Universidade Federal de Campina Grande

Centro de Engenharia Elétrica e Informática

Coordenação de Pós-Graduação em Ciência da Computação

Ednaldo Dilorenzo de Souza Filho

Uma Abordagem para Recomendação de Casos de Teste em Projetos Ágeis Baseados no Scrum

Campina Grande - PB

2021

Universidade Federal de Campina Grande
Centro de Engenharia Elétrica e Informática
Coordenação de Pós-Graduação em Ciência da Computação

Uma Abordagem para Recomendação de Casos de Teste em Projetos Ágeis Baseados em Scrum

Ednaldo Dilorenzo de Souza Filho

Tese submetida à Coordenação do Curso de Pós-Graduação em Ciência da Computação da Universidade Federal de Campina Grande - Campus I como parte dos requisitos necessários para obtenção do grau de Doutor em Ciência da Computação.

Área de Concentração: Ciência da Computação
Linha de Pesquisa: Engenharia de Software

Hyggo Almeida
(Orientador)

Campina Grande, Paraíba, Brasil

S729a Souza Filho, Ednaldo Dilorenzo de.
Uma abordagem para recomendação de casos de teste em projetos ágeis baseados em scrum / Ednaldo Dilorenzo de Souza Filho. – Campina Grande, 2021.
138 f. : il. color.

Tese (Doutorado em Ciência da Computação) – Universidade Federal de Campina Grande, Centro de Engenharia Elétrica e Informática, 2021.
"Orientação: Prof. Dr. Hyggo Oliveira de Almeida".
Referências.

1. Engenharia de Software. 2. Sistemas de Recomendação. 3. Reuso de Software. 4. Inteligencia Artifical. 5. Scrum. 6. Métodos Ágeis. I. Almeida, Hyggo Oliveira de. II. Título.

CDU 004.41(043)

FICHA CATALOGRÁFICA ELABORADA PELA BIBLIOTECÁRIA MARIA ANTONIA DE SOUSA CRB 15/398

MINISTÉRIO DA EDUCAÇÃO
UNIVERSIDADE FEDERAL DE CAMPINA GRANDE
POS-GRADUACAO CIENCIAS DA COMPUTACAO
Rua Aprigio Veloso, 882, - Bairro Universitario, Campina Grande/PB, CEP 58429-900

FOLHA DE ASSINATURA PARA TESES E DISSERTAÇÕES

EDNALDO DILORENZO DE SOUZA FILHO

UMA ABORDAGEM PARA RECOMENDAÇÃO DE CASOS DE TESTE EM PROJETOS ÁGEIS BASEADOS EM SCRUM

Tese apresentada ao Programa de Pós-Graduação em Ciência da Computação como pré-requisito para obtenção do título de Doutor em Ciência da Computação.

Aprovada em: 23/08/2021

Prof. Dr. HYGGO OLIVEIRA DE ALMEIDA, UFCG, Orientador

Prof. Dr. EVANDRO DE BARROS COSTA, UFAL, Examinador Interno

Profa. Dra. PATRICIA DUARTE DE LIMA MACHADO, UFCG, Examinador Interno

Prof. Dr. FREDERICO ARAUJO DURÃO, UFBA, Examinador Externo

Prof. Dr. ANDRÉ CAVALCANTE HORA, UFMG, Examinador Externo

Documento assinado eletronicamente por **Evandro de Barros Costa**, **Usuário Externo**, em 23/08/2021, às 12:25, conforme horário oficial de Brasília, com fundamento no art. 8º, caput, da Portaria SEI nº 002, de 25 de outubro de 2018.

Documento assinado eletronicamente por **Frederico Araújo Durão**, **Usuário Externo**, em 23/08/2021, às 12:25, conforme horário oficial de Brasília, com fundamento no art. 8º, caput, da Portaria SEI nº 002, de 25 de outubro de 2018.

Documento assinado eletronicamente por **HYGGO OLIVEIRA DE ALMEIDA**, **PROFESSOR(A) DO MAGISTERIO SUPERIOR**, em 23/08/2021, às 12:41, conforme horário oficial de Brasília, com fundamento no art. 8º, caput, da Portaria SEI nº 002, de 25 de outubro de 2018.

Documento assinado eletronicamente por **Andre Cavalcante Hora**, **Usuário Externo**, em 23/08/2021, às 13:40, conforme horário oficial de Brasília, com fundamento no art. 8º, caput, da Portaria SEI nº 002, de 25 de outubro de 2018.

Documento assinado eletronicamente por **PATRICIA DUARTE DE LIMA MACHADO**, **PROFESSOR(A) DO MAGISTERIO SUPERIOR**, em 24/08/2021, às 12:28, conforme horário oficial de Brasília, com fundamento no art. 8º, caput, da Portaria SEI nº 002, de 25 de outubro de 2018.

A autenticidade deste documento pode ser conferida no site https://sei.ufcg.edu.br/autenticidade, informando o código verificador **1690072** e o código CRC **8432A157**.

Referência: Processo nº 23096.049542/2021-05 SEI nº 1690072

Resumo

A utilização de metodologias ágeis para o desenvolvimento de software vem aumentando significativamente nos últimos anos, trazendo consigo muitas mudanças em relação ao modelo tradicional. Essas mudanças tiveram um grande impacto na forma como os testes são desenvolvidos e executados, uma vez que a responsabilidade da qualidade do produto final, e assim a execução de atividades relativas a testes, é de todo o time. Contudo, apesar dos benefícios alcançados, como o rápido *feedback* do cliente em relação ao produto, as metodologias ágeis também apresentam muitos desafios para o desenvolvimento e execução de testes como a alta quantidade e tipos de teste que devem ser executados no curto período de tempo de uma iteração. Além disso, os membros de times ágeis costumam negligenciar a estimativa de execução de testes durante o planejamento das iterações, fazendo assim com que o produto seja entregue sem a qualidade esperada ou ainda que a entrega seja postergada. Esses desafios são agravados quando o time de desenvolvimento é inexperiente. Neste trabalho, tem-se como objetivo a concepção de uma abordagem para recomendação de casos de teste em projetos ágeis baseados em Scrum. A abordagem compara Estórias de Usuário sendo desenvolvidas com as que existem em um banco de dados gerado contendo 217 Estórias de Usuário e 1077 casos de teste de duas orgnanizações para encontrar quais delas apresentam as similaridades mais significativas. Então, a abordagem recomenda casos de teste associados às Estórias de Usuário mais similares para permitir que o testador possa reusá-las. Para encontrar a melhor configuração pro sistema de recomendação, foi considerada a métrica *F-measure(β)*, adicionando relevância ao *recall* e não descartando a precisão, uma vez que, nesse contexto, o *recall* é mais importante. Dada a melhor configuração, foi desenvolvida uma ferramenta chamada *TestRec* para identificar a viabilidade da solução proposta. Para validação da abordagem, foram coletadas 177 Estórias de Usuário de três projetos ágeis, e comparados os casos de teste desenvolvidos pelas equipes de teste com os recomendados pela ferramenta *TestRec*. Os resultados mostraram que, utilizando a abordagem, os testadores poderiam ter reutilizado $65,09\%$ dos casos de teste desenvolvidos e incrementado a suíte de teste em $38,79\%$.

Abstract

The use of agile methodologies for software development has increased significantly in recent years, bringing many changes to the traditional model. These changes had a great impact on the way the tests are developed and executed, since the responsibility for the quality of the final product, and thus the execution of activities related to tests, belongs to the whole team. However, despite the benefits achieved, such as the customer's quick feedback regarding the product, agile methodologies also present many challenges for the development and execution of tests such as the high quantity and types of tests that must be performed in the short period of an iteration. In addition, members of agile teams often neglect to estimate test execution when planning iterations, thus causing the product to be delivered without the expected quality or even its delivery postponed. These challenges are harder when the development team is inexperienced. In this work, the objective is to design an approach for recommending test cases in agile projects based on Scrum. The approach compares User Stories being developed with those in a generated database containing 217 User Stories and 1077 test cases from two companies to find which ones have the most significant similarities. The approach then recommends test cases associated with more similar User Stories to allow the tester to reuse them. In order to find the best configuration for the recommendation system, the *F-measure (β)* metric was considered, adding relevance to the *recall* and not discarding the precision, since, in this context, the *recall* is more important. Given the best configuration, we developed a tool called *TestRec* to identify the solution viability. To validate our approach, we collected 177 User Stories from three agile projects and compared the test cases developed by project testers against the ones recommended by *TestRec*. The results showed that, using our approach, the testers could reuse 65.09% of the developed test cases, as well as increase their test suite in 38.79%.

Agradecimentos

Agradeço primeiramente a Deus por estar sempre ao meu lado e guiar meus caminhos, dando-me paz, saúde, forças e todas as condições possíveis para exercer as tarefas do dia a dia.

À minha esposa e amor da minha vida, Juliana de Castro, que, apesar de não ter escrito uma linha deste trabalho, também é responsável por sua escrita com todo suporte necessário aos nossos filhos, apoio emocional e motivacional para que fosse possível a conclusão do mesmo.

Aos meus pais Maria Ágida e Ednaldo Dilorenzo e ao meus irmãos Albert George e Patrícia Souza, pelo apoio e incentivo em todos os momentos da minha vida.

Ao meu orientador Hyggo Almeida, fonte de inspiração profissional, por toda paciência, compreensão, apoio e pela orientação deste trabalho no sentido real da palavra.

Ao meu grande amigo e co-orientador Mirko Perkusish, pelo excelente auxílio, suporte e conversas extremamente enriquecedoras durante todo o período do doutorado.

Aos professores e funcionários do Programa de Pós-Graduação em Ciência da Computação da Universidade Federal de Campina Grande.

Aos amigos e colegas do doutorado e do grupo de pesquisa *Intelligent Software Engineering* pelo apoio constante durante toda essa trajetória, em especial Felipe Ramos, Antonio Alexandre Moura Costa, Alexandre Braga, Emanuel Dantas, Danyllo Albuquerque, Renata Saraiva, José Ferdinandy, Ademar Neto, Luiz Antônio Pereira e Dálton Cezane.

Por fim, a todos que de alguma forma contribuíram para a realização deste trabalho.

Conteúdo

Lista de Símbolos

SisRec - *Sistemas de Recomendação*

EU - *Estória de Usuário*

FDD - *Feature Driven Development*

XP - *Extreme Programming*

DSDM - *Dynamic Systems Development Method*

DoD - *Definition of Done*

TDD - *Test Driven Development*

ATDD - *Acceptance Test Driven Development*

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

Metodologias ágeis vêm sendo cada vez mais adotadas por empresas de desenvolvimento de software [94]. Elas representam uma mudança significativa em relação às metodologias tradicionais [68], tendo iterações mais curtas contemplando todas as fases do ciclo de desenvolvimento de software e resultando em um incremento do produto final potencialmente implantável. Além disso, as metodologias tradicionais se baseiam na especificação das necessidades do cliente através de longas documentações, ao contrário das metodologias ágeis, que têm como princípio a atuação do cliente junto ao time de desenvolvimento, priorizando as funcionalidades do sistema a serem desenvolvidas e tendo o sistema funcionando como reflexo de suas necessidades [86]. As equipes de desenvolvimento devem ser auto-gerenciáveis nos métodos ágeis e todos os membros do time são responsáveis pela qualidade das entregas realizadas.

O processo de testes também teve uma mudança significativa nas metodologias ágeis no sentido de que todo o desenvolvimento deve ser voltado para a qualidade e orientado a testes [33]. Além disso, os membros do time responsáveis por testes – os testadores – não são responsáveis apenas pela elaboração de *scripts* de teste e execução dos mesmos. Eles são responsáveis por ajudar os clientes do projeto a expressarem suas necessidades na forma de requisitos que possam ser automatizados; ajudar os desenvolvedores a entenderem as necessidades do cliente tendo uma visão do todo na perspectiva do usuário; ajudar os desenvolvedores a implementarem a solução de forma simples e que permita a testabilidade do sistema [26].

Métodos ágeis devem responder a mudanças de requisitos solicitadas pelo cliente de

forma efetiva, e os testes também devem estar preparados para refletir essas mudanças [23]. Assim, os testadores ágeis devem sempre buscar ferramentas e métodos que facilitem o trabalho de desenvolver e adaptar testes de forma simples e eficiente. Os testes executados fornecem *feedback* da qualidade do produto desenvolvido aos envolvidos no projeto e permitem o acompanhamento do andamento do mesmo.

Em projetos ágeis, mudanças e entregas de incrementos do produto final são mais frequentes do que na metodologia tradicional [97]. Essa característica é reforçada por práticas adotadas como *Integração Contínua* [67] e vem da consequência da mentalidade ágil de validação de produtos em ciclos curtos, o que leva a mudanças na visão do produto e em seus requisitos. Como consequência, atividades de teste são mais intensas do que em abordagens tradicionais. Devido ao curto ciclo de entregas, a qualidade do produto deve ser continuamente conservada, além dos testes definidos e executados em períodos curtos. Assim, para conseguir o sucesso em ambientes caóticos, testadores devem colaborar com os desenvolvedores [7], automatizar sempre que possível e executar testes manuais de forma mais eficiente [2].

Neste trabalho realizamos um Mapeamento Sistemático [76] na literatura e um Estudo de Caso [87] com projetos reais para identificar os principais desafios no desenvolvimento e execução de testes em métodos ágeis onde pudemos constatar que esses métodos trouxeram muitos desafios, contribuindo assim para o estado da arte. A entrega de um incremento do produto final potencialmente implantável ao término de cada iteração leva ao desenvolvimento de um conjunto de testes para as funcionalidades desenvolvidas, além da execução de testes de regressão, para garantir a qualidade das funcionalidades desenvolvidas nas iterações anteriores, dando a confiança necessária para que desenvolvedores efetuem mudanças complexas durante o desenvolvimento de sistemas. Com o avanço do desenvolvimento, os testes se tornam mais complexos e numerosos, tornando o gerenciamento [14] e execução [73] desses testes uma atividade complexa e suscetível a erros.

Outro desafio encontrado é a dificuldade em estimar o esforço para desenvolvimento e execução de testes em métodos ágeis [64]. Durante a reunião de planejamento da iteração, o time de desenvolvimento seleciona as funcionalidades que serão desenvolvidas levando em consideração sua capacidade. Contudo, muitos times não conseguem identificar nesse momento o esforço para desenvolvimento e execução de testes. Situações como essa trazem

como consequência a não entrega de funcionalidades prometidas ao término da iteração, quando o esforço para desenvolvimento e execução de testes é muito maior do que o previsto.

Um dos princípios ágeis é a preferência pelo software funcionando em detrimento a vastas documentações [12]. Porém, a falta de uma documentação detalhada muitas vezes se torna um grande desafio para o desenvolvimento de testes [50], o que é agravado quando o cliente do produto não tem uma participação efetiva no desenvolvimento [74]. Muitas vezes, apesar do desenvolvimento de Estórias de Usuário em conjunto com critérios de aceitação, cenários importantes não são contemplados, exigindo assim um profundo conhecimento do negócio e experiência em testes exploratórios por parte dos testadores. Adicionalmente, esse fato dificulta o rastreamento da cobertura dos testes executados no produto.

Os desafios listados acima se tornam mais evidentes ainda quando os testadores são inexperientes [14]. Empresas de desenvolvimento de software nem sempre podem contar com equipes experientes, ou ainda disponibilizar testadores experientes para colaboração em vários projetos dadas as limitações de orçamento e a disponibilidade desses profissionais no mercado de trabalho. Uma vez que a correta execução de testes é um fator de sucesso para projetos de software [98], esses desafios podem implicar na baixa qualidade do produto entregue e, consequentemente, na baixa satisfação do cliente.

É neste contexto de teste em métodos ágeis que se insere este trabalho. Mais especificamente, propõe-se utilizar dados históricos de projetos para recomendar testes para projetos futuros.

1.1 Problemática

Nas metodologias ágeis, a alta quantidade de entregas de incrementos do produto exige a criação e execução de uma alta quantidade de testes de funcionalidades desenvolvidas na iteração atual e de regressão de funcionalidades desenvolvidas nas iterações anteriores. Assim, mudanças nos requisitos desenvolvidos previamente implicam no desenvolvimento de novos casos de testes refletindo essas mudanças e no rastreamento de todos os casos que contemplam a funcionalidade alterada para posterior atualização. Para minimizar os impactos desse fato, Hotomski et al. [49] propôs uma ferramenta chamada *GuideGen* para o rastreamento entre requisitos e testes de aceitação, possibilitando a geração de instruções textuais para

alterações sempre que requisitos forem alterados. A abordagem descrita apresenta bons resultados, porém, foca apenas no problema do rastreamento, além de ter um número reduzido de avaliações dos resultados por especialistas por questões de disponibilidade dos mesmos.

Durante o planejamento da iteração, o time de desenvolvimento seleciona as funcionalidades que devem finalizar durante a mesma. Contudo, a estimativa da capacidade da equipe muitas vezes não leva em consideração a complexidade para execução dos testes. Nidagundi et al. [72] descreve como exemplo os testes em aplicativos móveis, que, para o desenvolvimento de funcionalidades simples requerem testes complexos, considerando a presença de rede móvel disponível ou não, consumo da bateria do dispositivo, usabilidade, interrupções durante o uso da funcionalidade, dentre outras. Neste mesmo trabalho os autores propõem uma abordagem utilizando *Lean Canvas* para melhorar a estratégia de testes de uma equipe ágil. Contudo, o trabalho sugere a abordagem, mas deixa claro que a mesma necessita ser evoluída e, para isso, requer validações adicionais para atestar sua eficiência.

Um dos princípios ágeis é que o software funcionando é mais importante do que a documentação descrevendo o mesmo [12]. Contudo, essa falta de documentação muita vezes dificulta a compreensão mais profunda do negócio do qual o software em desenvolvimento faz parte. Adicionalmente, esse fato leva à execução de testes superficiais, deixando de exercitar partes importantes das funcionalidades implementadas [50]. Otaduy et al. [74] propôs uma abordagem chamada *TestMind* para incrementar os testes de aceitação do usuário utilizando mapas mentais com o intuito de melhorar a colaboração entre cliente e time de desenvolvimento, e assim adicionar informações relevantes e que possam ser utilizadas para melhorar os testes executados no software. Contudo, a validação da abordagem foi executada com apenas três participantes e sua conclusão demonstrou, segundo o próprio autor, a necessidade da evolução da mesma para contemplar outros tipos de teste.

Não foram encontradas na literatura abordagens para lidar com a falta de possível experiência por parte de testadores, para casos em que não haja disponibilidade para auxílio e treinamento por parte de testadores mais experientes, o que pode ter como consequência a execução de forma superficial das várias atribuições do testador, descritas neste trabalho. Esse problema se torna mais grave em metodologias ágeis, que tem todo o desenvolvimento orientado a testes, tornando assim essa área de fundamental importância para o sucesso do projeto.

Enuncia-se, então, o problema de negócio abordado neste trabalho: como fornecer suporte automatizado aos times de desenvolvimento ágeis baseados em *Scrum* com o propósito de melhorar os casos de teste definidos e reduzir o esforço para implementação dos mesmos?

Com base na definição do problema de negócio, enuncia-se o problema técnico abordado neste trabalho: como reutilizar casos de teste de projetos passados para recomendar casos de teste em projetos futuros?

1.2 Objetivos

O objetivo geral neste trabalho é a concepção de uma abordagem para recomendação de casos de teste de aceitação em projetos ágeis. Por casos de teste de aceitação utilizamos a definição do *SWEBOK* [15], ou seja, testes de aceitação são os testes desenvolvidos e executados para garantir que os critérios de aceitação definidos pelo cliente do produto sejam satisfeitos e assim, possibilite a definição de uma Estória de Usuário como concluída. O foco deste trabalho é melhorar a completude das suítes de teste em projetos de software, mais especificamente os que utilizam o método ágil *Scrum*, além de promover a reutilização de casos de teste no desenvolvimento dessas suítes com base no conhecimento previamente acumulado nos dados históricos de projetos. A restrição de utilização do método *Scrum* se deve à sua popularidade entre os métodos ágeis e a disponibilidade de projetos que o utilizam para validação [102].

A abordagem proposta é implementada como um Sistema de Recomendação (SisRec) que utiliza os dados de projetos ágeis desenvolvidos como base de treinamento para, a partir dela, recomendar casos de teste de aceitação que são definidos a fim de identificar se o sistema em desenvolvimento atende ao comportamento definido pelos clientes [15]. Esses casos de teste recomendados devem ser analisados pelos testadores dos projetos em desenvolvimento que, caso julguem viável, podem reutilizar as informações neles contidos para o desenvolvimento de novos casos de teste e atualizar os mesmos para que atenda ao contexto do projeto em desenvolvimento. A recomendação é baseada na avaliação da semelhança entre Estórias de Usuários em desenvolvimento com as que existem na base de dados de projetos. As estórias mais semelhantes terão os casos de teste recomendados para avaliação do testador.

Dentre os desafios citados na seção anterior, este trabalho endereça a falta de informações em projetos ágeis, pela falta de documentação escrita em conjunto com a dificuldade de participação efetiva do cliente durante o desenvolvimento e especificação de casos de teste, além da eventual falta de experiência de testadores em empresas de desenvolvimento de sistemas.

A fim de alcançar o objetivo principal neste trabalho, foram definidos os seguintes objetivos específicos:

1. Estruturar e popular uma base de dados históricos de estórias de usuários de projetos ágeis de software que utilizam o método Scrum.
2. Criar a abordagem de recomendação de casos de teste com base em dados históricos.
3. Validar a abordagem de recomendação proposta através da implementação de uma ferramenta de Sistema de Recomendação (SisRec).
4. Avaliar sua utilização em projetos reais de software.

1.3 Metodologia

O presente trabalho utiliza o ciclo de engenharia da metodologia definida por Wieringa et al. [106] para definição do problema de pesquisa e as questões de pesquisa em conjunto com suas respostas, para definição de um artefato para resolução do problema. Como mencionado anteriormente, o mapeamento sistemático e estudo de caso embasaram a confirmação do problema de pesquisa e a concepção da solução.

Assim, definem-se as seguintes questões de pesquisa:

- **RQ01:** Quais os principais desafios encontrados por testadores em projetos ágeis que impactam na qualidade dos testes executados e consequentemente na qualidade do produto final?
- **RQ02:** É possível estruturar uma base de dados históricos sobre artefatos de projetos de forma a comparar itens visando recomendação?

- **RQ03:** A solução proposta melhora o desenvolvimento de casos de teste em projetos ágeis?

Para a realização da pesquisa, foram definidas as seguintes atividades e seus relacionamentos com as questões de pesquisa apresentadas anteriormente:

1. Identificar os principais desafios encontrados por testadores em projetos ágeis na academia e em projetos reais. (**RQ01**);

2. Propor uma abordagem para estruturar dados históricos de projetos ágeis e avaliar sua viabilidade (**RQ02**);

3. Avaliar os benefícios trazidos pela abordagem no uso em projetos ágeis reais através de um estudo de caso (**RQ03**).

1.4 Relevância

Os resultados esperados com este trabalho poderão impactar positivamente o processo de detalhamento das atividades para execução de testes em projetos ágeis, viabilizando a utilização de Estórias de Usuário semelhantes para auxiliar na definição de novos casos de teste.

Outra contribuição importante é a possibilidade de utilização da ferramenta de forma experimental, dada a definição da melhor configuração encontrada para a base de treinamento desenvolvida, o que pode contribuir com a produtividade da equipe de testes e melhorar a qualidade dos casos de teste em desenvolvimento através do aproveitamento do conhecimento intrínseco aos casos de teste reutilizados.

Por fim, este trabalho tem relevância para o avanço nas pesquisas do grupo *Intelligent Software Engineering* (ISE) [1] do VIRTUS/UFCG, que investiga a aplicação de técnicas inteligentes para a melhoria da produtividade na prática de Engenharia de Software.

Dentre as contribuições esperadas neste trabalho, pode-se destacar as seguintes:

- Levantamento dos principais desafios encontrados no desenvolvimento e execução de testes em projetos ágeis em projetos reais e na academia.

[1] https://www.virtus.ufcg.edu.br/pesquisa

- Identificação dos principais fatores que influenciam o desenvolvimento de casos de testes em projetos ágeis.

- Definição de um modelo de estruturação de Estórias de Usuário e critérios de aceitação que permite a comparação entre elas e, consequentemente, a identificação de similaridades.

- Definição de um sistema de recomendação baseado em conteúdo e dados demográficos de Estórias de Usuário, com o acréscimo de heurística de seleção de Estórias de Usuário.

- Uma ferramenta para gestão de projetos ágeis utilizando a abordagem definida neste trabalho.

1.5 Estrutura do Documento

Os capítulos restantes que compõem este documento estão estruturados da seguinte forma:

Capítulo 2: Fundamentação Teórica. Apresentam-se definições gerais dos temas abordados neste documento, que servem para dar embasamento teórico aos leitores acerca de *Scrum*, testes em ágil e sistemas de recomendação.

Capítulo 3: Trabalhos Relacionados. Discutem-se os trabalhos relacionados na área, especificamente de recomendações de casos de teste.

Capítulo 4: Recomendações de Casos de Teste em Projetos Ágeis Baseados no *Scrum*. Apresenta-se o Modelo Estruturado de Estórias de Usuário, bem como detalhes das etapas de sua construção. Além disso, apresentamos todos os componentes da abordagem definida para resolução do problema de pesquisa.

Capítulo 5: Validação Off-line do Sistema de Recomendação. Apresenta-se o experimento realizado para validação do sistema de recomendação definido.

Capítulo 6: Avaliação da Solução Definida. Apresentam-se os passos definidos para avaliação da abordagem definida em projetos de desenvolvimento de sistemas reais. Além disso, os resultados dessa avaliação são descritos e discutidos em relação aos objetivos definidos.

Capítulo 7: Considerações Finais. Apresentam-se as considerações finais do trabalho, incluindo uma revisão da validação das hipóteses, contribuições e trabalhos futuros.

Capítulo 2

Fundamentação Teórica

Neste capítulo, apresentamos os principais conceitos relacionados a este trabalho. Falaremos de conceitos relacionados a métodos ágeis, em particular o *Scrum*, como os testes são executados nesse método, além de Sistemas de Recomendação (SisRec), que usaremos como ferramenta para resolução do problema.

2.1 Metodologias Ágeis

Metodologias ágeis vieram como um novo paradigma para o desenvolvimento de software, a partir do Manifesto Ágil [12], em contrapartida ao modelo tradicional. Uma das motivações para a criação dos métodos ágeis veio do fato do desenvolvimento tradicional possuir longos ciclos de desenvolvimento, fato que retarda o acesso a *feedbacks* sobre o produto por parte dos *stakeholders* do projeto e, consequentemente, aumenta a possibilidade de retrabalho.

Os longos ciclos de desenvolvimento também dificultam a inspeção da execução do projeto, uma vez que, no modelo tradicional, essa atividade ocorre com uma menor frequência e contempla um longo período, fazendo com que problemas ocorridos no início do ciclo não sejam recordados, além de haver a execução de muitas etapas do processo sem a análise do que poderia ter sido melhorado [62].

Outra motivação para o surgimento dos métodos ágeis é o excesso de documentos produzidos nas metodologias tradicionais [21]. A alta volatilidade da indústria e da tecnologia faz com que requisitos de sistemas mudem constantemente, e a definição de uma especificação completa (requisitos, projeto, arquitetura, etc.) antes da execução da codificação faz com

que qualquer mudança no sistema implique na revisão e atualização de todos os documentos, demandando um alto esforço e custo para o projeto. Essas mudanças são mais evidentes em projetos cujo escopo não é suficientemente claro e os requisitos vão sendo descobertos no decorrer do projeto.

Várias abordagens surgiram com base nos princípios ágeis, dentre elas *Feature Driven Development* (FDD) [19], *Crystal* [20], *Extreme Programming* (XP) [11], *Dynamic Systems Development Method* (DSDM) [83] e *Scrum* [86]. *Scrum* é um método ágil comumente utilizado para o desenvolvimento de produtos de software complexos organizando o fluxo de vários tipos de trabalho e será abordado em profundidade nesse trabalho.

2.2 O Arcabouço Scrum

O *Scrum* tem suas origens no artigo intitulado "The New New Product Development Game" [99] da *Harvard Business Review*, que relatava a importância de times de desenvolvimento serem auto-organizáveis, descrevendo o papel da gerência no processo de desenvolvimento. Em 1993, Jeff Sutherland criou o processo *Scrum* para uso no desenvolvimento de software combinando conceitos do artigo de *1986*. *Scrum* é um arcabouço (do inglês, *framework*), ou seja, um conjunto de valores, princípios, práticas e regras para o desenvolvimento ágil iterativo e incremental para o gerenciamento de produtos e desenvolvimento de serviços complexos [86], dado que nessas situações a habilidade para explorar e responder a mudanças é crítica.

Scrum é um arcabouço simples, centrado em pessoas e baseado nos valores de honestidade, abertura, coragem, respeito, foco, confiança, empoderamento e colaboração [86]. Na Figura 2.1 ilustra-se o esqueleto do processo iterativo e incremental do *Scrum*. O círculo representa as seguidas iterações de desenvolvimento que duram entre duas a quatro semanas [90]. Ao término de cada iteração, devemos ter como resultado um incremento potencialmente implantável do produto final [91]. O coração do *Scrum* reside nas iterações, onde no início de cada iteração, chamada *Sprint*, a equipe avalia os requisitos disponíveis para implementação, avalia sua capacidade técnica e, conjuntamente, decide o que será desenvolvido dentro da iteração.

Ao fim de cada iteração o produto resultante da mesma é apresentado aos *stakeholders* do

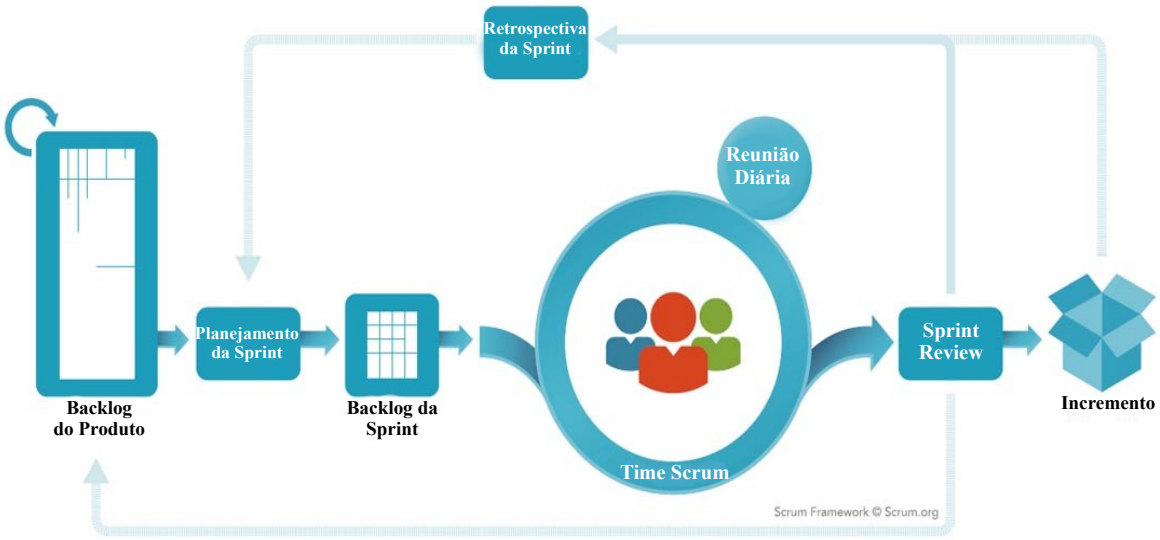

Figura 2.1: Ciclo de vida de um projeto *Scrum*.

projeto. Em seguida, é feita a retrospectiva da iteração, onde a execução é avaliada e ajustes são sugeridos pelo time de desenvolvimento visando a melhoria do processo. O *Scrum* é composto pelos **papeis**: *dono do produto*, *time de desenvolvimento* e o *ScrumMaster*. Esses papeis são detalhados na Seção 2.2.1. As principais **atividades** do *Scrum* são: *Planejamento da Sprint*, *Execução da Sprint*, *Revisão da Sprint* e *Retrospectiva da Sprint* detalhadas na Seção 2.2.2. Os principais **artefatos** definidos pelo *Scrum* são: O *backlog do produto*, o *backlog da Sprint* e o *incremento potencialmente implantável*, resultante da *Sprint* discutidos em detalhes na Seção 2.2.3.

2.2.1 Papeis no Scrum

Projetos que utilizam *Scrum* são formados por um ou mais times. Cada time *Scrum* define apenas três papeis para o desenvolvimento de projetos, podendo haver outros papeis definidos de acordo com a natureza do projeto e da organização [86]. Podermos observar na Figura 2.2 esse papeis: O **Dono do produto**, o **ScrumMaster** e o **Time de desenvolvimento**.

Dono do Produto

O **Dono do produto** (do inglês, *Product Owner*) é o ponto central do projeto no desenvolvimento do projeto. Ele é o responsável por decidir quais funcionalidades devem ser desenvolvidas no projeto e em que ordem essas funcionalidades devem ser desenvolvidas, definindo o valor agregado. O dono do produto também define o planejamento das entregas junto com o time de desenvolvimento. Adicionalmente, ele é o ponto focal entre o time de desenvolvimento e os *stakeholders* do projeto no lado do cliente (usuários, financiadores, etc.).

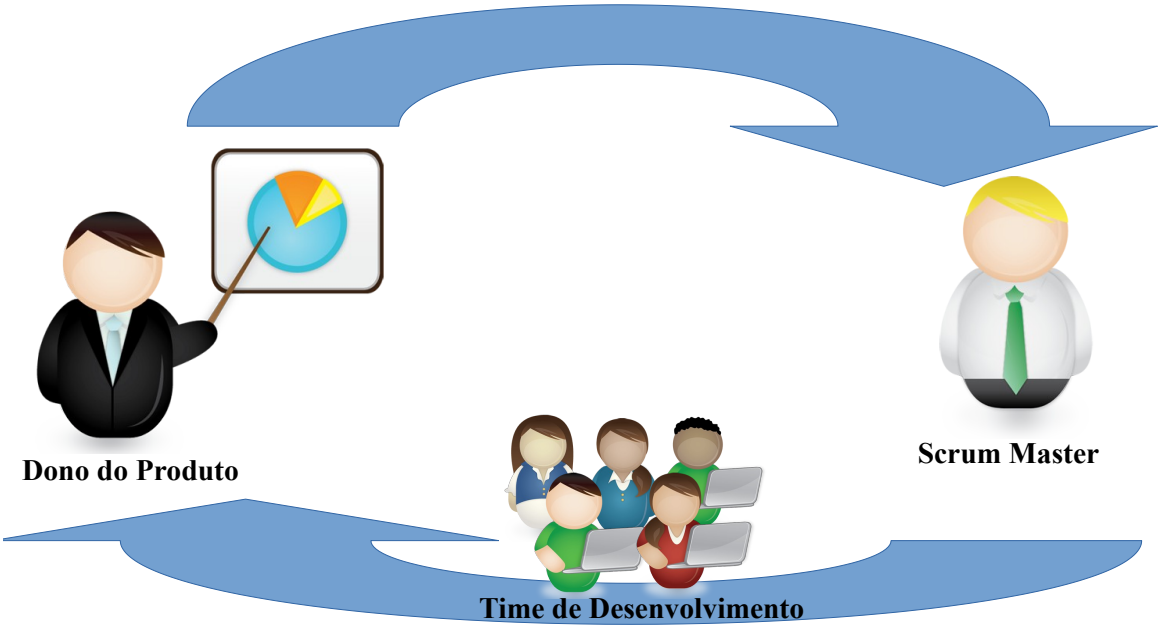

Figura 2.2: Papeis do *Scrum*.

Uma das principais responsabilidades do **Dono do produto** é a de fornecer os requisitos para que o time de desenvolvimento possa desenvolver o produto e coletar os mesmos, garantindo que as necessidades de todas as partes interessadas sejam atendidas. Também é responsabilidade do dono do produto participar de todos os eventos do *Scrum* além de validar cada incremento do produto desenvolvido através da criação e execução de casos de teste de aceitação.

ScrumMaster

O **ScrumMaster** é o responsável por garantir a correta execução do *Scrum*, ou seja, garantir que seus eventos aconteçam no momento correto, se preciso, ensinando a metodologia para cada membro do time [91] e fazendo com que o time entenda e considere os valores, princípios e práticas [86].

Também é responsabilidade do *ScrumMaster* garantir que os impedimentos relatados pelo time de desenvolvimento sejam solucionados, fazendo assim com que o desenvolvimento do produto flua naturalmente. O *ScrumMaster* não tem autoridade sobre o time de desenvolvimento, ou seja, sua função não é semelhante a de um gerente de projetos tradicional.

Time de Desenvolvimento

Times de Desenvolvimento devem ser auto-gerenciáveis, auto-organizáveis e multidisciplinares. Eles são responsáveis pelo desenvolvimento das funcionalidades priorizadas pelo dono do produto e, no final de cada *Sprint*, gerar um novo incremento do produto que será validado. Os times como um todo são responsáveis pelo sucesso do projeto e devem possuir as habilidades técnicas necessárias para superar os desafios impostos pelas necessidades do produto [91].

Os times de desenvolvimento devem contar com entre cinco e nove membros [86], o que permite a constante comunicação entre eles, a fim de permitir que as atividades sejam executadas de maneira coordenada. É possível que se use o *Scrum* para times maiores, contudo, ao invés de apenas um time, as pessoas devem ser divididas em vários times de desenvolvimento, contendo no máximo nove pessoas.

2.2.2 Atividades do Scrum

Na Figura 2.1 ilustram-se as atividades executadas em um processo que utiliza *Scrum*. Essas atividades são executadas durante o andamento de um projeto *Scrum*. Porém, Schwaber et al. [90] descreve uma fase antes da execução chamada *Pregame* (ou pré-jogo, em Português), onde é feita a concepção do projeto, ou seja, definição dos requisitos iniciais, arquitetura de alto nível da aplicação, estimativas do projeto, definição de cronogramas e entregas baseadas

no custo do projeto.

Durante o desenvolvimento do projeto, as iterações são executadas, também chamadas de *Sprints*. O tempo de execução de uma *Sprint* é definido pelo projeto e deve durar entre duas e quatro semanas. O *Scrum* possui o princípio chamado de *time-box*, que descreve como o tempo deve ser limitado nas atividades do projeto. Isso significa que ao término do período estipulado para execução da *Sprint* a mesma deve ser encerrada, mesmo que o seu objetivo não tenha sido cumprido.

No início de cada *Sprint*, o time de desenvolvimento, em conjunto com o *Dono do produto* e o *ScrumMaster* executam o **planejamento da Sprint**, definindo o objetivo da mesma. Nela, eles decidirão quais funcionalidades serão desenvolvidas e qual estratégia será utilizada dentro do período da *Sprint* para cumprir seu objetivo, baseado na prioridade das mesmas definida pelo *Dono do produto*, podendo ser consideradas outras questões, como as dependências entre as funcionalidades.

O time de desenvolvimento deve levar em consideração, na seleção de funcionalidades da *Sprint*, a capacidade de concluir as atividades selecionadas para que, ao término da *Sprint*, seja possível a entrega de um incremento do produto potencialmente implantável. Assim, as funcionalidades selecionadas são divididas em tarefas que detalham como essas funcionalidades devem ser desenvolvidas e que são expostas a todo o time de desenvolvimento para a execução da *Sprint*.

Durante a execução, os membros do time de desenvolvimento selecionam tarefas definidas durante o planejamento e executam as mesmas. Cada projeto define o significado de pronto (*Definition of Done*, ou DoD em inglês), e a completude de cada funcionalidade desenvolvida deve respeitar essa definição [95]. Todos os dias no decorrer da execução da *Sprint*, os membros do time devem se reunir para acompanhar o fluxo do desenvolvimento, inspecionando e adaptando o plano da execução da *Sprint* na atividade denominada **Reunião Diária** (*Daly Meeting*, em inglês). Essa reunião deve ter duração máxima de quinze minutos e geralmente é executada com os membros de pé, por isso também é denominada em inglês *Stand up Meeting*. Essa reunião deve ter como objetivo apenas monitorar como está o andamento da execução da *Sprint* e não deve ser utilizada para discussão de detalhes do projeto, o que deve ser feito em outras reuniões.

Ao término do período da *Sprint* acontece a reunião chamada de **Revisão da Sprint**

(*Sprint Review* em inglês), quando o time de desenvolvimento deve apresentar a todo o time *Scrum* o novo incremento do produto gerado a partir das funcionalidades completadas durante o período da *Sprint*. O *Dono do produto* pode convidar outros *stakeholders* do projeto para participar. Assim, durante a revisão da *Sprint*, é feita a confirmação do entendimento dos requisitos traduzidos no incremento do produto e sugestões de mudanças ou melhorias podem ser feitas e devem entrar como funcionalidades a serem desenvolvidas nas próximas *Sprints*.

A última atividade durante a execução é a reunião de **Retrospectiva da Sprint**. Nela, o time se reúne e avalia os pontos positivos que devem ser repetidos na execução das próximas *Sprints*, além dos pontos que devem ser melhorados ou não repetidos. Essa constante revisão e adaptação promove os ajustes necessários para melhoria da execução do projeto. Assim, o ciclo se repete durante um número de *Sprints* necessário para criação de um incremento do produto que o *Dono do produto* julgue que possa ser implantado e disponibilizado para seus usuários.

2.2.3 Artefatos do Scrum

Um projeto *Scrum* tem seus requisitos armazenados em um **Backlog do produto**, contendo um conjunto de *Estórias de Usuário* definidas pelo *Dono do produto*. Essas estórias são ordenadas do maior valor agregado, ou do retorno do investimento, para o menor. Segundo Mike Cohn [22], *Estórias de Usuário* são compostas de três aspectos:

- **Cartão:** uma descrição por escrito usada para planejamento. Visa representar os requisitos do cliente em vez de documentá-los;
- **Conversação:** conversas sobre a *estória* que servem para especificar os detalhes da funcionalidade;
- **Confirmação:** testes que documentam detalhes e que podem ser usados para determinar quando uma *estória* é concluída.

Além disso, *Estórias de Usuário* são escritas em linguagem natural tipicamente no formato:

como um [*ator*], eu quero [*ação*] de modo que [*funcionalidade*],

em que *ator* representa um tipo de usuário, *ação* representa algo que o *ator* realiza e *funcionalidade* representa o que é esperado com a *ação* do *ator*.

Estórias de Usuário também contêm condições que devem ser satisfeitas. Essas condições são chamadas de *critérios de aceitação*, que esclarecem qual o comportamento desejado para a funcionalidade descrita pela estória [86]. Elas são utilizadas para que o time de desenvolvimento tenha uma melhor compreensão do que deve ser construído e devem ser escritas com o apoio do *dono do produto* para o mesmo confirmar que as estórias foram desenvolvidas de forma que satisfaçam suas decisões.

O critérios de aceitação são associados às *Estórias de Usuário* e podem ser escritas na forma de *testes de aceitação* no verso do cartão, sendo muito importantes para representar a perspectiva do *dono do produto*. Os critérios de aceitação demonstram exemplos de comportamento e utilização do produto em desenvolvimento, por isso, também são chamadas de *Especificação pelo Exemplo* [86].

O *backlog do produto* também pode conter outros tipos de atividades, como mudanças em funcionalidades desenvolvidas, defeitos que precisam ser corrigidos e melhorias técnicas no produto, desde que estejam em comum acordo com o *dono do produto*. Ele deve estar em constante evolução dado que com o andamento do projeto, o entendimento do negócio vai sendo esclarecido e assim, novas estórias podem ser adicionadas, excluídas ou revisadas, e consequentemente re-priorizadas pelo *time de desenvolvimento* em conjunto com o *dono do produto*. Esse constante ajuste no *backlog do produto* é chamado de **Grooming** [86], e deve acontecer periodicamente durante o projeto.

Conforme descrito na seção anterior, no início de cada *Sprint* o *time de desenvolvimento* seleciona as *Estórias de Usuário* com maior prioridade, e que consigam completar, do *backlog do produto*, definindo o artefato chamado **Backlog da Sprint**. Esse *backlog* contém todas as estórias que serão desenvolvidas no decorrer da *Sprint* em andamento e, ainda durante o planejamento da *Sprint*, é dividido em *tarefas* que detalham como a funcionalidade descrita pela estória será implementada.

Ao término de cada *Sprint* o objetivo é sempre a finalização das estórias definidas no *backlog da sprint* de acordo com a definição de pronto, gerando assim o artefato *Scrum* que é um novo **incremento do produto** desenvolvido pelo projeto. Esse incremento deve ser potencialmente implantável, cabendo ao *dono do produto* decidir a viabilidade do mesmo

quanto a sua implantação ou não.

2.3 Testes Ágeis

Assim como todas as atividades em desenvolvimento de software, os testes também sofreram mudanças com o surgimento dos métodos ágeis em relação ao processo de desenvolvimento tradicional. Testes em metodologias ágeis não significam necessariamente testes executados dentro de processos que utilizam a metodologia ágil, mas sim testes que são executados seguindo os princípios ágeis, seja em projetos ágeis, ou até mesmo em projetos tradicionais.

No modelo tradicional de desenvolvimento, as fases são sequenciadas e divididas em longos ciclos de desenvolvimento, ou seja, cada fase inicia ao término da fase anterior, conforme descrito na Figura 2.3. Os testadores no modelo tradicional recebem como entrada os documentos de requisitos e desenvolvem o plano de testes para futura execução. Assim, a fase de testes se inicia ao término da fase de desenvolvimento, visando garantir a qualidade de todos os requisitos levantados e implementados durante todo o longo período de desenvolvimento. Em muitos casos, essa fase é reduzida por atrasos na fase de desenvolvimento [33, 73].

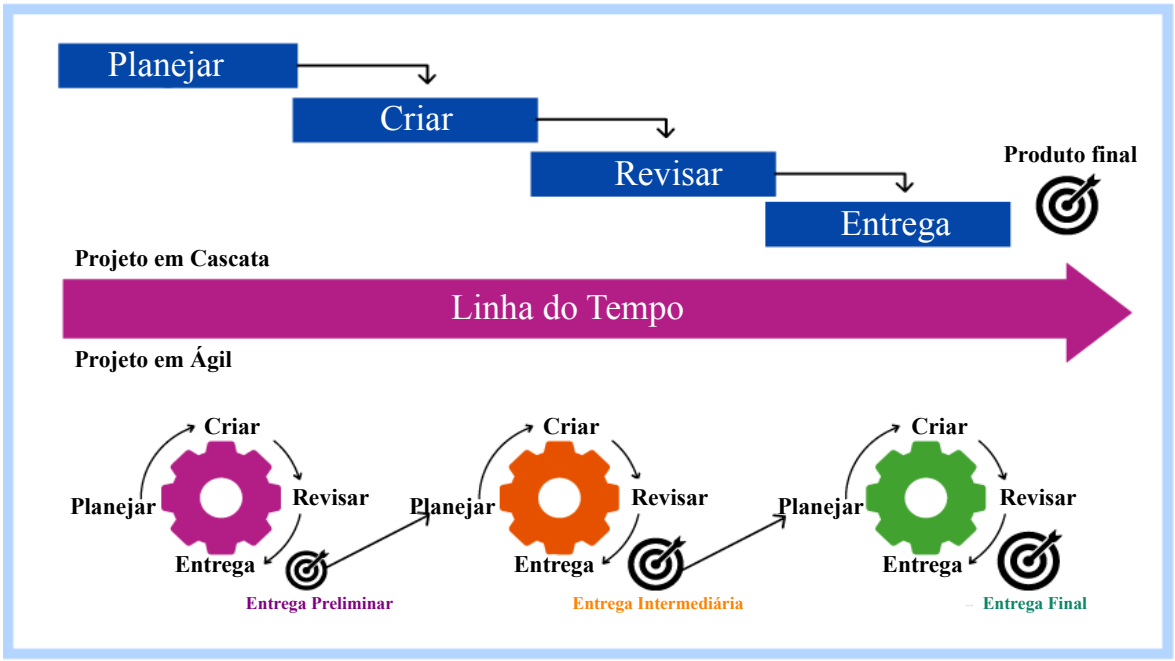

Figura 2.3: Metodologia tradicional versus ágil.

Por outro lado, nas metodologias ágeis são executados vários ciclos curtos de desenvol-

vimento, passando por todas as fases, onde todos os membros do time devem contribuir com o objetivo da iteração. Nas metodologias ágeis, pequenas funcionalidades são entregues conforme são desenvolvidas, permitindo assim com que os testadores já possam validar e garantir a qualidade do produto, baseado nas necessidades do cliente, que o mesmo ajudou a levantar em um momento anterior. Enquanto isso, novas funcionalidades são desenvolvidas para futuramente serem testadas fazendo com que os testes não fiquem apenas para o fim do projeto.

Além dos papeis descritos na Seção 2.2.1, membros de time executam uma ou mais funções específicas, como analista de requisitos, testadores, arquiteto de software, dentre outros. Crispin et al. [25] sugere duas divisões nesses papeis que atuam de forma conjunta para o sucesso do projeto: o **time do cliente**, que compreende analistas de negócio, gerentes de produto, especialistas no domínio, ou seja, todos envolvidos no lado do negócio do projeto; e o **time de desenvolvimento**, que compreende os responsáveis por implementar o código necessário para funcionamento do incremento resultante das *Sprints*.

Os **testadores** são membros do time responsáveis por todas as atividades relacionadas a testes, e se encontram no time do cliente, discutindo as funcionalidades e ajudando os especialistas do negócio na elaboração de *Estórias de Usuário* e criação de testes de aceitação; e no time de desenvolvimento, ajudando os desenvolvedores na correta implementação das funcionalidades e na execução de testes a fim de garantir a qualidade do produto gerado por eles [40]. Os testadores devem entender os dois mundos para compreender o ponto de vista do cliente e as complexidades técnicas de implementação.

Um dos princípios definidos no manifesto ágil [12] é o código funcionando ao invés de vastas documentações. Assim, várias práticas de testes se popularizaram no movimento ágil. Dentre elas o *Test Driven Development* (TDD) [10] e o *Acceptance Test Driven Development* (ATDD) [41], que permitem que testes sejam automatizados e façam parte do código produzido para o projeto, o que faz com que esse código também faça parte da documentação do produto.

Dada a grande quantidade de incrementos produzidos durante o ciclo de vida de um projeto ágil, uma prática de testes que também ganhou força e é considerado um fator de sucesso foi a automatização dos testes [23]. Ela permite que testes de regressão sejam executados rapidamente e que haja a garantia de que novas funcionalidades implementadas não afetem

negativamente as funcionalidades entregues nos incrementos anteriores.

Testes ágeis podem ser utilizados para vários propósitos. Brian Marick [65] definiu em seu blog uma classificação para esses propósitos, posteriormente atualizado por Lisa Crispin [26], chamada os **quadrantes de testes ágeis**. Podemos ver nas Figura 2.4 que os quadrantes são divididos por estarem mais voltados para o negócio ou para tecnologia e também por estarem voltados para suporte do time e para criticar o produto sendo desenvolvido.

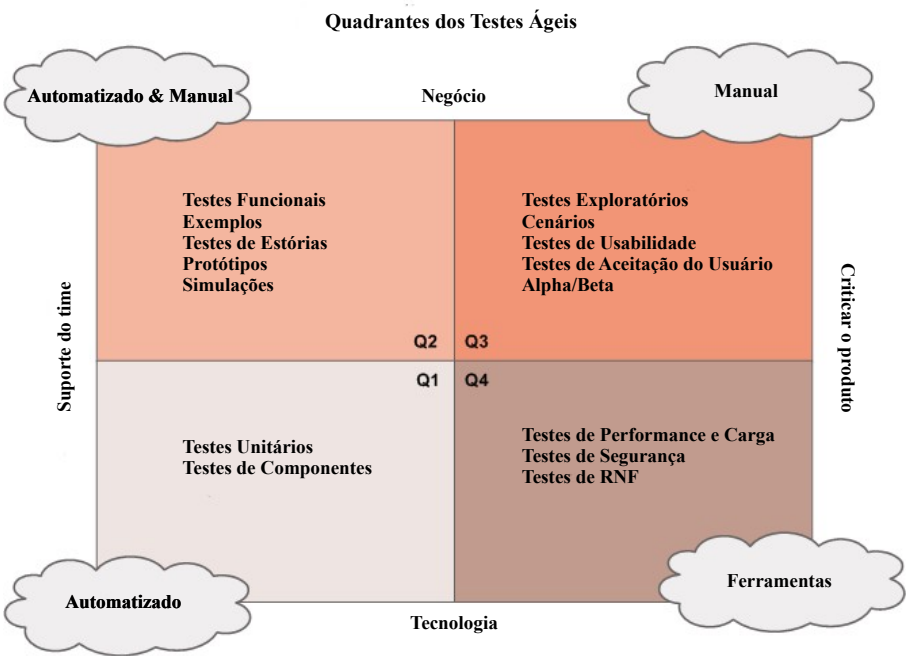

Figura 2.4: Quandrantes representando os propósitos para testes ágeis.

Os tipos de testes de suporte ao time – Quadrantes **Q1** e **Q2** – se referem aos que são utilizados para garantir a qualidade do código produzido durante o desenvolvimento do projeto, permitindo assim que alterações no mesmo possam ter uma verificação facilitada e possibilite sua rápida correção. Esses tipos de testes podem ser executados de forma automatizada como *Testes Unitários* e *Testes de Componente* voltados para a tecnologia utilizada, assim como existem tipos que podem ser executados tanto manualmente como automaticamente, como os *Testes Funcionais* e *Testes de Estória* voltados para validação do negócio sendo implementado.

Já os quadrantes **Q3** e **Q4** se referem aos testes utilizados para criticar o produto e verificar se o mesmo atende às necessidades de negócio de forma manual com *Testes Exploratórios*, *Testes de Usabilidade* e *Testes de Aceitação do Usuário*, ou através de ferramentas

voltadas para a tecnologia utilizada no desenvolvimento do produto, como os *Testes de Performance e Carga*, *Testes de Segurança* e demais *Testes de Requisitos Não Funcionais*.

2.3.1 Testes de Aceitação

Com o objetivo de garantir que as necessidades do cliente do projeto sejam atendidas, e o time de desenvolvimento possa considerar uma funcionalidade finalizada, utilizamos os *testes de aceitação*. De acordo com o padrão IEEE Standard 1012-1986 [1, 82] testes de aceitação são testes que visam identificar se o cliente deve aceitar o sistema em desenvolvimento. Conforme exemplificado na Seção anterior, os testes de aceitação são utilizados para criticar o produto e deve ser definido pelo time de desenvolvimento do sistema em conjunto com o time do cliente. Testes de aceitação são definidos e executados a partir dos critérios de aceitação definidos para cada Estória de Usuário. Além disso, testes de aceitação devem contemplar também requisitos não funcionais do sistema, uma vez que o não cumprimento dos mesmos pode influenciar negativamente na satisfação do cliente.

É extremamente desejável que os testes de aceitação sejam automatizados. Testes de aceitação automatizados favorecem práticas como a integração contínua, além de possibilitar que testes manuais sejam executados com maior velocidade e de forma menos repetitiva, além de permitir que os testadores possam focar em testes exploratórios e de usabilidade. Testes de aceitação automatizados também permitem que testes de regressão sejam feitos para qualquer mudança no sistema, o que dá ao time de desenvolvimento uma maior confiança para realizar alterações mais complexas.

Existem várias ferramentas que tornam a especificação dos testes de aceitação executáveis como Cucumber [1], JBehave [2], Concordion [3] e FitNesse [4]. Essas ferramentas requerem a escrita dos testes de aceitação na linguagem Gherkin [85], o que visa ser uma linguagem comum entre o time de desenvolvimento e o time do cliente, facilitando assim a compreensão da especificação. A listagem abaixo mostra o exemplo da especificação dos testes de aceitação para a Estória de Usuário *Realizar Login*. Nela, podemos ver o cenário *Login com sucesso*, que descreve o critério para que a funcionalidade seja considerada aceita. A

[1] https://cucumber.io/
[2] https://jbehave.org/
[3] https://concordion.org/
[4] http://fitnesse.org/

seção do texto representada pela palavra **Dado** informa a pré-condição para execução do teste, já a seção representada pela palavra **Quando** informa a ação a ser executada no teste, enquanto que a seção representada pela palavra **Então** informa o resultado esperado do teste.

Funcionalidade: Realizar Login
Como usuário
Eu quero executar login no sistema
Para que eu possa utilizá-lo

Cenário: Login com sucesso
Dado que estou acessando a aplicação
Quando informo usuário "user"e senha "123"
E clico no botão entrar
Então visualizo a página inicial

2.4 Sistemas de Recomendação

Sistemas de recomendação (SisRec) são ferramentas ou técnicas utilizadas para sugerir *itens* que possivelmente sejam de interesse de *usuários* [35]. O termo "itens" corresponde a tudo que o SisRec pode recomendar para o usuário. A ideia é semelhante ao que acontece no cotidiano, ou seja, pessoas que têm interesses semelhantes a outras recomendam itens consumidos umas às outras, aumentando assim a possibilidade de satisfação com o item consumido. Hoje em dia, existe uma grande variedade de SisRec, recomendando uma grande quantidade de itens como músicas [17], livros [61], filmes [96] e muitos outros. Os SisRec se tornaram uma área independente de pesquisa na metade dos anos 90 focando apenas em problemas que se baseiam em notas dadas por pessoas que consumiram itens, aliado à previsão de notas desses itens para pessoas que ainda não o consumiram [4].

A Engenharia de Software também tem adotado os benefícios dos SisRec para ajudar a melhorar a produtividade no desenvolvimento de sistemas [38]. Eles têm sido utilizados para a recuperação de diversos itens, como exemplos de código fonte, requisitos, arquite-

tura, práticas de desenvolvimento e muitos outros itens a partir de uma vasta quantidade de informações espalhadas e de difícil acesso. Essa utilização pode trazer vários benefícios aos projetos de desenvolvimento, como o aumento do reúso de artefatos utilizados em projetos anteriores e a melhoria da qualidade do software produzido dada a recuperação de conhecimento previamente alcançado.

De acordo com Ricci et al. [35], podemos distinguir os SisRec em seis classes de abordagens de recomendação:

- **Baseado em conteúdo** (*Content-based*): O SisRec aprende quais os conteúdos preferidos do usuário baseados na avaliação ou consumo do mesmo e recomenda itens semelhantes de acordo com critério selecionado.

- **Filtragem Colaborativa** (*Collaborative filtering*): O SisRec aprende os conteúdos preferidos do usuário, baseado no consumo de itens por usuários que possuem preferências semelhantes a ele. Existem várias maneiras de avaliar quais usuários possuem preferências semelhantes dentre elas a *baseada em usuários*, *baseada em atributos do usuário* e a *baseada em itens*.

- **Demográfica**: Esse tipo de SisRec recomenda itens baseado no perfil demográfico do usuário. Isso quer dizer que usuários com características diferentes (idade, sexo, profissão, etc.) serão agrupados e, possivelmente, terão recomendações diferentes.

- **Baseada no conhecimento** (*Knowldge-based*): Esse tipo de SisRec recomenda itens baseados no conhecimento de como as características dos itens atende as necessidades e preferências dos usuários. Eles recebem como entrada a descrição do problema e calculam quais itens mais se assemelham às caracaterísticas descritas.

- **Baseado na comunidade** (*Community-based*): Como o próprio nome já diz, esse SisRec se baseia nas preferências dos usuários ou relacionamentos. Dada a popularidade das redes sociais, informações de relacionamentos entre usuários são utilizadas com o objetivo de identificar preferências dentro de uma comunidade e recomendar itens que atendam essas preferências aos usuários que fazem parte da mesma.

- **SisRec Híbridos**: Esses SisRec são baseados em uma combinação das técnicas mencionadas acima. Dado que um sistema de recomendação tenha características que fa-

voreçam mais de uma técnica, utilizá-las em conjunto pode trazer um grande benefício nas recomendações de itens.

2.4.1 Filtragem Baseada em Conteúdo

A filtragem baseada em conteúdo recomenda itens similares aos que o usuário consumiu no passado [35]. O sistema define o perfil do usuário baseado no consumo e, a partir desse perfil, busca itens não consumidos que atendam as preferências do usuário. Os itens consumidos pelos usuários são agrupados em categorias e essas são atribuídas às preferências do usuário. Assim as buscas são facilitadas, considerando apenas itens dessas categorias [88].

Nesse tipo de filtragem, consideramos que usuários operam independentemente, ou seja, ele não necessita de informações de outros usuários para a correta recomendação [101] como é observado em outras abordagens como a *filtragem colaborativa*, descrita na próxima seção. As principais vantagens da filtragem baseada em conteúdo são:

- **Independência do Usuário:** Dado que esse tipo de filtragem não depende do consumo de outros usuários, a falta desses dados não prejudica o funcionamento do SisRec.
- **Transparência:** A explicação sobre como a recomendação foi feita pode ser facilmente identificada dadas as características dos itens recomendados serem preferidos pelo usuário, baseado no consumo de itens passados.
- **Novo Item:** A adição de novos itens no sistema de recomendação não prejudica o mesmo, que pode recomendar automaticamente esse novo item para o usuário, caso sua categoria seja de preferência do usuário.

2.4.2 Filtragem Colaborativa

Os métodos colaborativos têm como objetivo prever a utilidade de itens para usuários baseados no histórico de consumo de itens por outros usuários [93], ou seja, eles dependem da colaboração de outros usuários para que funcionem corretamente. A filtragem colaborativa é considerada a abordagem mais comum e amplamente implementada em SisRec [35].

Dentre as vantagens da filtragem colaborativa, podemos citar a habilidade de filtrar qualquer tipo de conteúdo, como textos, música, amigos, notícias, filmes, etc; a habilidade de

filtrar conteúdos baseados na complexidade e conceitos difíceis de representar; e a capacidade de realizar recomendações aleatórias [46].

Os métodos de filtragem colaborativa podem ser classificados em duas classes: métodos **baseados na vizinhança** e métodos **baseados em modelo**. O métodos baseados em vizinhança tentam prever as preferências dos usuários levando em consideração as semelhanças entre as relações dos usuários e itens e podem ser classificados em **baseados no usuário** e **baseados nos itens**. Na Figura 2.5 ilustra-se a diferença entre as filtragens baseadas no usuário e baseadas no item.

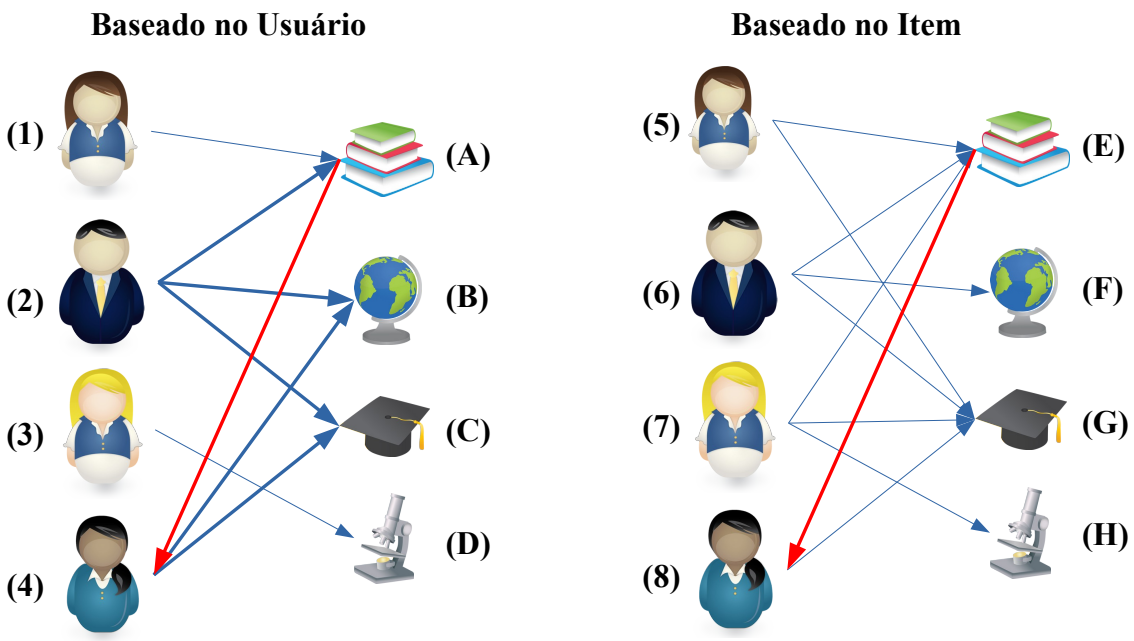

Figura 2.5: Diferença entre a filtragem colaborativa baseada no usuário e baseada no item.

A filtragem colaborativa baseada no usuário avalia os interesses de um usuário alvo (4), e busca usuários que possuem interesses semelhantes que são chamados de *vizinhos*. Assim, o sistema verifica os itens consumidos pelos vizinhos mais próximos – aqueles que possuem interesses mais semelhantes ao do usuário alvo – que não foram consumidos pelo usuário alvo e recomenda esses itens para ele. Na figura podemos ver que o usuário 2 e o usuário 4 possuem consumo de itens semelhante e identifica que o item A foi consumido pelo usuário 2 e não foi consumido pelo usuário 4, recomendando assim esse item.

Já a filtragem colaborativa baseada no item avalia o consumo de itens do usuário alvo e busca itens que são consumidos de forma semelhante sendo considerados os vizinhos [61].

A partir daí, o sistema verifica os padrões de consumo de itens mais semelhantes ao dos itens consumidos pelo usuário alvo e que ainda não foram consumidos pelo mesmo, que serão assim recomendados. Na figura podemos ver que o item E que não foi consumido pelo usuário 8 tem um padrão de consumo semelhante ao item G que foi consumido pelo usuário 8. Assim, o sistema de recomendação considera o item E como um vizinho próximo ao item G e o recomenda para o usuário 8.

Dentre as vantagens da filtragem colaborativa baseada na vizinhança, podemos destacar:

- **Simplicidade:** Os métodos baseados na vizinhança são simples de entender e de implementar, requerendo como configuração, apenas o número de vizinhos a serem considerados.

- **Justificabilidade:** O uso desses métodos favorece a justificativa das recomendações dado que o sistema computa as mesmas pelo consumo dos usuários.

- **Eficiência**: Os métodos baseados em vizinhança não necessitam exaustivas fases de treinamento que devem ser executadas em intervalos frequentes. Apesar da fase de recomendação ter processamento mais pesado, esse método pode ter o processamento de vizinhos em um passo anterior *off-line* melhorando assim o desempenho do sistema.

- **Estabilidade**: Uma característica interessante desses tipos de SisRec é que eles são pouco afetados pela constante adição de usuários e itens, o que é observado em grandes sistemas comerciais.

2.4.3 Características dos Sistemas de Recomendação

Os SisRec possuem várias características que podem ser exploradas neste trabalho. Destacamos aqui as principais características de acordo com [56].

Coverage

Coverage (do inglês, cobertura) se refere à relação entre o total de todos os possíveis itens a serem recomendados (casos de teste) e o total de itens cadastrados na base do recomendador. Neste quesito, consideramos como requisito a maior *coverage* possível, pois, dado que o

objetivo do recomendador é fornecer todos os bons itens, estão incluídos itens registrados no sistema até o momento da recomendação.

Diversity

Diversity (do inglês, diversidade) é uma característica relativa a quão vasta é a quantidade de itens que o recomendador oferece ao usuário. Em um recomendador comum, uma grande variedade pode ser considerada se muitos casos de itens são recomendados ao usuário.

Novelty

Novelty (do inglês, novidade) é uma característica de que não apenas os itens mais populares, ou seja, os mais selecionados para os usuários, sejam recomendados. Para o artefato proposto, a novidade deve ser apresentada, porém, não estamos considerando a mesma com um bom ranqueamento, dado que novos casos de teste cadastrados, podem ser muito específicos para uma aplicação em particular.

Responsiveness

Responsivess (do inglês, capacidade de resposta) define o quão rápido o comportamento dos usuários reflete nas recomendações do sistema. Dado que o uso da ferramenta não atinge altas proporções, definimos que o sistema de recomendação proposto deve possuir uma alta capacidade de resposta, ou seja, as ações dos usuários devem ser refletidas no sistema assim que forem tomadas.

Churn

Churn está relacionado a quanto as recomendações feitas pelo sistema para o mesmo usuário mudam. Como já foi descrito, neste trabalho consideramos um usuário do sistema de recomendação uma nova Estória de Usuário sendo registrada ou associada a uma *Sprint*. No sistema de recomendação proposto, cada recomendação será executada apenas uma vez, dado que as Estórias do Usuário, uma vez finalizadas não são novamente cadastradas no sistema. Assim, se considerarmos o *Churn* para Estórias de Usuário semelhantes, as recomendações podem mudar drasticamente devido à alta responsividade, descrita na seção anterior.

Explanation

Explanation (do inglês, explicação) diz respeito à capacidade do sistema de recomendação facilitar a explicação dos resultados obtidos [46]. Apesar do objetivo do projeto de pesquisa ser permitir que usuários possam reutilizar casos de teste previamente definidos, os mesmos são responsáveis pela escolha destes. Por isso, entendemos que é importante que o sistema de recomendação facilite a explicação dos resultados para permitir que o usuário possa justificar a seleção dos casos de teste recomendados.

2.5 Considerações Finais do Capítulo

Neste capítulo apresentamos de forma detalhada os principais conceitos que servem de suporte para o presente trabalho. Definimos o conceito de metodologias ágeis e apresentamos os papeis, eventos e artefatos utilizados em processos de desenvolvimento baseados em *Scrum*. Detalhamos também os principais conceitos relacionados a Sistemas de Recomendação (SisRec) e discutimos seus tipos e características, focando no conteúdo que em algum momento é utilizado durante o trabalho.

Capítulo 3

Trabalhos Relacionados

A qualidade dos testes no desenvolvimento de software é um fator muito importante para o sucesso de um projeto [74] de software ágil ou tradicional. Assim, muitos trabalhos relatam e analisam os desafios encontrados por testadores e práticas utilizadas para minimizar os efeitos destes desafios encontrados. Outros trabalhos propõem abordagens específicas para solucionar determinados problemas encontrados durante o desenvolvimento ou execução de testes em métodos ágeis.

Contudo, neste capítulo iremos focar em abordagens que visam promover a reutilização de casos de teste em projetos de software ágeis ou tradicionais, com o objetivo de auxiliar os testadores nesta atividade. Várias abordagens usam diversas técnicas para permitir o reúso de casos de teste. Assim, agrupamos as soluções encontradas na literatura de acordo com a tecnologia empregada.

3.1 Reutilização de Casos de Teste Através de Recomendação

Nesta seção descrevemos soluções similares à desenvolvida neste trabalho que utilizam sistemas de recomendação para reutilização do conhecimento adquirido em projetos passados. Ge et al. [39] propôs um sistema de recomendação baseado em filtragem colaborativa para a recomendação de casos de teste. Na abordagem definida, a similaridade entre os casos de uso do projeto é calculada considerando seus atributos de gerenciamento e os seus atributos

básicos. Como atributos de gerenciamento de casos de uso, os autores consideram o sistema operacional para o qual ele deve ser desenvolvido e o banco de dados utilizado. Já como atributos básicos, os autores consideram a descrição, o objetivo e as pré-condições do caso de uso.

Dados todos os atributos, os autores identificam que utilizar apenas um dos tipos de atributo não permite uma recomendação viável e propõem uma combinação entre os atributos de gerenciamento dos casos de uso e os atributos básicos dos mesmos. Também são coletados nos dados históricos informações sobre os testadores responsáveis por testar os casos de uso, o que contribui na acurácia do algoritmo. De posse desses dados, a proposta utiliza o algoritmo *K Nearest Neighbors (KNN)* para identificar os casos de uso similares e assim recuperar os casos de teste associados a esses casos de uso.

A abordagem utiliza elementos similares aos desenvolvidos nesse trabalho. Porém, não consideramos os dados dos testadores responsáveis como relevantes para a recomendação de casos de teste para sistemas. Outro ponto importante é que a validação da abordagem é feita através da adaptação da base de dados aberta *Movie Lens* [1] para representar dados de usuários e casos de teste a serem avaliados, o que indica que a mesma não foi avaliada utilizando dados de empresas reais. Os autores concluem que a acurácia do algoritmo é satisfatória, porém, acreditam que o custo computacional da abordagem é muito alto, dificultado sua utilização.

Por outro lado, o trabalho descrito por Janjic et al. [53] apresenta uma abordagem para recomendação de testes unitários desenvolvidos e executados para auxiliar desenvolvedores de sistemas na definição de novos testes. A abordagem utilizada tem como entrada classes de testes executados em projetos com código fonte aberto de vários repositórios como *SourceForge* ou *GitHub*. Adicionalmente, o trabalho apresenta uma ferramenta na forma de um *plugin* para o ambiente de desenvolvimento Eclipse, que executa a busca e fornece a recomendação de classes de testes semelhantes às desenvolvidas pelo usuário.

Segundo o autor, 200.000 arquivos de teste utilizando o *JUnit* foram utilizados para montagem do que é chamado de mecanismo para busca de testes. Esses arquivos então passam por uma análise semântica com o objetivo de classificá-los. Assim, um agente que é executado em *background* realiza a análise semântica da classe de teste em desenvolvimento e fornece os dados para o mecanismo de busca de testes, que recomenda classes semantica-

[1] https://grouplens.org/datasets/movielens/

mente semelhantes para que o usuário possa consultar e, consequentemente, ter um auxílio no seu desenvolvimento. A abordagem possui um interessante mecanismo de classificação de classes de teste e recomendação, porém, ela se restringe à reutilização de código para testes, enquanto que o objetivo deste trabalho é prover mecanismos para o auxílio ao desenvolvimento de casos de teste funcionais e não funcionais.

3.2 Reutilização de Casos de Teste Através de Ontologia

Algumas abordagens propõem a reutilização de casos de teste através do uso de Ontologias. Landhäußer et al. [57] propuseram uma ontologia do código desenvolvido em projetos de software em conjunto com Estórias de Usuário e os testes funcionais associados a essas em projetos utilizando metodologia ágil. A ideia principal é utilizar a ontologia para identificar passos de teste similares desenvolvidos em projetos passados e reutilizar seu código. Os autores argumentam que o conhecimento fornecido pelos artefatos desenvolvidos é essencial para o desenvolvimento e testes de novos projetos. Adicionalmente, a abordagem foi validada através de um estudo de caso utilizando dados de um projeto real, e a medição da acurácia pelas métricas *precisão* e *recall*. Os autores concluíram que a abordagem atingiu um bom *recall* (77.5%), porém, uma precisão não satisfatória (8.68%).

No trabalho realizado por Li et al. [60], a ontologia é utilizada para representar o conhecimento intrínseco aos casos de teste. Para tal fim, casos de testes desenvolvidos são armazenados em um repositório específico. Esses casos de teste devem ser certificados antes de serem armazenados no repositório para garantir que os mesmos possuam informações necessárias de acordo com a ontologia definida. A partir desse repositório de testes, testadores podem fazer buscas nos casos de teste para serem reutilizados nos projetos em desenvolvimento, ajudando assim os mesmos no desenvolvimento e, consequentemente, melhorando a qualidade do produto. As principais dificuldades da abordagem são: a necessidade de manutenção do repositório, dado que os casos de teste podem e devem evoluir; e a possibilidade de duplicação de casos de teste, dificultando a busca dos mesmos.

A mesma ideia é apresentada no trabalho realizado por Guo et al. [43], onde após analisar um alto número de casos de teste, os autores definiram uma ontologia para descrever os mesmos e armazená-los em uma biblioteca. A partir dessa biblioteca, testadores podem

fazer buscas e, caso os resultados satisfaçam suas necessidades, podem reutilizar os casos de teste retornados. Uma vez que as essas buscas não resultem em casos de teste reutilizáveis, os testadores podem gerar o caso de teste em questão e adicioná-lo à biblioteca de casos de teste para futuro reúso. O trabalho conclui que seu método de busca não se apresenta com a eficiência desejada. Além disso, não apresenta validação da abordagem em projetos com grandes quantidades de casos de teste com o objetivo de avaliar sua eficiência.

De maneira similar, Cai et al. [16] também desenvolveu uma ontologia mais completa para definir casos de teste em projetos de software baseado nas definições do *SWEBOK* [15] e no modelo de qualidade de software da *ISO 9126*. Baseada na ontologia, a abordagem calcula o que chama de distância semântica dos conceitos dos casos de teste. Com isso, é possível identificar a semelhança entre o caso de teste em desenvolvimento e os casos de teste armazenados para sua consequente busca e possibilidade de reúso pelos testadores a partir de uma biblioteca de casos de teste. A busca de casos de teste é realizada através de uma linguagem definida na biblioteca de casos de teste. O trabalho não apresenta a validação da abordagem, nem como é feito o armazenamento de casos de teste, além de sua manutenção.

3.3 Reutilização de Casos de Teste Utilizando Modelos

Várias outras abordagens foram avaliadas no intuito de se atingir o reúso de casos de teste em projetos de software. Os trabalhos apresentados por von Mayrhauser et al. [103], posteriormente por Jovanovikj et al. [54] e finalmente por Bera et al. [13] apresentam abordagens que se utilizam de modelos para a reutilização de casos de teste baseado na geração dos mesmos. No primeiro trabalho o autor aborda o uso de modelos de domínio utilizados em Linhas de Produto de Software (LPS), demonstrado no domínio de aplicações de linha de comando. O segundo utiliza métodos de reengenharia para a extração de modelos de testes a partir de projetos previamente desenvolvidos e, a partir desses modelos, realiza a geração de novos casos de teste para novos projetos. Já a terceira abordagem prevê a modelagem de requisitos para sistemas em desenvolvimento, para assim reutilizar não só os requisitos modelados, como também os artefatos associados a eles, incluindo os casos de teste.

A utilização de modelos para reutilização de casos de teste pode trazer uma dificuldade extra ao processo de desenvolvimento. O acréscimo de mais um artefato, principalmente

em projetos ágeis, pode ser um gargalo para o entendimento do time de desenvolvimento, que deve ter o conhecimento a respeito da modelagem escolhida, e para o entendimento dos membros de time que fazem parte do cliente, dado que os mesmos possuem uma visão menos técnica e, consequentemente, a dificuldade e falta de motivação para o aprendizado de novas ferramentas.

3.4 Considerações Finais do Capítulo

Neste capítulo, analisamos abordagens que têm como objetivo a reutilização de casos de teste em projetos de software. Como vimos, os autores utilizaram várias técnicas nas propostas em suas abordagens. Porém, foi identificada nos trabalhos a falta de uma abordagem amplamente verificada em projetos reais e que apresente dados que concluam que sua utilização é viável e traz benefícios para o desenvolvimento de casos de teste.

Assim, iniciamos o desenvolvimento da abordagem proposta neste trabalho que será apresentado em detalhes no próximo capítulo, tendo desde sua concepção o foco em dados de organizações reais com o objetivo de maximizar a viabilidade de aplicação da abordagem nesse ambiente. Além da abordagem definida, apresentaremos a metodologia utilizada para identificar os problemas relatados em projetos reais e posterior definição da abordagem para contemplar os mesmos.

Capítulo 4

Recomendação de Casos de Teste em Projetos ágeis Baseados em Scrum

Neste capítulo apresentamos a metodologia utilizada para identificação e confirmação do problema de pequisa e, em seguida, os passos para definição da abordagem desenvolvida para solução do problema. Além disso, apresentamos e detalhamos a solução proposta que consiste em um Sistema de Apoio à Decisão (SAD) [77] baseado em Sistemas de Recomendação (SisRec) para casos de teste em metodologias ágeis.

4.1 Metodologia de Pesquisa

Para desenvolvimento da pesquisa foi utilizada a metodologia *Design Science* [106]. O ciclo de vida dessa metodologia, descrita na Figura 4.1, inicia-se com a **investigação do problema**, onde os objetivos da pesquisa são definidos, o problema é identificado e uma investigação é iniciada com o objetivo de confirmar que se trata de um problema científico.

Dado que o problema é identificado e confirmado, a próxima fase consiste no **projeto do artefato** que será utilizado para resolução do problema identificado. Nesta fase, os requisitos do artefato são definidos, é identificado como eles contribuem com os objetivos definidos na fase anterior, é avaliado se já existem artefatos que cumpram esses objetivos e, caso não existam, um artefato é desenvolvido.

Uma vez que o artefato para atingir os objetivos definidos é desenvolvido, o mesmo deve ser **validado em ambiente experimental** com o intuito de avaliar se esses objetivos

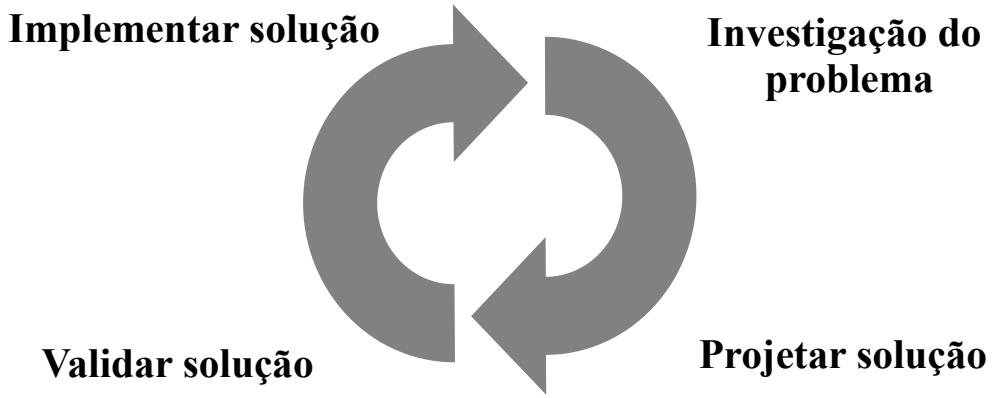

Figura 4.1: Ciclo de vida de um projeto de pesquisa utilizando a metodologia Design Science.

estão sendo alcançados e, caso o objetivo seja a melhoria de um artefato existente, existe a possibilidade de comparação entre artefatos para que se identifique a efetividade do artefato desenvolvido.

Finalmente, o sucesso da validação do artefato desenvolvido possibilita sua **implementação**, que significa a implantação do artefato no ambiente onde o mesmo será utilizado de fato. Esta fase não consiste apenas em implantar, mas também em monitorar como o artefato se comporta no ambiente real em relação aos objetivos definidos no início do projeto e como ele atende aos requisitos definidos.

4.1.1 Identificação do Problema de Pesquisa

O problema de pesquisa a que se refere este trabalho foi identificado a partir da observação do dia a dia de profissionais de testes no desenvolvimento de sistemas de informação. Observamos continuamente muitos profissionais, que ao desenvolver novos casos de testes, verificavam casos de teste de projetos anteriores, ou no próprio projeto, relativos às funcionalidades semelhantes e reutilizavam os mesmos fazendo os devidos ajustes de acordo com o contexto do projeto em desenvolvimento. Além disso, também observamos que em vários

casos, profissionais da área de teste mais experientes costumavam desenvolver uma quantidade de casos de teste mais abrangente do que profissionais iniciantes para funcionalidades com características semelhantes.

Assim, iniciando a metodologia *Design Science*, realizamos uma pesquisa com o objetivo confirmar os principais desafios identificados para testes em metodologias ágeis, ordenados pelo impacto no desenvolvimento e execução de casos de teste e a análise das principais práticas utilizadas em testes ágeis utilizando um *mapeamento sistemático na literatura* [76] e um *estudo de caso* [87] em uma instituição de pesquisa e desenvolvimento com projetos na área de software. Para atingirmos os objetivos, definimos a seguinte questão de pesquisa a ser respondida:

RQ01: Quais os principais desafios encontrados por testadores em projetos ágeis que impactam na qualidade dos testes executados e consequentemente na qualidade do produto final?

A busca por artigos do mapeamento sistemático, cujos passos são apresentados na Figura 4.2, foi realizada nos principais bancos de dados que armazenam trabalhos científicos (Scopus [1], IEEE Xplore [2], SpringerLink [3], and ACM Digital Library [4]). O mapeamento sistemático foi realizado no período de Outubro de 2018 a Fevereiro de 2019 e após a execução da busca e triagem dos trabalhos encontrados, selecionamos inicialmente *75* trabalhos após uma triagem da busca inicial. Esses artigos foram submetidos a uma leitura completa, onde foram aplicados os critérios de inclusão e exclusão, resultando em *43* trabalhos identificados como relevantes. Em seguida à seleção de artigos identificados como relevantes, executamos um processo de análise através da codificação dos dados baseada no *Guideline* proposto por Cruzes e Dyba [27] para extração de dados relevantes para pesquisa.

O estudo de caso foi realizado na instituição de pesquisa e desenvolvimento VIRTUS/UFCG [5], que executa projetos na área de software utilizando processo baseado em *Scrum* [92]. Por questões de sintetização iremos chamá-la de organização *X*. Atualmente,

[1] https://www.scopus.com/
[2] https://ieeexplore.ieee.org/
[3] https://link.springer.com/
[4] https://dl.acm.org/
[5] https://www.virtus.ufcg.edu.br/

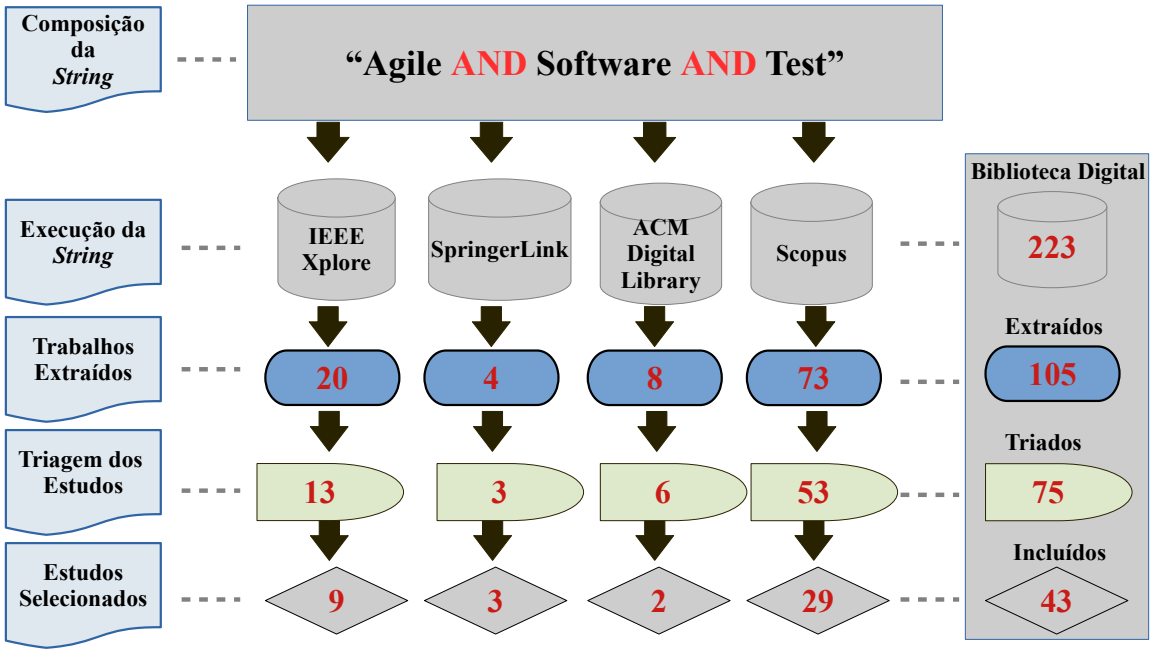

Figura 4.2: Resultados obtidos no processo de mapeamento sistemático na literatura.

organização *X* executa mais de 40 projetos em conjunto com parceiros multinacionais tendo mais de 200 colaboradores.

Os projetos são executados utilizando métodos ágeis, com processo de desenvolvimento baseado no *Scrum* desde o início, e membros de time executam papéis diferentes: gerentes de projeto, analistas de qualidade e desenvolvedores. Os times de desenvolvimento têm entre 4 e 5 membros dependendo dos requisitos do projeto, e a maioria dos desenvolvedores tem conhecimento em mais de uma linguagem de programação.

Foram gravadas entrevistas semi-estruturadas com membros da organização e, posteriormente, transcritas em documento texto para análise. Na análise dos dados, foi utilizado o mesmo processo de codificação do mapeamento sistemático, para facilitar a comparação de resultados, permitindo assim chegarmos a conclusões sobre as questões buscadas na pesquisa.

De acordo com a questão de pesquisa **RQ01**, dentre os principais desafios identificados na pesquisa realizada destacam-se:

- **Dificuldade em gerenciar alta quantidade de testes:** Uma vez que no desenvolvimento ágil a quantidade de entregas é significativamente maior do que no processo de desenvolvimento tradicional, a quantidade de testes executados também é maior, além

da necessidade de constante execução dos testes de regressão a fim de garantir que as funcionalidades implementadas na entrega atual não incluem erros nas funcionalidades desenvolvidas nas entregas anteriores. Gerenciar (incluir, localizar, alterar, etc.) essa grande quantidade de testes foi reportado como uma grande dificuldade nos testes em metodologias ágeis.

- **Pouco tempo para desenvolvimento e execução dos testes:** No desenvolvimento ágil, as iterações têm duração entre duas e quatro semanas, envolvendo todas as atividades do ciclo de desenvolvimento de software. Assim, o tempo para execução de testes é curto e, dependendo da maturidade do software, exige que vários tipos de teste sejam desenvolvidos e executados (unitário, integração, aceitação, etc.).

- **Falta de documentos detalhados:** De acordo com o manifesto ágil [12], software em funcionamento deve ter prioridade sobre vasta documentação. Isso se justifica pelo fato de tratar mudanças como um evento natural, o que aumenta a chance de se entregar um produto que agrade ao cliente [86]. Porém, a vinculação da atualização do sistema com a concomitante atualização de documentos de projeto inviabiliza a agilidade e traz riscos de sincronização entre sistema e documentos. Em um contexto onde o cliente não colabora como membro do time de desenvolvimento, a falta de documentos detalhados de projeto dificulta o desenvolvimento e a entrega de valor ao cliente e seus usuários finais.

- **Falta de envolvimento dos stakeholders:** A falta de engajamento de clientes, desenvolvedores, analistas de negócio e usuários em reuniões, definição de funcionalidades e definição de cenários, assim como a falta de tempo alocado e motivação para escrita de casos de teste pelo cliente dificulta a definição das funcionalidades do sistema, assim como a execução de testes.

- **Falta de comunicação:** A falta de comunicação entre clientes e o time de desenvolvimento utilizando uma linguagem comum como Gherkin [84] quando os clientes não têm experiência em sua utilização traz sérias dificuldades ao desenvolvimento e execução de testes de aceitação.

- **Falta de interesse dos desenvolvedores em testes:** Durante o estudo foi identifi-

cada a falta de interesse dos desenvolvedores de sistemas em atividades de teste por considerá-las monótonas. A consequência da falta de testes executados por desenvolvedores em métodos ágeis é uma possível sobrecarga na equipe de testes [51].

- **Falta de experiência dos testadores:** A falta de experiência de testadores na execução de projetos ágeis foi reportada como uma dificuldade, principalmente porque mesmo testadores experientes na metodologia tradicional sentem dificuldade ao executar projetos ágeis devido às mudanças de paradigma refletidas nas obrigações e responsabilidades de um testador ágil.

- **Dificuldade em estimar esforço para testes:** Em projetos ágeis é comum o erro na estimativa de execução de testes, visto que algumas funcionalidades no sistema, mesmo sendo facilmente desenvolvidas, apresentam uma grande complexidade para execução. A falha na estimativa de execução dos testes eventualmente provoca o término de iterações sem sua finalização, o que, dependendo da definição de pronto do projeto, pode acarretar a não entrega de funcionalidades e, consequentemente, no atraso de entrega.

Na Figura 4.3 ilustram-se os problemas identificados durante o estudo no mapeamento sistemático (academia) e no estudo de caso (projetos reais). Além disso, a figura apresenta quais problemas foram considerados mais e menos relevantes na revisão sistemática, identificando a quantidade de vezes em que o mesmo foi citado e em projetos reais através dos questionamentos feitos aos entrevistados durante o estudo de caso.

4.2 Solução Proposta

Métodos ágeis têm se tornado cada vez mais populares na indústria [48] e a utilização do *Scrum* no desenvolvimento de software tem trazido vários benefícios para clientes e fábricas de software no desenvolvimento e testes de aplicações [71]. Contudo, surgiram algumas dificuldades decorrentes de suas características conforme demonstrado em estudo realizado, contemplando as atividades de especificação e execução de casos de teste.

Assim, com o objetivo de auxiliar membros do time de desenvolvimento ágil a criar e executar casos de teste, nós propomos neste trabalho um Sistema de Apoio à Decisão

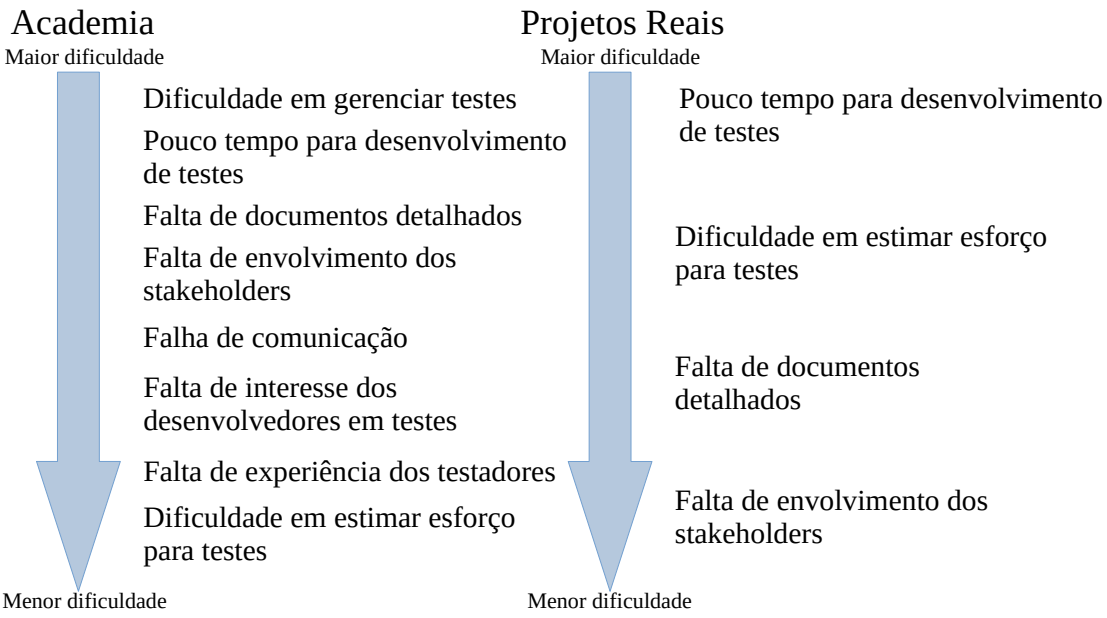

Figura 4.3: Comparativo entre problemas encontrados em projetos reais e na academia.

(SAD) [77] baseado em Sistemas de Recomendação (SisRec) [56] para auxiliar na produção de casos de testes através da reutilização de casos de teste recomendados previamente desenvolvidos em projetos anteriores ou no próprio projeto em desenvolvimento.

Essa abordagem recebe como entrada um requisito, que será comparado aos desenvolvidos previamente para identificar requisitos semelhantes e assim recomendar casos de teste associados a ele, uma vez que os mesmos são definidos para verificar requisitos do sistema utilizando um conjunto controlado e pré-definido de entradas para garantir que o sistema se comporta como esperado. Assim, assumimos que casos de teste são orientados a requisitos e, se requisitos são semelhantes, os casos de teste associados a eles também são. Em projetos ágeis, o principal artefato que armazena o conhecimento do produto são as Estórias de Usuário [8, 52, 104] que, por sua simplicidade, facilitam as discussões entre os *stakeholders* do projeto.

Formalizando o problema, temos EU o conjunto de toda as Estórias de Usuário desenvolvidas e armazenadas na base de dados de uma determinada empresa $EU = \{e_1, e_2, ..., e_n\}$ e e_a, a *Estória Alvo*, ou seja, uma nova Estória de Usuário a ser desenvolvida pelo projeto corrente e alvo de recomendações, onde $e_a \notin EU$. Seja C o conjunto de todos os casos de teste cadastrados na base de dados da mesma empresa $C = \{c_1, c_2, ..., c_m\}$, temos que cada

Estória de Usuário e_i que pertence a EU possui um conjunto de casos de teste associados $C_i \subset C$ e uma similaridade com a Estória Alvo $sim(e_a, e_i) \in (0, 1)$. Assim, nossa proposta consiste em encontrar um subconjunto de Estórias de Usuário $EU' \subset EU$ que possuam a maior similaridade com e_a e recomendar os casos de teste associados a esse subconjunto de Estórias C'.

A proposta consiste nos principais elementos:

- **Taxonomia para Classificação de Estórias de Usuário:** Definimos uma taxonomia para classificação de Estórias de Usuário para permitir a comparação entre as mesmas e assim identificar artefatos relacionados, possibilitando a recomendação dos mesmos.

- **Instrumentação do Scrum:** Para viabilizar a classificação das Estórias do Usuário e assim a recomendação de casos de teste, foi necessário o acréscimo pontual de atividades no modelo padrão do *Scrum*, assim como a definição dos papeis responsáveis por essas.

- **Seleção dos Critérios de Aceitação:** Dada a importância de critérios de aceitação para o desenvolvimento de casos de teste, surgiu a necessidade de estruturação e padronização de critérios de aceitação a fim de favorecer a análise e recomendação de casos de teste.

- **Recomendação de Casos de Teste:** A recomendação se dá pela coleta de informações das fontes de dados, análise dos dados coletados para geração de recomendações personalizadas de casos de teste para requisitos funcionais de projetos de software.

A instrumentação e o sistema de recomendação de casos de teste são detalhados nas subseções seguintes. A instrumentação do *Scrum* foi realizada em conjunto com o grupo de pesquisa *Intelligent Software Engineering* (ISE), do VIRTUS/UFCG.

4.3 Taxonomia de Estórias do Usuário

Com o objetivo de operacionalizar o SisRec, precisamos de um mecanismo para comparar Estórias de Usuário, surgindo assim a segunda questão de pesquisa que direciona este trabalho:

RQ02: É possível estruturar uma base de dados históricos sobre artefatos de projetos de forma a comparar itens visando recomendação?

Na tentativa de respondermos essa questão, definimos uma taxonomia para adicionar uma ligação semântica entre Estórias de Usuário, facilitando assim a comparação entre elas em termos de características desenvolvidas [100]. Essa taxonomia foi desenvolvida em conjunto com o grupo *Intelligent Software Engineering* (ISE) sendo descrita em detalhes no trabalho desenvolvido por Dilorenzo et al. [32] e utilizada nos trabalhos desenvolvidos por Dantas et al. [29] e Ramos et al. [80]. Definimos as Estórias de Usuário com a inclusão de novos atributos com o termo *Estórias de Usuário Semi-Estruturadas*.

Durante a execução do projeto de pesquisa, coletamos Estórias de Usuário a partir de dados de projetos reais e definimos uma base de categorias inicial, com o objetivo de viabilizar sua utilização em empresas que não possuem tais informações. Para essa atividade, foram analisados *5 backlogs* de projetos de desenvolvimento de software compostos de *118* Estórias de Usuário.

No decorrer da atividade, foram encontradas diversas dificuldades. Dentre elas, podemos citar a dificuldade de entendimento do propósito de algumas estórias devido à falta de clareza de sua descrição, de modo que algumas estórias foram descartadas por não ser possível a identificação de seu propósito. Também foram descartadas Estórias de Usuário referentes a atividades técnicas, como por exemplo, estórias criadas para refatoração de código, ou para ajustes na arquitetura do sistema.

A constante análise das Estórias para definição da taxonomia nos levaram a classificá-las em dois níveis: *Módulo* e *Operação*. As operações devem ser sempre relacionadas com seus módulos. A taxonomia inicial resultou em um total de *3* módulos e *18* operações. Na Tabela 4.1 demonstramos exemplos de itens da taxonomia definida com todos os módulos definidos: **Cadastro**, que consiste na especificação do desenvolvimento de funcionalidades referentes a *inserção*, *busca*, *alteração* e *exclusão* de dados; **Autenticação**, referente ao desenvolvimento de funcionalidades referentes ao controle de autenticação e acesso aos sistemas; e **Gerencial**, que consiste em funcionalidades vinculadas à parte operacional do sistema (dashboards, relatórios, etc.).

Após o desenvolvimento inicial, várias atualizações vem sendo continuamente realizadas a partir da análise de Estórias de Usuários de novos projetos. Assim, classificações são feitas

Tabela 4.1: Taxonomia para classificação de Estórias do Usuário.

Módulo	**Operação**
Autenticação	Executar login com nome de usuário e senha
	Executar login com OAuth
	Recuperação de senha
	Primeiro login
	Validar permissões do usuário
	Atualizar perfil do usuário
	Criar conta
	Remover conta
Cadastro	Recuperar dados
	Atualizar dados
	Inserir dados
	Remover dados
	Alterar dados
Gerenciamento	Visualizar Dashboard
	Exportar relatório para PDF
	Exportar relatório para XLS
	Notificar através da aplicação
	Notificar por e-mail

constantemente, e reuniões são realizadas para se analisar novas classificações possíveis, se aprovadas pelos membros do grupo *ISE*.

Exemplos de Estórias de Usuário com suas respectivas classificações são demonstradas na Tabela 4.2. Como pode ser percebido, a descrição de várias Estórias de Usuário apresentadas na tabela são comuns a vários projetos, o que significa que uma Estória de Usuário definida em outro projeto semelhante a de identificador *1* da tabela, deve ter a mesma classificação. Isso é importante, pois para o sistema de recomendação, definido na próxima seção, facilita a comparação entre duas estórias sem necessidade da análise do texto completo usando uma técnica como PLN [24].

Tabela 4.2: Exemplos de classificação de *USs* de acordo com suas características.

#	Descrição da Estória	Módulo	Operação
1	Como administrador, quero poder efetuar o *login* pela aplicação Web, para acessar a aplicação.	Autenticação	Fazer *login* com usuário e senha
2	Como administrador, quero poder solicitar que minha senha seja enviada por e-mail, para que eu possa recuperá-la em caso de esquecimento.	Autenticação	Recuperar senha
3	Como Administrador desejo fazer *login* para ter o primeiro acesso ao sistema	Autenticação	Acessar pela primeira vez o sistema
4	Como Admin desejo visualizar todos os usuários cadastrados para ver suas informações	Cadastro	Recuperação de dados
5	Como Scrum Master quero editar as respostas do questionário para melhor representar um fator	Cadastro	Atualizar dados
6	Como Scrum Master quero adicionar métricas para avaliação dos fatores da Sprint	Cadastro	Inserir dados
7	Como Scrum Master, desejo visualizar gráficos de radar plot no dashboard do projeto	Gerencial	Visualizar painel
8	Como administrador, quero poder configurar alarmes na aplicação Web, para receber notificações quando a água estiver no nível mínimo, máximo ou em um valor determinado.	Gerencial	Notificar via aplicação

Vale salientar que a taxonomia definida independe do projeto em desenvolvimento e as mesmas podem ser reutilizadas para qualquer projeto. Porém, durante a classificação inicial e posterior executada, foram cobertos apenas projetos no domínio de aplicações corporativas (Web) com o objetivo de viabilizar a demostração de utilidade da taxonomia.

4.4 Instrumentação do Scrum

O *Scrum* não é um processo de desenvolvimento, mas um arcabouço (do inglês, *framework*) para organização e, principalmente, gerenciamento do trabalho [86] baseado em valores e princípios e que contém eventos, papéis, artefatos e regras.

Com o objetivo de criar uma ferramenta que nos permita avaliar a resolução dos problemas de pesquisa, identificamos que muitos desses problemas possivelmente seriam solucionados através da reutilização de artefatos de desenvolvimento de software previamente desenvolvidos, seja no projeto atual ou em projetos anteriores, cujos dados estejam acessíveis a essa ferramenta, o que pode trazer os seguintes benefícios:

- **Reutilizar o conhecimento de outros testadores:** o reuso de casos de teste possibilita o aproveitamento do desenvolvimento de casos de teste executados por profissionais experientes, ajudando assim profissionais menos experientes que podem não ter capacidade para desenvolvê-los.

- **Diminuir o esforço do desenvolvimento de casos de teste:** dada a possibilidade da reutilização e adaptação de casos de teste para Estórias de Usuário semelhantes às que estão sendo desenvolvidas, o esforço para criação de uma suíte de casos de teste diminui proporcionalmente à quantidade reutilizada, aumentando assim a produtividade da equipe.

- **Facilitar a estimativa de desenvolvimento e execução de testes:** Uma vez que casos de teste reutilizados já foram executados, eles podem ser facilmente estimados. A diminuição do esforço do desenvolvimento dos casos de teste permite que essa estimativa seja realizada mais cedo, dentro de uma *Sprint*.

Uma vez que os artefatos de desenvolvimento como código, requisitos funcionais e casos de teste são vinculados às funcionalidades do sistema em desenvolvimento, e que essas fun-

cionalidades são representadas no *Scrum* através de Estórias do Usuário, identificamos que, para reutilizar os artefatos relacionados às Estórias do Usuários, deveríamos promover meios de possibilitar a comparação entre essas estórias, e assim, encontrar estórias semelhantes.

Na Figura 4.4, é possível observarmos as alterações definidas nas atividades padrão do *Scrum*. Utilizando a metodologia *Scrum* padrão, o *Product Owner* define os critérios de aceitação para cada Estória do Usuário com o intuito de possibilitar o time identificar a conclusão dos mesmos. Na instrumentação proposta neste trabalho o passo seguinte é a seleção de critérios de aceitação previamente definidos através de uma busca simples (A). Em seguida, o *time Scrum*, em comum acordo com o *Product Owner*, deve categorizar as Estórias de Usuário de acordo com a taxonomia também definida na seção anterior (B). É importante frisar que nenhum elemento do framework *Scrum* foi removido. Nas próximas subseções detalhamos as alterações sugeridas nessa proposta.

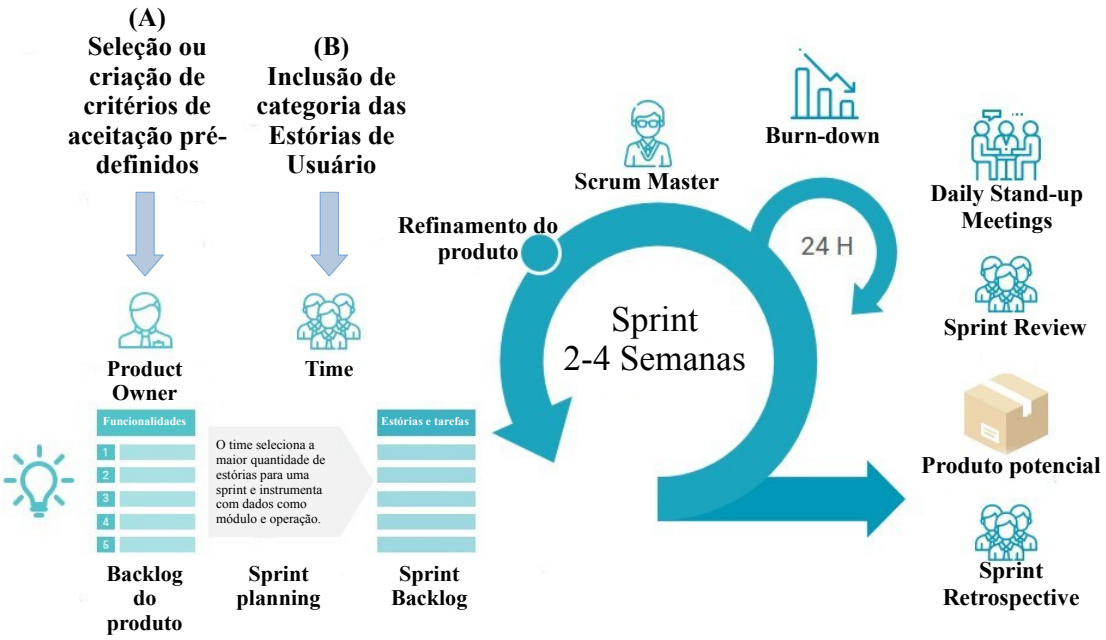

Figura 4.4: Processo de desenvolvimento utilizando *Scrum* com etapas de instrumentação.

4.4.1 Seleção dos Critérios de Aceitação

Critérios de aceitação, em conjunto com a descrição das Estórias do Usuário são muitas vezes utilizadas como fontes de requisitos para o desenvolvimento utilizando *Scrum* [58]. Por isso, o *Product Owner* – geralmente em conjunto com a equipe de desenvolvimento – especifica

os critérios de aceitação, que são as características ou qualidades do sistema expressas sob a perspectiva do negócio.

Os critérios de aceitação devem ser utilizados ao fim da *Sprint* com o objetivo de identificar que uma Estória do Usuário foi corretamente desenvolvida pelo time *Scrum* [34]. Assim, os critérios de aceitação são essenciais para a definição de testes que confirmem que os mesmos são válidos. Esses testes são chamados de **testes de aceitação** [66]. Uma vez que critérios de aceitação são essenciais para o desenvolvimento de casos de teste, já que estes podem representar os requisitos de um sistema utilizando métodos ágeis, eles devem ser um item a ser considerado na recomendação de casos de teste.

Uma forma de identificar e comparar critérios de aceitação seria a utilização de Processamento de Linguagem Natural (PLN) [24] com base nos textos de sua descrição. Contudo, sua utilização traz uma complexidade extra ao sistema de recomendação e, mesmo sendo utilizada, pode ter seu desempenho prejudicado pela qualidade da escrita dos critérios em projetos de desenvolvimento, além dos critérios de aceitação fornecerem uma quantidade reduzida de dados para análise.

Assim, para identificar e comparar critérios de aceitação visando futura utilização dos mesmos na recomendação de casos de teste, analisamos uma base de dados contendo *200* critérios para definir um conjunto padrão, com o objetivo de permitir que o usuário possa selecionar quais devem ser utilizados para cada Estória de Usuário. No total, foram definidos *59* critérios de aceitação padrão sendo classificados como **funcionais** e **não funcionais**. Na Tabela 4.3 apresentamos exemplos de variados critérios de aceitação padrão extraídos da base de dados coletada de projetos reais e cuja construção será detalhada em seções posteriores.

Na tabela, é possível identificarmos que os critérios padrão foram estruturados no formato de *template* para serem utilizados da forma mais genérica possível e permitir que critérios desconhecidos ao sistema desenvolvido sejam evitados. De acordo com nosso processo, descrito na Figura 4.4, novos *templates* podem ser criados caso exista a necessidade, ou seja, na percepção de que o mesmo será utilizado em futuras Estórias do Usuário, sendo uma decisão a ser tomada pelo *Product Owner*.

Tabela 4.3: Exemplos de critérios de aceitação padrão.

ID	Descrição	Tipo
c_1	Ao cadastrar dados da <entidade> a mesma deve ser persistida no banco de dados	Funcional
c_2	Ao tentar alterar uma <entidade> cujo identificador já esteja cadastrado no sistema o mesmo deverá exibir mensagem	Funcional
c_3	Ao informar login cadastrado e senha incorreta o sistema deve exibir mensagem de senha incorreta.	Funcional
c_4	Ao inserir um link de um <entidade> inexistente um erro 404 deve ser exibido	Funcional
c_5	O conteúdo deve ser exibido nos principais browsers (IE, Chrome, Firefox)	Não Funcional

4.4.2 Categorização de Estórias de Usuário

Uma vez definida a taxonomia para classificação de Estórias de Usuário, definimos o processo para categorização e reutilização das categorias definidas apresentado na Figura 4.5. Conforme podemos observar na figura, o processo se inicia com a definição de uma nova Estória do Usuário que deve ser analisada durante o processo para identificação de sua categoria. De acordo com o conteúdo da nova Estória, deve ser localizada uma categoria que seja adequada. Em seguida, ao identificar a categoria, a mesma é utilizada para classificar a nova Estória.

Contudo, caso nenhuma categoria seja adequada ao conteúdo da estória, o usuário pode criar uma nova categoria e associá-la a estória para, em seguida, persistir a mesma na base de dados, permitindo que essa categoria possa ser utilizada na classificação de outras estórias a serem desenvolvidas no futuro. Esse processo permite uma constante evolução da taxonomia, fazendo assim com que seja, cada vez menos, necessária a criação de novas categorias para classificação de novas Estórias de Usuário. Adicionalmente, a flexibilidade do processo permite que possamos construir uma classificação para outros domínios de aplicação e tipos de projeto no futuro.

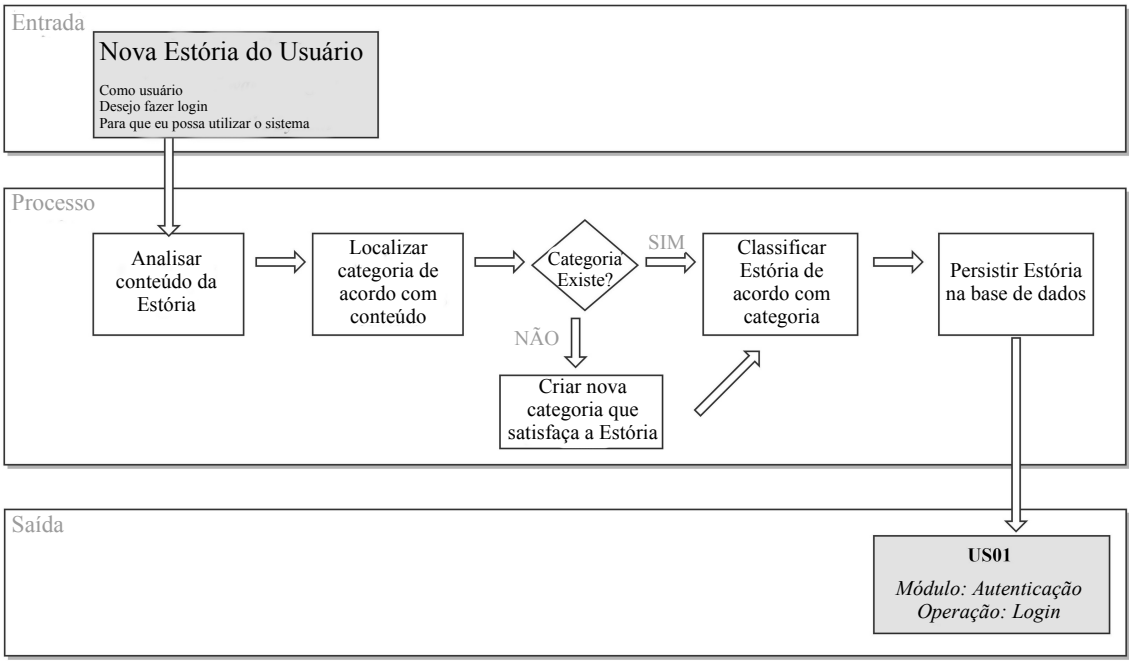

Figura 4.5: Processo para armazenamento e recuperação de categorias de Estórias do Usuário.

4.5 Recomendação de Casos de Teste

Dados os problemas relacionados ao desenvolvimento e execução de casos de teste encontrados durante a pesquisa na literatura e descritos na Seção 4.1.1, buscamos uma solução para resolver boa parte deles e entendemos que a reutilização de artefatos desenvolvidos previamente é uma maneira eficiente para tal fim, ajudando e dando suporte ao desenvolvimento de software baseado em *Scrum*.

A instrumentação do *Scrum* alinhada à criação da taxonomia de Estórias de Usuário e a padronização dos critérios de aceitação possibilita a operacionalização da solução proposta neste trabalho. Sistemas de recomendação têm como objetivo utilizar ferramentas e técnicas para recomentar itens para usuários [35]. Tipicamente, sistemas de recomendação são utilizados para se recomendar itens específicos (livros, músicas, produtos, notícias, etc.), contudo, nossa abordagem propõe a recomendação de casos de teste (item de um SisRec) a fim de serem selecionados como artefatos relacionados às Estórias do Usuário (usuário de um SisRec), adaptação semelhante feita por Han et al. [44].

Assim, adaptando a definição genérica de problemas de recomendação apresentada por Adomavicius e Tuzhilin [4] para o contexto deste trabalho, tem-se: seja E o conjunto de

todas as Estórias de Usuário e C o conjunto de todos os casos de teste que podem ser recomendados. Seja u a função de utilidade de um caso de teste c para uma Estória de Usuário e (u: $E \times C \rightarrow T$, onde T é o conjunto total ordenado), então, para cada Estória de Usuário $e \in E$, pretende-se recomendar os casos de teste $c' \in C$ que maximizem a utilidade da Estória do Usuário e. Mais formalmente, o problema pode ser representado pela Equação 4.1 [4], onde $\arg\max_{c \in C} u(e, c)$ representa a maior similaridade entre estória de usuário e e o conjunto de Estórias de Usuário E.

$$\forall e \in E, {c_e}' = \arg\max_{c \in C} u(e, c). \tag{4.1}$$

4.5.1 Requisitos do Artefato

Um dos passos definidos na metodologia *Design Science* é a definição de requisitos do artefato a ser definido para atingir os objetivos dos *stakeholders* do projeto. Dado que o sistema para recomendação visa fornecer o maior número de casos de teste possíveis de acordo com os objetivos definidos, utilizamos a abordagem *Encontrar todos os bons itens* [35], ou seja, fornecer todos os possíveis casos de teste que o usuário possa selecionar. A ideia é que ao planejar uma *Sprint* no método ágil, o usuário selecione as Estórias do Usuário que farão parte da mesma e assim o sistema de recomendação possa oferecer possíveis casos de teste a serem executados naquela *Sprint*, que devem ser analisados e selecionados pelo usuário do sistema. Assim, analisamos, com o apoio do grupo de pesquisa *ISE*, propriedades de sistemas de recomendação descritos em Ricci [35] e Kane [56] que podem ser avaliados.

O artefato resultante do projeto de *Design Science* tem como principal requisito analisar os requisitos em desenvolvimento durante um projeto ágil e recomendar o maior número possível de casos de teste que possam ser aceitos pelo usuário do artefato. Os requisitos neste trabalho são as Estórias de Usuário, critérios de aceitação e requisitos não-funcionais do projeto.

Os casos de teste devem ser recomendados em formato de lista, sem tamanho pré-determinado e ordenada pela possível relevância do caso de teste de acordo com os dados coletados pelo sistema. O principal objetivo é possuir o maior número de acertos possível, independente do usuário ter uma alta quantidade de casos de teste para avaliar a relevância.

Para suportar os requisitos definidos, tomando como base o trabalho realizado por Ramos

et al. [79] em conjunto com o grupo de pesquisa ISE, decompomos o sistema de recomendação em três componentes:

- **Coletor de dados:** responsável por buscar informações das Estórias de Usuário Semi-Estruturadas em conjunto com informações do projeto para criação dos perfis de Estórias do Usuário Semi-Estruturadas;
- **Transformador de dados:** responsável pela transformação dos perfis de Estórias de Usuário Semi-Estruturadas em vetores de característica para permitir o cálculo de proximidade e, consequentemente, a recomendação de casos de teste;
- **Recomendador:** responsável por analisar os vetores de características gerados pelo transformador de dados para Estórias de Usuário, calcular a proximidade das mesmas e recomendar seus casos de teste associados, ranqueados por relevância.

Na Figura 4.6 ilustra-se como os componentes desenvolvidos interagem para possibilitar a recomendação de casos de teste. Nas próximas seções, detalharemos o funcionamento de cada componente especificamente.

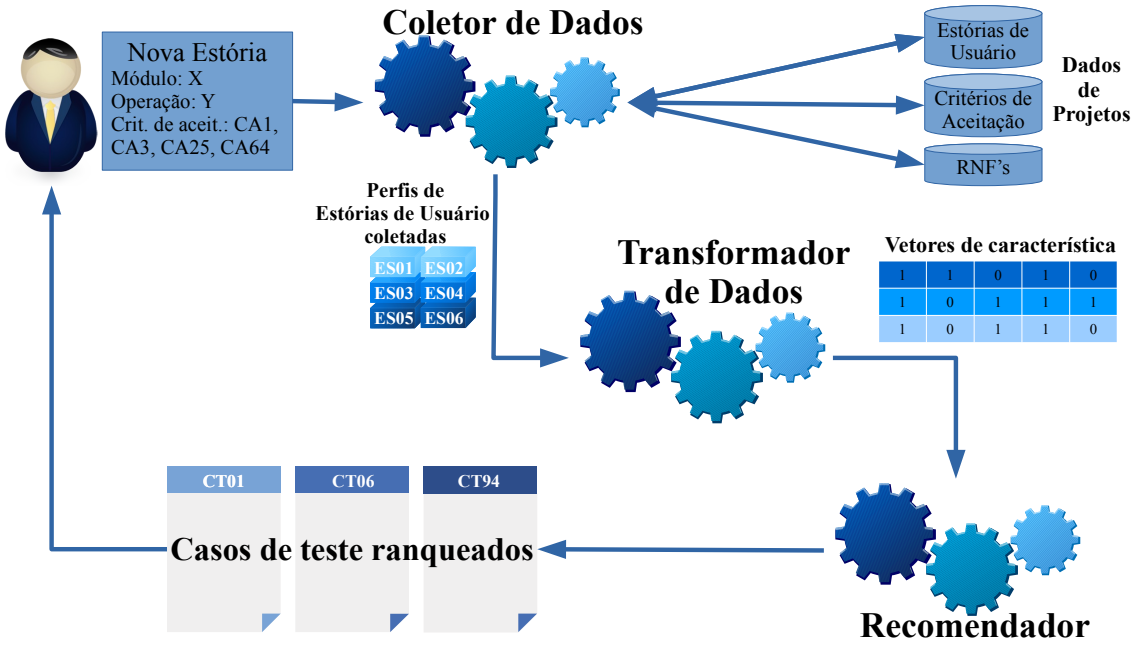

Figura 4.6: Componentes utilizados para a recomendação de casos de teste a partir de uma nova Estória de Usuário.

4.5.2 Coletor de Dados

Conforme descrito na seção anterior, a responsabilidade do componente coletor de dados é buscar informações necessárias para a recomendação vindas da base de dados da empresa, contendo dados de projetos anteriores e em andamento, tendo como entrada dados da nova Estória de Usuário cujo registro ou atualização se encontra em andamento. As informações selecionadas são o que chamamos de *Estórias de Usuário Semi-Estruturadas*, ou seja, as estórias contendo atributos relevantes para definição de casos de teste, critérios de aceitação e os requisitos não funcionais associados ao projeto.

Durante o projeto de *Design Science* surgiu a necessidade de identificarmos quais dados seriam necessários para montagem de um vetor de características que representasse os atributos necessários à definição e reutilização de casos de teste em projetos ágeis. Para definição dos dados necessários a serem recuperados dos projetos, realizamos primeiramente uma busca inicial nos principais mecanismos de busca acadêmicos como Google Scholar [6], Scopus [7], and IEEExplore [8].

Contudo, os resultados atingidos não nos revelaram informações suficientes para proceder com os próximos passos do projeto de *design science*, nos induzindo a executar um *Survey* [107] com projetos reais para coletar dados de praticantes e maximizar a credibilidade dos dados coletados. Adicionalmente, optamos por selecionar e entrevistar testadores devido à dificuldade em atingir um número estatisticamente relevante de respostas através do uso de questionários online.

No *Survey*, realizamos entrevistas semi-estruturadas [87], utilizando o questionário apresentado no Apêndice A, em dois institutos de pesquisa e desenvolvimento que executam projetos de software em parceria com a indústria, sendo a primeira a organização *X* descrita na Seção 4.1.1 e a segunda, o ASSERT [9], que por questões de conveniência, vamos chamar de *Y*. Em relação à organização *X*, a organização *Y* é menor, executando 6 projetos e contando com pouco mais de 40 colaboradores. A logística dos times de testes nas organizações são bastante distintas. Orgnização *X* adota o modelo de equipes independentes de testes com testadores integrados à equipe de desenvolvimento e tendo as atividades definidas

[6] https://scholar.google.com
[7] https://www.scopus.com/
[8] https://ieeexplore.ieee.org/
[9] http://assert.ifpb.edu.br/

dentro da própria equipe, enquanto a organização *Y* adota o modelo de uma equipe independente de testes separada da equipe de desenvolvimento, atendendo demandas das equipes de desenvolvimento [25].

O foco de ambas as organizações é a execução de projetos de inovação, contando com vários projetos que demandam esforços temporários e objetivos pré-definidos para criar novos produtos, serviços ou processos. O foco para este trabalho foi nos projetos de software para a *Web*.

Nelas, selecionamos respondentes e identificamos o tempo de experiência de cada um na parte de testes ágeis e no desenvolvimento de software como um todo, descritos na Tabela 4.4. As principais questões levantadas na pesquisa refletiram quais atributos os profissionais de testes em projetos ágeis consideravam mais importantes para o desenvolvimento de casos de teste. Além disso, questionamos também se os profissionais costumam reutilizar casos de teste e, em caso positivo, quais atributos eles levam em consideração para tal atividade. As entrevistas foram gravadas e para análise dos dados utilizamos a abordagem *Análise Temática* [28].

Tabela 4.4: Perfil dos testadores entrevistados.

Entrevistado	Empresa	Exp. Projetos	Exp. Teste Ágeis	Escolaridade
A	X	6	6	Graduado
B	X	3	1	Graduado
C	X	6	4	Graduado
D	Y	5	5	Especialista
E	Y	6	4	Graduado
F	Y	7	6	Graduado

Após coleta dos dados, fizemos o processo de codificação seguindo a abordagem definida por Cruzes et al. [27], e, a partir da análise dos códigos, identificamos a ordem de prioridade dos atributos na construção e reuso de casos de teste ágeis. Em caso de divergências na definição dos códigos, membros do grupo de pesquisa eram acionados para discussão, assim, concluindo o entendimento. Para definir essa ordem, utilizamos, tanto no mapeamento sistemático como no estudo de caso, o critério da quantidade de vezes que o tema foi encontrado. Quanto maior o número de vezes em que um tema foi encontrado, consideramos

maior sua importância. Na Figura 4.7 compilamos os atributos identificados como mais e menos importantes no desenvolvimento e reuso de casos de teste ágeis respectivamente.

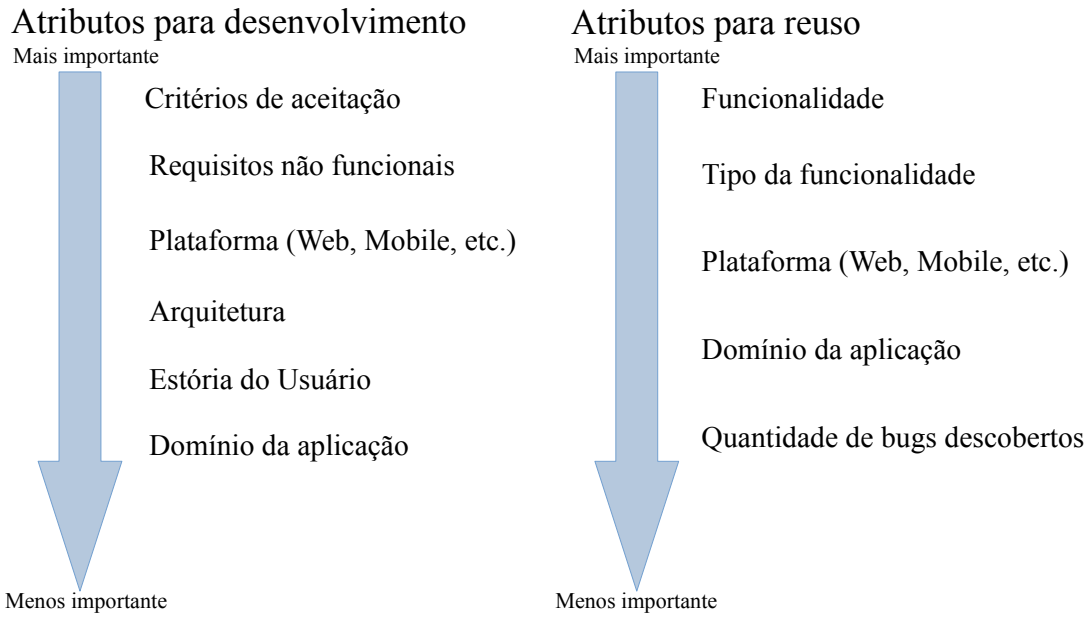

Figura 4.7: Atributos para necessários para desenvolvimento e reuso e casos de teste.

É importante observar que durante a codificação das entrevistas foram levantados outros atributos que não foram incluídos nos itens finais, pelo fato de não serem atributos padrão de metodologias ágeis, como por exemplo *"casos de uso"* ou ainda *"protótipos de tela"*. Para o desenvolvimento de casos de teste, os entrevistados enfatizaram a importância dos critérios de aceitação, dado que nas metodologias ágeis, os requisitos do sistema são definidos na forma da descrição das Estórias do Usuário em conjunto com os critérios de aceitação [78]. Ao final, concluímos que os critérios de aceitação, Requisitos não funcionas, plataforma, domínio da aplicação são os atributos a serem considerados neste trabalho para a reutilização de casos de teste a partir das Estórias de Usuário.

Muitos atributos são comuns entre os necessários para o reuso e desenvolvimento de casos de teste, e isso foi levado em consideração no momento de definir quais os dados necessários para reuso de casos de teste e, consequentemente, para montagem de vetores de características para recomendação de casos de teste. Na seção seguinte, descrevemos como é definida a montagem de vetores de características com base nos dados levantados resultantes da pesquisa apresentada.

4.5.3 Transformador de Dados

Uma vez que os dados de Estórias de Usuário necessários foram definidos, ao qual chamamos de *Perfil de Estórias de Usuários*, o transformador de dados tem a responsabilidade de receber os perfis retornados pelo coletor de dados como entrada e gerar vetores de características para elas em relação à *Estória Alvo*, ou seja, a Estória de Usuário para a qual o SisRec irá fornecer as recomendações, com o objetivo de permitir a definição de proximidade entre Estórias, processo que será descrito em mais detalhes na seção seguinte.

Na Tabela 4.5, podemos ver exemplos de perfis de Estórias de Usuário Semi-Estruturada contendo seus dados demográficos, considerando o tipo da Estória de acordo com a taxonomia definida na Seção 4.3 com os atributos **módulo** e **operação**, além dos atributos definidos a partir da pesquisa apresentada na Seção 4.5.2. Como exemplo, a linha cujo ID é e_1 possui módulo *"Cadastro"* e operação *"modificar_inserção_de_dados"* e é uma Estória referente ao desenvolvimento para plataforma *Web* – existem projetos cujo objetivo é o desenvolvimento de funcionalidades para várias plataformas – e o domínio da aplicação é o de suporte ao usuário.

Tabela 4.5: Exemplos de perfis de Estórias de Usuário.

ID	Módulo	Operação	Plataforma	Domínio	CA's	RNF's
e_1	Cadastro	modificar_inserção _de_dados	Web	Suporte	8|9|10|11|12| 13|14	1|2
e_2	Cadastro	inserir_dados	Web	Fábrica	1|2|3|6|43	1|2
e_3	Cadastro	recuperar_dados	Web	Fábrica	15|17|20|21|57	1|2
e_4	Autenticação	fazer_login_com _usuário_e_senha	Web	Fábrica	25	1|2
e_5	Cadastro	remover_dados	Web	Gestor	49|50	1|2

É importante notar que as colunas Critérios de Aceitação (CA's) e Requisitos não funcionais (RNF's) são multi-valoradas. Isso se deve ao fato de uma Estória de Usuário poder ter vários CA's associados, assim como vários RNF's. Contudo, os RNF's não necessariamente são associados às Estórias dos Usuários, mas podem vir de outras fontes de dados, como o projeto, a definição de pronto, ou ainda ter uma própria Estória que define o requisito [80].

Nos exemplos descritos na tabela, é possível vermos que os identificadores da coluna multi-valorada de critérios de aceitação refletem os definidos durante análise dos critérios de projetos em andamento descritos na Seção 4.4.1. Os requisitos não funcionais também foram padronizados para possibilitar a identificação dos mesmos definidos durante o projeto seguindo a definição presente em [79] . Na tabela, os identificadores 1 e 2 representam os RNF's de *"Segurança"*, referente aos atributos *"Autenticação"* e *"Autorização"*, respectivamente.

As Estórias de Usuário são recuperadas da base de dados pelo **Coletor de dados** e em seguida, utiliza-se a *Estória alvo* para transformação dos dados retornados em vetores de características semelhante ao apresentado na Tabela 4.6. Na tabela, é possível verificar que definimos uma coluna para cada critério de aceitação associado à estória alvo (CA1, CA2) e também para requisitos não funcionais (RNF1, RNF2). Se consideramos a estória alvo a estória e_5 da Tabela 4.5, teríamos para o critério *CA1* o de identificador 49 e para o *CA2* o de identificador 50, além dos requisitos não funcionais *RNF1* sendo o de identificador 1 e o *RNF2* sendo o de identificador 2.

Tabela 4.6: Exemplos de vetores de características gerados com base nas características extraídas de uma Estória alvo e_a, sendo a estória e_5 da Tabela 4.5.

	Módulo	Operação	Plataforma	Domínio	CA1	CA2	RNF1	RNF2
e_a	1	1	1	1	1	1	1	1
e'_1	1	0	1	0	0	0	1	1
e'_2	1	0	1	0	0	0	1	1
e'_3	1	0	1	0	0	0	1	1
e'_4	0	0	1	0	0	0	1	1

Assim, os vetores de características são definidos através da comparação dos dados do perfil de Estória do Usuário entre a estória alvo e as estórias selecionadas utilizando a técnica *one-hot encoding* [18], significando que os valores das estórias selecionadas iguais ao da *Estória Alvo* recebem o valor 1, e os valores das estórias selecionadas diferentes da *Estória Alvo* recebem o valor 0. Com isso, fazemos com que as estórias selecionadas que possuem uma maior quantidade de dados do perfil igual ao da *Estória Alvo* apresente uma proximidade maior e, assim, tenha seus casos de teste recomendados.

4.5.4 Recomendador

A geração das recomendações de casos de teste é feita pelo componente **Recomendador**. Neste trabalho, utilizamos a filtragem baseada em conteúdo utilizando dados demográficos dos usuários [46], seguindo os requisitos definidos na seção anterior. Nessa abordagem, consideramos a recomendação de casos de teste relacionados aos vizinhos mais próximos da Estória de Usuário sendo cadastrada ou selecionada para execução na *Sprint*.

A proximidade entre estórias é dada pela semelhança de suas características, apresentadas na Tabela 4.5. Essa abordagem foi adotada pelo fato das novas Estórias de Usuário não terem nenhum teste relacionado no momento de sua criação ou de sua seleção para *Sprint*, problema conhecido como *User Cold-Start* [89]. Portanto, não há a possibilidade da utilização de filtragem colaborativa baseada em itens (casos de teste). O processo de recomendação é apresentado na Figura 4.8.

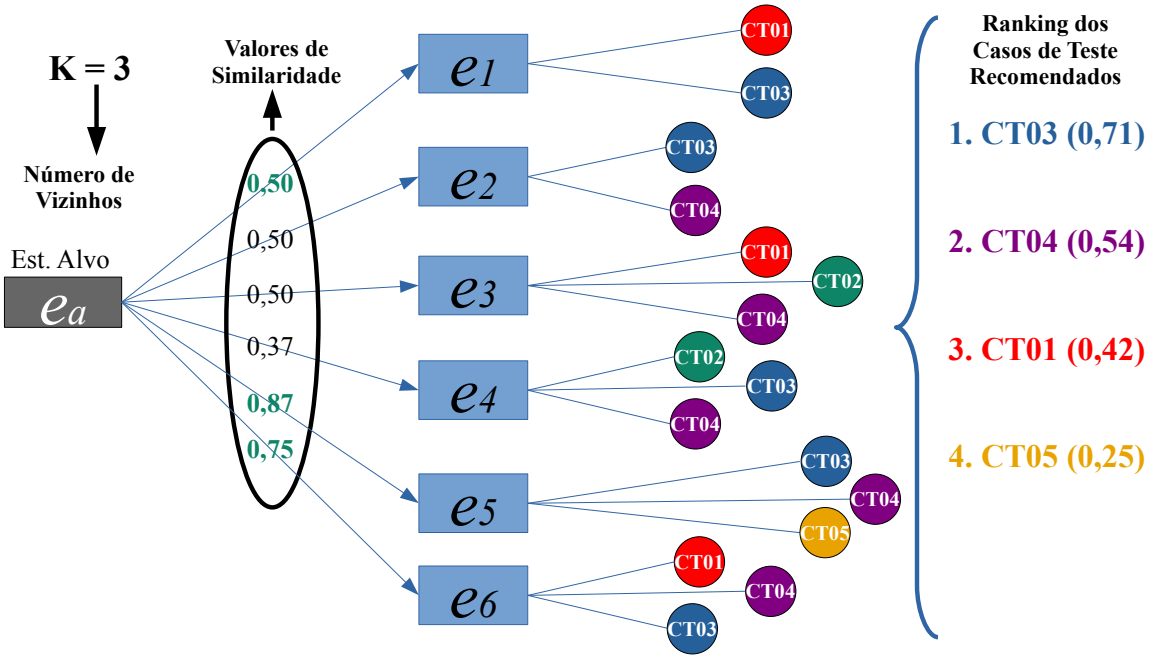

Figura 4.8: Cálculo dos $K = 3$ vizinhos mais próximos para recomendação de casos de teste associados a Estórias de Usuário.

O papel do recomendador é executar o cálculo dos K vizinhos mais próximos, utilizando a abordagem *K Nearest Neighbors* (KNN) [31]. Para fins de cálculo, utilizaremos vários algoritmos, para comparação de desempenho. Dentre eles podemos citar a distância do *Cosseno* [4], *Euclidiana* [3], *Manhattan* [3], *Jaccard* [6], *Canberra* [55] e *Chebyshev* [55].

Dado o cálculo das distâncias entre a Estória do Usuário alvo e_a e as Estórias retornadas pelo coletor de dados e'_i, os valores de similaridade [59] são calculados de acordo com a Equação 4.2, normalizando os resultados da distância em um intervalo fechado de $[0, 1]$. As *K* estórias com maior valor de similaridade – portanto mais similares – têm seus casos de teste selecionados para recomendação. No exemplo apresentado na figura, supondo o cálculo da distância e similaridade hipotética, vemos que o *K* definido tem o valor 3, ou seja, as três estórias mais similares, e_1, e_5 e e_6, com valores de similaridade $0{,}50$, $0{,}87$ e $0{,}75$, respectivamente, têm seus casos de teste selecionados.

$$sim(e_a, e') = 1 - \frac{\sum_{i=1}^{n} |e_{ai} - e'_i|}{n}. \tag{4.2}$$

Em seguida é feito o ranqueamento dos casos de teste relacionados às Estórias de Usuário selecionadas c_{ij} para recomendação. O critério utilizado é a quantidade de vezes que um caso de teste aparece em relação às estórias selecionadas descritas na Equação 4.3, além da utilização da similaridade como peso na avaliação do ranqueamento. No exemplo da figura, vemos que o caso de teste *CT03* aparece nas três estórias de usuário selecionadas (com *K* igual a 3, e_1, e_5 e e_6) e com valor de ranqueamento $0{,}71$, exemplificado na Equação 4.4. Por isso, é a primeira recomendação da lista, seguido do caso de teste *CT04* que aparece nas Estórias e_5 e e_6 tem o valor de ranqueamento $0{,}54$, e o caso de teste *CT01* que é relacionado apenas com a estória e_6 com valor de ranqueamento $0{,}42$.

$$rat(ct_j) = \frac{\sum_{i=1}^{n} sim(e_a, e'_i)}{K}, \tag{4.3}$$

$$rat(ct_{03}) = \frac{0,50 + 0,87 + 0,75}{3} = 0,71. \tag{4.4}$$

Na Tabela 4.7 podemos visualizar exemplos de casos de teste hipotéticos retornados pelo recomendador, que foram criados para uma funcionalidade de cadastro de projetos em um projeto previamente executado. Nela, exibimos apenas o título e os passos para execução do caso de teste no formato Gherkin. Adicionalmente, outros atributos do caso de teste são recuperados na recomendação, como o propósito do caso de teste, se o mesmo é executado manualmente ou automaticamente, dentre outros. Porém, todos os dados desse caso de teste podem ser alterados, uma vez que ao reutilizar o caso de teste, uma cópia do mesmo será

feita. Essa definição tem como objetivo adequar a descrição do caso de teste ao contexto do projeto. Por exemplo, se os casos de teste descritos na tabela forem recomendados para uma Estória de Usuário cujo título é "Cadastrar Usuários", após reutilizá-lo, o usuário do recomendador poderá alterar sua descrição, ou qualquer outro atributo do caso de teste para adequar o mesmo a essa estória.

Tabela 4.7: Exemplos de casos de teste recomendados.

ID	Título	Passos
CT01	Cadastrar projeto com sucesso	**Dado** que estou logado no sistema **Quando** clico no menu "Projetos" **E** clico no botão "Novo" **E** preencho os campos obrigatórios **E** clico no botão "Salvar" **Então** a mensagem "registro cadastrado com sucesso" é exibida
CT03	Cadastrar projeto com campos obrigatórios não preenchidos	**Dado** que estou logado no sistema **Quando** clico no menu "Projetos" **E** clico no botão "Novo" **E** não preencho os campos obrigatórios **E** clico no botão "Salvar" **Então** a mensagem "campo obrigatório" é exibida

4.5.5 Heurística para Seleção de Vizinhos mais Próximos

O exemplo da Figura 4.8 nos mostrou o processo de seleção de vizinhos mais próximos a partir da similaridade entre a estória de usuário alvo e as estórias armazenadas na base de dados do sistema de recomendação. Contudo, ao observar recomendações, foi identificado que, em caso de empate na similaridade de duas ou mais estórias de usuário, o algoritmo padrão de seleção dos vizinhos mais próximos elege aleatoriamente as estórias mais similares.

A Figura 4.9 demonstra o mesmo exemplo hipotético exibido na figura 4.8. Nela, podemos observar que as estórias e_1, e_2 e e_3 possuem o mesmo valor de similaridade. No

algoritmo padrão de seleção de vizinhos mais próximos as estórias selecionadas para $K = 3$ foram e_1, e_5 e e_6. Dentre as que possuem a mesma similaridade, e_1 foi escolhida pelo fato de ser a primeira identificada. Então, decidimos adicionar uma heurística que considera a quantidade de casos de teste associados a uma Estória de Usuário como critério de desempate. Essa heurística vem do fato de que queremos identificar se uma Estória de Usuário que possui mais casos de teste associados, e que consequentemente fornece mais casos de teste para recomendação, melhora o desempenho do SisRec.

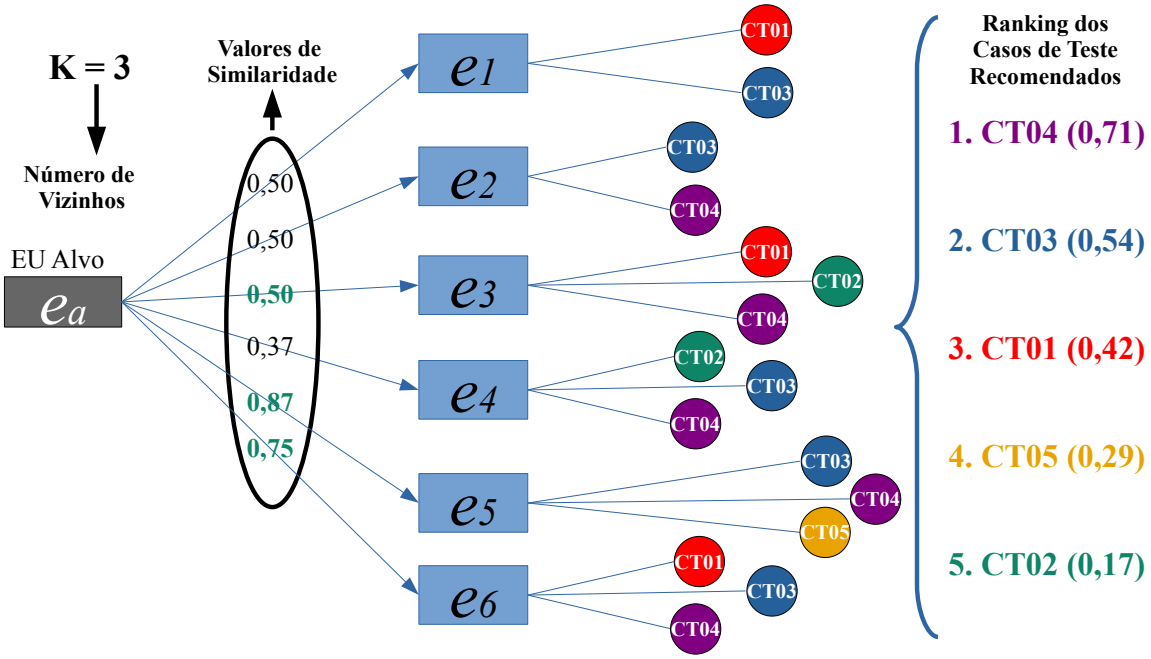

Figura 4.9: Cálculo dos K vizinhos mais próximos para recomendação de casos de teste associados a Estórias de Usuário utilizando heurística.

Assim, no exemplo hipotético na Figura 4.9 com a utilização dessa heurística e o mesmo K, as estórias selecionadas são as estórias e_5, que possui maior similaridade $(0,87)$, e_6, com a segunda maior similaridade $(0,75)$ e e_3, que possui a terceira maior similaridade $(0,50)$, porém, uma maior quantidade de casos de teste associados em relação às outras estórias com a mesma similaridade. Ainda utilizando o exemplo da figura, vemos que o mesmo cálculo definido na seção anterior é utilizado para o ranqueamento dos casos de teste selecionados. Dado que, utilizando essa heurística, selecionaremos estórias com maior quantidade de casos de teste, existe uma indicação de que ela possa melhorar o desempenho do algoritmo de recomendação. Para tanto, essa possibilidade foi investigada durante o experimento *off-line*

da solução, descrito no próximo capítulo.

4.6 Considerações Finais do Capítulo

Neste capítulo apresentamos um Sistema de Apoio à Decisão baseado em Sistemas de Recomendação para auxiliar na definição de casos de teste em projetos de software baseados em *Scrum*. A operacionalização da solução proposta é possibilitada por uma instrumentação do *Scrum*, que consiste em adicionar elementos à estrutura original do arcabouço. A instrumentação tem como principal objetivo possibilitar a coleta de informações que serão utilizadas na geração das recomendações, e baseia-se na adição de dois elementos ao arcabouço original de *Scrum*, a seleção dos critérios de aceitação padrão e a classificação das Estórias de Usuário de acordo com a taxonomia definida.

Além disso, uma primeira versão do sistema de recomendação de casos de teste foi desenvolvida com base em uma abordagem colaborativa baseada em vizinhança. Para a geração das recomendações, uma base de dados contendo informações reais de projetos de software foi coletada com o auxílio de 12 colaboradores, que forneceram dados de projetos *Scrum* com foco na associação entre Estórias de Usuário e casos de teste. Para finalizar, a definição da abordagem nos deu a possibilidade de criar uma ferramenta para avaliar sua viabilidade, e assim garantir que a integração da solução é possível e traz os benefícios esperados.

Capítulo 5

Validação Off-line do Sistema de Recomendação

Neste capítulo, apresentamos os passos executados para validação do artefato gerado para resolução do problema de pesquisa. Nele, descrevemos uma visão do protocolo definido para realização de um experimento para validação *off-line* da solução e em seguida apresentamos os resultados obtidos com esse experimento. Posteriormente, analisaremos os resultados e identificaremos as conclusões sobre a viabilidade de utilização do artefato e definimos a melhor configuração para utilização do sistema de recomendação. Finalmente, demonstraremos uma ferramenta desenvolvida com base na configuração encontrada para atestar a viabilidade da solução.

5.1 Avaliação do Sistema de Recomendação

Seguindo os passos da metodologia *Design Science*, após a definição da solução para resolver o problema de pesquisa, a próxima atividade é validar a solução definida, com o propósito de identificar a viabilidade de sua futura implantação em ambiente real. Em sistemas de recomendação, a avaliação antes de sua entrega é preferencialmente feita de forma *off-line*, ou seja, utilizando conjuntos de dados conhecidos em ambiente isolado do ambiente real [35].

Avaliações *off-line* são mais fáceis de conduzir e requerem menos custos, dado que não envolvem a participação de usuários reais, além de permitir a avaliação de grandes quantidades de algoritmos, suas configurações e traz indícios de como o desempenho de cada um se

reflete na base de dados utilizada [35]. Contudo, avaliações *off-line* refletem apenas dados de usuários do passado, que podem não ser condizentes com os dados de uso futuro dos usuários durante a implantação.

O principal objetivo da avaliação *off-line* é descartar algoritmos e configurações com baixo desempenho ou acurácia, deixando apenas um conjunto pequeno de soluções candidatas à execução de testes com usuários reais, que requerem um maior custo para execução. Um protocolo comum para avaliação é, a partir da base de dados conhecida, determinar uma quantidade de itens para treinamento – chamado de dados para treinamento – e comparar a saída do algoritmo com a parte da base não utilizada para treinamento – também chamada de dados de teste – considerando que em um ambiente real as escolhas feitas pelos usuários da base de teste seriam as escolhas feitas por usuários no sistema real [42].

Neste sentido, uma abordagem popular para a avaliação de sistemas de recomendação é a validação cruzada (em inglês, *cross-validation*), onde os dados de treinamento e testes são divididos em partições variando em diversas iterações e fazendo assim com que essas variações evitem a alta especialização do algoritmo em uma base de testes fixa. Uma especialização dessa abordagem é a validação cruzada *K-fold*, em que a base de dados total é divida em *K* partições ou pastas (do inglês, *fold*) e *K-1* delas são utilizadas para treinamento do algoritmo, enquanto que a partição restante é utilizada para teste das recomendações [85]. Podemos verificar o processo de manipulação dos dados conhecidos na validação cruzada *K-fold* na Figura 5.1.

Como podemos observar na figura, o processo de validação cruzada *K-fold* nos fornece *K* resultados que podem ser agregados de várias formas, como a média, ou outras análises estatísticas mais avançadas. Neste trabalho, optamos por avaliar a recomendação de casos de teste utilizando a validação cruzada *10-fold*, pois nos permite reduzir o viés dos dados de treinamento, pela separação em *folds*. De acordo com Refaeilzadeh [85], a repetição de testes com dados para validação e treino independentes promove a viabilidade estatística para medição do desempenho do algoritmo em avaliação. Com *K* igual a *10*, possibilitamos uma massa de dados para treino (*90%*) muito similar à massa de dados total, enquanto diminuímos a quantidade de sobreposições entre as massas de dados para treinamento. Com *K* igual a *10* cada massa de dados para treino compartilha *8/9* de seus valores com outras 9 massas de treino. O aumento do *K* minimiza o tamanho da massa para teste, diminuindo a precisão na

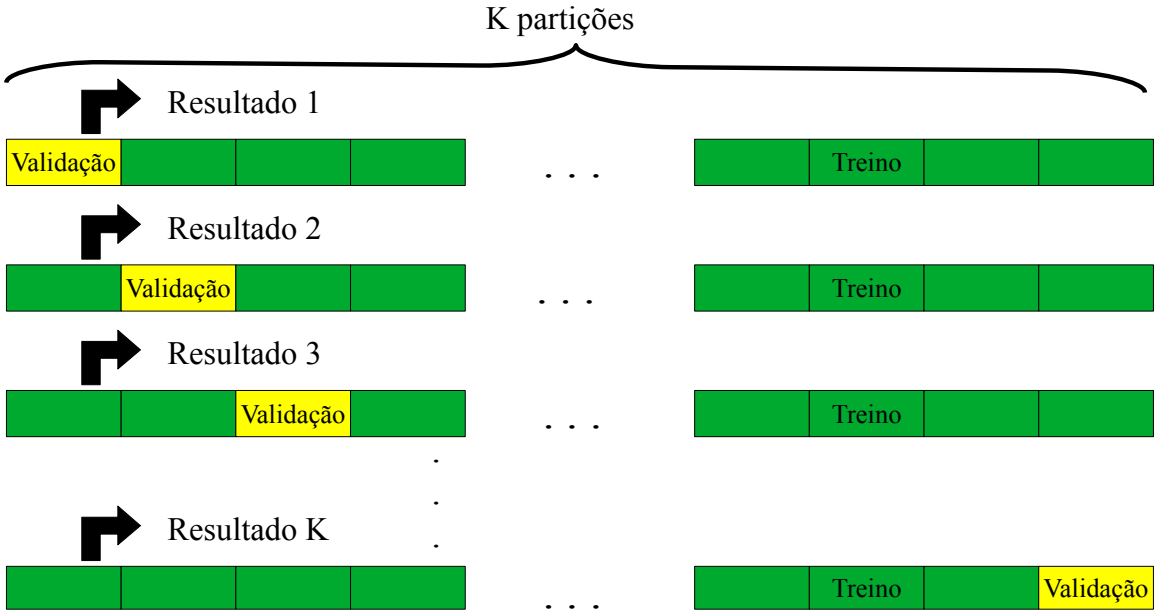

Figura 5.1: Manipulação da base de dados na validação cruzada K-fold.

medição do desempenho do algoritmo.

Portanto, o experimento realizado neste trabalho tem as seguintes definições:

- **Protocolo:** Configuração experimental *off-line*.
- **Métricas avaliadas:** Na avaliação preliminar coletamos *precisão*, *recall* e *F-meaure(β)*. Essas métricas serão detalhadas nas seções que seguem.

5.2 Definição da Base de Dados

Conforme descrito na seção anterior, para viabilizar o experimento e executar a validação cruzada se faz necessário utilizar uma base de dados o mais próximo possível da base de dados real. Contudo, iniciamos o sistema de recomendação a partir da instrumentação do *Scrum* descrita no capítulo anterior, além do mesmo depender das Estórias de Usuário Semi-Estruturadas e a padronização de critérios de aceitação e requisitos não funcionais. Isso nos trouxe a falta de dados instrumentados que permitisse o experimento. Além disso, os dados em bases reais como Estórias de Usuário, critérios de aceitação e casos de teste não são unificados, dificultando assim sua comparação e recomendação.

O problema da falta de dados de usuários e de itens para possibilitar a recomendação é chamada de *User Cold-Start* e *Item Cold-Start*, respectivamente [89]. Conforme descrito na seção anterior, temos como usuários do sistema de recomendação as Estórias de Usuário que terão itens recomendados no formato de casos de teste. É importante frisar que os sistemas de recomendação comuns utilizam *ratings* (notas) para, a partir deles, decidir quais itens devem ser apresentados aos usuários [105]. No problema descrito neste trabalho, para evitar uma mudança mais acentuada no processo de desenvolvimento, que incluiria classificar um caso teste após a sua reutilização, utilizamos apenas a associação de um caso de teste a uma Estória de Usuário para executar a recomendação. Com isso, os usuários não necessitam definir notas para os casos de testes selecionados, além de proporcionar uma baixa escassez de dados, como acontece no caso dos *ratings* de filmes, dado que muitos usuários não dão notas para filmes [81] e consequentemente reduz o poder da recomendação.

Assim, para resolver o problema do *Item Cold-Start* no presente trabalho, coletamos dados de Estórias de Usuário, critérios de aceitação e casos de teste da base de dados de duas organizações que utilizam o *Scrum*. Ao todo, coletamos 217 Estórias de Usuário de 13 projetos, contando com atividade nas plataformas *Mobile*, *Web* e *Desktop*. Dentre estes, a maior parte é relativa a platamforma *Web*, enquanto que foram coletados dados de uma grande quantidade de projetos que possuiam várias plataformas, como *Mobile* e *Web* simultaneamente. Foram coletados também 530 critérios de aceitação e 1077 casos de teste distintos. Todas as Estórias de Usuário foram manualmente classificadas de acordo com a taxonomia definida na Seção 4.3, assim como os critérios de aceitação (Tabela 4.3) e casos de teste foram associados aos casos de teste padrão definidos para utilização e recomendação.

A dinâmica do processo de criação de casos de teste favorece a recomendação de novos casos de teste criados pelo fato dos mesmos serem sempre associados a uma estória de usuário. O *User Cold-Start*, que no presente trabalho reflete a criação de novas estórias de usuário, é tratado através da recomendação de itens mais populares entre as estórias de usuários semelhantes [36], conforme descrito no capítulo anterior.

O processo de definição da base de dados contou com vários desafios. Dentre eles, podemos citar:

- **Dificuldade de classificação das Estórias de Usuário:** Em muitos casos, a descrição da Estória de Usuário não indicava o propósito da mesma, o que dificultou a classifica-

ção. Para minimizar o problema, analisamos, além da descrição da estória, os critérios de aceitação, quando presente, ou outras informações disponíveis. As estórias em que não foi possível a identificação do propósito mesmo com a análise de outros artefatos foram descartadas.

- **Vários itens da taxonomia em apenas uma Estória de Usuário:** Durante a classificação das Estórias de Usuário, percebemos que algumas estórias poderiam ter várias classificações, seguindo a taxonomia definida. Como até o momento o processo não suporta mais de uma classificação, adotamos a abordagem de dividir estórias de acordo com sua classificação. Por exemplo, estórias que representassem atividades de inserção e busca de uma entidade seriam classificadas como módulo *Cadastro* e Operações *Inserir Dados* e *Recuperação de Dados*. Nesse caso, separamos as estórias em uma estória com operação *Inserir Dados* e outra com operação *Recuperação de Dados*.

- **Estórias de Usuário sem critérios de aceitação ou casos de teste definidos:** Dado que em algum momento as organizações analisadas não possuíam algum processo de análise de artefatos, foram encontradas Estórias de Usuário sem critérios de aceitação associados, ou ainda com poucos critérios de aceitação ou poucos casos de teste associados. Essa ausência impacta diretamente nas recomendações do sistema definido, uma vez que os critérios de aceitação são informações requeridas para montagem do vetor de características e os casos de teste são fundamentais para seleção dos itens a serem recomendados. Para minimizar o problema, analisamos os dados disponíveis das estórias, associamos critérios de aceitação padrão e definimos casos de teste que julgamos ser necessários. Em alguns casos foi possível o contato direto com o time responsável pela definição da estória e validar as associações feitas.

- **Dificuldade de identificação de critérios de aceitação ou do seu propósito:** Durante a busca dos critérios de aceitação associados às Estórias de Usuário, identificamos que a escrita dos critérios foi realizada de formas diferentes mesmo dentro da mesma organização, o que dificultou a localização do critério de aceitação. Em alguns casos, mesmo com a identificação do critério, sua descrição dificultou o entendimento do propósito, forçando a análise de outros artefatos relacionados às Estórias de Usuário para correta associação com os critérios de aceitação padrão ou definição de novos

critérios padrão.

- **Dificuldade no entendimento de casos de teste:** Em ambas as organizações, aparentemente o padrão de escrita de casos de teste varia entre projetos, e até mesmo dentro do mesmo projeto. Além disso, muitas descrições dos casos de teste não deixaram claro seu propósito, o que dificultou na associação com casos de teste padrão para treinamento do algoritmo. Para minimizar o impacto, analisamos outras informações disponíveis para o teste, além da Estória de Usuário associada. Em determinados casos, não foi possível adicionar o caso de teste à base.

- **Dificuldade na identificação dos RNF's:** O registro de requisitos não funcionais em ambas as organizações não é uma prática adotada. Além disso, as ferramentas utilizadas por ambas não dão suporte ao cadastro de RNF's, o que dificultou a identificação dos mesmos para definição dos vetores de características. Para minimizar o problema, outras informações referentes ao projeto ou Estórias de Usuário foram consultadas. Por exemplo, todos os projetos possuíam estórias de login e cadastro de autorizações para usuários, o que nos levou a identificar que todos possuem requisitos não funcionais de segurança com acesso apenas para quem está autenticado e autorizado no sistema. Outro fato favorável à identificação de RNF's foi que, na maioria dos projetos, os mesmos são definidos por projeto ao invés de por Estória de Usuário.

Seguindo o *Guideline* para avaliação de novos SisRecs [47], geramos também uma base de dados sintética. O propósito dessa base de dados é o de executar uma simulação da validação cruzada para avaliar o funcionamento do código gerado para validação, assim como o código do SisRec, permitindo assim a correção de *bugs* antes da validação principal. Para a base de dados sintética, registramos Estórias de Usuário para todas as categorias definidas na taxonomia definida com critérios de aceitação, requisitos não funcionais e casos de teste específicos.

5.3 Ferramentas

Para o desenvolvimento da solução, levamos em consideração a facilidade do desenvolvimento e as tendências atuais para as práticas de aprendizado de máquina. Assim, foram

selecionadas e utilizadas as seguintes ferramentas:

- **Python** [1]**:** Python é uma linguagem interpretada orientada a objetos, cujo primeiro *release* ocorreu em 1991 e é vastamente utilizada em aplicações com aprendizado de máquina, graças a sua interatividade de alto nível [75].

- **Scikit-learn** [2]**:** O scikit-learn é uma biblioteca Python utilizada em aplicações voltadas ao aprendizado de máquina. Ela fornece os principais algoritmos para seus usuários com uma interface que facilita sua utilização [75].

- **Google Colab** [3]**:** O Google Colab é um ambiente colaborativo para o desenvolvimento de aplicações Python na nuvem que não requer configuração para execução e fornece as principais bibliotecas para o desenvolvimento.

- **Google Sheets** [4]**:** O Google Sheets é uma ferramenta colaborativa para criação e manipulação de planilhas executada na nuvem. Ela é uma ferramenta grátis e parte da suíte de aplicações office do Google [5]. Essa ferramenta foi utilizada para registro da base de dados para treinamento do algoritmo de recomendações devido à facilidade de colaboração e customização dos dados, além de sua análise.

5.4 Avaliação Experimental

Uma vez que o artefato para resolução do problema foi desenvolvido e a base de dados foi preparada, realizamos uma avaliação experimental com o objetivo de validar o desempenho e analisar os resultados do processamento para recomendação de casos de teste através da validação cruzada, descrita na Seção 5.1. A avaliação tem dois objetivos principais: 1) identificar a melhor configuração para o sistema de recomendação; 2) Avaliar a qualidade dos resultados obtidos pelo algoritmo na validação cruzada e julgar a viabilidade de sua utilização em conjunto com a base de dados gerada.

[1] https://www.python.org/
[2] https://scikit-learn.org/stable/
[3] https://colab.research.google.com/
[4] https://docs.google.com/spreadsheets
[5] https://google.com

A melhor configuração do sistema de recomendação consiste em analisar as variáveis independentes do experimento, que neste trabalho consideramos o algoritmo para cálculo da distância entre a Estória de Usuário recuperada da base de dados e a estória alvo, o valor do *K* do algoritmo dos *K* vizinhos mais próximos, a base de dados utilizada para treinamento e a utilização ou não da heurística para seleção de vizinhos descrita na Seção 4.5.5. As variáveis independentes são detalhadas na Figura 5.2. Neste trabalho, consideramos o valor dos *K* vizinhos mais próximos em 10 níveis (*K = 1*, *K = 2*, *K = 3*, *K = 4*, *K = 5*, *K = 6*, *K = 7*, *K = 8*, *K = 9* e *K = 10*). Já os algoritmos para calcular distância entre as estórias selecionadas da base e a estória alvo foram descritos na Seção 4.5.4.

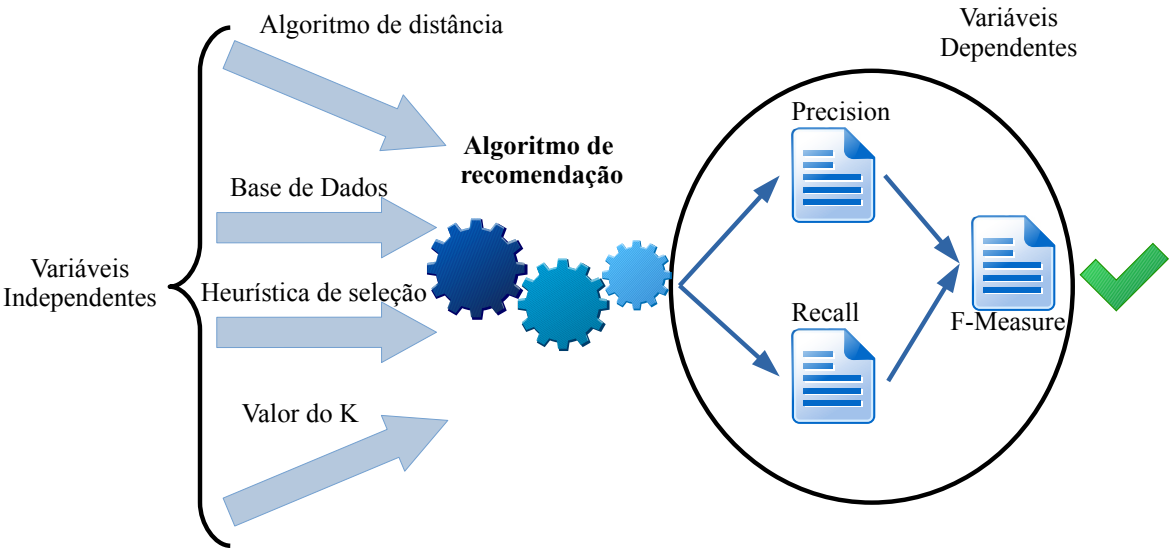

Figura 5.2: Definição de variáveis dependentes e independentes do experimento.

De acordo com Gunawardana [42], para avaliar a acurácia de um sistema de recomendação, é necessário classificá-lo de acordo com a tarefa de recomendação à qual ele é designado. Dentre as classificações, podemos destacar: **recomendar bons itens**, ou seja, recomendar itens que possivelmente o usuário possa utilizar; **otimizar a utilidade**, que tem como objetivo usar a recomendação para alcançar um objetivo, por exemplo, aumentar as vendas de uma loja de produtos; e por fim, **prever notas**, que visa prever a nota que um usuário daria para um item que ele ainda não consumiu, com o objetivo de recomendar esse item, caso a nota prevista atinja um limiar pré-determinado. Neste trabalho, consideramos

o objetivo do sistema a recomendação de bons itens, especificamente, **todos os bons itens**, o que significa que ainda que seja necessário exibir uma lista com muitos itens ao usuário, todos os itens que possivelmente devem ser relacionados à Estória do Usuário devem estar contidos nessa lista seguindo os requisitos definidos na Seção 4.5.1.

Por isso, na avaliação estamos interessados apenas em notas binárias, ou seja, ou o caso de teste foi selecionado para a Estória de Usuário (1) ou não (0). Assim, ao retornar uma lista de sugestões para o usuário, temos os seguintes possíveis resultados: **Falso-Positivo**, ou seja, um item que foi recomendado, mas não é aceito; **Verdadeiro-Positivo**, que é um item que foi recomendado e que é aceito; **Falso-Negativo**, correspondendo a um item que seria aceito mas não foi recomendado; e o **Verdadeiro-Negativo**, correspondendo a um item que não foi recomendado e, se fosse, não seria aceito [42]. Para fins de esclarecimento, consideramos um caso de teste aceito quando ele se encontra no conjunto de casos de teste definidos para uma determinada Estória de Usuário na base de treinamento definida.

Portanto, as métricas utilizadas para avaliação do algoritmo são *precision*, *recall* e *F-measure(β)*. A *precision* é o valor percentual das recomendações relevantes para usuário em relação à quantidade de recomendações retornadas descrita na Equação 5.1, ou seja, o valor da precisão mais perto de *1* significa um maior número de recomendações relevantes retornadas para o usuário. Uma vez que o requisito do sistema de recomendação é fornecer todos os bons itens, nesse caso todos os itens relevantes, a precisão não é a característica mais importante a ser considerada na avaliação, contudo, não deve ser totalmente ignorada.

$$precision = \frac{\mid \{recomendacoes\ \ relevantes\} \cap \{recomendacoes\ \ retornadas\} \mid}{\mid \{recomendacoes\ \ retornadas\} \mid} \quad (5.1)$$

O *recall* fornece o valor percentual da quantidade de recomendações relevantes fornecidas para o usuário em relação a todas as recomendações relevantes possíveis descritas na Equação 5.2, sendo as recomendações relevantes os casos de teste registrados na base de teste da validação. No contexto deste trabalho, quanto mais próximo de *1* for o *recall*, significa que o algoritmo está mais perto de recomendar todos os bons itens, ou todos os itens relevantes.

$$recall = \frac{\mid \{recomendacoes\ \ relevantes\} \cap \{recomendacoes\ \ retornadas\} \mid}{\mid \{recomendacoes\ \ relevantes\} \mid} \quad (5.2)$$

O *F-mesure* é uma medida de sumarização que calcula a média harmônica das duas métricas anteriores (*precision* e *recall*) descrito na Equação 5.3. Seu propósito é devido à relação inversa entre *precision* e *recall*, uma vez que listas de recomendação maiores melhoram o *recall* e que, quanto maior a lista de recomendações, maior a probabilidade dessa lista conter um número maior de itens relevantes ao usuário, assim como reduzem a precisão, uma vez que longas listas podem trazer muitos itens não relevantes.

$$\textit{F-measure} = 2 \cdot \frac{\textit{precisão} \cdot \textit{recall}}{\textit{precisão} + \textit{recall}}. \tag{5.3}$$

Dado que a validação executada é a validação cruzada *10-fold*, cada configuração – o valor do *K* e o algoritmo de cálculo da distância – trará um conjunto de resultados para cada configuração. Para a apresentação dos dados retornados no experimento, consideraremos o valor total da *precision* e *recall* para cada configuração a média de todos os resultados obtidos, e consequentemente o *F-measure* por questões de simplificação.

Uma vez que temos como requisito fornecer o maior número de recomendações certas possíveis mesmo cometendo um número considerável de erros, para o experimento, o *recall* deve ter uma maior importância em relação à *precision*. Por isso, ajustamos o cálculo do *F-Measure* para contemplar essa importância sem desconsiderar totalmente a *precision*, utilizando a definição de *Weighted F-Measure* [45, 70], descrito na Equação 5.4, que considera uma ênfase no *recall* caso o peso β seja maior do que *1* e na *precision*, caso o valor do peso β seja menor do *1*. O ajuste do peso é definido segundo experimento e avaliação de resultados.

$$\textit{F-measure}(\beta) = \frac{(\beta^2 + 1) \cdot \textit{precision} \cdot recall}{(\beta^2 \cdot \textit{precision}) + recall} \tag{5.4}$$

5.4.1 Objetivo e Hipóteses

O principal objetivo do experimento *off-line* é o de identificar qual a configuração mais eficaz para utilização do Sistema de Recomendação (SR) para casos de teste em projetos ágeis de acordo com o algoritmo utilizado e o número de vizinhos mais próximos (*K*), dada a base de dados utilizada para treinamento e a consideração da heurística de seleção de vizi-

nhos ou não, conforme descrito na Figura 5.2. De acordo com o método *GQM* (do inglês, *Goal/Question/Metric*) [9], o objetivo do experimento pode ser descrito da seguinte forma:

Analisar diferentes configurações do SR para casos de teste
para o propósito de sua avaliação
com respeito à sua eficácia com base na métrica *F-measure(β)*
do ponto de vista de um conjunto de dados históricos de projetos reais baseados em *Scrum*
no contexto de Projetos de desenvolvimento software ágeis.

Desta forma, pretende-se responder a seguinte questão de pesquisa:

RQ: A solução proposta é viável?

Para tanto, foram formuladas as seguintes questões de pesquisa secundárias:

RQ_1: As configurações definidas no experimento afetam os resultados obtidos no experimento?

RQ_2: Qual configuração apresenta melhor resultado de *F-measure(β)* na recomendação de casos de teste do ponto de vista do conjunto de dados histórico apresentado na Seção 5.2?

A fim de responder a questão de pesquisa secundária RQ_1, é necessário primeiramente identificar se as variáveis independentes realmente influenciam nos resultados das recomendações, e, para isso, devemos identificar se as configurações apresentam resultados estatisticamente diferentes. Por isso, formulamos as seguintes hipóteses:

H_{0-1}: não há diferença entre configurações distintas do SisRec de casos de teste em relação à métrica *F-measure(β)* das recomendações.

H_{1-1}: há diferença entre as configurações distintas do SisRec de casos de teste em relação à métrica *F-measure(β)* das recomendações.

5.4.2 Variáveis

O *design* de experimento aplicado considera quatro fatores de entrada (variáveis independentes) e uma variável resposta (variável dependente). As variáveis independentes são as seguintes:

- ***K***: quantidade de vizinhos considerada no algoritmo *K-NN* para a geração das recomendações de casos de teste, com 10 níveis, ou seja, K variando de 1 a 10;

- **Medida de similaridade**: medida de distância/similaridade usada para o cálculo de vizinhança com seis níveis, são eles: similaridade do *Cosseno* [4], distância *Euclidiana* [3], distância de *Manhattan* [3], distância de *Jaccard* [6], distância de *Canberra* [55] e distância de *Chebyshev* [55].

- **Heurística de seleção de vizinhos**: para avaliar a eficácia da heurística de seleção de vizinhos mais próximos descrita na Seção 4.5.5, deve ser feita a alternância entre o seu uso e não uso para comparação de dados coletados.

- **Base de dados**: A base de dados coletada em projetos de desenvolvimento ágil é considerada uma variável independente, pois sua alteração pode acarretar alterações nos resultados. Para esse estudo, esta variável não será alterada dado que a mesma será utilizada para recomendação de casos de teste a partir da definição da melhor configuração.

A variável resposta considerada no experimento é a seguinte:

- ***F-measure(β)***: média harmônica da *precision* e do *recall* das recomendações de casos de teste com β maior que *1* para enfatizar a importância do *recall* sobre a *precision*.

5.5 Resultados

Conforme descrito na seção anterior, antes da execução da validação cruzada *10-fold*, fizemos uma avaliação manual para identificar a coerência das recomendações aos olhos de um usuário do sistema utilizando uma base sintética gerada e a base real de avaliação. Em seguida, fizemos uma avaliação prévia do algoritmo utilizando essa base sintética a partir da geração de entradas aleatórias e observação das recomendações retornadas para identificar se estavam de acordo com o esperado. Para isso, geramos entradas simulando a criação de várias Estórias de Usuário alvo também variando de acordo com a taxonomia definida e com os critérios de aceitação padrão. O ponto positivo foi, ao contrário da validação cruzada, a utilização de toda a base de dados gerada para treinamento do algoritmo.

Dada a coerência nos resultados obtidos através da base sintética, executamos o mesmo procedimento agora com a base de dados real, o que nos permitiu realizar alguns ajustes no

algoritmo, como correção de *bugs* e melhorias, além de ajustes no procedimento de avaliação do mesmo. Também executamos uma validação cruzada "prévia", utilizando a base sintética, sendo possível assim, testar o código de automatização da própria validação cruzada, o que nos permitiu efetuar ajustes no mesmo.

5.5.1 Execução

A validação cruzada foi executada utilizando em todas as configurações inicialmente com $\beta = 2$ para posterior análise dos resultados, considerando a heurística de seleção dos vizinhos mais próximos e sem considerar a heurística de seleção, resultando em um total de 60 configurações. Durante as primeiras execuções e análises iniciais dos dados, identificamos que as métricas *Canberra*, *Euclidiana*, *Jaccard* e *Manhattan* apresentaram os mesmos valores para todas as métricas selecionadas. Essa semelhança de valores se dá pelo fato dessas distâncias serem derivadas da distância *Euclidiana* e os pontos cujas distâncias são calculadas serem multidimensionais com valores *0* ou *1* [5]. No exemplo abaixo, demonstramos o cálculo da distância entre as Estórias de Usuário e'_1 e e_a da Tabela 4.6.

$$\sum_{i=1}^{n}|x_i + y_i| = 8 \textit{ (Manhattan)}$$

$$\sqrt{\sum_{i=1}^{n}(x_i + y_i)^2} = 8 \textit{ (Euclidiana)}$$

Assim, com o objetivo de reduzir a quantidade de comparações entre algoritmos na análise estatística, consideramos apenas a distância Euclidiana. Na Figura 5.3 apresentamos os valores de *recalls* para os algoritmos de cálculo de distância *Euclidiana*, *Chebyshev* e *Cosseno* considerando o uso da heurística de seleção de vizinhos ou não grupos diferentes, enquanto que na Figura 5.4 e 5.5 apresentamos também as precisões e F-Measures(2), respectivamente. Os valores de *K* representam a média dos valores coletados para cada configuração e a distribuição de cada configuração possui um total de 185 registros.

Na Figura 5.3 vemos que os valores de *recall* crescem de maneira suave de acordo com o aumento da quantidade de vizinhos mais próximos selecionados. Isso pode ser explicado pelo fato de que quanto maior a quantidade de vizinhos, maior o número de casos de teste que serão selecionados para avaliação, aumentando assim o percentual de acertos. A utilização da

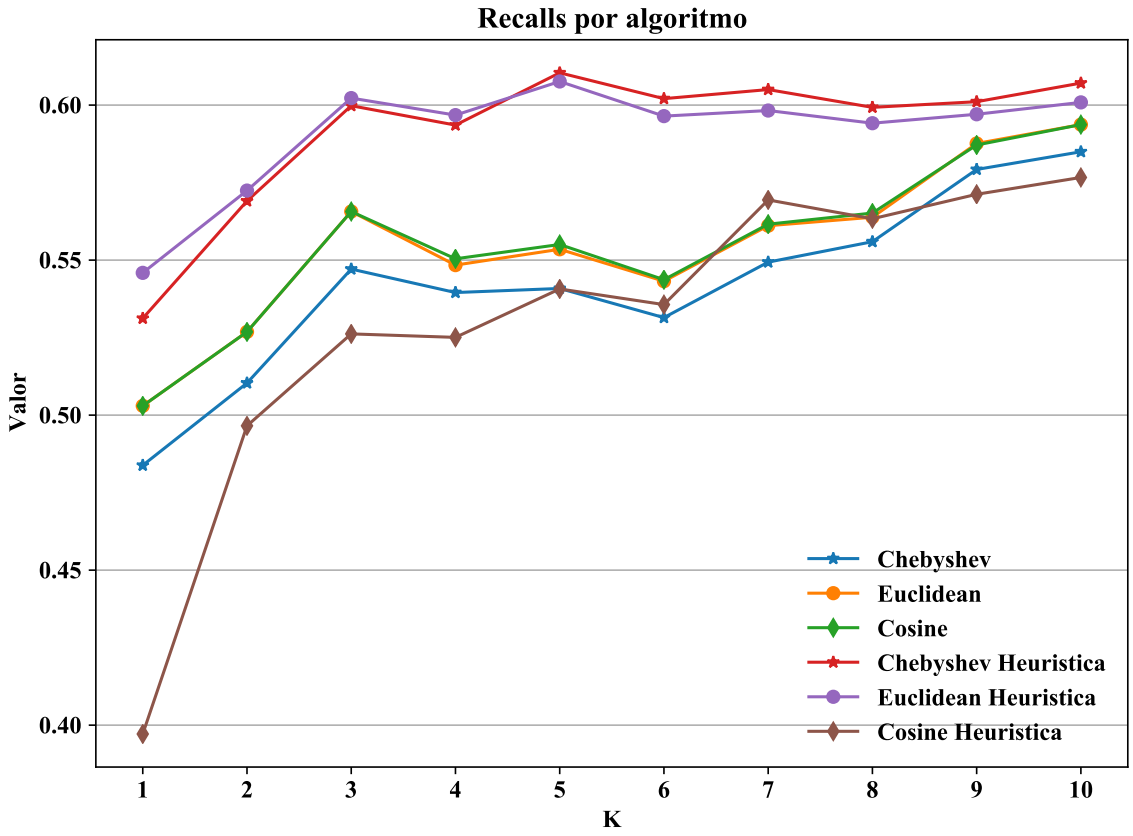

Figura 5.3: Distribuição de *recalls* entre as configurações selecionadas para avaliação.

heurística de seleção de vizinhos teve um efeito positivo no *recall* nos algoritmos *Euclidiano* e *Chebyshev*, porém, o *Cosseno* apresentou melhores resultados de *recall* não considerando a heurística.

Já nos valores de *precision*, descritos na Figura 5.4, é possível perceber uma queda acentuada de todos os algoritmos em relação ao aumento dos K vizinhos mais próximos. O motivo aparente é o significativo aumento do número de casos de teste selecionados. Por exemplo, considerando que o K seja 10 e que cada vizinho (Estória de Usuário) possua 5 casos de teste distintos, teríamos um total de 100 casos de teste selecionados o que leva a quantidade destes a ser muito maior do que a quantidade de recomendações relevantes (Verdadeiro-Positivo). Ainda é possível percebermos que os algoritmos que utilizam a heurística de seleção de vizinhos obtiveram resultados de *precision* inferiores aos que não utilizaram a heurística.

Finalmente, temos o gráfico resultante da *F-Measure(2)*, descrito na Equação 5.4 e apre-

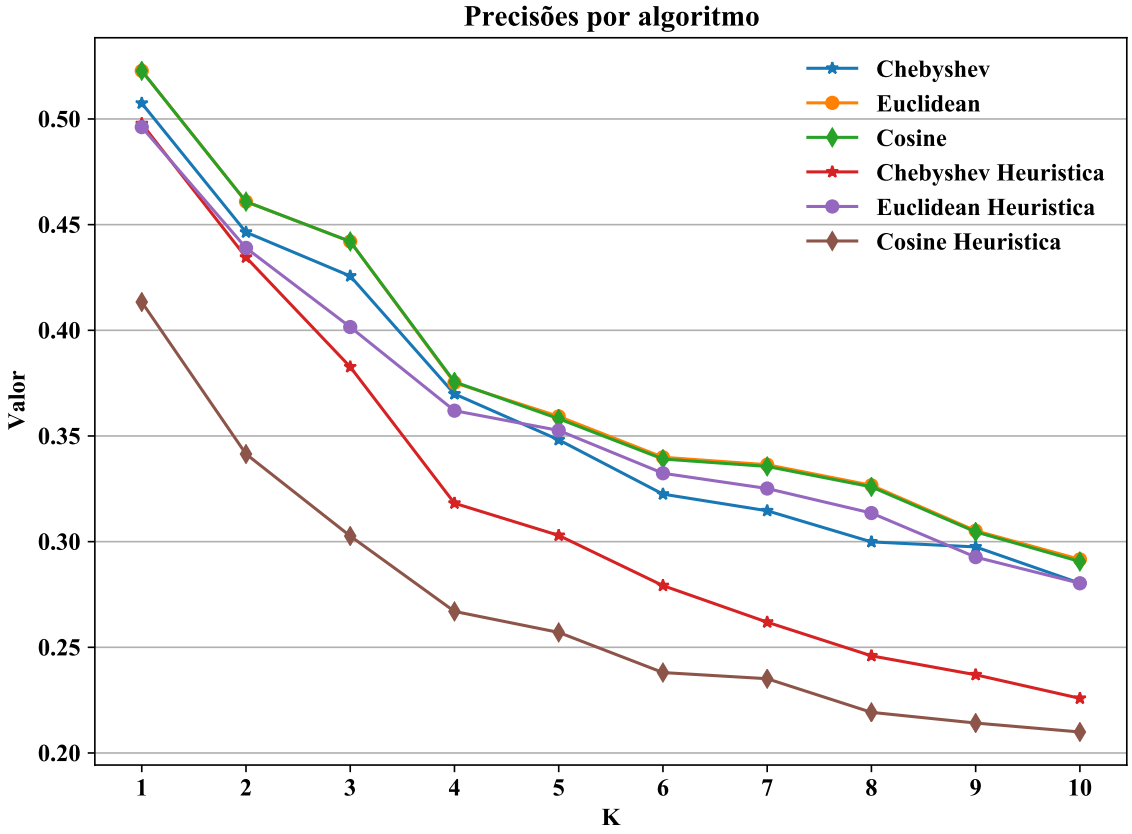

Figura 5.4: Distribuição de precisões entre as configurações selecionadas para avaliação.

sentado na Figura 5.5. Nele podemos observar que para quantidade de vizinhos menor ele possui valores mais altos. Isso é explicado pelo fato de que, com menos vizinhos temos valores de precisão mais altos e com mais vizinhos, apesar termos um pequeno aumento no *recall*, a precisão tem uma redução significativa, indicando que, nesse caso, temos um pequeno benefício em relação ao *recall*, porém, uma grande perda em relação à precisão, fato que pode comprometer a utilização da ferramenta.

Podemos observar nas distribuições de dados de *precision* e *recall* que os melhores resultados ainda não se aproximam do valor máximo *1*. Esse fato pode ser atribuído à dificuldade de encontrar dados próximos do ideal nas bases de dados das organizações avaliadas. Acreditamos que o uso da ferramenta de recomendação somado à retro-alimentação da base de dados com informações mais precisas gera a tendência de aumento das métricas e consequente melhoria das recomendações, fato que acrescenta motivação para sua criação e utilização.

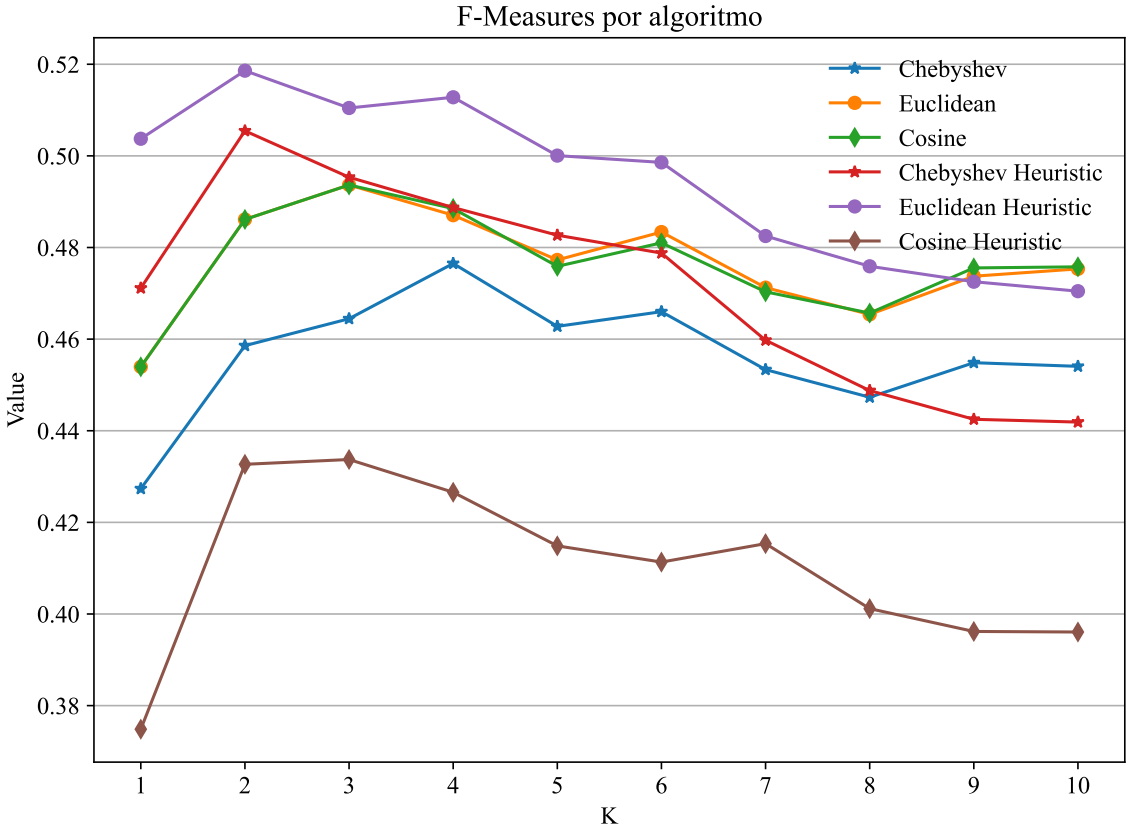

Figura 5.5: Distribuição de *F-Measure(2)* entre as configurações selecionadas para avaliação.

5.6 Análise Estatística

Dada a execução das configurações, é necessária análise estatística dos dados com o objetivo de responder as questões de pesquisa definidas na Seção 5.4.1 e, assim, identificar qual das configurações estudadas apresenta melhores resultados na recomendação de casos de teste. Essa análise é feita através dos *Testes de Comparações Múltiplas* utilizados na comparação de mais de dois algoritmos [37]. Esse teste executa a comparação entre os resultados de todas as configurações, e é ideal em um cenário onde não existe um grupo de controle.

Antes de executarmos testes estatísticos de comparação entre os algoritmos, faz-se necessário identificar quanto a normalidade dos dados coletados durante a execução do experimento com o objetivo de possibilitar a decisão sobre os tipos de testes utilizados para identificarmos se existe uma diferença estatística significativa entre os grupos estudados refutando ou aceitando as hipóteses nula H_{0-1} e alternativa H_{1-1}. No entanto, para atestar a normalidade dos dados, faz-se necessário o teste de hipótese de acordo com as hipóteses

formuladas abaixo:

- H_0**:** A população de dados provê uma distribuição normal.
- H_1**:** A população de dados não provê uma distribuição normal.

Para verificar as hipóteses mencionadas, existem várias abordagens disponíveis. Porém, o estudo feito por Razali et al. [69] demonstrou que a abordagem *Shapiro-Wilk* possui melhor desempenho para todos os tipos de distribuição e tamanhos de amostra em comparação com outras abordagens, portanto será a utilizada nesse trabalho com nível de significância de *5%*, ou seja, $\alpha = 0,05$. A Tabela 5.1 apresenta os resultados de *p*-valor do teste *Shapiro-Wilk* nos dados obtidos a partir da execução da validação cruzada com os algoritmos selecionados não utilizando a heurística de seleção de vizinhos Euclidiana (EUC), Chebyshev (CHE) e Cosseno (COS), e os mesmos algoritmos utilizando a heurística, Euclidiana (EUC H.), Chebyshev (CHE H.) e Cosseno (COS H.).

Tabela 5.1: Resultados dos testes de normalidade de *Shapiro-Wilk* para os resultados de *F-measure* obtidos na validação cruzada, em que cada valor da tabela representa o *p*-valor do teste.

K	**EUC**	**CHE**	**COS**	**EUC H.**	**CHE H.**	**COS H.**
1	2.39e-29	3.94e-08	2.39e-29	2.39e-29	7.68e-08	6.68e-07
2	2.41e-29	3.54e-09	2.41e-29	2.37e-29	4.21e-07	8.77e-07
3	2.39e-29	8.49e-08	2.39e-29	2.36e-29	1.58e-06	1.14e-06
4	7.57e-09	5.30e-09	7.69e-09	6.82e-08	2.63e-07	3.09e-07
5	7.48e-09	1.20e-08	7.90e-09	1.28e-07	8.13e-07	2.00e-07
6	3.98e-09	5.12e-09	3.78e-09	6.47e-08	7.11e-07	2.10e-29
7	1.53e-08	1.53e-08	1.51e-08	1.09e-07	6.04e-07	1.36e-08
8	9.81e-09	6.96e-09	1.17e-08	7.04e-08	7.50e-08	7.99e-09
9	1.19e-08	6.57e-09	1.17e-08	5.42e-08	7.81e-08	4.13e-08
10	2.23e-08	1.48e-08	2.30e-08	9.08e-08	3.43e-08	1.01e-07

A partir dos dados da tabela, é possível observar que para todas as *60* configurações o *p*-valor atingiu um valor inferior a α (*0,05*). Isso indica a possibilidade de rejeitarmos a

hipótese nula H_0, ou seja, temos a informação de que as distribuições para todas as configurações não apresentam uma distribuição normal. Essa informação é essencial para a escolha do teste estatístico a ser utilizado para identificação da existência de uma diferença estatística significativa entre as configurações, o que é um indicativo de que as variáveis independentes utilizadas no experimento produzem algum efeito nos dados.

Conforme discutido neste capítulo, foi executado o *Teste de Comparações Múltiplas*. Esse teste, além das comparações entre grupos, controla a possibilidade de erros Tipo I durante o processo – a possibilidade de encontrar falso positivos. Métodos de comparações múltiplas realizam ajustes dos níveis de significância em relação à família de comparações – nome dado ao conjunto de configurações a serem comparadas – ajustando o valor do nível de significância α considerando a quantidade de comparações a serem realizadas α_F, de acordo com a fórmula abaixo:

$$\alpha_F = 1 - (1 - \alpha)^N$$

Onde *N* é o número de testes de uma família. A partir dos dados do nosso experimento, onde foram definidas *60* possíveis configurações *C*, podemos calcular o valor de *N* e consequentemente α_F, demostrado a seguir:

$$N = \frac{(C * (C - 1))}{2} = \frac{(60 * (60 - 1))}{2} = 1770$$

$$\alpha_F = 1 - (1 - 0,05)^{1770} \cong 1$$

Os resultados mostram que para uma distribuição com *60* configurações temos um total de *1770* comparações e o nível de significância α_F aproximadamente *1*. A redução na quantidade de configurações disponíveis, ao descartar algoritmos que retornaram os mesmos resultados, tiveram como principal objetivo a redução significativa de comparações, e um consequente aumento ainda maior do nível de significância.

Um dos métodos mais comuns para esse tipo de comparação é o ANOVA [30], porém, a utilização desse método tem como premissa que os dados devem vir de populações independentes e com uma distribuição normal. A distribuição dos dados não normal restringe os testes de verificação da diferença estatística dos grupos a *não paramétrica*. Dentre os

métodos para verificação de dados não paramétrica, Garcia et al [37] identificaram que para comparações com mais de *5* grupos, o método *Friedman* possui um melhor desempenho.

Os resultados da execução do teste de *Friedman* podem ser vistos na Tabela 5.2 com nível de significância de *5%*, ou seja, $\alpha = 0,05$. Nela é possível verificarmos que o *p*-valor resultante da comparação entre todas as configurações é inferior a *0,05*, permitindo assim rejeitarmos a hipótese nula $H_{0-2.1}$. Esse resultado nos indica que existe uma diferença estatisticamente relevante entre as configurações, ou seja, que as variáveis independentes têm influência nos resultados do *F-Measure(2)* extraídos da execução do experimento.

Tabela 5.2: Resultados dos testes de *Friedman*.

Métrica	**Valor**
F-Statistic	784,29
P-Valor	2,6656659044626407e-127

Uma vez identificada a diferença entre os grupos sendo estudados, o próximo passo é identificar como esses grupos diferem no intuito de classificarmos o(s) algoritmo(s) que possuem melhor resultado considerando o *F-Measure(2)*. Para isso, faz-se necessária uma análise *post hoc* ainda utilizando o método de *Friedman*. O resultado dessa execução é apresentado na Tabela 5.3 onde um ranking é gerado de acordo com capacidade de diferenciação entre as médias dos resultados do *F-Measure(2)*.

Na tabela, as configurações são ranqueadas através de grupos ordenados por ordem alfabética. Porém, configurações que estão no mesmo grupo não podem ser consideradas melhores do que as outras, mas estatisticamente elas são semelhantes. Por exemplo, as configurações *Euclidiana* com *K = 3* e com *K = 5* ambas utilizando heurística de seleção de vizinhos (ID's 1 e 10 da tabela), possuem o grupo "a" no ranqueamento, o que significa que elas não são estatisticamente diferentes e não é possível afirmar estatisticamente que a utilização de uma ou outra irá trazer um melhor desempenho na recomendação de casos de teste. A tabela mostra as 13 configurações que possuem os dois melhores desempenhos (a e b) por questões de sumarização dentre as *60* configurações avaliadas.

Para seleção da medida a ser adotada, foi escolhida dentre as que se encontram no grupo de melhor desempenho ("a"), a configuração que apresentou o maior valor absoluto de *F-Measure* descrito na Tabela 5.5, por isso, optamos pela métrica *Euclidiana* com *K = 3* consi-

Tabela 5.3: Resultados dos testes de *Friedman*.

ID	Algoritmo	K	Heurística	Grupos
1	Euclidiana	3	Sim	a
2	Euclidiana	2	Sim	ab
3	Cosseno	3	Não	ab
4	Euclidiana	3	Não	ab
5	Chebyshev	2	Sim	abc
6	Chebyshev	3	Sim	abcd
7	Cosseno	2	Não	abcde
8	Euclidiana	2	Não	abcde
9	Euclidiana	4	Sim	abcde
10	Euclidiana	5	Sim	abcde
11	Cosseno	4	Não	bcdef
12	Euclidiana	1	Sim	bcdefg
13	Euclidiana	4	Não	bcdefg

derando as heurística de seleção de vizinhos. Portanto, a configuração escolhida juntamente com seus resultados obtidos no experimento são os seguintes (RQ):

- medida de similaridade: similaridade de Euclidiana;
- número de vizinhos (k): 3;
- Precisão: *37,8%*;
- *Recall*: *67,5%*;
- *F-Measure(2)*: *80%*.

5.7 Ameaças à Validade

Nesta seção, descrevemos as ameaças à validade do experimento seguindo as definições apresentadas em [107]: **Ameaça à validade da conclusão**, **Ameaça à validade interna**, **Ameaça à validade de construção** e **Ameaça à validade externa**.

As **ameaças à validade de conclusão** são aquelas que podem dificultar a conclusão do experimento baseado nos resultados obtidos. Nesse sentido, a base de dados montada para treinamento forneceu uma quantidade significativa de dados, porém, em algumas categorias da taxonomia definida para análise dos dados. Em outras categorias, como exemplo, *Módulo de Autenticação*, a quantidade de dados foi baixa – aproximadamente uma estória por projeto – principalmente pelo fato de projetos de software definirem apenas uma Estória de Usuário para o tratamento dessas funcionalidades. Para mitigar esse risco, durante a execução do experimento foram descartadas estórias de categorias que não continham no *fold* de treinamento a quantidade de vizinhos K definida na configuração.

Já as **ameaças à validade interna** estão relacionadas ao que pode afetar a variável independente do experimento. No experimento relatado neste trabalho, a coleta de dados e classificação foi realizada apenas pelo autor, o que pode ser considerada uma ameaça à validade, por poder conter viés do pesquisador. Como estratégia para mitigação da ameaça, a definição das categorias foram discutidas durante reuniões com o grupo de pesquisa, para avaliar a corretude das classificações e necessidade de criação ou alterações de categorias já definidas. Uma outra ameaça nesse sentido se deve à qualidade dos dados coletados nas organizações que utilizam o ágil, possivelmente por falta de maturidade na documentação de casos de teste. Fato que foi analisado durante a pesquisa resultando no descarte de vários registros não aptos para validação.

As **ameaças à validade de construção** dizem respeito à generalização dos resultados obtidos no experimento em relação à teoria por trás do estudo. Neste experimento foi utilizada apenas uma técnica de recomendação – K vizinhos mais próximos – além de um número limitado de algoritmos para similaridade por questões de viabilidade, o que é uma ameaça à validade de construção. No entanto, mesmo na avaliação entre as opções selecionadas, a execução com os algoritmos que obtiveram o pior desempenho não causaram um impacto tão significativo no resultado da recomendação.

As **ameaças à validade externa** são as que dizem respeito à generalização dos resultados em ambiente diferente do executado. Uma ameaça nesse sentido é que os níveis de *precision* e *recall* são considerados aceitáveis nos resultados obtidos a partir de estudo qualitativo com testadores de organizações, porém, no ambiente onde o mesmo deve ser aplicado, essa aceitação deve ser dependente da interface através da qual os usuários utilizarão a aborda-

gem, podendo ser necessário o aumento significativo das métricas para tornar sua utilização viável.

5.8 A Ferramenta TestRec

A ferramenta TestRec foi desenvolvida como parte da análise da viabilidade da solução desenvolvida. O desenvolvimento desta ferramenta foi executado pelo autor deste trabalho, sendo seu projeto uma parceria entre a SOFTEX [6] e o VIRTUS/UFCG [7]. O principal objetivo da ferramenta TestRec é ser uma prova de conceito da abordagem definida neste trabalho, utilizando as configurações definidas a partir da validação *off-line* descrita nas seções anteriores. Nela foram concebidas as principais funcionalidades de um sistema para gestão de projetos ágeis, como o cadastro de projetos, Sprints, Estórias de Usuário, requisitos não funcionais, critérios de aceitação, e sua principal funcionalidade, que é o cadastro manual de casos de teste, ou através da reutilização de um caso de teste previamente desenvolvido que consta na base de dados.

Na Figura 5.6, podemos observar a arquitetura da ferramenta, que compreende quatro componentes. A **Base de dados** possui todas as informações armazenadas do projeto. Nela, registramos as Estórias de Usuário contendo a taxonomia definida neste trabalho e casos de teste contendo informações de que indicam a partir de qual caso de teste o mesmo foi reutilizado, caso tenha sido. Os dados iniciais de projeto, Estórias de Usuário e casos de teste foram importados através dos dados coletados para validação *Off-line* – descrito na Seção 5.2 – uma vez que esses dados já estariam de acordo com os requisitos para o correto funcionamento do Sistema de Recomendação.

Na figura, podemos observar o componente **Client**, que é o responsável pela interação com o usuário, fornece e recupera dados do componente *Server*. O componente **Server** é o responsável por executar todas as regras de negócio referentes às funcionalidades do sistema, armazenando e buscando informações da *Base de Dados* do sistema e fornecendo uma API (Application Programming Interface), com formato de mensagem REST (Representational State Transfer) para o componente *Client*. Também é papel deste componente fornecer a

[6] https://softex.br/

[7] https://www.virtus.ufcg.edu.br/

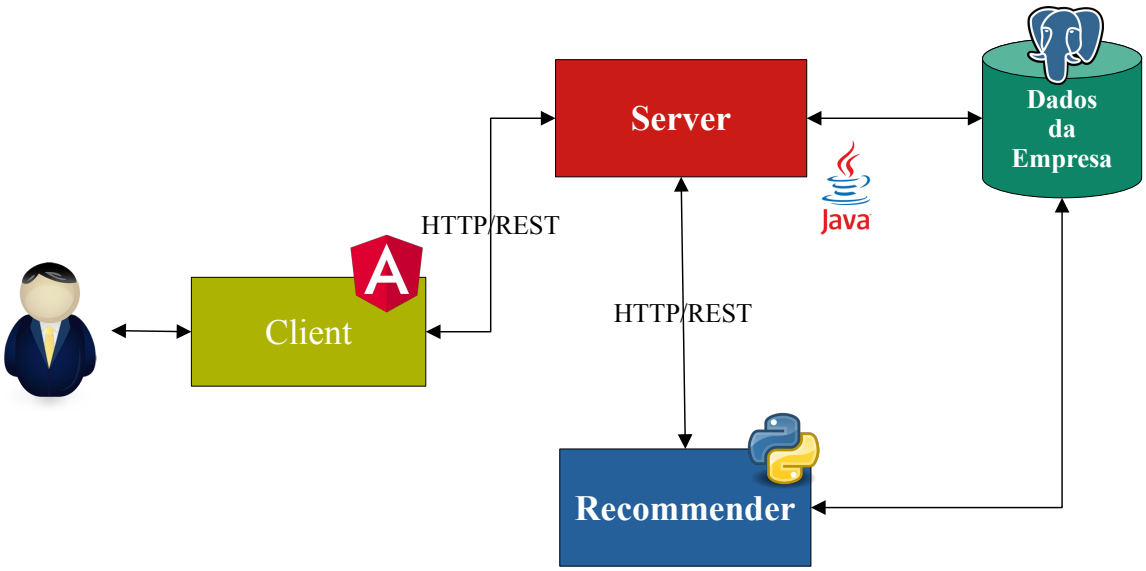

Figura 5.6: Arquitetura da ferramenta TestRec.

entrada para o componente *Recommender*, receber a saída e fornecer para o componente *Client* para que o usuário final, que acessa esse componente, possa efetuar a reutilização dos casos de teste a partir destas recomendações.

O componente **Recommender** é responsável por efetuar a recomendação de casos de teste dentro da ferramenta cuja representação é descrita através da Figura 4.6. Ele recebe como entrada o identificador da Estória de Usuário que se deseja recomendar casos de teste através da exposição de uma API REST, coleta as informações necessárias para recomendação na *Base de dados*, processa as informações para efetuar a recomendação e fornece como saída os registros dos casos de teste a serem recomendados. As configurações deste componente seguem as definições feitas a partir da validação *Off-line* descrita nas seções anteriores deste capítulo. O *Recommender* foi desenvolvido na linguagem de programação *Python* [8], pois seu código foi concebido a partir do programa utilizado para validação *Off-line* da solução, que conta com várias ferramentas específicas para ciência de dados.

Na Figura 5.7, é possível vermos a tela de recomendação de casos de teste. Nela, podemos selecionar para qual Estória de Usuário queremos que o sistema busque recomendações (A). Ao selecionarmos uma estória, o sistema busca as recomendações de acordo com o

[8]https://www.python.org/

cadastro da mesma, segundo processo descrito na capítulo anterior, resultando em uma listagem de casos de teste recomendados (B). Esses casos de teste recomendados podem ser arrastados a fim de serem reutilizados pelas estórias de usuário do projeto (D), ou novos casos de teste podem ser cadastrados, caso exista algum caso de teste não recomendado, mas que necessite ser registrado e posteriormente executado (C).

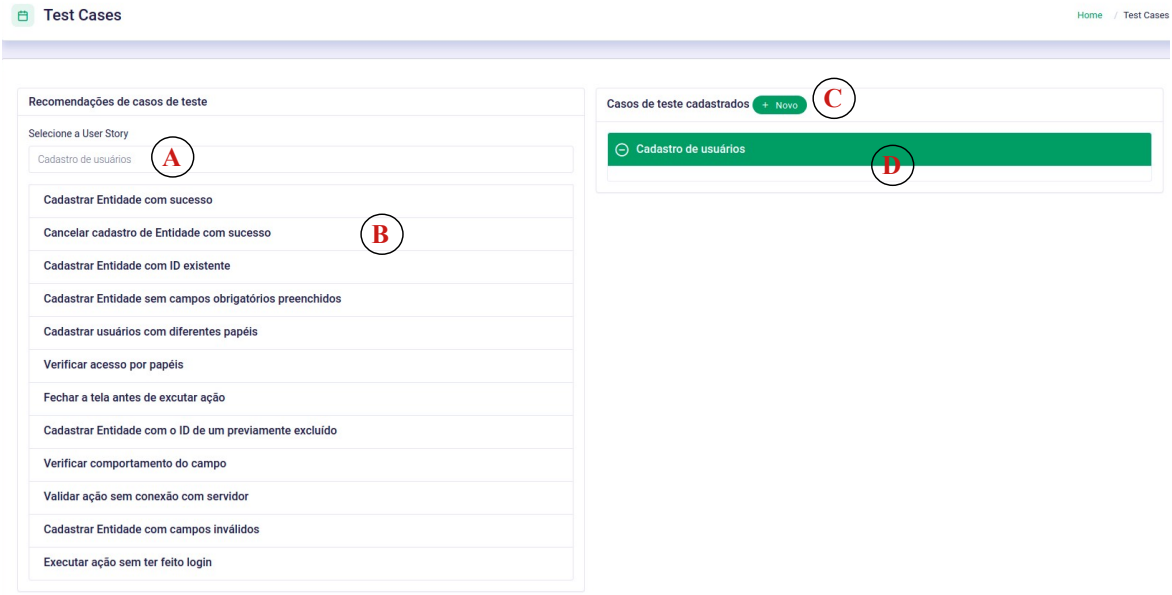

Figura 5.7: Recomendação de casos de teste para Estória de Usuário Cadastro de Usuários.

Ao clicarmos e arrastarmos casos de teste recomendados para as Estórias de Usuário selecionadas, cópias dos casos de teste são criadas e relacionadas a essa estória, como pode ser visto na Figura 5.8. Por isso, caso o usuário deseje fazer alguma alteração nos casos de teste recomendados, pode fazer sem afetar o caso de teste original. Essa funcionalidade foi concebida dessa maneira, pois, apesar de serem semelhantes, muitos casos de teste reutilizados podem ter detalhes muito específicos para um determinado domínio, equanto que o projeto que reutiliza esse caso de teste também pode possuir suas especificidades. Ao realizarmos essa cópia, o casos de teste copiado possui uma referência para o caso de teste original, para que assim possamos ter a informação da quantidade de vezes que um caso de teste é reutilizado. O manual do usuário contendo todos os detalhes da ferramente é descrito no Apêndice B.

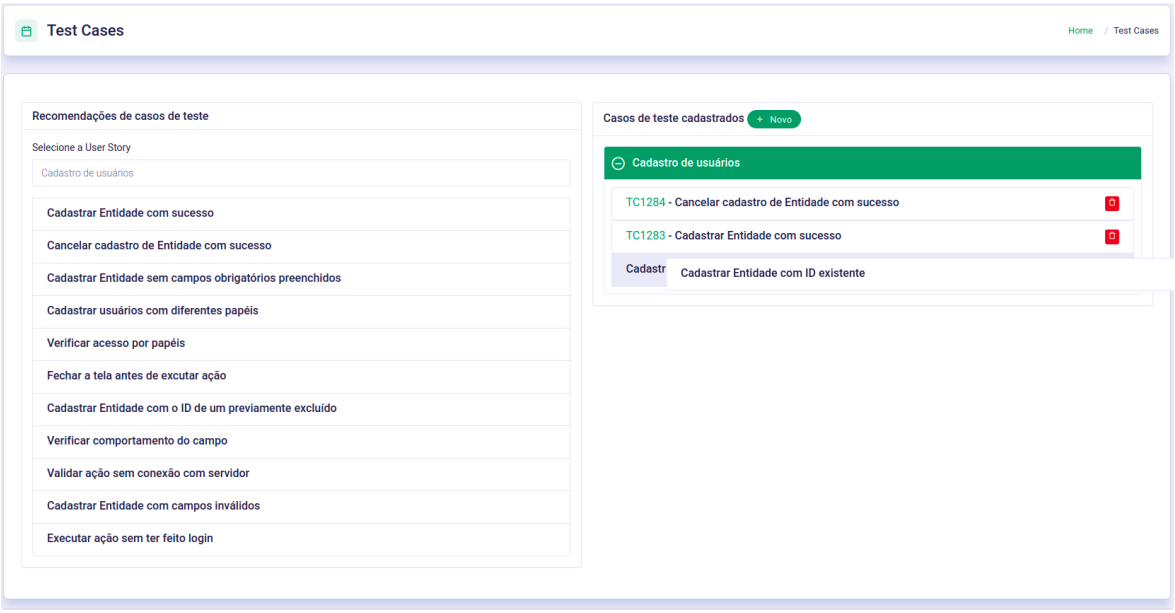

Figura 5.8: Cópia de casos de teste recomendados.

5.9 Considerações Finais do Capítulo

Neste capítulo, apresentamos uma descrição completa do experimento *off-line* utilizando o sistema de recomendação definido como solução para o problema científico descrito neste trabalho. Nele consideramos o algoritmo de K vizinhos mais próximos e testamos várias configurações com variáveis independentes, o número de vizinhos mais próximos (K), o algoritmo de cálculo de distância entre os vizinhos, a base de dados utilizada para treino e o uso ou não da heurística de seleção de vizinhos mais próximos.

Para evitar o problema do *cold start*, coletamos dados de Estórias de Usuário e casos de teste de duas organizações que utilizam o método ágil. Apesar de problemas encontrados nos dados registrados pelas organizações, conseguimos um número aceitável de registros para iniciar o treinamento e em seguida executar o experimento. Nele, fizemos um validação cruzada *10-fold* para evitar a superespecialização do algoritmo e viabilizar a validação, na qual identificamos a *precision* e o *recall* das recomendações sugeridas. A partir desses dados, fizemos uma análise estatística, na qual identificamos as melhores configurações para, a partir da base de dados de treino, executar recomendações em projetos reais. A partir da melhor configuração encontrada, respondemos a questão de pesquisa (**RQ - A solução pro-**

posta é viável?) e definimos a ferramenta TestRec para gerenciamento de projetos ágeis que fornece a recomendação de casos de teste para seus usuários.

Capítulo 6

Avaliação da Solução

Após a conclusão sobre a melhor configuração a ser utilizada pelo sistema de recomendação desenvolvido, dada a base de dados coletada, devemos utilizar a abordagem em projetos reais para avaliar o quanto ela pode ajustar os testadores no desenvolvimento de casos de teste.

6.1 Metodologia

O objetivo do estudo é a coleta de evidências a respeito do impacto da abordagem desenvolvida no desenvolvimento de casos de teste em projetos reais. Assim, o propósito da ferramenta desenvolvida é minimizar o esforço para o desenvolvimento de casos de teste, além de aumentar a completude da suíte de testes desenvolvida. Esses propósitos devem ser atingidos através da reutilização de casos de teste previamente desenvolvidos pela empresa utilizando a ferramenta.

6.2 Protocolo do Estudo

O método científico utilizado para o estudo foi um Estudo de Caso descritivo [87] para coleta de evidências sobre o impacto do sistema de recomendação de casos de teste no desenvolvimento em projetos reais. Assim, o objetivo da ferramenta é o reuso de casos de teste de projetos previamente desenvolvidos, além do aumento da completude das suítes de teste.

Caso: O processo ágil para o desenvolvimeto de casos de teste utilizando o sistema de recomendação.

Unidade de Análise: Um projeto real de desenvolvimento de software ágil.

Propósito: Identificar o impacto do sistema de recomendação no desenvolvimento de casos de teste ágeis.

Assim, pretendemos responder a seguinte questão de pesquisa:

RQ03: A solução proposta melhora o desenvolvimento de casos de teste em projetos ágeis?

Para detalharmos o entendimento do problema, foram formuladas as seguintes questões de pesquisa secundárias:

$\mathbf{RQ}_{3.1}$: O sistema de recomendação de casos de teste aumenta a completude da suíte de testes de um projeto ágil?

$\mathbf{RQ}_{3.2}$: O sistema de recomendação de casos de teste promove o reuso de casos de testes em um projeto ágil?

$\mathbf{RQ}_{3.3}$: Os benefícios encontrados na utilização do sistema de recomendação evoluem com o tempo?

Na questão $\mathbf{RQ}_{3.1}$, desejamos identificar se a suíte de testes resultante das recomendações e dos casos de teste desenvolvidos pelos testadores do projeto são mais completas do que as suítes de teste originais, ou seja, a suíte contendo apenas testes desenvolvidos por testadores. Essa informação é crucial para identificarmos se o sistema de recomendação ajuda os testadores a incluir na suíte casos de teste que não identificaram durante o desenvolvimento do projeto, casos de teste que não identificaram por falta de conhecimento do negócio, ou ainda casos de teste que não identificaram por falta de experiência em projetos de software. Para avaliação das questões foram levantadas as seguintes hipóteses:

- $H_{0-3.1}$**:** O sistema de recomendação não aumenta a completude da suíte de testes de um projeto ágil.
- $H_{1-3.1}$**:** O sistema de recomendação aumenta a completude da suíte de testes de um projeto ágil.

Já na questão $\mathbf{RQ}_{3.2}$, nosso objetivo é identificar se o sistema de recomendação fornece aos testadores casos de teste que eles iriam desenvolver, ou seja, se ele antecipa casos de teste que seriam desenvolvidos e, assim, permitem ao testador reutilizá-los. De fato, a reutilização

de casos de teste – ainda que exista a necessidade de ajustes nos mesmos para contemplar especificidades do negócio – pode trazer uma série de outros benefícios ao projeto de software, como a redução do esforço de desenvolvimento dos mesmos, uma vez que se os testes são reutilizados, não há a necessidade de se dispender esforço para o seu desenvolvimento. Para avaliação das questões foram levantadas as seguintes hipóteses:

- $H_{0-3.2}$**:** O sistema de recomendação não promove o reuso no desenvolvimento de casos de testes em um projeto ágil.
- $H_{1-3.2}$**:** O sistema de recomendação promove o reuso no desenvolvimento de casos de testes em um projeto ágil.

Dado que as Estórias de Usuário desenvolvidas durante a utilização do sistema de recomendação, junto com seus casos de teste desenvolvidos e recomendados são armazenados na base de dados, os mesmo serão utilizados no processo de recomendação de futuras Estórias de Usuário a serem desenvolvidas. Esse fato nos dá indícios de que a contínua utilização do sistema de recomendação permite uma evolução do mesmo e possibilita a execução de recomendações mais assertivas. São esses indícios que desejamos verificar para responder a questão $\mathbf{RQ}_{3.3}$. Para avaliação das questões foram levantadas as seguintes hipóteses:

- $H_{0-3.3}$**:** Os benefícios encontrados na utilização do sistema de recomendação não evoluem com o tempo.
- $H_{1-3.3}$**:** Os benefícios encontrados na utilização do sistema de recomendação evoluem com o tempo.

6.3 Ameaças à Validade do Estudo

Nesta seção, apresentamos as ameaças à validade do estudo, de acordo com a classificação apresentada por Wholin et al. [107] já descritas na Seção 5.7. Dado que as classificações das Estórias de Usuário foram feitas pelo pesquisador, a reprodução da execução em ambiente real com todo o processo sendo executado pelos usuários pode não ter os mesmos resultados obtidos nesse estudo, configurando uma ameaça de validade interna. Para mitigar essa ameaça, as classificações foram revisadas por outros pesquisadores durante o processo.

Os dados para treinamento foram focados em módulos e operações específicos, devido à quantidade disponível dos mesmos. Assim, caso a natureza do projeto fuja das características da base de dados para treinamento, os resultados podem ser minimizados, não permitindo uma conclusão correta a respeito dos resultados configurando a validade externa. Para mitigar essa ameaça, fizemos os cálculos dos resultados considerando as Estórias de Usuário não mapeadas de acordo com a taxonomia definida durante o trabalho.

O fato do estudo de caso ser executado em uma empresa pode enviezar os resultados para empresas que possuam característica de projeto semelhante, dificultando sua generalização, o que configura em uma ameaça de validade externa. Para mitigar essa ameaça, consideramos analisar projetos de clientes diversos e de domínios diversos, o que proporcionou uma grande variabilidade de dados para o estudo.

A falta de qualidade dos dados obtidos poderia ter influência negativa nos resultados do estudo. Assim, para evitar que isso acontecesse, nos casos em que tivemos dificuldade em identificar as características da Estória de Usuário ou dos critérios de aceitação, entramos em contato com os testadores responsáveis pelo desenvolvimento dos mesmos a fim de identificar a corretude dos dados.

6.4 Coleta de Dados

Para coleta de dados, a instituição participante da avaliação identificou os projetos que se adequaram a mesma, considerando questões de natureza, criticidade e disponibilidade de seus membros para participação da avaliação. Como resultado, foram selecionados dois projetos: um sistema web para classificação de riscos de projetos, e um sistema para auxílio a profissionais da educação, possuindo uma interface web e outra interface com dispositivos móveis.

Uma vez que a ferramenta definida e validada no capítulo anterior possui uma interface gráfica com o usuário, mas não é a ferramenta padrão utilizada pela instituição, a participação do autor deste trabalho se fez necessária para execução da mesma. Podemos ver todo o processo de coleta de dados na Figura 6.1. Ao término de cada iteração dos projetos o pesquisador identificou as Estórias de Usuário desenvolvidas pelo projeto utilizando a ferramenta utilizada pela instituição, registrou essas estórias na ferramenta *TestRec* classificando

cada uma de acordo com a taxonomia definida neste trabalho, uma vez que a mesma não é utilizada por projetos da supracitada empresa. Outros dados como critérios de aceitação e requisitos não funcionais relacionados à Estória de Usuário e ao projeto respectivamente também foram coletadas para possibilitar a recomendação por parte da abordagem definida. Para os dados implícitos nas ferramentas, ou seja, não levantados pela equipe durante o desenvolvimento, foram consultadas outras fontes, inclusive o próprio time de desenvolvimento.

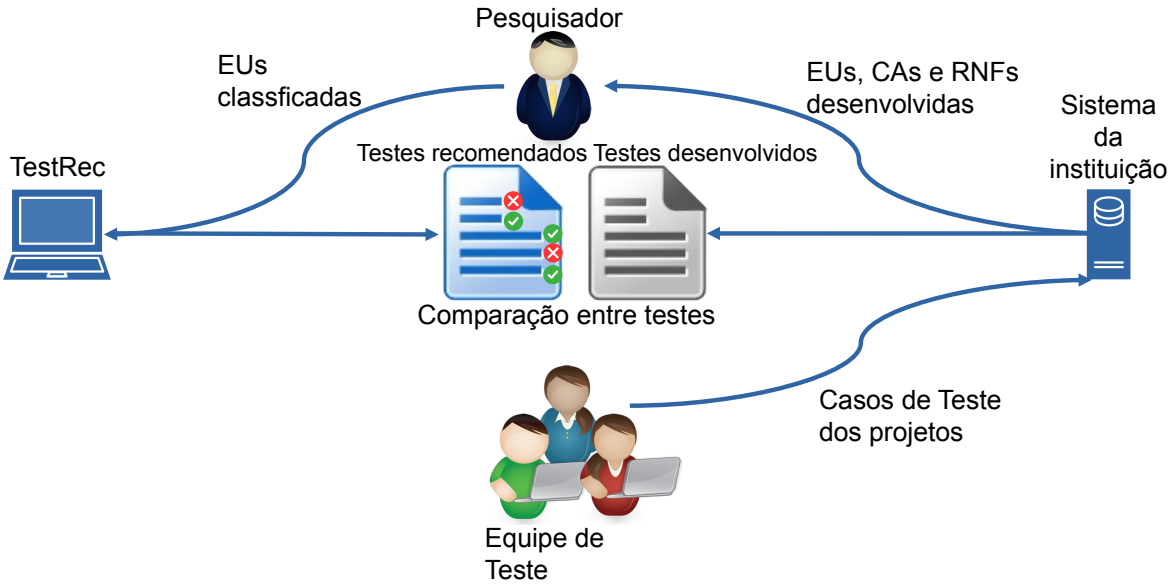

Figura 6.1: Cópia de casos de teste recomendados.

Dadas as informações coletadas, o sistema de recomendação foi executado pelo pesquisador para as Estórias de Usuário desenvolvidas coletadas, fornecendo como resultado uma lista de casos de teste para execução dessas estórias. De posse da lista de casos de teste recomendados, o pesquisador acessou os casos de teste desenvolvidos pela equipe de desenvolvimento do projeto, a fim de identificar quais casos de teste recomendados foram desenvolvidos. Adicionalmente, essa informação nos traz indícios de que, caso utilizassem a abordagem proposta no projeto, os desenvolvedores poderiam reutilizar esses casos de teste no projeto, reduzindo o esforço de desenvolvimento da suíte de testes.

Uma vez coletada a informação dos casos de teste que foram desenvolvidos pelos testadores e recomendados pelo sistema de recomendação, foi realizada uma reunião entre o

pesquisador e membros da equipe de desenvolvimento responsáveis pelos testes dos projetos, para avaliação dos casos de teste recomendados restantes. A ideia principal foi verificar se os casos de teste remanescentes poderiam ser acrescentandos à suíte de testes dos projetos. Para cada caso de teste, discutimos com a equipe a viabilidade de adição à suíte de testes e, para os casos de teste que os desenvolvedores entendessem como viável o acréscimo à suíte de testes, entendemos que os mesmos deveriam ser desenvolvidos pela equipe, mas não foram. O acréscimo de casos de teste recomendados às suítes dos projetos nos traz indícios de que a utilização da abordagem faz com que os desenvolvedores gerem uma suíte de testes mais completa em relação à não utilização.

Ainda durante a reunião com o time de desenvolvimento, foram identificados os casos de teste que não poderiam fazer parte de suíte de testes dos projetos analisados, ou seja, o SisRec recomendou um caso de teste não relevante. Adicionalmente, também identificamos as razões pelas quais esses casos de teste não poderiam ser incorporados com o objetivo de identificar possíveis ajustes na solução. Por fim, identificamos casos de teste que não foram recomendados, mas foram desenvolvidos pelo time do projeto. Isso significa que mesmo utilizando a abordagem definida, esses casos de teste não seriam recomendados e, consequentemente, deveriam ser desenvolvidos. É importante frisar que esses casos de teste foram incorporados à base de treinamento durante a coleta dos dados, fazendo assim com que os mesmos possam ser recomendados nas próximas iterações. Essa incorporação dos dados reflete o processo real de utilização da ferramenta, onde os casos de teste não recomendados devem ser desenvolvidos e assim estejam prontos para reutilização no desenvolvimento das próximas Estórias de Usuário.

6.5 Métricas

Baseado nas informações coletadas sobre os casos de teste recomendados e desenvolvidos pelo time de desenvolvimento, definimos rótulos para classificar esses casos de teste com o objetivo de ajudar a definir as métricas para avaliar as hipóteses definidas e responder as questões de pesquisa. Durante as comparações entre os casos de teste recomendados e os casos de teste desenvolvidos, identificamos que parte dos testes recomendada, também foi desenvolvido pelo time de desenvolvimento e outra parte não foi desenvolvida. Nas situações

em que os casos de teste foram recomendados e os mesmos também foram desenvolvidos pelo time de desenvolvimento, definimos o rótulo **Reuso**.

Já os casos de teste recomendados, não desenvolvidos pelo time e aceitos pelo mesmo para incorporação na suíte de testes, foram rotulados **Aceito**. Além dessa informação, durante as reuniões entre o pesquisador e os desenvolvedores também foi identificado para cada caso de teste aceito qual a relevância do mesmo para o projeto. Neste trabalho, classificamos a relevância do casos de teste adicionado em: **alta**, para casos que são muito importantes para o projeto, ou seja, aqueles que não deveriam não serem contemplados durante a fase de testes do projeto; **média**, para casos de teste que possuem uma relevância intermediária, ou seja, os casos de teste que trazem relevância para o projeto, mas caso não estejam o impacto para qualidade do projeto não é alta; e finalmente, **baixa**, para os casos de teste que acrescentam valor ao produto, mas caso não fossem executados, não teriam impacto na qualidade do produto final.

Ainda dentre os casos de teste recomendados e não desenvolvidos, identificamos durante reunião com o time que alguns não poderiam ser incorporados a suíte de testes, aos quais rotulamos como **Rejeitado**. Além disso, identificamos e classificamos as razões pelas quais os casos de teste foram rejeitados junto ao time de desenvolvimento. Assim, os casos de teste rejeitados foram classificados conforme as razões seguir:

- **Negócio:** Essa razão para rejeição significa que o SisRec recomendou um caso de teste que está de acordo com o requisito, mas ataca uma regra de negócio não definida para a Estória de Usuário em questão.

- **Outro requisito:** Em alguns casos, o SisRec recomendou casos de teste que não pertencem ao requisito para o qual foram recomendados, sendo assim rejeitados pelo time de desenvolvimento.

- **Não aplicável:** Essa razão para rejeição especifica casos de teste que não podem ser executados, principalmente por estarem mal definidos na base de treinamento.

- **Muito simples:** Casos de teste que especificam uma parte muito pequena do requisito ao qual foram definidos, sendo assim rejeitados.

Existiram ainda os casos de teste que não foram recomendados pela TestRec, mas que

foram desenvolvidos pelo time de desenvolvimento. Esses casos de teste foram classificados como **Desenvolvido**. A Tabela 6.1 sumariza os rótulos adotados para classificar os casos de teste mesclados entre os desenvolvidos pelo time de desenvolvimento e os recomendados pelo sistema de recomendação.

Tabela 6.1: Rótulos definidos para os casos de teste.

Rótulo	**Descrição**
Desenvolvido	Não recomendado pelo SisRec e desenvolvido pelo time.
Reuso	Recomendado e desenvolvido pelo time.
Aceito	Recomendado, não desenvolvido e aceito pelo time.
Rejeitado	Recomendado, não desenvolvido e não aceito pelo time.

Uma vez classificados os casos de teste, o próximo passo é definir as métricas que ajudem a avaliar as hipóteses definidas e responder as questões de pesquisa. Assim, definimos as métricas: percentual de crescimento da suíte de testes (Test suite increase percent *ts_inc_perc*), descrita na Equação 6.1, que considera o percentual de casos de teste recomendados e não desenvolvidos pelo time de desenvolvimento; o percentual de casos de teste recomendados que poderiam ser reutilizados pelo time de desenvolvimento (reuse percent *reuse_perc*), descrito na Equação 6.2; A precisão dos casos de teste recomendados (reuse precision *reuse_precision*), descrita na Equação 6.3, que considera, dos casos de teste recomendados, o percentual dos aceitos; e finalmente, o percentual de rejeição (rejection percent *reject_perc*), que considera a proporção de casos de teste rejeitados dentre todos os casos de teste recomendados, descrita na Equação 6.4.

$$ts\_inc\_perc = \frac{\#Aceito}{\#Desenvolvido + \#Reuso} \tag{6.1}$$

$$reuse\_perc = \frac{\#Reuso}{\#Desenvolvido + \#Reuso} \tag{6.2}$$

$$reuse\_precision = \frac{\#Aceito + \#Reuso}{\#Aceito + \#Reuso + \#Rejeitado} \tag{6.3}$$

$$reject\_perc = \frac{\#Rejeitado}{\#Aceito + \#Reuso + \#Rejeitado} \tag{6.4}$$

6.6 Resultados

No estudo realizado, coletamos dados de 41 Estórias de Usuário e 203 casos de teste criados pelo testadores dos projetos. Conforme descrito na seção anterior, para cada iteração foram executadas as recomendações para as Estórias coletadas e em seguida foram realizadas reuniões com os testadores dos projetos para identificar os casos de teste que poderiam ser adicionados à suite de testes do projeto. Como resultado, 26 casos de teste foram desenvolvidos pelos testadores e não recomendados pela abordagem definida, ou seja, rotulados como **Desenvolvido**.

Além disso, 177 casos de teste foram desenvolvidos pelos testadores e também recomendados pelo SisRec, ou seja, rotulados como **Reuso**. Adicionalmente, outros 90 casos de teste foram aceitos pela equipe de desenvolvimento como viáveis para serem inseridos na suíte de testes dos projetos, e assim rotulados como **Aceito**. Dentre os casos de teste aceitos pelos testadores, identificamos a relevância dos mesmos para entender sua importância para os projetos em desenvolvimento. A Figura 6.2 apresenta a proporção de casos de teste com alta, média e baixa relevância de acordo com os testadores dos projetos participantes do estudo.

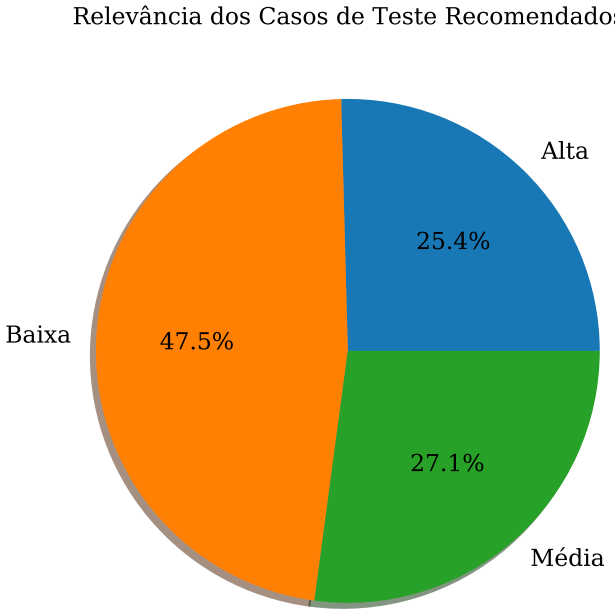

Figura 6.2: Propoção da relevância de casos de teste recomendados aceito pelos testadores.

Finalmente, dentre os os recomendados, 144 casos de teste foram rejeitados – rotulados

como **Rejeitado** – pelos testadores dos projetos, que indicaram não ser viável incorporá-los à suíte de testes. Os dados ainda indicaram um aumento da suíte de testes dos projetos de 203 para 267 casos de teste. Podemos verificar na Tabela 6.2 exemplos de casos de teste classificados para um Estória de Usuário intitulada *Registro de projeto*, para um sistema de gestão de projetos ágeis.

Tabela 6.2: Análise dos casos de teste para a EU registro de projeto.

Caso de Teste	**Status**
Registro de projeto com sucesso	Reuso
Cancelar registro de projeto	Aceito
Registrar projeto com ID existente	Rejeitado
Verificar papeis de usuário	Rejeitado
Fechar a janela antes de executar a ação	Aceito
Registrar com ID previamente excluído	Rejeitado
Executar a ação sem conexão com servidor	Aceito
Executar a ação sem ter feito login	Aceito
Registro com data inicial após a data final	Desenvolvido
Registro com campos obrigatórios não preenchidos	Reuso
Registrar com campos inválidos	Reuso

Podemos verificar o resultado das métricas definidas na seção anterior na Tabela 6.3. É interessante notar que a abordagem definida recomendou 85.31% dos casos de teste desenvolvidos. Esse resultado nos mostra que, se os projetos utilizassem o SisRec durante o desenvolvimento de casos de teste, seria necessário apenas o desenvolvimento completo de 14.69% dos casos de teste, que são representados pelos casos de teste marcados com o rótulos **Desenvolvido** (26 casos de teste).

Na Tabela 6.3, ainda podemos ver que o uso de SisRec pode trazer um aumento relevante da suíte de casos de teste ($50,85\%$), indicando que um número de casos de teste considerável não foi considerado durante o desenvolvimento. Dado que os testadores aceitaram os casos de teste recomendados, temos a indicação de que o SisRec melhora a completude da suíte de testes.

Entre os casos de teste rejeitados, a causa mais comum foi a regra de negócio, conforme

Tabela 6.3: Métricas definidas para avaliação do projeto piloto e resultado.

Métrica	**Resultado**
Percentual de incremento da suíte (ts_inc_perc)	$50,85\%$
Percentual de reuso (reuse_perc)	$85,31\%$
Precisão (reuse_precision)	$63,09\%$
Percentual de rejeição (reject_perc)	$36,91\%$

pode ser visualizado na Tabela 6.4. Na tabela, também é possível verificarmos que o SisRec desenvolvido tem uma baixa quantidade de erros na recomendação de casos de teste não relacionados à Estória de Usuário em desenvolvimento com base na baixa quantidade de casos de teste rejeitados pelo motivo *Outro requisito* $(0,70\%)$.

Tabela 6.4: Resumo dos casos de teste rejeitados.

Motivo	**Resultado**
Regra de negócio	$85,81\%$
Muito simples	$4,96\%$
Não aplicável	$7,80\%$
Outro requisito	$0,70\%$

Para identificar a evolução das métricas, também fizemos a análise dos dados obtivos no decorrer do tempo como pode ser visto na Figura 6.3. Como pode ser visto, o *Percentual de reuso*, que representa a proporção de casos de teste que foram desenvolvidos pelos testadores e também recomendado pelo RecSis tem um aumento significativo, de menos de 40% na primeira coleta de dados para 100% nas últimas coletas. Já a *Precisão* se mostra praticamente constante durante todo o período de coletas, assim como o percentual de rejeição de casos de teste. Enquanto isso, a métrica *Percentual de incremento* possui uma alta variação, onde nas coletas com valores mais altos chega a mais que dobrar a suíte de testes, enquanto que nos valores mais baixos, não possui incremento da suíte de testes.

Ainda com o objetivo de identificar a eficácia da ferramenta na reutilização de casos de teste em projetos ágeis dadas as Estórias de Usuário, identificamos, para cada projeto, dentre as estórias de negócio, quantas conseguimos mapear utilizando a taxonomia e adicionamos os casos de teste de estórias não mapeadas ao total de casos de teste do projeto,

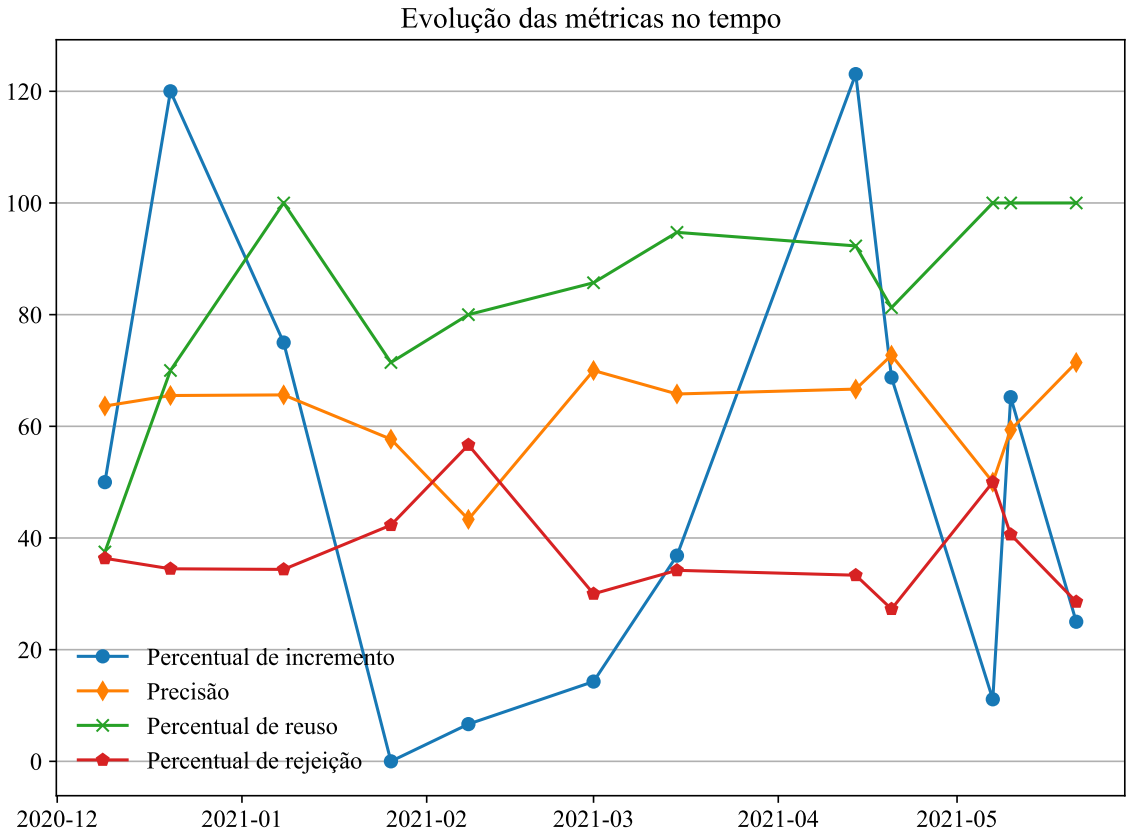

Figura 6.3: Evolução das métricas no tempo.

classificando-as como **Desenvolvido**. Essa abordagem nos dá uma visão real do reuso de casos de teste em projetos, uma vez que acreditamos que, na maioria dos projetos, não será possível o mapeamento de todas as Estórias de Usuário, e para as mesmas, casos de teste devem ser desenvolvidos e requerirá esforço por parte dos testadores, dado que o SisRec não contemplará Estórias de Usuário não mapeadas de acordo com a taxonomia definida neste trabalho.

Para atingir esse objetivo, verificamos dentre os projetos selecionados a quantidade de casos de teste para as Estórias de Usuário não classificadas utilizando a taxonomia definida. Esses casos de teste foram acrescentados aos classificados como **Desenvolvido**, e assim aumentando a quantidade de casos de teste classificadas nesse item. Essa ação impacta em duas métricas que se baseam nessa classificação: o percentual de incremento da suíte de teste (*ts_inc_perc*), onde o aumento de itens classificados como **Desenvolvido** implica na redução dessa métrica; e o percentual de reuso (*reuse_perc*), onde o aumento de casos de teste

classificados como **Desenvolvido** também implica na redução da métrica. Essas informações podem ser confirmadas através da consulta às equações 6.1 e 6.2.

Ao analisar os projetos, identificamos 55 casos de teste relacionados às Estórias de Usuário classificando-as como **Desenvolvido**. Assim, reajustamos os valores descritos na Tabela 6.3 considerando as novas quantidades, resultando na Tabela 6.5. Nela podemos perceber que o percentual de incremento da suíte de testes caiu de $50,85\%$ para $38,79\%$, enquanto que o percentual de reuso caiu de $85,31\%$ para $65,09\%$.

Tabela 6.5: Métricas definidas para avaliação do projeto piloto e resultados considerando EUs não classificadas de acordo com taxonomia.

Métrica	**Resultado**
Percentual de incremento da suíte (ts_inc_perc)	**38,79%**
Percentual de reuso (reuse_perc)	**65,09%**
Precisão (reuse_precision)	$63,09\%$
Percentual de rejeição (reject_perc)	$36,91\%$

6.7 Discussão

Nesta seção, discutiremos os dados apresentados na seção de resultados e responderemos as questões de pesquisa definidas. Apresentamos duas tabelas com o resultado das métricas atingidas: na primeira, não consideramos os dados de Estórias de Usuário não classificadas utilizando a taxonomia definida neste trabalho; na segunda acrescentamos os casos de teste das Estórias de Usuário não classificadas e adicionamos a eles o rótulo de **Desenvolvido**. Para fins de avaliação utillizaremos os dados reportados na segunda tabela, pois entendemos que o mesmo reflete a realidade, uma vez que acreditamos que a maioria dos projetos reais possuirão Estórias de Usuário não classificadas pela taxonomia.

RQ$_{3.1}$: O sistema de recomendação de casos de teste aumenta a completude da suíte de testes de um projeto ágil?

Dada a métrica de *ts_inc_perc*, observamos um aumento de $38,79\%$ na suíte de testes desenvolvida pelos projetos considerando as Estórias de Usuário dos projetos classificadas ou não, de acordo com a taxonomia definida. Esse aumento nos traz evidências de que

durante o desenvolvimento dos casos de teste, os testadores não contemplam todos os casos de teste que julgam importantes, uma vez que esse aumento no tamanho da suíte de teste se dá pela aceitação dos casos de teste recomendados que consideram relevantes para o projeto.

Vale observar que, conforme apresentado na Figura 6.2, a maioria dos casos de teste aceitos para incorporação na suíte de testes dos projetos tem baixa relevância. Contudo, a quantidade de casos de teste com relevância média e alta apresentaram valores significativos em relação ao total e, mesmo os casos de teste menos relevantes podem trazer benefícios quando incorporados à suíte de testes de um projeto de software. Adicionalmente, apesar de não fazer parte da avaliação deste estudo, o registro de casos de teste importantes para o projeto, por menor que seja, pode trazer um aumento na qualidade do produto final. Porém, é possível afirmarmos que o SisRec proposto, dentro do contexto utilizado, favorece a construção de uma suíte de testes mais completa do que a gerada apenas pelo desenvolvimento dos testadores.

RQ$_{3.2}$: O sistema de recomendação de casos de teste promove o reuso de casos de testes em um projeto ágil?

Analisando a métrica **percentual de reuso** (*reuse_perc*), podemos observar um percentual de $65,09\%$ de reuso de casos de teste, ou seja, de todos os casos de teste desenvolvidos pelos testadores dos projetos analisados, mais da metade destes poderiam ter sido reutilizados e não desenvolvidos, caso o time de desenvolvimento utilizasse o SisRec durante a execução do projeto.

Esse valor é atingido considerando que nem todos os casos de teste dos projetos são mapeados. Adicionalmente, quando analisamos a métrica do ponto de vista onde apenas as Estórias de Usuário dos projetos são mapeadas, temos um percentual de reuso ainda maior ($85,31\%$). O que implica dizer que, quanto maior a quantidade de Estórias de Usuário mapaeada, maior o percentual de reuso de casos de teste dentro de um projeto.

Vale salientar que o percentual de reuso de casos de teste não está diretamente relacionado ao percentual de redução de esforço para o desenvolvimento destes casos de teste reutilizados. Isso se deve ao fato de que ao reutilizar um caso de teste, o testador deve fazer ajustes no mesmo para contemplar as especificidades do projeto em questão. Porém, esse esforço deve ser minimizado considerando o valor da precisão encontrada de $63,09\%$, que implica que os testadores poderiam selecionar mais da metade dos casos de teste recomenda-

dos. Assim, é possível concluir que o SisRec proposto promove o reuso no desenvolvimento de casos de teste. Além disso, esse reuso de casos de teste pode trazer vários benefícios, como a redução do esforço do desenvolvimento dos casos de teste, o que permite que os testadores possam focar em outras atividades, como por exemplo os testes exploratórios, gerando um produto com maior qualidade. Essa reutilização também pode ser estendida, abrindo o caminho para outros benefícios como a reutilização de código de testes automatizados, gerando uma economia de recursos para os projetos que utilizam essa abordagem.

RQ$_{3.3}$: Os benefícios encontrados na utilização do sistema de recomendação evoluem com o tempo?

Considerando o gráfico exibido na Figura 6.3, identificamos as variações das métricas durante o período de coleta dos dados. Nela, é possível destacarmos alguma evolução para umas métricas, mas para outras não é possível. O percentual de reuso (*reuse_perc*) demonstra um crescimento siginificativo no decorrer do projeto. Durante a nossa análise, percebemos que esse fato se deve à evolução do conhecimento dos testadores conforme as sprints de desenvolvimento vão evoluindo. Isso significa que os testes recomendados e aceitos pelo time de teste nas primeiras sprints foram desenvolvidos pelos mesmos nas sprints subsequentes. Esse fato fez com que, quando recomendados novamente, os casos de teste fossem rotulados como **Reuso**.

Podemos perceber também que tanto a *precisão* quanto o *percentual de rejeição* mantiveram baixas variações durante todo o período. Os valores da precisão demonstraram que, dos casos de teste recomendados, uma boa quantidade deles foi rotulada como *Aceito* ou *Reuso*. Esse resultado está relacionado ao percentual de rejeição, uma vez que esse percentual foi calculado a partir dos casos de teste que não foram aceitos ou reutilizados.

É importante perceber também a grande variação do percentual de incremento da suíte de testes. Essas variações coincidem com a entrada de novos projetos na coleta de dados, onde os picos de aumento acontecem nas primeiras sprints dos projetos, onde a quantidade de casos de teste aceitos é maior. Vale salientar que durante o tempo o percentual de incremento da suíte de casos de teste tende a cair. Isso se deve pelo fato de que a maioria dos casos de teste tendem a ser reutilizados, e assim não mais contabilizados nesse percentual.

6.8 Considerações Finais do Capítulo

Neste capítulo, apresentamos a validação da abordagem definida, dada a melhor configuração identificada. Para isso, realizamos um processo de recomendação em paralelo ao desenvolvimento de dois projetos de software e comparamos os casos de teste desenvolvidos com os recomendados. Essa comparação nos mostrou que $65,09\%$ dos casos de teste poderiam ter sido reutilizados pelos testadores e que a suíte de testes total teve um aumento de $38,79\%$.

Essas informações nos trazem indícios de que a abordagem definida traz benefícios aos projetos que a utilizam, como a inclusão de casos de teste não previstos pelos testadores do projeto, que permite que importantes casos de teste que não seriam executados sejam adicionados à suíte de testes do projeto, e a redução do esforço no desenvolvimento de casos de teste, pode permitir que o produto tenha uma maior qualidade na entrega, além da economia de recursos por parte das empresas. Esses benefícios aumentam com o passar do tempo, dado que a base para consulta do recomendador aumenta, e assim dados mais precisos são utilizados.

Capítulo 7

Considerações Finais

Os métodos ágeis vêm sendo utilizados com frequência cada vez maior nos projetos de desenvolvimento de software [94]. Esses métodos permitem que, com a ativa participação do cliente, os sistemas em desenvolvimento possam atender as necessidades do negócio e possibilitar o maior retorno do investimento possível, através da priorização das entregas com base no valor agregado pelas mesmas. Além disso, a auto-organização e empoderamento das equipes trazem uma maior autonomia e efeciência para o desenvolvimento, dada a capacidade técnica das equipes definirem a ordem das ações a serem executadas e análise da viabilidade das soluções propostas.

Essa agilidade no desenvolvimento e abertura a mudanças por parte da equipe de desenvolvimento faz com que a atividade de testes tenha uma profunda mudança em relação à mesma atividade nas metodologias tradicionais. Os curtos ciclos de desenvolvimento requerem uma alta execução de funcionalidades tanto no desenvolvimento, como nos testes, o que dificulta o gerenciamento das suítes de testes. Dadas as constantes mudanças no sistema em desenvolvimento, testes de regressão devem ser frequentemente executados a fim de garantir que as funcionalidades já desenvolvidas não sejam impactadas pelas alterações.

Em muitos casos, a participação do time do cliente é insuficiente, gerando assim o desenvolvimento de soluções que não atendem plenamente às necessidades dos usuários. Esse problema é agravado pela maior parte do conhecimento utilizado para o desenvolvimento de sistemas em métodos ágeis ser tácito, ou seja, os documentos escritos servem apenas para discussão entre os times e não como fonte de consulta para as decisões tomadas. A crescente demanda por profissionais de desenvolvimento tem feito com que as empresas contratem

cada vez mais pessoas com pouca experiência, o que pode ter um alto impacto na qualidade dos sistemas, dado que o desenvolvimento e execução de testes ágeis exigem um alto conhecimento por parte da equipe.

Assim, este trabalho se propõe a resolver o problema de como fornecer suporte automatizado aos times de desenvolvimento ágeis baseados em *Scrum* com o propósito de melhorar os casos de teste definidos e reduzir o esforço para implementação dos mesmos. Para isso, definimos uma abordagem para reutilizar casos de teste de projetos passados para recomendar casos de teste em projetos futuros, e assim obter as melhorias desejadas no desenvolvimento de casos de teste.

A solução proposta para reutilização de casos de teste em projetos ágeis foi um Sistema de Apoio à Decisão (SAD) baseado em Sistemas de Recomendação (SisRec). A proposta inicia com a instrumentação do Scrum, no sentido de que o dono do projeto não apenas defina os critérios de aceitação para uma Estória de Usuário, mas também os selecione a partir de uma base de dados pré-definida em conjunto com o time de desenvolvimento, incluindo também a categorização das Estórias de Usuário por parte do time de desenvolvimento com base na taxonomia definida pelo grupo *Intelligent Software Enginnering* (ISE), proporcionando assim uma forma para a comparação entre as estórias.

Dada a instrumentação do Scrum, desenvolvemos um SisRec que, ao se desenvolver uma nova Estória de Usuário, busca as estórias mais similares previamente cadastradas por projetos desenvolvidos anteriormente, ou até mesmo pelo projeto atual na base de dados da empresa, identifica os casos de teste associados a essas estórias e recomenda ao usuário do SisRec para que o mesmo possa definir quais desses casos de teste recomendados fazem sentido para a reutilização na estória em desenvolvimento. Mesmo reutilizando um caso de teste, o usuário ainda tem a opção de fazer as alterações necessárias no mesmo, a fim de o adequar ao contexto do projeto, sem perder a rastreabilidade para retroalimentação do sistema de recomendação. Isso significa que ao desenvolver novos casos de teste, esses já serão candidatos a serem reutilizados no caso da Estória de Usuário à qual pertencem ser similar às próximas a serem desenvolvidas. Para identificar a viabilidade da solução construímos uma ferramenta utilizando a abordagem definida chamada *TestRec*.

A validação da abordagem foi realizada através da utilização da ferramenta *TestRec* em três projetos que utilizam a metodologia ágil. A validação contou com a participação do

autor do trabalho executando a ferramenta, uma vez que a empresa na qual a validação foi executada possui uma ferramenta de gestão ágil própria. Em seguida, comparamos os casos de teste recomendados pela ferramenta com os desenvolvidos pelas equipes de teste dos projetos, e assim identificamos casos recomendados que foram desenvolvidos pela equipe de testes e casos de teste que não foram desenvolvidos, mas foram recomendados.

7.1 Contribuições

Seguindo a metodologia *Design Science* foram formuladas várias questões de pesquisa, que foram respondidas e deram origem outras questões, conforme fluxo definido no Capítulo 4. Dentre as questões definidas, as questões mais importantes e que guiaram a solução do problema de pesquisa foram as que seguem.

RQ01: Quais os principais desafios encontrados por testadores em projetos ágeis que impactam na qualidade dos testes executados e consequentemente na qualidade do produto final? Nessa questão, procuramos identificar se o problema de pesquisa identificado pelo autor a partir de experiências práticas em desenvolvimento de projetos ágeis tinham um respaldo acadêmico e em outros projetos de desenvolvimento reais. Para isso, utilizamos um mapeamento sistemático na literatura e um estudo de caso em uma empresa de desenvolvimento de software. Ambos os métodos empíricos convergiram nos desafios atuais do desenvolvimento de casos de teste em projetos ágeis, além de convergirem também com os problemas enfrentados pelos pesquisadores. Porém, segundo nosso estudo, a relevância dos desafios divergiram entre a academia e a indústria.

RQ02: É possível estruturar uma base de dados históricos sobre artefatos de projetos de forma a comparar itens visando recomendação? Uma vez que identificamos os desafios do desenvolvimento de casos de teste em projetos ágeis, buscamos uma forma de estruturar uma base de dados de projetos ágeis de forma a possibilitar a comparação entre Estórias de Usuários e assim poder recomendar casos de teste de estórias similares. Para isso, definimos uma taxonomia para classificação de Estórias de Usuário a partir da análise de 118 Estórias de Usuário de 5 projetos de empresas que utilizam métodos ágeis e validamos essa classificação utilizando um estudo de caso considerando dois projetos em desenvolvimento contendo um total de 59 Estórias de Usuário. Em termos de cobertura, conseguimos classifi-

car 90.17% das Estórias de Usuário e concluímos que é possível utilizar essa taxonomia para comparar Estórias de Usuário e assim ideitificar a similaridade entre elas.

RQ03: A solução proposta melhora o desenvolvimento de casos de teste em projetos ágeis? Dada a estruturação dos dados e a solução definida, nossa próxima questão se refere a identificar indícios de que ela fornece os benefícios para os quais ela foi preparada. Para isso, executamos um estudo de caso visando entender em que pontos a solução proposta melhora o desenvovlimento de casos de teste em projetos ágeis. Com base nas possibilidades de coleta de dados, o foco do estudo foi indentificar a possibilidade de aumento na suíte de testes, além de identificar a relevância dos mesmos; identificar a possibilidade e quantificar o reuso de casos de teste em projetos ágeis utilizando a abordagem; e por fim, identificar se os benefícios identificados evoluem com o tempo, uma vez que a medida que o SisRec é utilizado, sua base de treinamento se torna mais completa, e assim pode produzir um aumento significativo dos resultados obtidos.

Para validação, realizamos um estudo de caso utilizando três projetos nos quais, após a análise dos dados coletados, identificamos que do total de casos de teste desenvolvidos pelos projetos **65,09%** foram recomendados pela ferramenta, significando que esses poderiam ter sido reutilizados caso a abordagem fosse utilizada pelos projetos. Além disso, identificamos também que as suítes de testes dos projetos cresceram em **38,79%**, indicando que casos de testes de projetos passados não foram considerados nos projetos em andamento.

Assim, podemos listar como principais contribuições deste trabalho:

- O levantamento dos principais desafios no desenvolvimento de testes em projetos ágeis junto com a relevância de cada, validados pela academia e projetos reais que podem ser utilizados como forma de levantar riscos aos projetos de software nesse contexto e, consequentemente, devem requerer planos de mitigação relacionados.

- A taxonomia para classificação de Estórias de Usuário criada a partir de uma análise profunda de estórias desenvolvidas em projetos de desenvolvimento ágeis reais e também validada utilizando informações levantadas em empresas reais, junto com o processo para evolução, que a permite ser utilizada em outros domínios de aplicação.

- O sistema de recomendação que utiliza dados de Estórias de Usuário utilizando técnicas de codificação em conjunto com seu código fonte, que pode ser estendido e testado

com várias bases de dados que contemplem a estrutura definida.

- A base de dados para testes e produção, utilizando dados de projetos reais codificada e que pode ser utilizada como ponto de partida para utilização do recomendador em empresas que não possuem dados históricos formatados com a estrutura requerida pelo recomendador.

Tais contribuições têm potencial de impacto nos seguintes aspectos relacionados à Engenharia de Software:

- Diminuição do esforço de desenvolvimento de casos de teste, dado o alto percentual de reutilização obtido para projetos de desenvolvimento de software que utilizam metodologias ágeis.

- Aumento da completude da suíte de testes a partir da seleção de casos de teste executados em Estórias de Usuário semelhantes às que estão sendo desenvolvidas, que pode trazer uma melhora na qualidade para o software em desenvolvimento.

- A reutilização de casos de teste comuns às estórias semelhantes permite que os testadores possam focar em atividades de teste exploratórias, podendo assim encontrar erros simulando a utilização por parte de usuários comuns e, consequentemente, melhorando a qualidade do software em desenvolvimento.

- O suporte aos testadores menos experientes, que, uma vez que possuem pouco conhecimento em testes ágeis e no negócio, podem utilizar as recomendações de casos de teste como base para manter a qualidade da suíte.

7.2 Trabalhos futuros

Além das contribuições mencionadas na seção anterior, este trabalho também abre possibilidades de extensão, além de alguns pontos de melhoria. Uma das limitações atuais da abordagem é considerar apenas uma classificação para cada Estória do Usuário. Essa limitação nos obriga a separar uma estória que contempla as funcionalidades de listagem, criação, edição e exclusão de uma entidade do sistema em quatro estórias distintas, dado que essa estória pode

ser classificada com apenas um item da taxonomia definida. Assim, uma possível solução para essa limitação é a possibilidade de registro de mais de uma categoria para a estória, que, no exemplo citado teria quatro classificações. Essa solução demanda um alto esforço para análise e retrabalho no sistema de recomendação, por essa razão, não se encontra contemplado neste trabalho. Além disso, o desenvolvimento e utilização de uma ontologia [63] deve ser considerada para facilitar a classificação e busca das Estórias de Usuário desenvolvidas.

A recomendação descrita neste trabalho é realizada a partir da seleção de Estórias de Usuário em desenvolvimento. Porém, o nosso processo permite que o usuário crie novos casos de teste uma vez que não encontre os casos de teste desejados durante a recomendação. No desenvolvimento de casos de teste, existe a possibilidade do usuário registrar um caso de teste existente na base de dados da empresa, o que favorece a duplicação de registros. No estado atual, a mitigação do risco de duplicação de registros é sanada pela verificação humana da duplicação de registros durante o cadastro de novos de teste, o que exige esforço do time de testes e é suscetível a erros. Nesse sentido, uma possível solução é a recomendção de casos de teste não só através de uma Estória de Usuário, mas também através da comparação entre casos de teste semelhantes, fazendo com que, ao cadastrar um novo caso de teste, seja possível o SisRec recomendar casos de teste semelhantes ao desenvolvido, e assim evitar com que casos de teste sejam duplicados de forma automatizada, eliminando a possibilidade de erros e duplicação.

O trabalho atual foca na reutilização de registro de casos de teste para que os usuários possam executar ao término do desenvolvimento de Estórias de Usuário. Porém, como mencionado na Seção 2.3.1, é extremamente desejável que os testes de aceitação sejam automatizados, o que favorece testes de regressão nos sistemas em desenvolvimento e o foco dos testadores em testes exploratórios. Assim, dado que o sistema de recomendação favorece a reutilização de casos de teste de aceitação e que estes estejam automatizados, existe a possibilidade de estendermos a solução para permitir que o usuário reutilize também o código de automação dos casos de teste, o que possibilita ainda mais a redução de esforço para o time de testes e, consequentemente, a economia de recursos por parte da empresa de desenvolvimento de software.

A ferramenta *TestRec*, desenvolvida com o objetivo de identificar a viabilidade da abordagem, é um sistema para gerenciamento de projetos ágeis. Através dela, pudemos observar

o comportamento da abordagem definida integrada a um sistema real, e assim realizar os ajustes necessários para essa adequação. Porém, comparada às ferramentas disponíveis atualmente, a ferramenta *TestRec* é bastante simples e fornece poucos recursos. Assim, temos como objetivo futuro a integração do mecanismo de recomendação desenvolvido em um sistema de gestão ágil existente, a fim de acrescentar essa funcionalidade e possibilitar que times de desenvolvimento se beneficiem desse recurso. Para isso, se faz necessária a análise de impacto da adequação das ferramentas existentes à adoção da taxonomia definida e os ajustes em termos de interface gráfica que possibilitem e maximizem os benefícios da abordagem.

Bibliografia

[1] Ieee standard for software verification and validation. *IEEE Std 1012-2004 (Revision of IEEE Std 1012-1998)*, pages 1–110, 2005.

[2] Challenges in adapting agile testing in a legacy product. *Proceedings - 11th IEEE International Conference on Global Software Engineering, ICGSE 2016*, pages 104–108, 2016.

[3] Gediminas Adomavicius, Nikos Manouselis, and YoungOk Kwon. *Multi-Criteria Recommender Systems*, pages 769–803. Springer US, Boston, MA, 2011.

[4] Gediminas Adomavicius and Alexander Tuzhilin. Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions. *IEEE Transactions on Knowledge and Data Engineering*, 17(6):734–749, 2005.

[5] Charu C. Aggarwal, Alexander Hinneburg, and Daniel A. Keim. On the surprising behavior of distance metrics in high dimensional space. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 1973:420–434, 2001.

[6] Xavier Amatriain, Alejandro Jaimes*, Nuria Oliver, and Josep M. Pujol. *Data Mining Methods for Recommender Systems*, pages 39–71. Springer US, Boston, MA, 2011.

[7] Tulasi Anand and V. S. Mani. Practices to make agile test teams effective: Challenges and solutions. *Proceedings - 2015 IEEE 10th International Conference on Global Software Engineering Workshops, ICGSEW 2015*, pages 7–11, 2015.

[8] Yanti Andriyani, Rashina Hoda, and Robert Amor. Understanding knowledge ma-

nagement in agile software development practice. In *International Conference on Knowledge Science, Engineering and Management*, pages 195–207. Springer, 2017.

[9] Victor R Basili, Gianluigi Caldiera, and H Dieter Rombach. The goal question metric approach. *Encyclopedia of Software Engineering*, 2:528–532, 1994.

[10] Beck. *Test Driven Development: By Example*. Addison-Wesley Longman Publishing Co., Inc., USA, 2002.

[11] Kent Beck. *Extreme Programming Explained: Embrace Change*. Addison-Wesley Longman Publishing Co., Inc., USA, 1999.

[12] Kent Beck, Mike Beedle, Arie van Bennekum, Alistair Cockburn, Ward Cunningham, Martin Fowler, James Grenning, Jim Highsmith, Andrew Hunt, Ron Jeffries, Jon Kern, Brian Marick, Robert C. Martin, Steve Mellor, Ken Schwaber, Jeff Sutherland, and Dave Thomas. Manifesto for agile software development, 2001.

[13] Palash Bera and Abhimanyu Gupta. A proposed method in agile practices to create requirements documentation and test cases. In *CEUR Workshop Proceedings*, volume 1859, pages 113–121, 2017.

[14] Elizabeth Bjarnason, Michael Unterkalmsteiner, Emelie Engström, and Marcus Borg. An Industrial Case Study on Test Cases as Requirements. *Lecture Notes in Business Information Processing*, 212:27–39, 2015.

[15] Pierre Bourque and Richard E. Fairley. *Guide to the Software Engineering - Body of Knowledge.* 2014.

[16] Lizhi Cai, Weiqin Tong, Zhenyu Liu, and Juan Zhang. Test case reuse based on ontology. *2009 15th IEEE Pacific Rim International Symposium on Dependable Computing, PRDC 2009*, pages 103–108, 2009.

[17] Òscar Celma. *Music Recommendation*, pages 43–85. Springer Berlin Heidelberg, Berlin, Heidelberg, 2010.

[18] Heng Tze Cheng, Levent Koc, Jeremiah Harmsen, Tal Shaked, Tushar Chandra, Hrishi Aradhye, Glen Anderson, Greg Corrado, Wei Chai, Mustafa Ispir, Rohan Anil, Zakaria Haque, Lichan Hong, Vihan Jain, Xiaobing Liu, and Hemal Shah. Wide deep learning for recommender systems. *ACM International Conference Proceeding Series*, 15-September-2016:7–10, 2016.

[19] Peter Coad, Jeff de Luca, and Eric Lefebvre. *Java Modeling Color with Uml: Enterprise Components and Process with Cdrom*. Prentice Hall PTR, USA, 1st edition, 1999.

[20] Alistair Cockburn. *Crystal Clear a Human-Powered Methodology for Small Teams*. Addison-Wesley Professional, first edition, 2004.

[21] David Cohen, Mikael Lindvall, and Patricia Costa. An Introduction to Agile Methods. *Advances in Computers*, 62(C):1–66, 2004.

[22] M. Cohn. *User Stories Applied: For Agile Software Development*. Addison-Wesley Signature Series (Beck). Pearson Education, 2004.

[23] Eliane Figueiredo Collins and Vicente Ferreira De Lucena. Software test automation practices in agile development environment: An industry experience report. *2012 7th International Workshop on Automation of Software Test, AST 2012 - Proceedings*, pages 57–63, 2012.

[24] Ronan Collobert, Jason Weston, Léon Bottou, Michael Karlen, Koray Kavukcuoglu, and Pavel Kuksa. Natural language processing (almost) from scratch. *J. Mach. Learn. Res.*, 12(null):2493–2537, November 2011.

[25] L. Crispin and J. Gregory. *Agile Testing: A Practical Guide for Testers and Agile Teams*. Addison-Wesley Signature Series (Cohn). Pearson Education, 2008.

[26] Lisa Crispin. Agile test quadrant, 2011.

[27] D. S. Cruzes and T. Dyba. Recommended Steps for Thematic Synthesis in Software Engineering. (7491):275–284, 2011.

[28] Daniela S. Cruzes and Tore Dybå. Research synthesis in software engineering: A tertiary study. *Information and Software Technology*, 53(5):440 – 455, 2011. Special Section on Best Papers from XP2010.

[29] Emanuel Dantas, Alexandre Costa, Marcus Vinicius, Mirko Perkusich, Hyggo Almeida, and Angelo Perkusich. An effort estimation support tool for agile software development: An empirical evaluation. pages 82–87, 07 2019.

[30] Janez Demšar. Statistical comparisons of classifiers over multiple data sets. *Journal of Machine Learning Research*, 7:1–30, 2006.

[31] Christian Desrosiers and George Karypis. *A Comprehensive Survey of Neighborhood-based Recommendation Methods*, pages 107–144. Springer US, Boston, MA, 2011.

[32] E. Dilorenzo, E. Dantas, M. Perkusich, F. Ramos, A. Costa, D. Albuquerque, H. Almeida, and A. Perkusich. Enabling the reuse of software development assets through a taxonomy for user stories. *IEEE Access*, 8:107285–107300, 2020.

[33] Andreia M. Dos Santos, Borje F. Karlsson, André M. Cavalcante, Igor B. Correia, and Emanuel Silva. Testing in an agile product development environment: An industry experience report. *LATW 2011 - 12th IEEE Latin-American Test Workshop*, 2011.

[34] C Ernani. Automated Acceptance Tests as Software Requirements: An Experiment to Compare the Applicability of Fit Tables and Gherkin Language. 149:104–119, 2018.

[35] B. Kantor F. Ricci, L. Rokach, B. Shapira. *Recommnender System Handbook*. 2010.

[36] Zeno Gantner, Lucas Drumond, Christoph Freudenthaler, Steffen Rendle, and Lars Schmidt-Thieme. Learning attribute-to-feature mappings for cold-start recommendations. *Proceedings - IEEE International Conference on Data Mining, ICDM*, pages 176–185, 2010.

[37] Salvador García, Alberto Fernández, Julián Luengo, and Francisco Herrera. Advanced nonparametric tests for multiple comparisons in the design of experiments in computational intelligence and data mining: Experimental analysis of power. *Information Sciences*, 180(10):2044–2064, 2010.

[38] Marko Gasparic and Andrea Janes. What recommendation systems for software engineering recommend: A systematic literature review. *Journal of Systems and Software*, 113:101–113, 2016.

[39] J. Ge and J. Liu. Software test cases recommendation system research based on collaborative filtering. In *2016 17th IEEE/ACIS International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing (SNPD)*, pages 557–562, 2016.

[40] Janet Gregory and Lisa Crispin. *More Agile Testing: Learning Journeys for the Whole Team*. Addison-Wesley Professional, 1st edition, 2014.

[41] Markus Grtner and Markus Grtner. *ATDD by Example: A Practical Guide to Acceptance Test-Driven Development*. Addison-Wesley Professional, 1st edition, 2012.

[42] Asela Gunawardana and Guy Shani. A survey of accuracy evaluation metrics of recommendation tasks. *The Journal of Machine Learning Research*, 10:2935–2962, 2009.

[43] Shaojie Guo, Juan Zhang, Weiqin Tong, and Zongheng Liu. An application of ontology to test case reuse. *Proceedings 2011 International Conference on Mechatronic Science, Electric Engineering and Computer, MEC 2011*, pages 775–778, 2011.

[44] Di Han, Jianqing Li, Lei Yang, and Zihua Zeng. A recommender system to address the Cold Start problem for App usage prediction. *International Journal of Machine Learning and Cybernetics*, 10(9):2257–2268, 2019.

[45] David Hand and Peter Christen. A note on using the F-measure for evaluating record linkage algorithms. *Statistics and Computing*, 28(3):539–547, 2018.

[46] J. L. Herlocker, J. A. Konstan, and J. Riedl. Explaining collaborative filtering recommendations. *Proceedings of the ACM Conference on Computer Supported Cooperative Work*, pages 241–250, 2000.

[47] Jonathan L Herlocker, Joseph A Konstan, Loren G Terveen, and John T Riedl. <Evaluating Collaborative Filtering.pdf>. 22(1):5–53, 2004.

[48] Rashina Hoda, Norsaremah Salleh, and John Grundy. The rise and evolution of agile software development. *IEEE Software*, 35(5):58–63, 2018.

[49] Sofija Hotomski, Eya Ben Charrada, and Martin Glinz. Keeping evolving requirements and acceptance tests aligned with automatically generated guidance. In Erik Kamsties, Jennifer Horkoff, and Fabiano Dalpiaz, editors, *Requirements Engineering: Foundation for Software Quality*, pages 247–264, Cham, 2018. Springer International Publishing.

[50] Sofija Hotomski, Eya Ben Charrada, and Martin Glinz. An Exploratory Study on Handling Requirements and Acceptance Test Documentation in Industry. In *Proceedings - 2016 IEEE 24th International Requirements Engineering Conference, RE 2016*, pages 116–125, 2016.

[51] Jezz Humble and David Farley. *Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation*. 2010.

[52] Irum Inayat, Siti Salwah Salim, Sabrina Marczak, Maya Daneva, and Shahaboddin Shamshirband. A systematic literature review on agile requirements engineering practices and challenges. *Computers in human behavior*, 51:915–929, 2015.

[53] Werner Janjic and Colin Atkinson. Utilizing software reuse experience for automated test recommendation. *2013 8th International Workshop on Automation of Software Test, AST 2013 - Proceedings*, pages 100–106, 2013.

[54] Ivan Jovanovikj, Marvin Grieger, and Enes Yigitbas. Towards a Model-Driven Method for Reusing Test Cases in Software Migration Projects. *Softwaretechnik-Trends, Proceedings of the 18th Workshop Software-Reengineering Evolution (WSRE) 7th Workshop Design for Future (DFF)*, 32(2):65–66, 2016.

[55] Giuseppe Jurman, Samantha Riccadonna, Roberto Visintainer, and Cesare Furlanello. Canberra distance on ranked lists. In *Proceedings of Advances in Ranking NIPS 09 Workshop*, pages 22–27. Citeseer, 2009.

[56] Frank Kane. *Building Recommender Systems with Machine Learning and AI: Help*

People Discover New Products and Content with Deep Learning, Neural Networks, and Machine Learning Recommendations. Independently published, 2018.

[57] Mathias Landhäußer and Adrian Genaid. Connecting User Stories and code for test development. *2012 3rd International Workshop on Recommendation Systems for Software Engineering, RSSE 2012 - Proceedings*, pages 33–37, 2012.

[58] Casper Lassenius, Torgeir Dingsøyr, and Maria Paasivaara. Agile processes, in software engineering, and extreme programming: 16th international conference, XP 2015 Helsinki, Finland, may 25-29, 2015 proceedings. *Lecture Notes in Business Information Processing*, 212:27–39, 2015.

[59] Ya Jun Leng, Qing Lu, and Chang Yong Liang. Survey of recommendation based on collaborative filtering. *Moshi Shibie yu Rengong Zhineng/Pattern Recognition and Artificial Intelligence*, 27(8):720–734, 2014.

[60] Xuexiang Li and Wenning Zhang. Ontology-based testing platform for reusing. *Proceedings - 6th International Conference on Internet Computing for Science and Engineering, ICICSE 2012*, pages 86–89, 2012.

[61] Greg Linden, Brent Smith, and Jeremy York. Amazon.com recommendations: Item-to-item collaborative filtering. *IEEE Internet Computing*, 7(1):76–80, 2003.

[62] Mikael Lindvall, Vic Basili, Barry Boehm, Patricia Costa, Kathleen Dangle, Forrest Shull, Roseanne Tesoriero, Laurie Williams, and Marvin Zelkowitz. Empirical findings in agile methods. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 2418:197–207, 2002.

[63] Alexander Maedche and Steffen Staab. *Ontology Learning*, pages 173–190. Springer Berlin Heidelberg, Berlin, Heidelberg, 2004.

[64] Mika V. Mäntylä, Bram Adams, Foutse Khomh, Emelie Engström, and Kai Petersen. On rapid releases and software testing: a case study and a semi-systematic literature review. *Empirical Software Engineering*, 20(5):1384–1425, 2015.

[65] Brian Marick. Agile test quadrant, 2003.

[66] Grigori Melnik, Kris Read, and Frank Maurer. Suitability of FIT user acceptance tests for specifying functional requirements: Developer perspective. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 3134:60–72, 2004.

[67] Mathias Meyer. Continuous integration and its tools. *IEEE Software*, 31(3):14–16, 2014.

[68] Emmanuel O.C. Mkpojiogu, Nor Lailyhashim, Abdullah Al-Sakkaf, and Azham Hussain. Software startups: Motivations for agile adoption. *International Journal of Innovative Technology and Exploring Engineering*, 8(8 S):454–459, 2019.

[69] Nornadiah Mohd Razali and Yap Bee Wah. Power comparisons of Shapiro-Wilk, Kolmogorov-Smirnov, Lilliefors and Anderson-Darling tests. *Journal of Statistical Modeling and Analytics*, 2(1):21–33, 2011.

[70] David Musicant, Vipin Kumar, and Aysel Ozgur. Optimizing f-measure with support vector machines. 03 2003.

[71] Padmaraj Nidagundi and Leonids Novickis. Introducing Lean Canvas Model Adaptation in the Scrum Software Testing. *Procedia Computer Science*, 104(December 2016):97–103, 2016.

[72] Padmaraj Nidagundi and Leonids Novickis. New method for mobile application testing using lean canvas to improving the test strategy. *2017 12th International Scientific and Technical Conference on Computer Sciences and Information Technologies (CSIT)*, pages 171–174, 2017.

[73] Padmaraj Nidagundi and Leonids Novickis. Possibilities about the design Lean canvas model and its adaptation in the agile testing. *CEUR Workshop Proceedings*, 1853:20–23, 2017.

[74] I. Otaduy and O. Diaz. User acceptance testing for Agile-developed web-based applications: Empowering customers through wikis and mind maps. *Journal of Systems and Software*, 133:212–229, 2017.

[75] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[76] Kai Petersen, Robert Feldt, Shahid Mujtaba, and Michael Mattsson. Systematic mapping studies in software engineering. In *Proceedings of the 12th International Conference on Evaluation and Assessment in Software Engineering*, EASE'08, pages 68–77, Swindon, UK, 2008. BCS Learning & Development Ltd.

[77] Daniel J. Power. Decision support systems: Concepts and resources for managers. 2002.

[78] M Rahman, J Gao, and Ieee. A Reusable Automated Acceptance Testing Architecture for Microservices in Behavior-Driven Development. *9th IEEE International Symposium on Service-Oriented System Engineering*, (May):321–325, 2015.

[79] Felipe Ramos, Alexandre Costa, Mirko Perkusich, Hyggo Almeida, and Angelo Perkusich. A non-functional requirements recommendation system for scrum-based projects. *Proceedings of the International Conference on Software Engineering and Knowledge Engineering, SEKE*, 2018-July(July):149–154, 2018.

[80] Felipe Ramos, Antonio Pedro, Marcos Cesar, Alexandre Costa, Mirko Perkusich, Hyggo Almeida, and Angelo Perkusich. Evaluating software developers' acceptance of a tool for supporting agile non-functional requirement elicitation. pages 26–31, 07 2019.

[81] Avi Rana and K. Deeba. Online book recommendation system using collaborative filtering (with jaccard similarity). *Journal of Physics: Conference Series*, 1362(1), 2019.

[82] Filippo Ricca, Marco Torchiano, Massimiliano Di Penta, Mariano Ceccato, and Paolo Tonella. Using acceptance tests as a support for clarifying requirements: A series of experiments. *Information and Software Technology*, 51(2):270–283, 2009.

[83] Barbara Roberts. *Dynamic Systems Development Method, The Standard for Rapid Application Development*, pages 16–33. Springer Berlin Heidelberg, Berlin, Heidelberg, 2001.

[84] S. Rose, M. Wynne, and A. Hellesøy. *The Cucumber for Java Book: Behaviour-driven Development for Testers and Developers*. Pragmatic programmers. Pragmatic Bookshelf, 2015.

[85] Kenneth A Ross. Cache-Aware Query Processing Cache-Conscious Query Processing. 2009.

[86] Kenneth S. Rubin. *Essential Scrum: A Practical Guide to the Most Popular Agile Process*. Addison-Wesley Professional, 1st edition, 2012.

[87] Per Runeson and Martin Höst. Guidelines for conducting and reporting case study research in software engineering. *Empirical Software Engineering*, 14(2):131–164, 2009.

[88] J. Ben Schafer, Joseph A. Konstan, and John Riedl. E-commerce recommendation applications. *Data Mining and Knowledge Discovery*, 5(1-2):115–153, 2001.

[89] Andrew I. Schein, Alexandrin Popescul, Lyle H. Ungar, and David M. Pennock. Methods and metrics for cold-start recommendations. *SIGIR Forum (ACM Special Interest Group on Information Retrieval)*, pages 253–260, 2002.

[90] Ken Schwaber. SCRUM Development Process. *Business Object Design and Implementation*, (April 1987):117–134, 1997.

[91] Ken Schwaber. *Agile Project Management with Scrum*. Microsoft Press, Redmond, WA, 2004.

[92] Ken Schwaber and Jeff Sutherland. The scrum guide. *Scrum Alliance*, 2016.

[93] James Seth and Henry Sidgwick. Toward the Next Generation of Recommender Systems: A Survey of the State-of-the-Art and Possible Extensions. *The Philosophical Review*, 12(5):548, 1903.

[94] Ahmed Sidky, James Arthur, and Shawn Bohner. A disciplined approach to adopting agile practices: The agile adoption framework. *Innovations in Systems and Software Engineering*, 3(3):203–216, 2007.

[95] Ana Silva, Thalles Araújo, João Nunes, Mirko Perkusich, Ednaldo Dilorenzo, Hyggo Almeida, and Angelo Perkusich. A systematic review on the use of definition of done on agile software development projects. In *Proceedings of the 21st International Conference on Evaluation and Assessment in Software Engineering*, EASE'17, page 364–373, New York, NY, USA, 2017. Association for Computing Machinery.

[96] Sanjeevan Sivapalan, Alireza Sadeghian, Hossein Rahnama, and Asad M. Madni. Recommender systems in e-commerce. *World Automation Congress Proceedings*, pages 179–184, 2014.

[97] Marian STOICA, Marinela MIRCEA, and Bogdan GHILIC-MICU. Software Development: Agile vs. Traditional. *Informatica Economica*, 17(4/2013):64–76, 2013.

[98] Goparaju Purna Sudhakar. A model of critical success factors for software projects. *Journal of Enterprise Information Management*, 25(6):537–558, 2012.

[99] Hirotaka Takeuchi and Ikujiro Nonaka. The new new product development game. *Harvard Business Review*, 1986.

[100] Muhammad Usman, Ricardo Britto, Jürgen Börstler, and Emilia Mendes. Taxonomies in software engineering: A Systematic mapping study and a revised taxonomy development method. *Information and Software Technology*, 85:43–59, 2017.

[101] Marco Vanetti, Elisabetta Binaghi, Barbara Carminati, Moreno Carullo, and Elena Ferrari. Content-based filtering in on-line social networks. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 6549 LNAI:127–140, 2011.

[102] VersionOne. 13th annual state of agile development survey results. https://www.stateofagile.com/#ufh-i-521251909-13th-annual-state-of-agile-report/473508, 2019. Accessed em: 07-01-2019.

[103] A. von Mayrhauser, Richard Mraz, Je Walls, and Pete Ocken. Domain based testing: increasing test case reuse. *Proceedings 1994 IEEE International Conference on Computer Design: VLSI in Computers and Processors*, pages 484–491, 1993.

[104] Xinyu Wang, Liping Zhao, Ye Wang, and Jie Sun. The role of requirements engineering practices in agile development: an empirical study. In *Requirements Engineering*, pages 195–209. Springer, 2014.

[105] Jian Wei, Jianhua He, Kai Chen, Yi Zhou, and Zuoyin Tang. Collaborative filtering and deep learning based recommendation system for cold start items. *Expert Systems with Applications*, 69:1339–1351, 2017.

[106] Roel Wieringa. *Design Science Methodology for Information Systems and Software Engineering*. 2014.

[107] Claes Wohlin, Per Runeson, Martin Höst, Magnus C. Ohlsson, Björn Regnell, and Anders Wesslén. *Experimentation in software engineering*, volume 9783642290. 2012.

Apêndice A

Questionário de Entrevista Semi-Estruturada de Fatores que Influenciam Definição e Reuso de Casos de Teste

A.1 Quantos anos de experiência em desenvolvimento de projetos você possui?

A.2 Quantos anos de experiência em testes ágeis você possui?

A.3 Qual o seu nível de escolaridade?

A.4 Quais fatores e artefatos influenciam o desenvolvimento de casos de teste em projetos ágeis?

A.5 Você constuma reutilizar casos de teste no desenvolvimento de novos? Se sim, Quais fatores e artefatos influenciam o reuso de casos de teste em projetos ágeis?

Apêndice B

Manual do Usuário da Ferramenta TestRec

B.1 Visão Geral da Ferramenta

O TestRec (Test Case Recommender) é uma aplicação cujo objetivo é auxiliar testadores em projetos ágeis, através da recomendação de casos de teste previamente desenvolvidos e associados à Estórias de Usuário. A ferramenta possui as seguintes funcionalidades: cadastro de projetos, cadastro de Estórias de Usuário pra um projeto com critérios de aceitação associados, cadastro de Sprints, planejamento de sprints, cadastro de requisitos não funcionais e cadastro de casos de teste através da reutilização de recomendação ou manualmente.

B.2 Cadastro de Projeto

Para que seja possível gerenciar projetos, primeiramente é necessário seu cadastro. Ao fazer o login no sistema, o usuário tem a opção de listar os projetos cadastrados, conforme descrito na Figura B.1. Ainda na tela de listagem, o usuário tem as opções de cadastrar um novo projeto e editar os projetos listados, além de entrar em algum dos projetos listados. A listagem dos projetos informa o identificador (ID), nome, data de início e data de término de cada projeto.

Ao clicar no botão *Novo* na tela de listagem de projetos, o usuário tem a opção de cadastrar um novo projeto. Ao iniciar a funcionalidade de cadastro de novo projeto, a tela

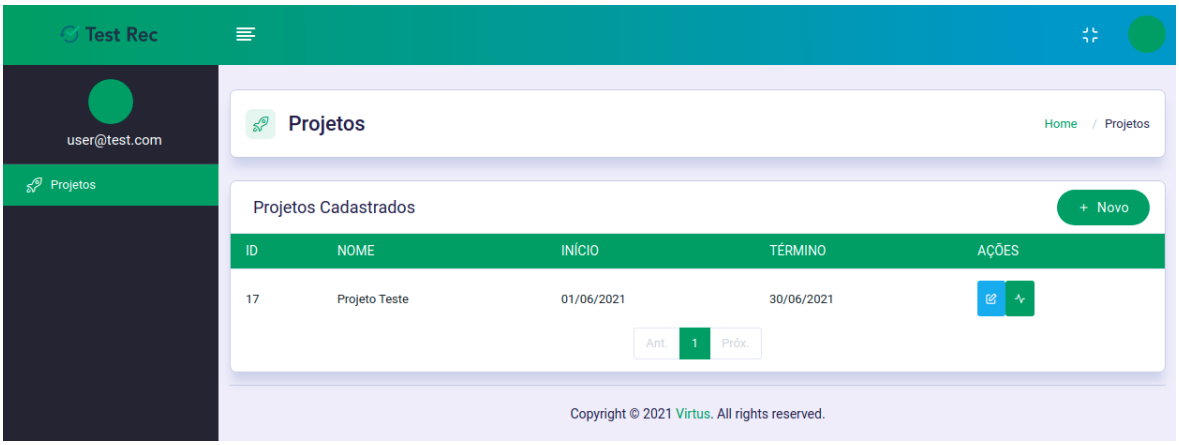

Figura B.1: Tela de listagem de projetos cadastrados.

de cadastro de projetos é exibida conforme apresentado na Figura B.2. Para o cadastro de projetos o usuário deverá informar o nome do projeto, descrição, data inicial e data final do mesmo. Após todos os dados informados, o usuário deverá clicar no botão *Salvar* e aguardar a mensagem *"Projeto salvo com sucesso"*. Caso o projeto tenha sido registrado, a tela de listagem de projetos deverá exibir os dados do novo projeto.

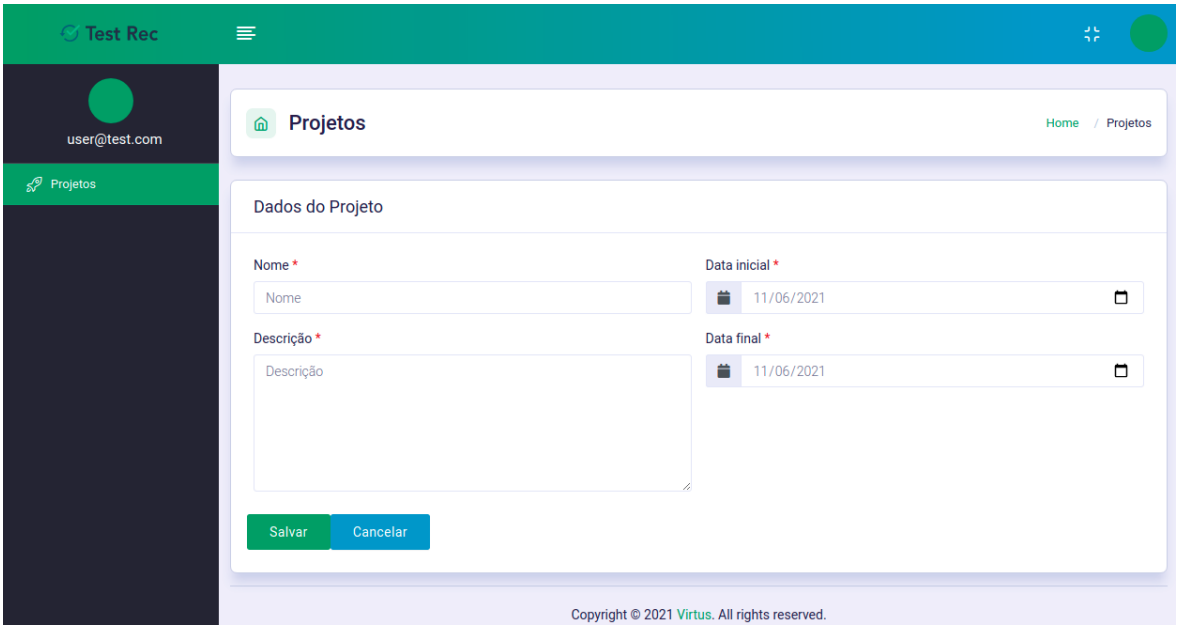

Figura B.2: Tela de cadastro de projetos.

Ainda na tela de listagem de projetos, a coluna ações representa as ações que usuário poderá tomar em relação a cada projeto. O primeiro ícone – na forma de caneta – permite que o usuário possa editar as informações do projeto cadastrado. Ao clicar nesse ícone a tela

apresentada na Figura B.3 é exibida contendo os dados do projeto. Assim, o usuário poderá alterar qualquer um dos dados do projeto e em seguida clicar salvar. Ao clicar em salvar, a mensagem *"Projeto atualizado com sucesso"* é exibida e a tela de listagem de projetos novamente é exibida com os dados do projeto atualizados.

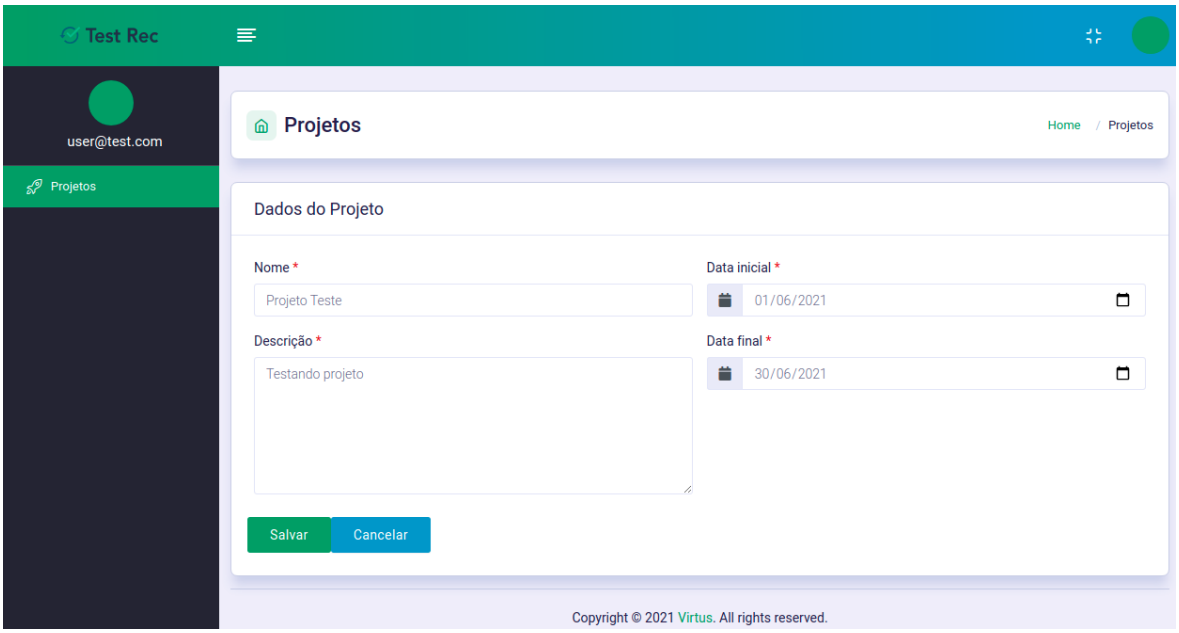

Figura B.3: Tela de edição de projetos.

B.3 Dashboard do Projeto

Após o cadastro do projeto, é possível acessar os dados de um projeto ao clicar no segundo link da listagem de projetos. Ao entrar no projeto, é possível vermos os itens do projeto através do novo menu. A primeira tela do projeto é o *Dashboard*, onde é exibida a *Sprint* em andamento e os casos de teste que já foram recomendados para o projeto como é apresentado na Figura B.4.

B.4 Cadastro de Estórias de Usuário

Para o gerenciamento ágil de um projeto, os requisitos são registrados como Estórias de Usuário. A Figura B.5 mostra a listagem das Estórias de Usuário cadastradas para o projeto em questão. Nela, além do identificador e do nome, podemos ver também a estimativa da

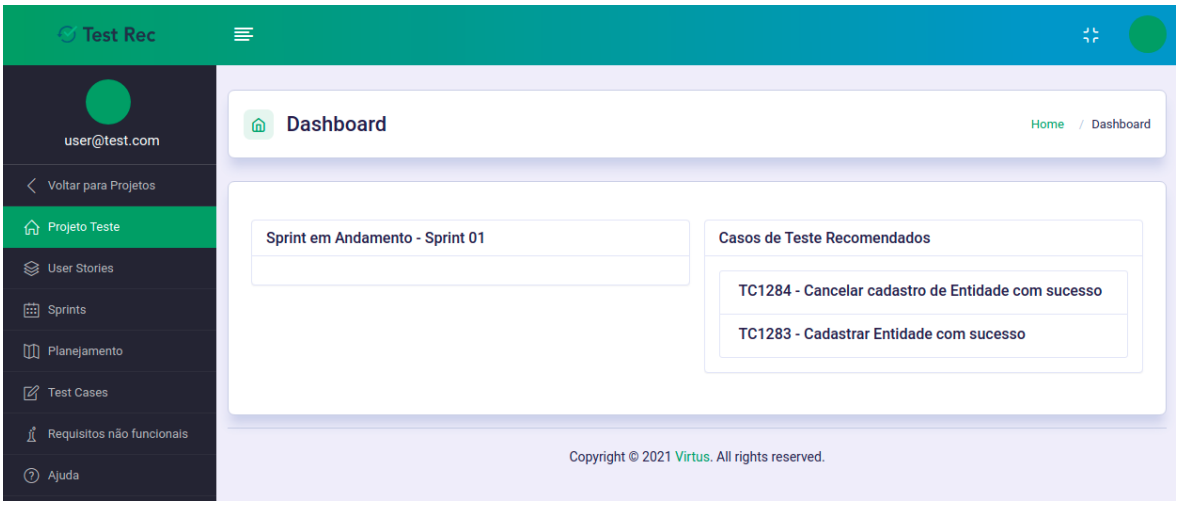

Figura B.4: Dashboard do projeto.

mesma em pontos. Além disso, na referida tela, o usuário poderá cadastrar uma nova estória clicando no botão *Nova*, editar ou excluir uma estória listada, a partir dos botões na coluna ações.

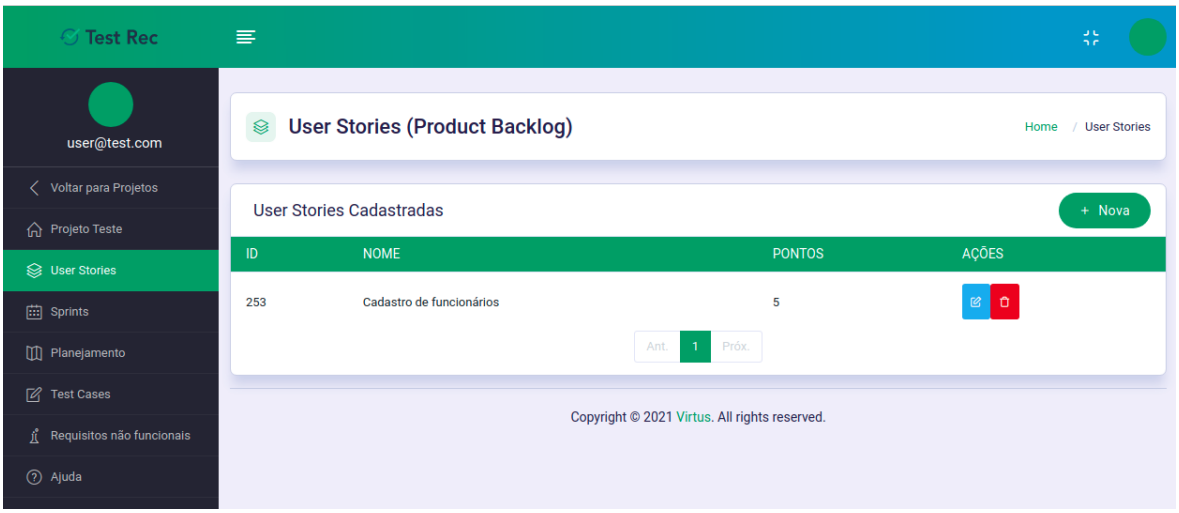

Figura B.5: Tela de listagem de Estórias de Usuário cadastradas para um projeto.

Ao clicar no botão *Nova*, a tela de cadastro de Estórias de Usuário é exibida conforme apresentado na Figura B.6. Nela, o usuário do sistema deverá informar o título da estória, a descrição da mesma, o módulo e operação, conforme taxonomia descrita no Capítulo 4 deste trabalho, a plataforma em que será desenvolvida a estória (Web, Mobile ou Desktop), o tipo de estória, sendo *Técnica* para atividades internas ao sistema e *Negócio* para atividades que agregam valor ao usuário, o status da estória e, opcionalmente, a estimativa em pontos de

estória. Além disso, o usuário poderá informar critérios de aceitação utilizados em outros projetos, digitando a descrição do mesmo no campo de seleção, ou adicionar novos critérios clicando no botão "+". Os critério de aceitação também podem ser removidos e um mesmo critério previamente cadastrado não poderá ser adicionar a estória mais de uma vez. Ao preencher os campos o usúario deverá clicar no botão *Salvar* caso queira confirmar a inclusão ou *Cancelar* caso não queira persistir as informações.

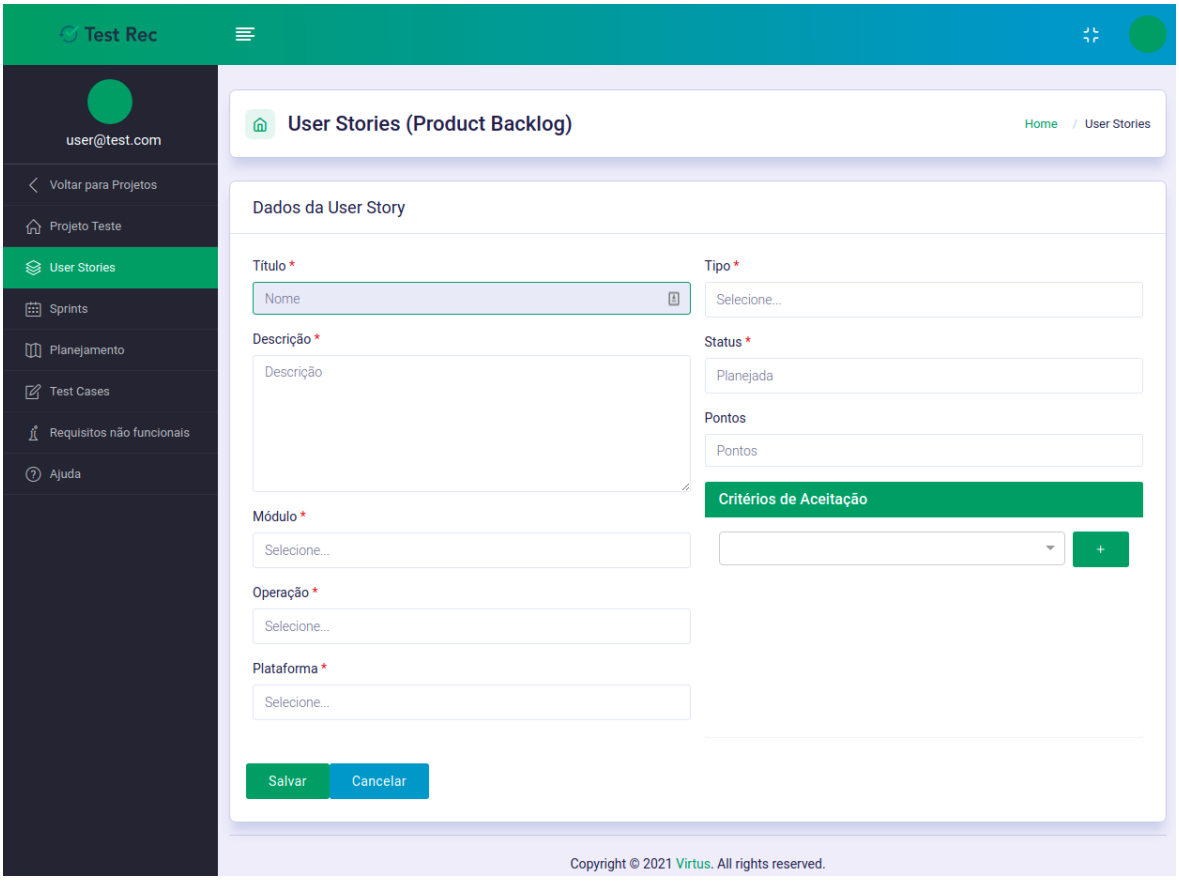

Figura B.6: Tela de cadastro de Estória de Usuário.

Na listagem das Estórias de Usuário, o usuário do sistema poderá editar qualquer estória cadastrada para o projeto clicando no botão de edição na coluna ações. Ao clicar nesse botão, a tela de edição de estória é exibida conforme conforme apresentado na Figura B.7. Nela, os dados da estória de usuário podem ser alterados, e critérios de aceitação podem ser adicionados ou retirados da estória. Ao finalizar as alterações, o usuário deverá clicar no botão *Salvar* para confirmar as mesmas, ou *Cancelar* para evitar a persistência dos dados alterados.

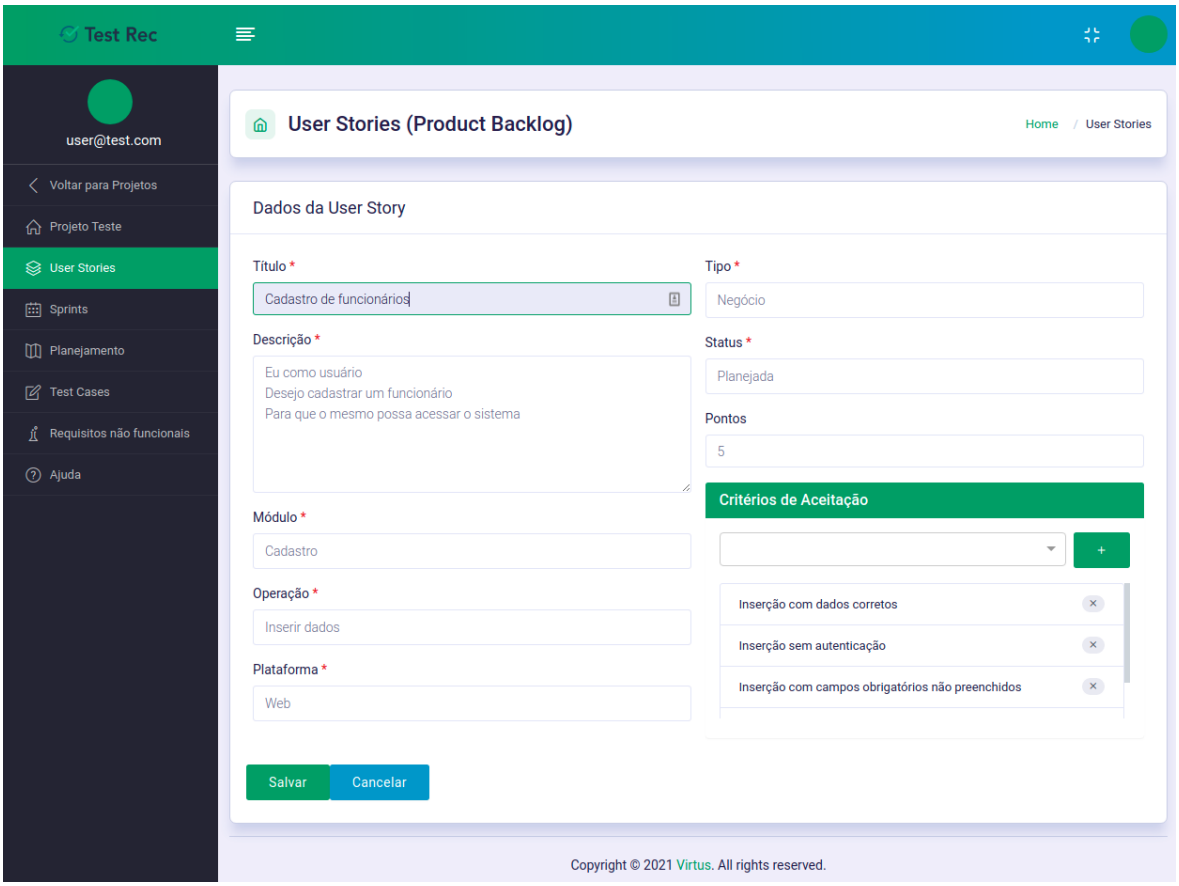

Figura B.7: Tela de edição de Estória de Usuário.

B.5 Cadastro de Sprints

Dado o período do projeto, registrado no cadastro de projetos, o usuário do sistema poderá registrar as iterações do projeto na forma de sprits. A Figura B.8 mostra a tela de listagem das sprints cadastradas para o projeto em questão. Nela, o identificador (ID), nome, data de início, data de término e status da Sprints são exibidos. Além disso, a coluna ações possui os botões relativos às ações que o usuário poderá executar sobre cada Sprint, que são a edição e exclusão.

Ainda na listagem de Sprints, caso o usuário queira registrar uma nova Sprint, o mesmo deverá clicar no botão *Nova*. Ao fazer isso, a tela de cadastro de sprints é exibida, conforme apresentado na Figura B.9. Nela, o usuário deverá infomar o nome da Sprint, o objetivo da mesma, o status, a data inicial e data final. É importante frisar que, em relação ao Status, apenas uma sprint pode ser registrada com o valor *Em andamento* para cada projeto, significando que o sistema não permitirá a persistência dos dados caso o usuário tente registrar

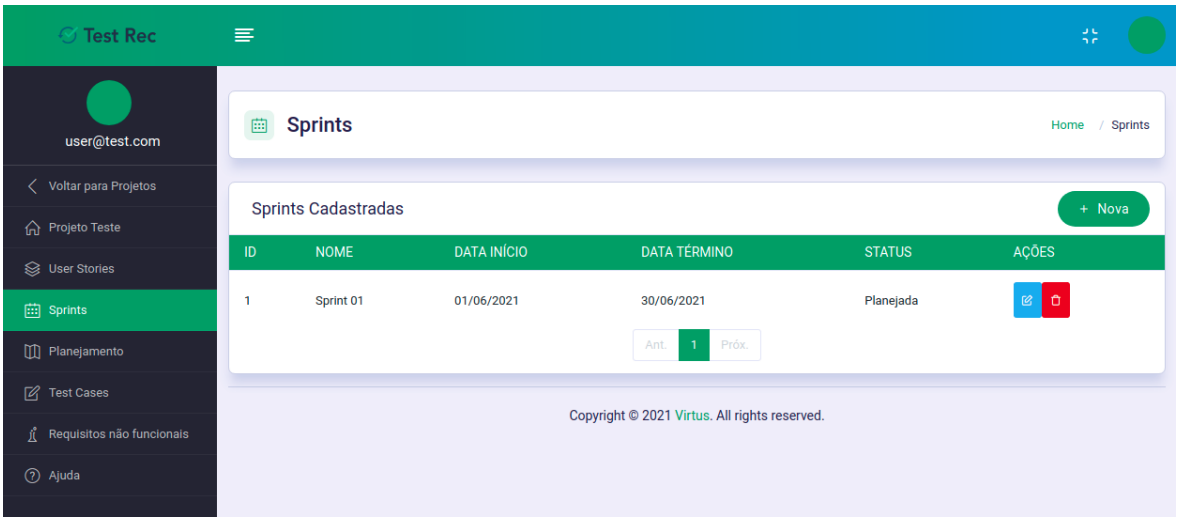

Figura B.8: Tela de listagem de Sprints cadastradas para um projeto.

mais de uma sprint com esse valor. Isso se deve ao fato de que, dentro de um projeto ágil, cada iteração deve ser executada em um determinado período específico, não sendo possível a execução de mais de uma iteração em paralelo. Também devemos observar que não será possível a persistência dos dados caso o período da sprint esteja fora do período do projeto, ou ainda, o período da sprint entre em colisão com o período de outra sprint cadastrada para o mesmo projeto.

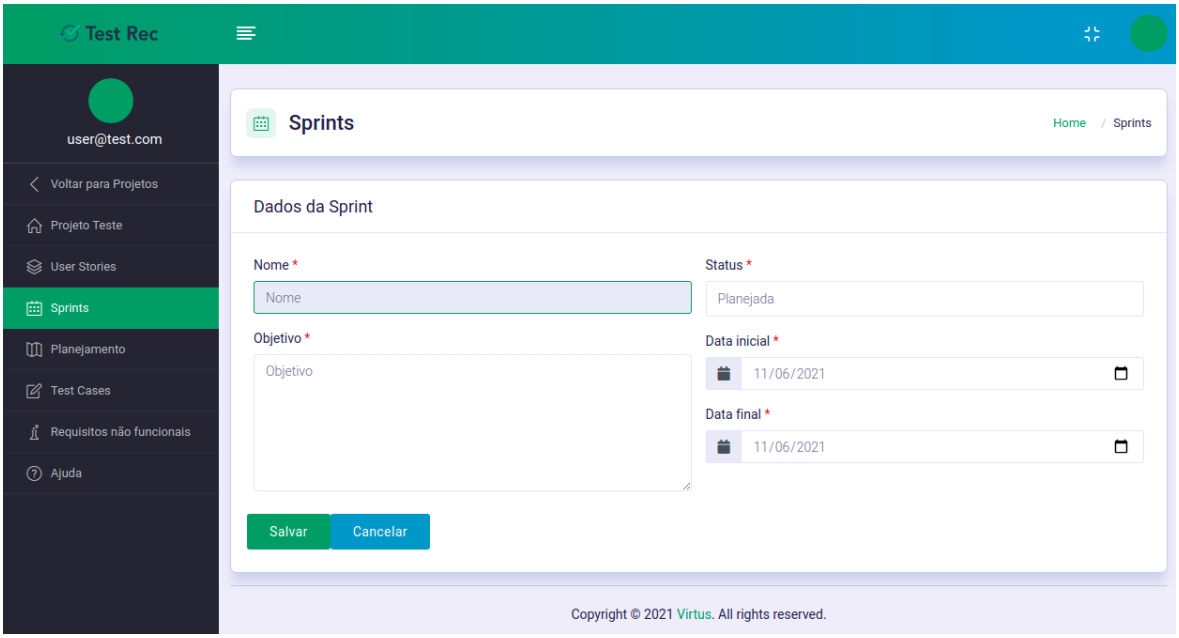

Figura B.9: Tela de cadastro de Sprints.

Como mencionado anteriormente, na listagem de sprints é possível selecionarmos uma Sprint cadastrada para edição, ou seja, alteração de seus dados. Para isso, o usuário deverá clicar no botão de edição, na coluna ação, referente à Sprint que se deseje alterar. Ao fazer isso, a tela de edição de Sprint é exibida, conforme Figura B.10. Nela, todos os valores da Sprint selecionada são exibidos, e o usuário tem a possibilidade de alterar qualquer um deles. Vale salientar que, as restrições descritas na tela de cadastro também valem para a edição, ou seja, não é possível que o período entre em colisão com outra Sprint, nem que o Status possa ser alterado para *Em andamento*, quando já houver outra Sprint com esse status.

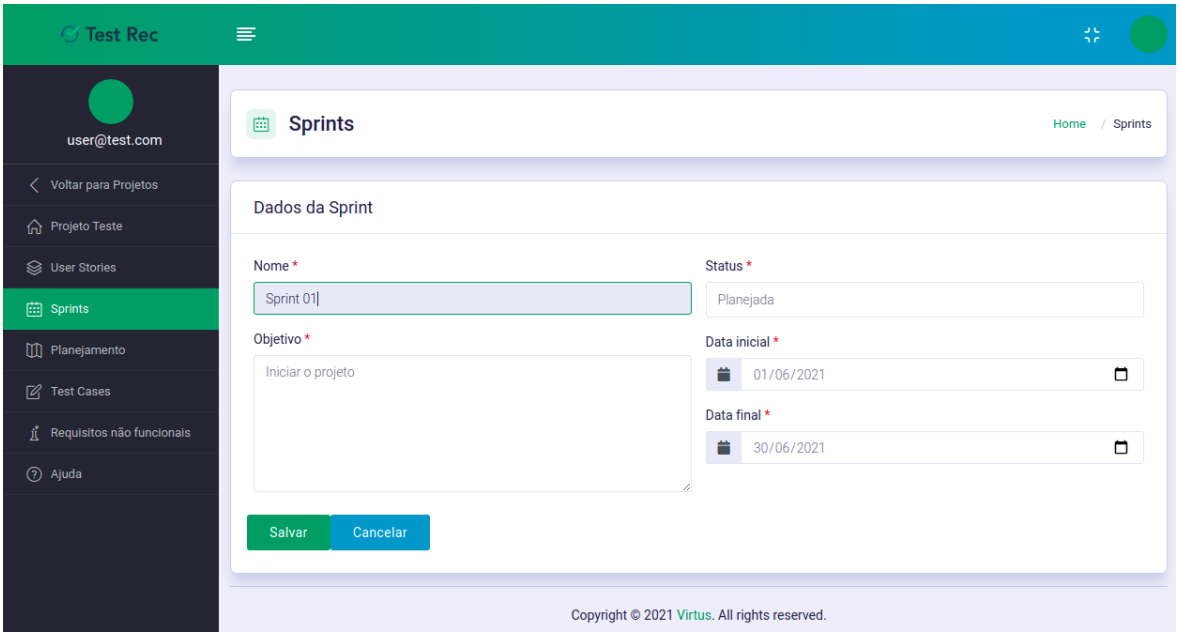

Figura B.10: Tela de edição de Sprint.

B.6 Planejamento de Sprints

Dado que temos requisitos a serem implementados no formato de Estórias de Usuário, e iterações no formato de Sprints, devemos iniciar o planejamento do que será feito em cada Sprint. A tela de planejamento, apresentada na Figura B.11 nos dá uma visão de como é feito esse processo. Ela possui duas colunas, uma representando os itens de backlog, ou seja, as Estórias de Usuário não planejadas, e, na segunda coluna, as Sprints registradas. Assim, o planejamento da sprint consiste em clicar no item de backlog e arrastá-lo para a Sprint em que se deseje que o mesmo seja desenvolvido.

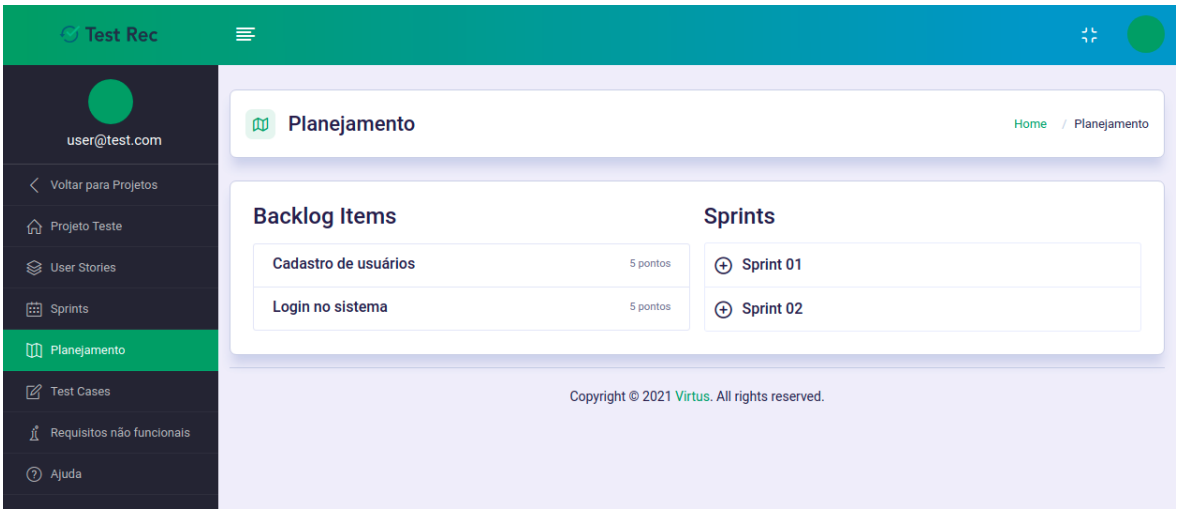

Figura B.11: Tela de planejamento de Sprints.

A Figura B.12 mostra a inclusão de um item de backlog à uma Sprint após o usuário clicar e arrastar o mesmo. Assim, uma vez que um item de backlog foi associado a uma Sprint, o mesmo não estará mais disponível na lista de itens de backlog disponível para o planejamento.

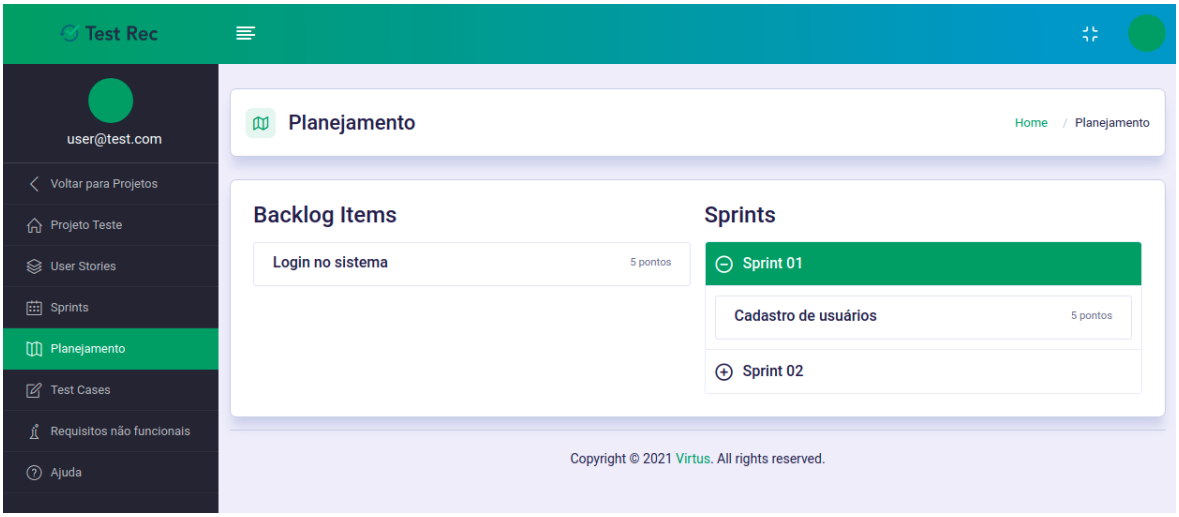

Figura B.12: Exemplo de associação de Estória de Usuário à Sprint 01.

B.7 Recomendação, Cópia e Cadastro de Casos de Teste

Para registro dos casos de teste a serem executados com o objetivo de garantir a qualidade do sistema em desenvolvimento é possível utilizarmos a funcionalidade de recomendação de

casos de teste já executados em projetos anteriores. Para isso, o usuário deverá selecionar a Estória de Usuário para a qual deseja que as recomendaçõe sejam executadas. Ao selecionar a estória, o sistema busca os casos de teste mais relevantes e exibe, conforme exibido na Figura B.13. Assim, o usuário tem a possibilidade de clicar e arrastar os casos de teste para qualquer uma das estórias de usuário registradas. Caso o usuário queria ver os passos do caso de teste recomendado para identificar seu propósito, o mesmo poderá clicar em cima da descrição do caso de teste, e uma tela será exibida com tal informação.

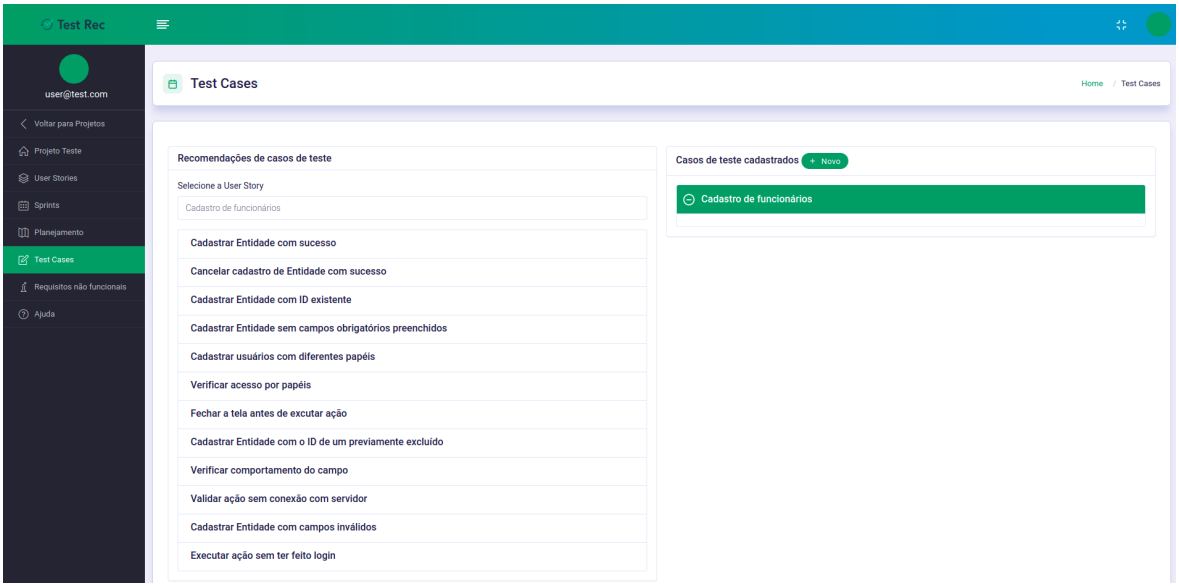

Figura B.13: Tela de listagem de casos de teste recomendados.

Na Figura B.14, é possível vermos dois casos de teste selecionados e associados à Estória de Usuário *Cadastro de Usuários*. Ao ser associado a uma estória, o caso de teste recomendado é copiado em um novo caso de teste, possuindo informação do caso de teste de origem. Esses casos de teste possuem um identificador próprio e podem ser selecionados para edição.

Na tela de listagem de casos de teste, ainda é possível registrarmos novos casos de teste sem ser pela via de recomendação. Para isso, o usuário deverá clicar no botão *Novo*, e assim, a tela de cadastro de casos de teste será exibida, conforme apresentado na Figura B.15. Nesta tela, o usuário deverá informar o nome do caso de teste, a Estória de Usuário à qual ele está associado, a prioridade, o propósito do projeto, o tipo de execução, a pré-condição e dos passos a serem executados para completude do caso. Após informarmos os dados, o usuário tem a opção de clicar no botão *Salvar* e persistir os dados no banco, ou cancelar e descartar

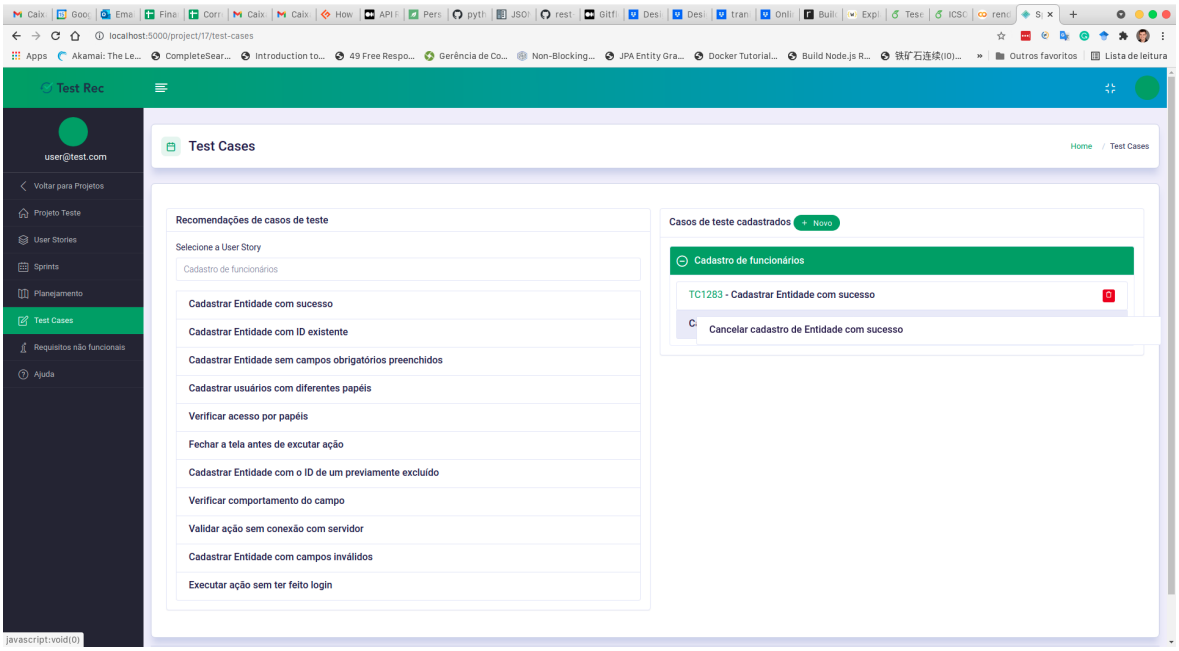

Figura B.14: Tela de cópia de caso de teste recomendado para Estória do Usuário.

os dados informados. Ao salvar o caso de teste, o mesmo será exibido na tela de listagem de casos de teste, abaixo da Estória de Usuário ao qual ele foi associado.

Na tela de listagem, é possível a seleção de um caso de teste para edição ao clicar no identificador. Assim, a tela de edição de casos de teste é exibida conforme exibido na Figura B.16. Nela, as informações do caso de teste são exibidas e o usuário pode alterá-las. Ao finalizar a edição, o usuário deverá clicar no botão *Salvar* para persistir as informações ou *Cancelar* para descartar as alterações feitas no registro.

B.8 Requisitos Não Funcionais

Em projetos de desenvolvimento de sistemas, gerenciar os requisitos não funcionais é de suma importância, a fim de garantir que o produto entregue atenda as expectativas dos clientes. O registro desses requisitos também é fundamental para o funcionamento do sistema de recomendação para casos de teste, uma vez que essa informação é utilizada na comparação entre Estórias de Usuário. Na Figura B.17, podemos observar a listagem de requisito não funcionais cadastrados para o projeto em questão. Nela são exibidos o identificador do requisito (ID), o tipo do requisito, o atributo, além das ações que usuário poderá executar

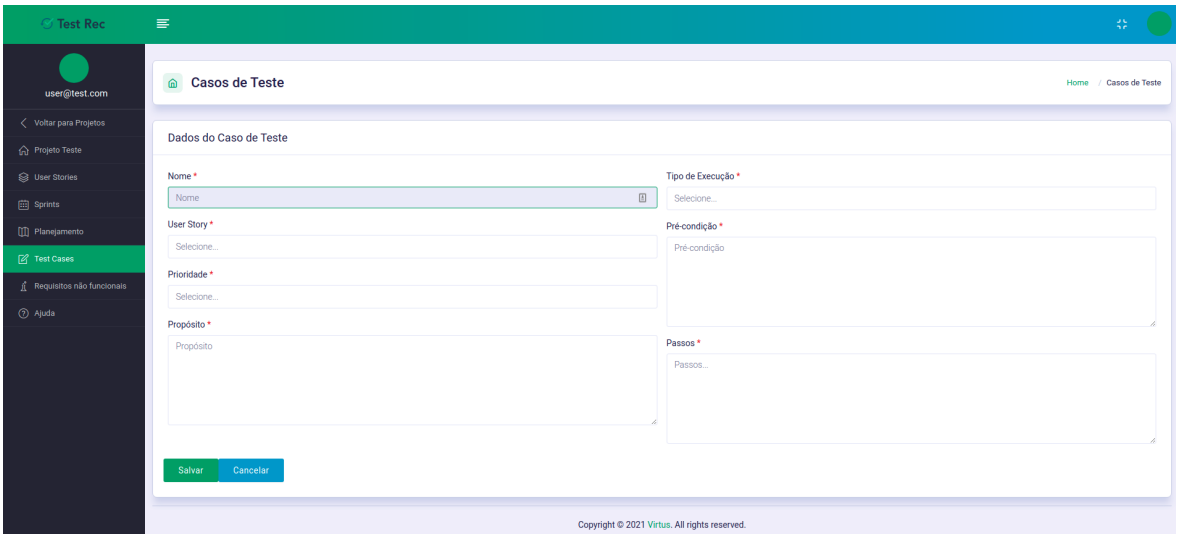

Figura B.15: Tela de cadastro de caso de teste.

sobre o mesmo. Para cada requisito, o usuário poderá editar ou excluir. Além disso, caso o usuário deseje registrar um novo requisito não funcional para o projeto, deve clicar no botão *Novo*.

Ao clicar no botão *Novo* na tela de listagem dos requisitos não funcionais, a tela de cadastro de requisitos não funcionais é exibida conforme Figura B.18. Nela o usuário deve apenas selecionar o tipo do requisito não funcional e, em seguida, descrever a sentença que explica como o requisito deve ser atingido. Ao finalizar o preenchimento das informações, o usuário deve clicar no botão *Salvar* para confirmar a persistência, ou *Cancelar*, para descartar os dados informados.

Ainda na tela de listagem de requisitos não funcionais, o usuário tem a opção de editar os requisitos cadastrados. Na Figura B.19 podemos ver a tela de edição dos requisitos não funcionais, onde os dados do requisito selecionado aparecem preenchidos. Assim, o usuário poderá alterar qualquer um dos dados existentes. Uma vez finalizas das alterações, o usuário poderá persistir as mesmas clicando no botão salvar, ou descartar as alterações clicando no botão cancelar.

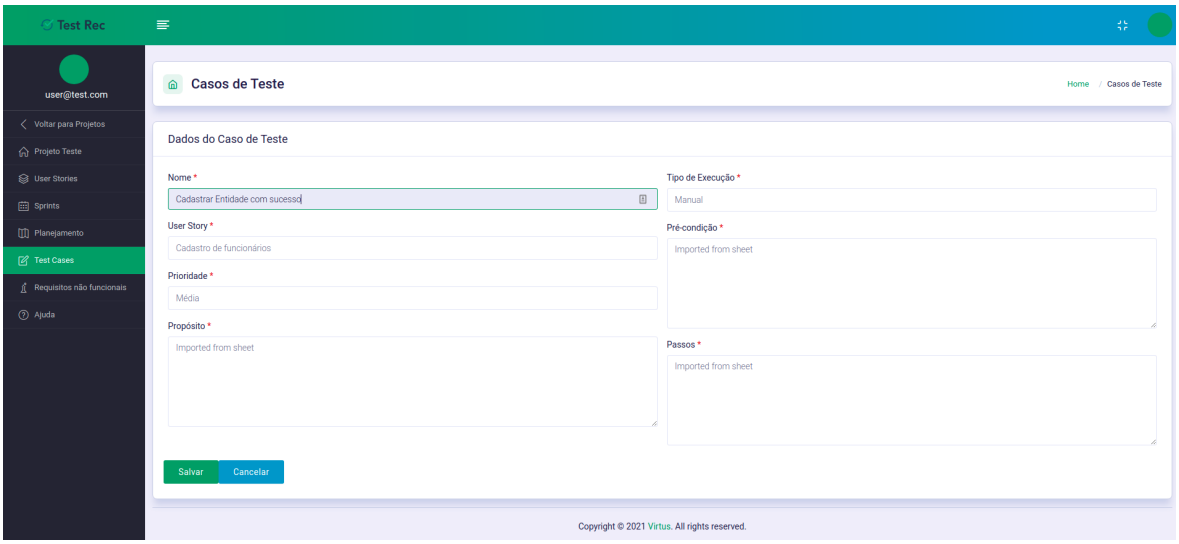

Figura B.16: Tela de edição de caso de teste.

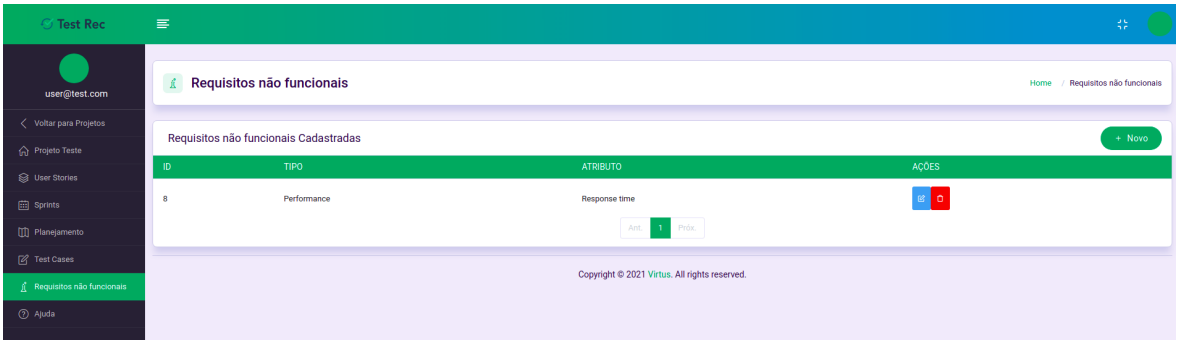

Figura B.17: Tela de listagem de requisitos não funcionais cadastrados para um projeto.

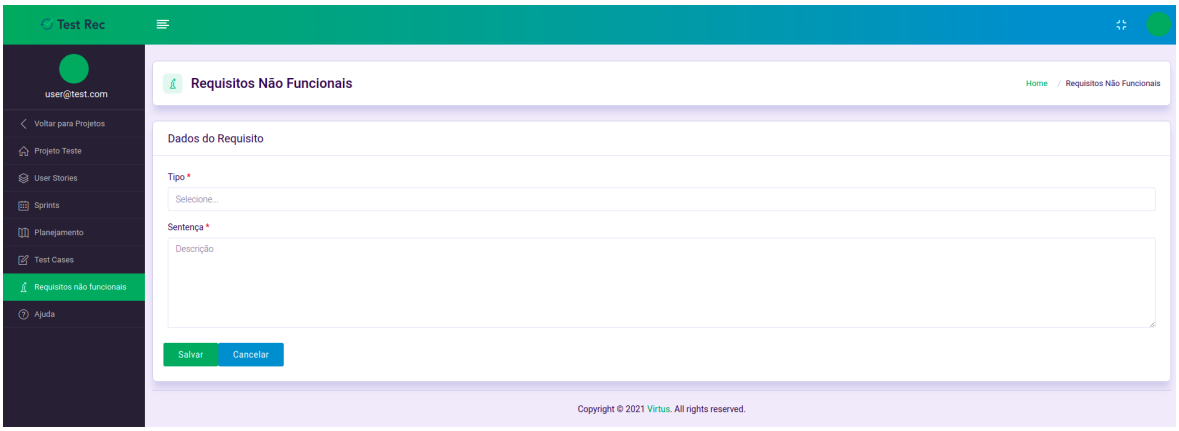

Figura B.18: Tela de cadastro de requisitos não funcionais.

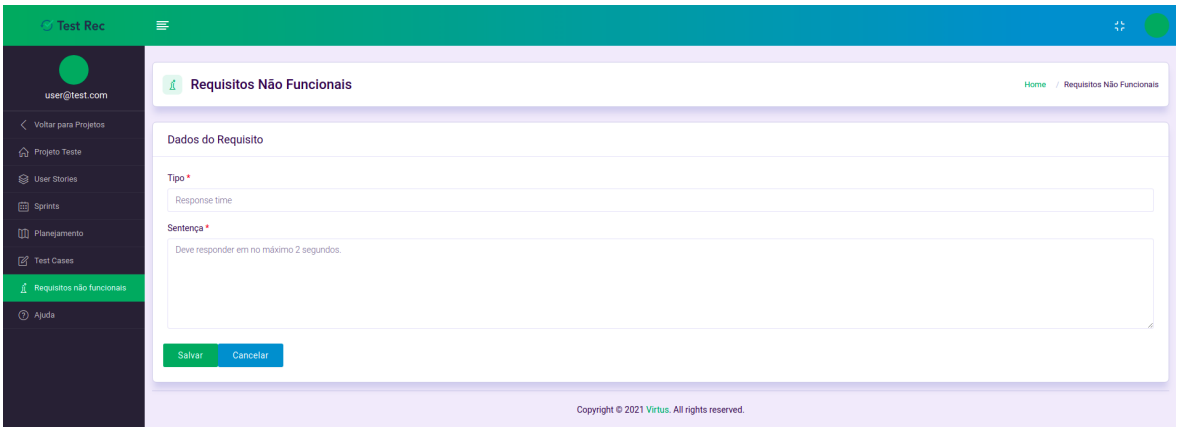

Figura B.19: Tela de edição de requisito não funcional.