimizar impactos no prazo e qualidade do produto desenvolvido.

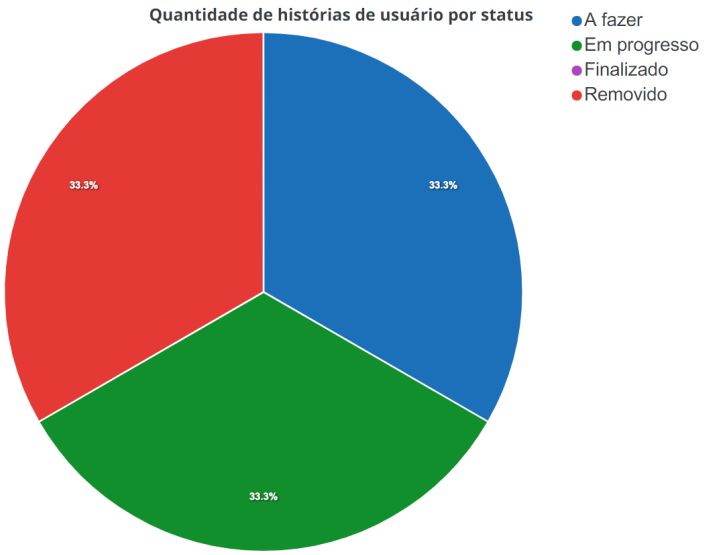
O terceiro relatório criado pelo sistema, ilustrado na Figura 3.14, é um gráfico de setores que classifica as histórias de usuário por status atual. O gráfico permite que o *Product Owner* tenha uma visão geral do andamento do projeto.

Figura 3.13: Exemplo de relatório gerado pelo *History Teller* de quantidade de mudanças ocorridas por motivo de mudança em um dado projeto.



Fonte: Elaborado pelo autor

Figura 3.14: Exemplo de relatório gerado pelo *History Teller* de status das histórias de usuário em um dado projeto.



Fonte: Elaborado pelo autor

Capítulo 4

Resultados e Discussão

Neste capítulo estão descritas a metodologia de avaliação e perfil dos participantes (Seção 4.1), além da análise dos resultados (Seção 4.2) obtidos a partir da avaliação realizada sobre o protótipo *History Teller*.

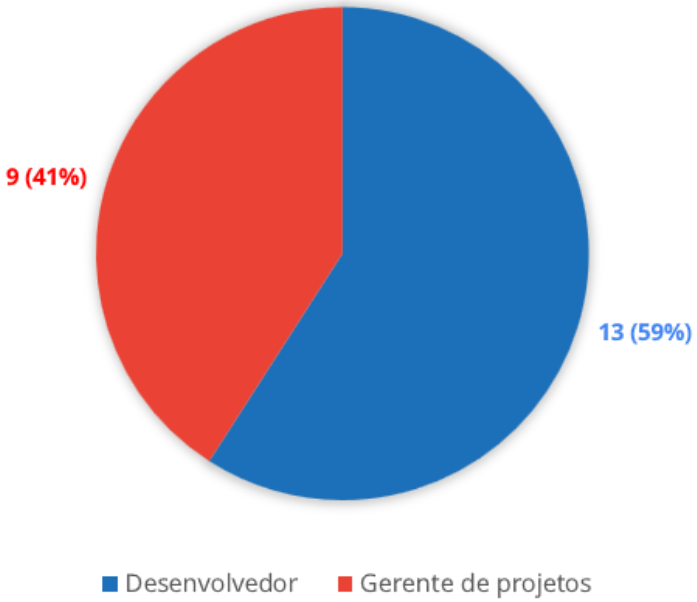
4.1 Metodologia de avaliação e perfil dos participantes

Os avaliadores, desevolvedores e gerentes de projeto, foram convidados pelo autor por meio de e-mail ou aplicativo de mensagem *Whatsapp*. O convite continha a apresentação do trabalho, seu objetivo, o link de acesso ao formulário eletrônico para a avaliação do protótipo e, como anexo, um documento instrucional (Apêndice A) para acesso ao protótipo *History Teller*. Tal documento descrevia as tarefas a serem seguidas para que todos os avaliadores pudessem qualificar a ferramenta baseado no mesmo conjunto de ações. Ao todo, 22 convidados aceitaram o convite.

Dentre os participantes, 13 são desenvolvedores e nove são gerentes de projeto, conforme pode ser visto na Figura 4.1. Além disso, os avaliadores possuem diferentes tempos de experiência no mercado de trabalho, ilustrado na Figura 4.2. Na Figura 4.3, verifica-se que aproximadamente 91% dos participantes já trabalhou em um ambiente de desenvolvimento ágil Scrum, evidenciando a difusão da metodologia no mercado

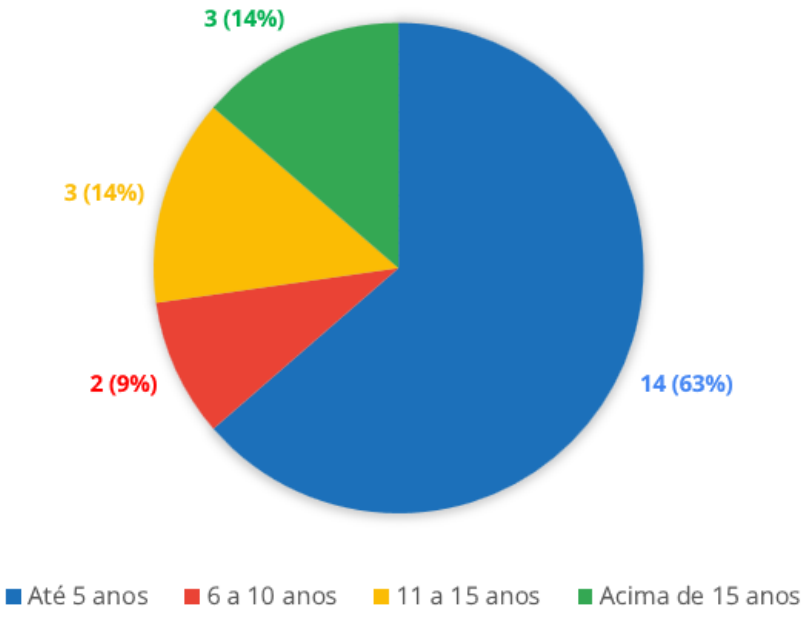
de trabalho atual.

Figura 4.1: Função exercida pelos avaliadores.



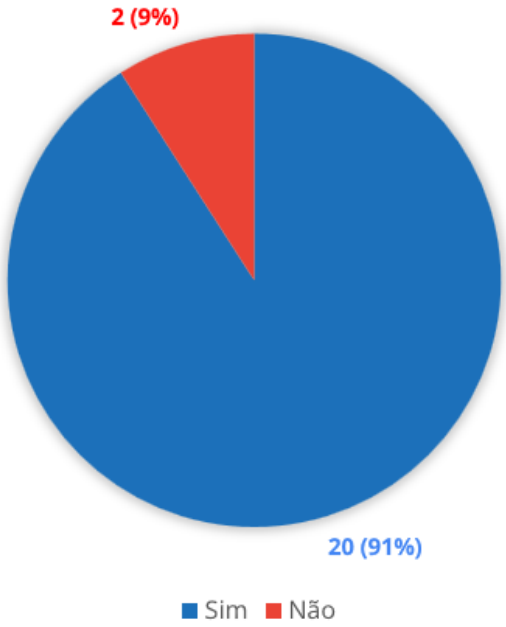
Fonte: Elaborado pelo autor

Figura 4.2: Experiência de mercado dos avaliadores.



Fonte: Elaborado pelo autor

Figura 4.3: Porcentagem dos avaliadores que já trabalhou em um ambiente Scrum.



Fonte: Elaborado pelo autor

A ferramenta foi disponibilizada de forma *online* por meio da plataforma gratuita *Heroku* para acesso dos usuários por 11 dias. Por meio de formulário eletrônico, contendo quatro afirmações e um campo aberto de preenchimento opcional para comentários gerais, buscou-se avaliar a usabilidade do protótipo, assim como a contribuição da abordagem proposta para o gerenciamento de mudanças de requisitos em um ambiente ágil Scrum e para a tomada de decisões relacionadas ao processo de desenvolvimento. Para avaliação das afirmações apresentadas no questionário foi usada uma escala Likert de cinco pontos: Discordo plenamente, Discordo, Neutro, Concordo e Concordo plenamente.

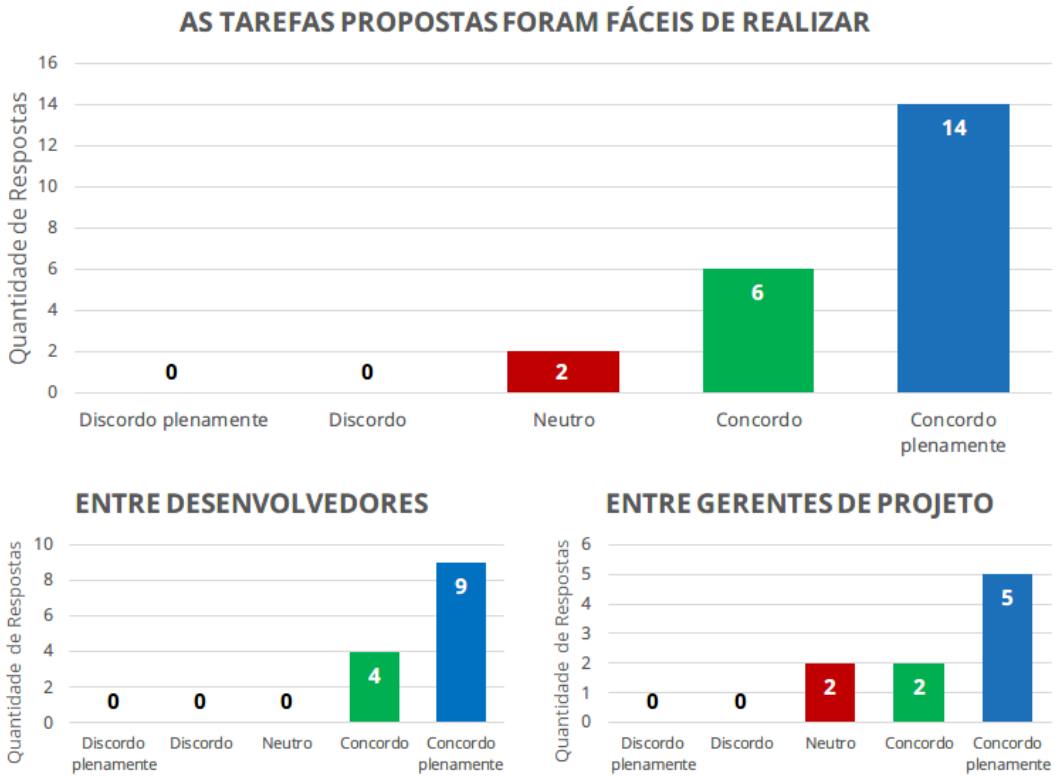
4.2 Análise dos resultados

Após seguirem as instruções e utilizar o *History Teller*, os participantes preencheram o formulário avaliativo.

Por meio da primeira afirmação, “As tarefas propostas foram fáceis de realizar”,

segundo respostas obtidas e apresentadas na Figura 4.4, é possível depreender que foi assegurada a usabilidade do *History Teller* visto que a ampla maioria dos avaliadores concordaram com a afirmação. Entre os avaliadores, apenas dois tiveram dificuldades e avaliaram como “Neutro” a afirmação.

Figura 4.4: Respostas dos avaliadores sobre usabilidade.

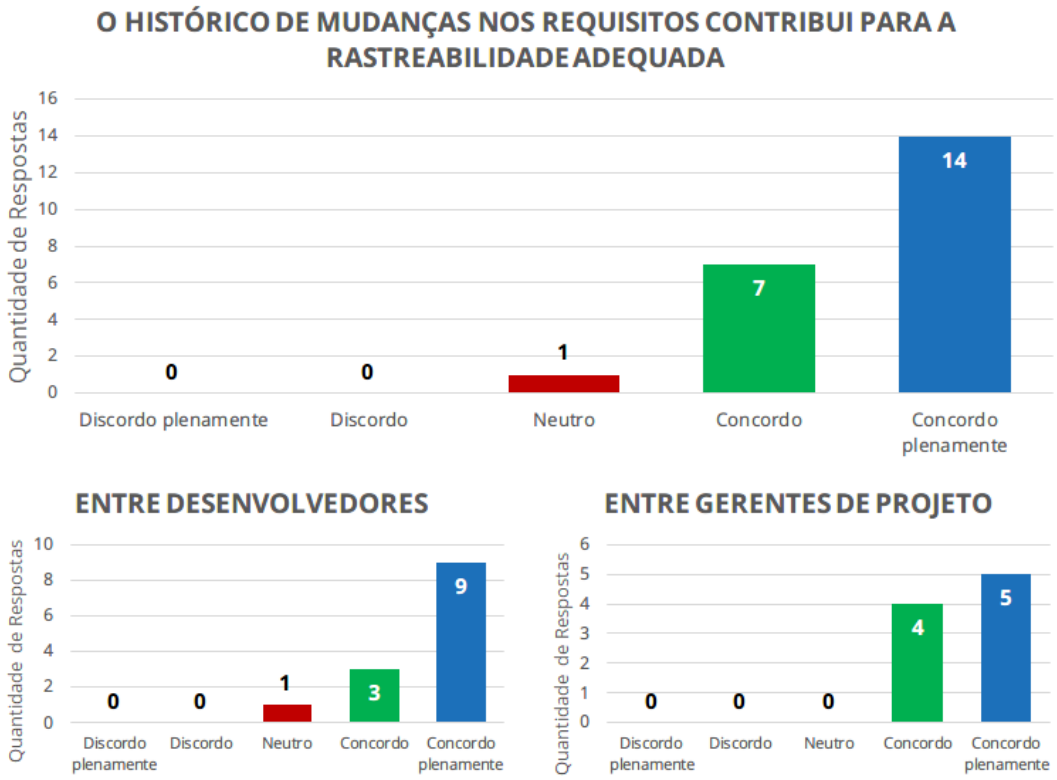


Fonte: Elaborado pelo autor

A segunda afirmação, "O histórico de mudanças nos requisitos contribui para a rastreabilidade adequada", segundo respostas obtidas e mostradas na Figura 4.5, ilustra que novamente a ampla maioria dos avaliadores concordaram com a afirmação, e o resultado não é diferente ao considerarmos os grupos separados de desenvolvedores e gerentes de projeto. Assim, é possível concluir que a abordagem proposta e implementada no protótipo pode auxiliar no rastreio das mudanças nos requisitos, confirmando a primeira hipótese (H1) formulada no presente trabalho (p. 2).

Para análise da terceira afirmação, "A ferramenta é útil para projetos em ambiente

Figura 4.5: Respostas dos avaliadores sobre a primeira hipótese.

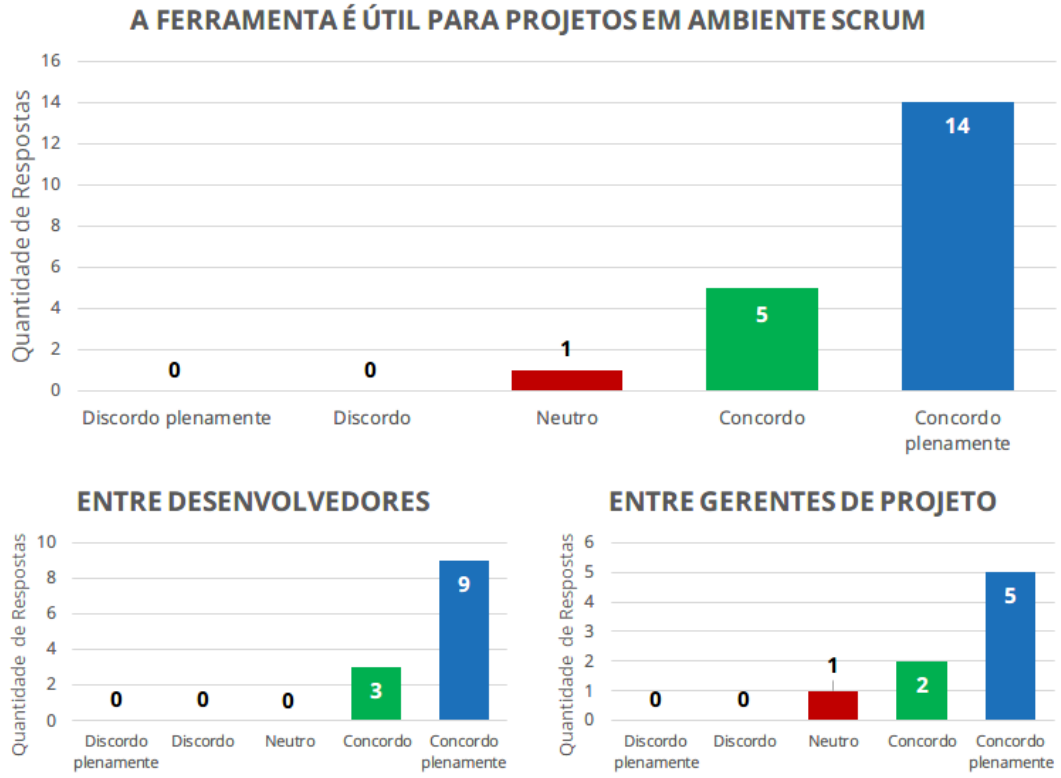


Fonte: Elaborado pelo autor

Scrum", observa-se que foram consideradas apenas 20 respostas, uma vez que foram desconsideradas aquelas apresentadas pelos participantes que nunca trabalharam em um ambiente Scrum (ver Figura 4.3). Os resultados, mostrados na Figura 4.6, expõem que a maior parte dos avaliadores concordaram plenamente que a ferramenta é útil para projetos em ambiente Scrum.

Como resultado da avaliação da quarta afirmação "Os relatórios gerados ajudam na tomada de decisões por fornecer uma visão geral sobre as mudanças ocorridas nos requisitos", envolvendo a segunda hipótese (H2) formulada no presente projeto (p. 2), conforme mostrado na Figura 4.7, verifica-se que os relatórios gerados pelo protótipo *History Teller* podem auxiliar na tomada de decisões pois a maioria dos avaliadores concordaram com a afirmação. Nota-se que, entre gerentes de projeto, um único avaliador discorda da afirmação sobre a utilidade dos relatórios. Dessa forma, nota-se que,

Figura 4.6: Respostas dos avaliadores sobre a contribuição do protótipo para o Scrum.



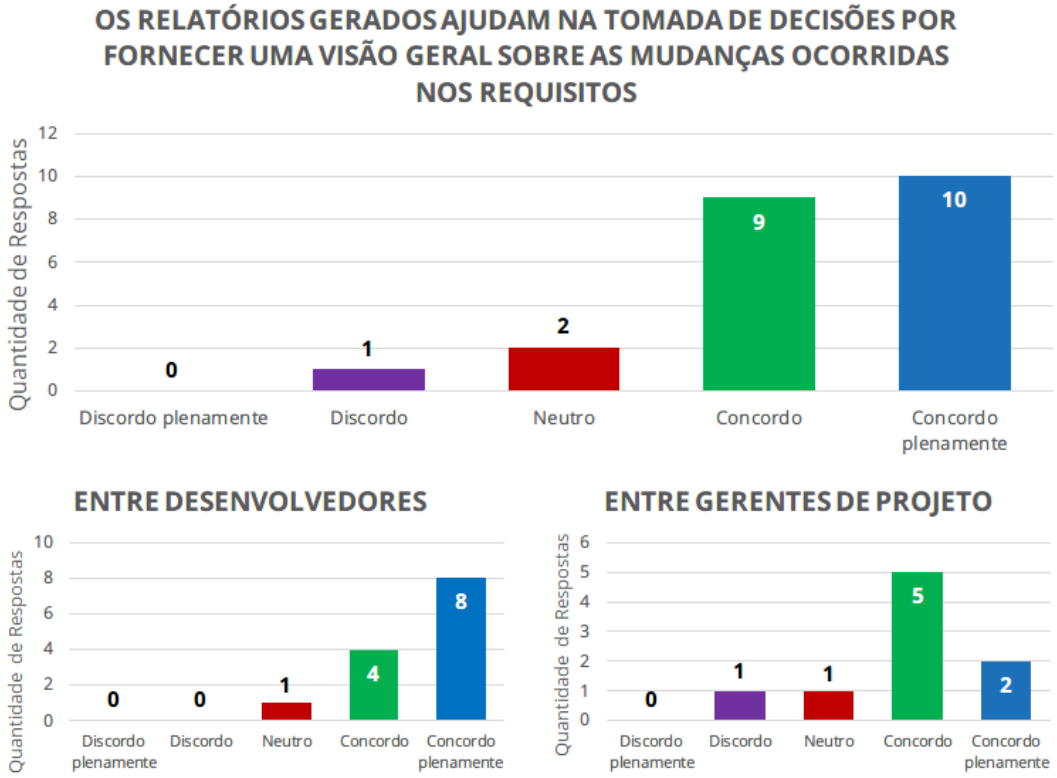
Fonte: Elaborado pelo autor

embora seja possível melhorar a funcionalidade de geração de Relatórios de forma que eles auxiliem com mais eficácia no acompanhamento das mudanças dos requisitos, a maioria dos avaliadores atestam a contribuição da solução desenvolvida.

Finalmente, tem-se o campo aberto disponibilizado para que os avaliadores fornecessem *feedback* sobre o protótipo *History Teller* para identificar pontos positivos e de carência observados que não puderam ser identificados por meio das afirmações avaliadas. Entre os *feedbacks* recebidos, múltiplos avaliadores responderam como ponto negativo a falta de um botão "Voltar" visto que, em alguns casos, é necessário reiniciar a navegação para acessar algumas funções disponibilizadas.

Outro *feedback* recebido foi a não possibilidade de remover histórias de usuário quando elas não estão associadas a nenhuma *sprint*, tornando necessário associá-la a uma *sprint* para então removê-la. Além disso, o formato definido para a definição da

Figura 4.7: Respostas dos avaliadores sobre a segunda hipótese.



Fonte: Elaborado pelo autor

história de usuário (“Quem, O Que, Para Que”), segundo opinião dos participantes da avaliação, é útil e válido no começo do projeto, mas pode se tornar difícil de usar em *sprints* no futuro, quando necessário descrever problemas detectados.

Além disso, a falta de familiaridade com o conceito de “Histórias de Usuário” de um dos avaliadores dificultou a utilização do protótipo, o que tornaria útil uma área para ajuda. Adicionalmente, o cadastro sequencial de itens do mesmo tipo (como, por exemplo, histórias de usuário) seria uma funcionalidade útil, com o intuito de facilitar o cadastro de dados.

Ainda, um avaliador sugeriu o desenvolvimento de um quadro Kanban, devido à gestão visual atrelada ao mesmo. Segundo Lei et al. (2017), Kanban é uma metodologia para gerenciamento de projetos de software cujo principal foco é definir com acurácia o que precisa ser trabalhado e quando ele precisa ser feito. Ainda sobre um

quadro Kanban, foi sugerido a possibilidade de arrastar tarefas entre as etapas/status do mesmo. Outro avaliador sugeriu a visualização comparando o que foi alterado entre versões, semelhante à funcionalidade *track changes* do sistema de controle de versionamento *Git*, em que o software destaca para o usuário quais exatamente foram as mudanças ocorridas em um determinado documento (VUORRE; CURLEY, 2018).

Finalmente, outros *feedbacks* foram positivos, em que os avaliadores elogiaram a funcionalidade de acompanhamento de mudanças implementada no *History Teller*.

Capítulo 5

Conclusão

Neste capítulo é realizada a análise das contribuições obtidas com o desenvolvimento do presente trabalho (Seção 5.1) e são apresentadas propostas de trabalhos futuros (Seção 5.2).

5.1 Contribuições

Mais do que nunca, projetos sofrem mudanças constantes e rápidas em seus requisitos, cada vez mais exacerbado por novas tecnologias e demandas que surgem à todo momento. Torna-se indispensável, então, uma forma simples e ágil de documentar as mudanças nos requisitos de um projeto. Neste contexto, o *History Teller* propõe uma abordagem para suprir esse requisito.

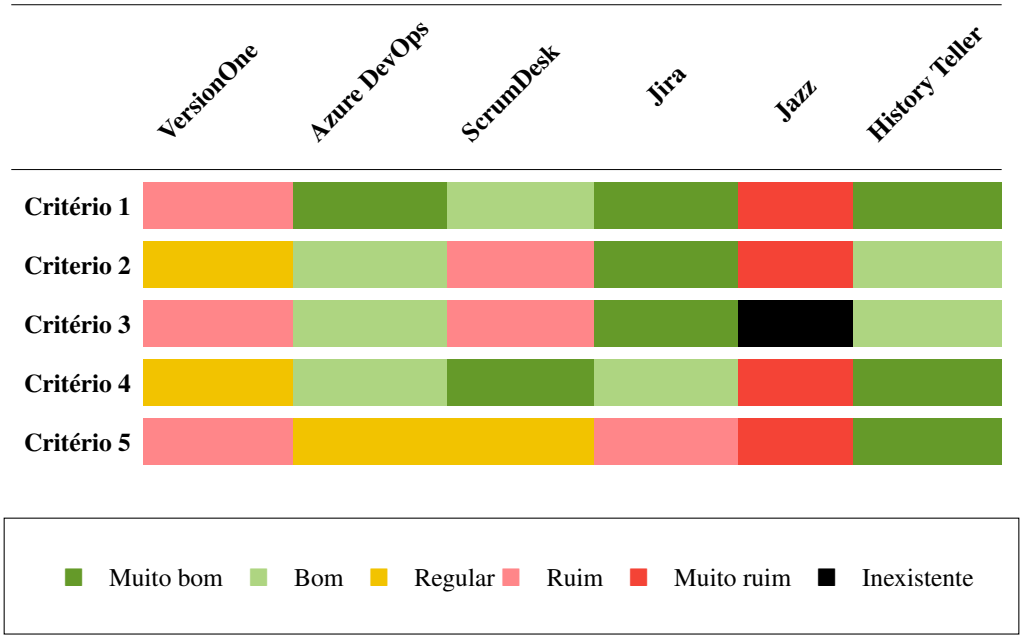
A partir de um processo avaliativo envolvendo profissionais experientes do mercado de trabalho que puderam manusear o *History Teller* por um período de onze dias, foi possível identificar as principais contribuições obtidas com o desenvolvimento deste trabalho. São elas:

- Melhora da rastreabilidade dos requisitos de um projeto desenvolvido em um ambiente Scrum, por meio da manutenção do histórico de mudanças;

- Apoio à tomada de decisões durante o processo de desenvolvimento de projetos Scrum, por meio da geração de relatórios contemplando informações sobre as mudanças ocorridas.

Na Capítulo 2 foram avaliadas algumas ferramentas de apoio à documentação de mudança nos requisitos existentes no mercado para identificar suas carências. Portanto, para evidenciar as contribuições obtidas com a conclusão deste trabalho é possível comparar o *History Teller* com estas ferramentas já existentes. O resultado de tal comparação é exibido na Tabela 5.1.

Tabela 5.1: Comparativo do History Teller com ferramentas existentes



Importante notar que o *History Teller* atende os critérios de qualidade estabelecidos no capítulo 2 de forma satisfatória. Para atingir "Muito bom" no critério 2, são necessárias melhorias na usabilidade do sistema, evidenciado pelos *feedbacks* fornecidos pelos avaliadores, como discutido no capítulo 4. Em relação ao critério 3, precisa-se que o sistema interaja com mais plataformas online *Git*.

5.2 Trabalhos Futuros

A partir dos *feedbacks* fornecidos pelos avaliadores do protótipo, por meio da identificação de pontos de melhoria almejados para o *History Teller*, é possível estabelecer algumas sugestões e objetivos para trabalhos futuros.

Em relação a funções existentes, possíveis aperfeiçoamentos futuros são: melhorias na experiência do usuário, como botão para voltar à tela anterior; possibilidade de exclusão de histórias de usuário não relacionadas a nenhuma *sprint*; adição de novos relatórios; e armazenar versões antigas da história de usuário ao ocorrer mudanças.

Se tratando da adição de novas funcionalidades, sugere-se: novo template para histórias de usuário, a fim de suprir eventuais situações em que o template proposto (O Que, Quem, Para Que) não se adequa; adição de um quadro *Kanban*; possibilidade de cadastro sequencial de dados; e adição de mecanismos de ajuda para explicar conceitos que possam não ser conhecidos pelo usuário.

Além disso, futuros trabalhos podem avaliar a efetividade da abordagem proposta neste trabalho em outras metodologias ágeis como *XP* ou *FDD*. Finalmente, é sugerido a continuação da pesquisa proposta por meio da análise de novas abordagens para efeitos comparativos.

Referências Bibliográficas

AGILE, M. *Manifesto for Agile Software Development*. 2001. Disponível em: <https://agilemanifesto.org/>.

AKBAR, M. A.; SANG, J.; KHAN, A. A.; SHAFIQ, M.; HUSSAIN, S.; HU, H.; ELAHI, M.; XIANG, H.; NASRULLAH; FAZAL-E-AMIN. Improving the quality of software development process by introducing a new methodology–az-model. *IEEE Access*, IEEE, v. 6, p. 4811–4823, 2017.

AKBAR, M. A.; SANG, J.; KHAN, A. A.; MAHMOOD, S.; QADRI, S. F.; HU, H.; XIANG, H.; NASRULLAH. Success factors influencing requirements change management process in global software development. *Journal of Computer Languages*, Elsevier, v. 51, p. 112–130, 2019.

AKBAR, M. A.; SHAFIQ, M.; AHMAD, J.; MATEEN, M.; RIAZ, M. T.; NASRULLAH. Az-model of software requirements change management in global software development. In: IEEE. *2018 International Conference on Computing, Electronic and Electrical Engineering (ICE Cube)*. Quetta, Pakistan, 2018. p. 1–6.

AL-ZAIDI, A.; QURESHI, R. Global software development geographical distance communication challenges. *The International Arab Journal of Information Technology (IAJIT)*, v. 14, n. 2, p. 215–222, 2017.

ALBUQUERQUE, D.; GUIMARAES, E.; PERKUSICH, M.; COSTA, A.; DANTAS, E.; RAMOS, F.; ALMEIDA, H. Defining agile requirements change management: a mapping study. In: ACM. *Proceedings of the 35th Annual ACM Symposium on Applied Computing*. Brno, Czech Republic, 2020. p. 1421–1424.

ALI, N.; LAI, R. A method of requirements change management for global software development. *Information and Software Technology*, Elsevier, v. 70, p. 49–67, 2016.

ALSALEMI, A. M.; YEOH, E.-T. A survey on product backlog change management and requirement traceability in agile (scrum). In: IEEE. *2015 9th Malaysian Software Engineering Conference (MySEC)*. Kuala Lumpur, Malaysia, 2015. p. 189–194.

AMBLER, S. W. *Agile Requirements Change Management*. 2005. Disponível em: <http://agilemodeling.com/essays/changeManagement.htm>.

ANANJEVA, A.; PERSSON, J. S.; BRUUN, A. Integrating ux work with agile development through user stories: An action research study in a small software company. *Journal of Systems and Software*, Elsevier, v. 170, p. 110785, 2020.

ANGUELOV, K. Research for usefulness of agile methods in creative business. In: IEEE. *2019 International Conference on Creative Business for Smart and Sustainable Growth (CREBUS)*. Sandanski, Bulgaria, 2019. p. 1–5.

AWAD, M. A comparison between agile and traditional software development methodologies. *University of Western Australia*, v. 30, 2005.

BHATTI, M. W.; HAYAT, F.; EHSAN, N.; ISHAQUE, A.; AHMED, S.; MIRZA, E. A methodology to manage the changing requirements of a software project. p. 319–322, 2010.

BLISCHAK, J. D.; DAVENPORT, E. R.; WILSON, G. A quick introduction to version control with git and github. *PLoS computational biology*, Public Library of Science San Francisco, CA USA, v. 12, n. 1, p. e1004668, 2016.

CERVONE, H. F. Understanding agile project management methods using scrum. *OCLC Systems & Services: International digital library perspectives*, Emerald Group Publishing Limited, 2011.

CHO, J. Issues and challenges of agile software development with scrum. *Issues in Information Systems*, v. 9, n. 2, p. 188–195, 2008.

CHOPADE, M. R. M.; DHAVASE, N. S. Agile software development: Positive and negative user stories. In: IEEE. *2017 2nd International Conference for Convergence in Technology (I2CT)*. Mumbai, India, 2017. p. 297–299.

COHN, M. *User stories applied: For agile software development*. Lafayette, Colorado: Addison-Wesley Professional, 2004.

DINGSØYR, T.; NERUR, S.; BALIJEPALLY, V.; MOE, N. B. *A decade of agile methodologies: Towards explaining agile software development*. Amsterdam, Netherlands: Elsevier, 2012.

ELALLAOUI, M.; NAFIL, K.; TOUAHNI, R. Automatic generation of uml sequence diagrams from user stories in scrum process. In: IEEE. *2015 10th international conference on intelligent systems: theories and applications (SITA)*. Estados Unidos, 2015. p. 1–6.

GASSER, U.; ALMEIDA, V. A. A layered model for ai governance. *IEEE Internet Computing*, IEEE, v. 21, n. 6, p. 58–62, 2017.

INAYAT, I.; SALIM, S. S.; MARCZAK, S.; DANEVA, M.; SHAMSHIRBAND, S. A systematic literature review on agile requirements engineering practices and challenges. *Computers in human behavior*, Elsevier, v. 51, p. 915–929, 2015.

JAYATILLEKE, S.; LAI, R. A method of specifying and classifying requirements change. In: *Proceedings of the 2013 22nd Australian Conference on Software Engineering*. Hawthorne, Austrália: ACM, 2013. p. 175–180.

JAYATILLEKE, S.; LAI, R. A systematic review of requirements change management. *Information and Software Technology*, Elsevier, v. 93, p. 163–185, 2018.

KUMAR, G.; BHATIA, P. K. Impact of agile methodology on software development process. *International Journal of Computer Technology and Electronics Engineering (IJCTEE)*, v. 2, n. 4, p. 46–50, 2012.

LEI, H.; GANJEIZADEH, F.; JAYACHANDRAN, P. K.; OZCAN, P. A statistical analysis of the effects of scrum and kanban on software development projects. *Robotics and Computer-Integrated Manufacturing*, Elsevier, v. 43, p. 59–67, 2017.

NIAZI, M.; MAHMOOD, S.; ALSHAYEB, M.; RIAZ, M. R.; FAISAL, K.; CERPA, N. Challenges of project management in global software development: initial results. In: IEEE. *2013 Science and Information Conference*. London, UK, 2013. p. 202–206.

NURMULIANI, N.; ZOWGHI, D.; POWELL, S. Analysis of requirements volatility during software development life cycle. In: IEEE. *2004 Australian Software Engineering Conference. Proceedings.* Melbourne, VIC, Australia, 2004. p. 28–37.

PEREIRA, J. C.; RUSSO, R. de F. Design thinking integrated in agile software development: A systematic literature review. *Procedia computer science*, Elsevier, v. 138, p. 775–782, 2018.

RUBIN, K. S. *Essential Scrum: A practical guide to the most popular Agile process*. Boston, United States: Addison-Wesley, 2012.

SAHER, N.; BAHAROM, F.; GHAZALI, O. Requirement change taxonomy and categorization in agile software development. In: IEEE. *2017 6th International Conference on Electrical Engineering and Informatics (ICEEI)*. Langkawi, Malaysia, 2017. p. 1–6.

SCHWABER, K. Scrum development process. In: *Business object design and implementation*. [S.l.]: Springer, 1997. p. 117–134.

SHAFIQ, M.; ZHANG, Q.; AKBAR, M. A.; KHAN, A. A.; HUSSAIN, S.; AMIN, F.-E.; KHAN, A.; SOOFI, A. A. Effect of project management in requirements engineering and requirements change management processes for global software development. *IEEE Access*, IEEE, v. 6, p. 25747–25763, 2018.

SHARMA, S.; SARKAR, D.; GUPTA, D. Agile processes and methodologies: A conceptual study. *International journal on computer science and Engineering*, Engg Journals Publications, v. 4, n. 5, p. 892, 2012.

SOMMERVILLE, I. *Software Engineering*. St. Andrews, UK: Addison-Wesley, 2007.

SRIVASTAVA, A.; BHARDWAJ, S.; SARASWAT, S. Scrum model for agile methodology. In: IEEE. *2017 International Conference on Computing, Communication and Automation (ICCCA)*. Greater Noida, India, 2017. p. 864–869.

STAVRU, S. A critical examination of recent industrial surveys on agile method usage. *Journal of Systems and Software*, Elsevier, v. 94, p. 87–97, 2014.

SUTHERLAND, J.; VIKTOROV, A.; BLOUNT, J.; PUNTIKOV, N. Distributed scrum: Agile project management with outsourced development teams. In: IEEE. *2007 40th annual Hawaii international conference on system sciences (HICSS'07)*. [S.l.], 2007. p. 274a–274a.

URBIETA, M.; ANTONELLI, L.; ROSSI, G.; LEITE, J. C. S. do P. The impact of using a domain language for an agile requirements management. *Information and Software Technology*, Elsevier, v. 127, p. 106375, 2020.

VUORRE, M.; CURLEY, J. P. Curating research assets: A tutorial on the git version control system. *Advances in Methods and Practices in Psychological Science*, Sage Publications Sage CA: Los Angeles, CA, v. 1, n. 2, p. 219–236, 2018.

Apêndice A

Documento instrucional para avaliação do *History Teller*

A.1 Introdução

O Scrum, uma metodologia ágil para gerenciamento de sistemas, não possui uma abordagem para gerenciar mudanças nos requisitos do projeto. Assim, foi proposto uma abordagem para gerenciar essas mudanças, além de um protótipo para avaliar a contribuição dos mecanismos propostos.

A.2 Instruções

Acesso ao sistema: https://tcc-frontend-felipe.herokuapp.com/

Credenciais: Como gerente de projeto, primeiramente, cadastre-se na tela de login, criando seu usuário e senha. Na sequência, faça o login com as credenciais criadas para iniciar a avaliação do protótipo.

A.2.1 Criação de projeto e cadastro de novos colaboradores

Ao entrar no sistema, crie um projeto. Para isso, clique no botão “Adicionar”. Na nova tela, preencha as informações sobre o projeto como desejar. Na sequência, cadastre dois novos colaboradores pressionando o botão “Cadastrar novos colaboradores”. Anote o usuário e senha gerados. Finalmente, selecione tais nomes na caixa de seleção apresentada para adicionar novos colaboradores ao projeto e pressiono o botão “Salvar”.

Obs: Por meio da criação dos novos colaboradores associados ao projeto é possível acessar com tais credenciais e, com isso, verificar que eles apenas terão acesso para visualizar as informações gerais do projeto cadastrado pelo gerente a que eles foram associados, bem como mudar o status de uma história de usuário. Logo, verifica-se, com isso, que apenas gerentes de projetos podem cadastrar novos projetos e associar colaboradores, assim como emitir relatórios.

A.2.2 Cadastro de histórias de usuário

Cadastre duas novas histórias de usuário. Para isso, na tela “Projetos”, selecione o seu projeto e, no lado direito, aperte o botão “Cadastrar história de usuário”. Na nova tela, defina a história de usuário como quiser. Lembre-se que, no Scrum, as histórias de usuário são curtas e seguem o formato Quem, O Que, Para Que. Finalmente, pressione o botão “Salvar”.

A.2.3 Cadastro de sprints e associação de histórias de usuário

Cadastre uma nova sprint no seu projeto. Para isso, na tela de projetos, selecione o seu projeto e, então, clique no botão “Sprints” na coluna à direita e, em seguida, clique no botão “Adicionar”.

Então, defina um nome para a sprint na esquerda e selecione as histórias de usuário para associar na direita. Finalmente, pressione o botão “Salvar”.

A.2.4 Detalhes de história de usuário e mudança no requisito

Novamente selecione o seu projeto, pressione o botão “Sprints” para selecionar a sprint criada por você e, na sequência, escolha uma história de usuário e visualize seus detalhes. Em seguida, aperte o botão “Editar”. Mude os detalhes como quiser. No lado direito, escolha uma razão para a mudança, e escreva uma breve descrição da mudança, e pressione o botão “Salvar”. Analise a interface com o histórico de mudanças.

A.2.5 Relatórios

No menu lateral esquerdo, selecione a opção “Relatórios”. Na nova tela, selecione um projeto cadastrado no seletor e veja os relatórios gerados.