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

Recomendação de Requisitos Não Funcionais em Projetos Ágeis Baseados em Scrum

Felipe Barbosa Araújo Ramos

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 Oliveira de Almeida
(Orientador)

Campina Grande, Paraíba, Brasil

“RECOMENDAÇÃO DE REQUISITOS NÃO FUNCIONAIS EM PROJETOS ÁGEIS BASEADOS EM SCRUM”

FELIPE BARBOSA ARAÚJO RAMOS

TESE APROVADA EM 25/02/2019

HYGGO OLIVEIRA DE ALMEIDA, Dr., UFCG
Orientador(a)

TIAGO LIMA MASSONI, Dr., UFCG
Examinador(a)

EVANDRO DE BARROS COSTA, Dr., UFAL
Examinador(a)

UIRA KULESZA, Dr., UFRN
Examinador(a)

MIRKO BARBOSA PERKUSICH, Dr., IFPB
Examinador(a)

CAMPINA GRANDE - PB

, ,

R175r Ramos, Felipe Barbosa Araújo.
Recomendação de requisitos não funcionais em projetos ágeis baseados em scrum / Felipe Barbosa Araújo Ramos. – Campina Grande, 2019.
153 f. : il. color.

Tese (Doutorado em Ciência da Computação) – Universidade Federal de Campina Grande, Centro de Engenharia Elétrica e Informática, 2019.
"Orientação: Prof. Dr. Hyggo Oliveira de Almeida".
Referências.

1. Requisitos não funcionais. 2. Recomendação. 3. Scrum. I. Almeida, Hyggo Oliveira de. II. Título.

CDU 004.42(043)

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

Resumo

O desenvolvimento ágil de software concentra-se em entregas rápidas e flexibilidade para mudanças de requisitos. Apesar de ser eficaz no fornecimento de requisitos funcionais de qualidade, as práticas ágeis tendem a negligenciar requisitos não funcionais até os últimos estágios do desenvolvimento do produto de software. Porém, não considerar devidamente requisitos não funcionais durante a análise de requisitos pode resultar em falhas de projetos iguais ou superiores a 60%. Nesta tese, propõe-se uma solução para auxiliar integrantes de equipes de projetos ágeis na definição de requisitos não funcionais durante o processo de análise de requisitos. A solução proposta é baseada em um sistema de recomendação baseado em memória que é viabilizado por uma adaptação do método ágil Scrum, que visa estruturar informações de projetos por meio da atribuição de *tags*. Para validar a solução, foram conduzidos experimentos *off-line* com dados coletados de 13 projetos de uma empresa de software brasileira. Os experimentos foram executados de acordo com o protocolo de validação cruzada *leave-p-out*. Como resultado, o sistema de recomendação proposto mostrou uma taxa de *F-measure* de até 79%. Além disso, foi realizada uma avaliação de usuário por meio de um estudo de caso com quatro equipes da mesma empresa citada anteriormente, em que foram avaliadas a utilidade prática de uma ferramenta que integra a solução proposta e a precisão das predições de requisitos não funcionais. De forma geral, as quatro equipes consideraram a ferramenta útil para o gerenciamento de requisitos não funcionais e foi obtida uma taxa de precisão de 81,8% das recomendações. Dessa forma, conclui-se que o sistema de recomendação é capaz de realizar previsões de requisitos não funcionais com eficácia satisfatória e, portanto, é útil para auxiliar nas tomadas de decisões de projetos baseados em Scrum.

Abstract

Agile software development focuses on quick delivery and flexibility to requirements change. Despite being effective in delivering quality functional requirements, agile practices tend to neglect non-functional requirements until the later stages of software development. However, neglecting non-functional requirements during requirements analysis can result in project failure ranges of 60% or higher. In this thesis, we propose a solution to assist agile project team members in the definition of non-functional requirements during the requirement engineering process. The proposed solution is based on a memory-based recommendation system that is enabled by an adaptation of the Scrum method, which aims to structure project information through the assignment of tags. To validate the solution, we conducted off-line experiments with data collected from 13 Scrum-based projects from a Brazilian software company. We performed the experiments according to the leave-p-out cross-validation protocol. As a result, our proposed recommendation system showed a F-measure rate of up to 79%. In addition, we conducted a user evaluation through a case study with four teams from the same company previously mentioned to evaluate the practical use of a tool that implements the proposed solution and the precision of the non-functional requirements predictions. Overall, the four teams considered the tool useful for the management of non-functional requirements, and our solution achieved a precision rate of 81.8% of the recommendations. Thus, we conclude that the recommendation system is capable of predicting non-functional requirements with satisfactory effectiveness, and hence, it is useful to support the decision making in Scrum-based projects.

Agradecimentos

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

Aos meus pais Adeziva Barbosa e Fernando Ramos e ao meu irmão Fernando Júnior, os quais me dão apoio e incentivo em todos os momentos da minha vida.

À minha namorada Karla Monik, pelo apoio e incentivo fornecido, principalmente durante a finalização deste documento de tese.

Ao meu orientador Hyggo Almeida, que sempre me auxiliou 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 Antonio Alexandre Moura Costa, Mirko Perkusich, Emanuel Dantas e Ednaldo Dilorenzo.

Aos alunos participantes do projeto de pesquisa e desenvolvimento do laboratório Embedded pelo suporte dado em diversas atividades apresentadas neste documento de tese, em especial Anderson Fellipe, Antonio Pedro, Ariel Roque, Caio Felipe, Marcos Vinícius e Thaynnara Raiany.

À Coordenação de Aperfeiçoamento de Pessoal de Nível Superior (CAPES) pelos investimentos na pós-graduação.

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

Conteúdo

Lista de Símbolos

AFFINE - *Arcabouço Ágil para a Integração de Engenharia de Requisitos Não Funcionais*

DAS - *Desenvolvimento Ágil de Software*

DoD - *Definição de Pronto*

ER - *Engenharia de Requisitos*

GQM - *Goal/Question/Metric*

IoT - *Internet das Coisas*

ISE - *Intelligent Software Engineering*

k-NN - *k-ésimo Vizinho mais Próximo*

LPS - *Linha de Produto de Software*

NERV - *Metodologia para Elicitação, Raciocínio e Validação de Requisitos Não Funcionais em Processos Ágeis*

NORMAP - *Modelagem de Requisitos Não Funcionais para Processos Ágeis*

NORMATIC - *Modelagem de Requisitos Não Funcionais em Processos Ágeis Semiautomática*

NORPLAN - *Planejamento de Requisitos Não Funcionais para Processos Ágeis*

OCR - *Reconhecimento Ótico de Caracteres*

PFU - *Percepção de Facilidade de Uso*

PO - *Product Owner*

PU - *Percepção de Utilidade*

RF - *Requisito Funcional*

RNF - *Requisito Não Funcional*

RQ - *Questão de Pesquisa*

SR - *Sistema de Recomendação*

TAM - *Modelo de Aceitação de Tecnologia*

UFCG - *Universidade Federal de Campina Grande*

US - *User Story*

VIRTUS - *Núcleo de Pesquisa, Desenvolvimento e Inovação em Tecnologia da Informação, Comunicação e Automação*

XP - *eXtreme Programming*

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

Desde a declaração do Manifesto Ágil em 2001, os métodos de desenvolvimento ágil de software (DAS), como Scrum [108] e *eXtreme Programming* (*XP*) [14], têm atraído a atenção das comunidades acadêmica [39, 64, 113] e industrial [23, 123]. Os métodos ágeis ganharam força com a aceitação de que a incerteza faz parte do mercado de desenvolvimento de software [39], emergindo como uma forma de acompanhar a elevada competitividade e volatilidade presente nesse mercado [24]. De forma geral, métodos ágeis seguem uma série de valores e práticas que visam permitir que as equipes de desenvolvimento entreguem software em funcionamento mais rapidamente, ao passo que respondam às mudanças de requisitos [86].

Ao contrário das abordagens tradicionais, que se apoiam em processos detalhados e extensivo planejamento [42], os métodos ágeis se concentram na entrega rápida de valor de negócio aos clientes, seguindo um processo de desenvolvimento empírico e incremental [110]. Adicionalmente, os métodos ágeis apresentam um foco elevado em aspectos humanos e sociais dos envolvidos no processo, promovendo a comunicação constante entre clientes e equipe de desenvolvimento. Por fim, apresentam um processo de desenvolvimento que encoraja práticas que acomodam mudanças nos requisitos em qualquer etapa do processo de desenvolvimento [39].

Nesse contexto, os requisitos são a base de todo produto de software e, consequentemente, a engenharia de requisitos (ER) desempenha um papel importante no desenvolvimento de sistemas [95]. Segundo Tripathi e Goyal [115], é importante ter requisitos claros, detalhados e sem ambiguidade para desenvolver software com sucesso. De acordo com

Vithana [118], requisitos incorretos e incompletos são um dos três principais motivos que causam a falha de projetos de desenvolvimento de software. Requisitos de software podem ser classificados em dois tipos, requisitos funcionais (RFs) e requisitos não funcionais (RNFs) [106]. RFs indicam o que um sistema deve fazer, enquanto que os RNFs referem-se a como o sistema deve se comportar [81].

Segundo Maiti e Mitropoulos [81], o sucesso de um sistema depende de ambos os tipos de requisitos, pois o sistema deve fazer o que se propõe, mas considerando atributos de qualidade definidos. Por exemplo, um sistema Web não deve levar mais de dois segundos para carregar uma página (requisito não funcional) e apresentar seu conteúdo para o usuário (requisito funcional) [6].

No DAS, o gerenciamento de requisitos é feito de forma contínua [24], ou seja, requisitos são inicialmente elicitados com os clientes, mas podem ser constantemente refinados durante o processo de desenvolvimento [47, 110]. Porém, assim como nas abordagens tradicionais, um processo crucial para o sucesso dos projetos ágeis é a Engenharia de Requisitos, denominada de Engenharia de Requisitos Ágil [66].

As práticas de ER ágil são consideradas efetivas na entrega rápida de requisitos funcionais [105], pois visando agregar valor de negócio já nas primeiras iterações, os profissionais tendem a focar o desenvolvimento nos principais requisitos funcionais do software. Porém, requisitos não funcionais - como desempenho e segurança – são frequentemente negligenciados nas fases iniciais dos projetos ou até mesmo ignorados durante o processo [105]. Esse fato também pode ser influenciado por questões como tempo limitado, falta de experiência da equipe, desinteresse ou desconhecimento do cliente e falta de uma cultura da organização para considerar RNFs [22].

Dada a importância do processo de ER para o sucesso de projetos ágeis, estudos recentes foram conduzidos para investigar suas práticas e desafios [24, 65, 66, 71, 96, 107, 118, 129]. Cao e Ramesh [24] realizaram um estudo empírico sobre práticas de ER ágil em 16 organizações e verificaram que, diferentemente da ER tradicional, a ER ágil requer uma abordagem de descoberta iterativa, sendo mais dinâmica e adaptativa. Nesse estudo, foram identificadas sete práticas diferentes adotadas nas organizações. Dentre elas, a prática considerada mais importante pelos profissionais entrevistados foi a comunicação intensiva entre desenvolvedores e clientes. Por outro lado, dentre os desafios presentes na ER ágil, foram apontadas

dificuldades na estimativa de custo e cronograma de projetos em fases iniciais e a negligência com requisitos não funcionais, que segundo os participantes do estudo, são frequentemente mal definidos e ignorados durante os primeiros ciclos de desenvolvimento. Ramesh et al. [96], em uma extensão do estudo empírico mencionado anteriormente, classificaram dois riscos de projeto como intratáveis pelas práticas ágeis convencionais: problemas com inabilidade de clientes e problemas relacionados à negligência com RNFs. Inayat et al. [66], em uma revisão sistemática da literatura com foco nas práticas e desafios do processo de ER ágil, indicaram a negligência com RNFs como um dos oito desafios provenientes da adoção das práticas ágeis.

Corroborando com os trabalhos previamente citados, em Schön et al. [107], estudo conduzido por meio de um processo iterativo baseado no julgamento de 26 especialistas, foram identificados 20 desafios provenientes das práticas da ER ágil, dos quais seis foram definidos como desafios chave. A negligência com requisitos não funcionais figurou entre os 20 desafios reportados e o gerenciamento contínuo de requisitos foi classificado como um dos seis desafios chave.

Como pode ser verificado nos trabalhos mencionados anteriormente, vários desafios relacionados à adoção de práticas ágeis são relatados na literatura, principalmente no contexto de gerenciamento contínuo de requisitos. Problemas relacionados à negligência com requisitos não funcionais são relatados por diversos trabalhos, incluindo estudos empíricos [24, 71, 96, 129], revisões da literatura [47, 109] e trabalhos baseados na análise de especialistas [107].

É nesse contexto de engenharia de requisitos ágil de software, com foco em Scrum, que se insere este trabalho. Mais especificamente, buscam-se mecanismos para dar suporte à equipe de desenvolvimento na elicitação de requisitos não funcionais, visando auxiliar o gerenciamento contínuo de requisitos.

1.1 Problemática

Negligenciar RNFs é um dos principais motivos de falhas em projetos. Segundo Bajpai e Gorthi [12], quando requisitos não funcionais não são considerados em projetos de software, as taxas de falhas observadas são iguais ou superiores a 60%. Adicionalmente, quando

RNFs não são considerados em fases iniciais do processo de desenvolvimento, os custos para atendê-los com adaptações do sistema são elevados, pois decisões na arquitetura do software dependem tanto de requisitos funcionais quanto de requisitos não funcionais [81].

Porém, no DAS, considerar RNFs em fases iniciais dos projetos não é uma tarefa trivial, pois nessa etapa os profissionais envolvidos dispõem de poucas informações acerca do escopo do projeto e suas decisões estão sujeitas a viés [118]. Adicionalmente, é comum que em projetos ágeis o gerenciamento de RNFs seja feito de forma *ad-hoc* [81], sem um processo definido ou o uso de ferramentas de apoio. Assim, esta lacuna no manuseio de requisitos não funcionais da ER ágil combinada com um cenário de mudanças encorajado pelas práticas ágeis pode resultar em problemas como aumento de custo e esforço em fases tardias do desenvolvimento para atingir os objetivos do projeto, renegociação de prazos de entrega e, em casos mais extremos, falha dos projetos [81].

Em projetos baseados em Scrum, o problema pode ser ainda mais acentuado, pois o *Product Owner* (PO), que é o responsável pelos requisitos, apresenta comumente um perfil mais voltado para os negócios e tende a considerar o progresso do projeto com base apenas nos RFs desenvolvidos, Dessa forma, é comum que a elicitação de RNFs fique implicitamente a cargo da equipe de desenvolvimento [104], que pode não ter experiência suficiente para defini-los ou também focar apenas em RFs para atender às demandas prioritárias do cliente.

Por isso, nos últimos anos, tem crescido o número de estudos abordando o tema de RNFs no contexto de DAS. Algumas pesquisas propuseram modelos conceituais visando modelar RNFs em projetos ágeis. Em Farid [52], foi proposto um arcabouço de modelagem de RNFs para o contexto de DAS. Por outro lado, Bourimi et al. [19] propuseram um arcabouço que visa impor conceitualmente a consideração precoce de RNFs. Adicionalmente, os autores introduziram o papel de parte interessada de RNFs (do inglês *NFR stakeholder*), que é responsável pela elicitação de RNFs específicos. Apesar de relevantes, esses trabalhos apenas propõem um reforço conceitual e/ou a adição de artefatos e papéis no processo ágil. Portanto, não são propostas soluções para dar suporte à elicitação de RNFs de forma automatizada.

Em outros trabalhos, são propostas abordagens de extração de RNFs a partir de artefatos de projetos. Em Domah e Mitropoulos [40], propõe-se a extração de RNFs a partir de documentos textuais disponibilizados antes do início dos projetos. Por sua vez, em Maiti e Mitropoulos [82] e [81], foi proposta a extração de RNFs de documentos textuais e imagens

disponibilizadas pelos arquitetos de software no início dos projetos. Porém, nem sempre esses documentos estão disponíveis nas fases iniciais dos projetos ágeis, pois um dos valores ágeis é "software em funcionamento mais que documentação abrangente". Também foram propostos trabalhos para melhorar a visualização dos RNFs como em [49] e [51].

Por fim, apesar de Scrum ser o método ágil mais utilizado na indústria [116], os estudos [19, 104, 105], que focam em Scrum, apresentam apenas processos para complementá-lo e consideram, por exemplo, a modelagem de RNFs como critérios da definição de pronto (DoD) para criar restrições para os requisitos funcionais [104], ficando mais uma vez a cargo da equipe de desenvolvimento identificar e definir os RNFs. Desta forma, pode-se concluir que ainda não há uma solução consolidada para a elicitação de RNFs em projetos ágeis.

Diante do exposto, enuncia-se o problema abordado neste trabalho: como prover suporte automatizado à equipe de desenvolvimento em projetos ágeis baseados em Scrum para reduzir o risco de negligência dos requisitos não funcionais de software?

1.2 Objetivos

O objetivo geral neste trabalho é propor um sistema de recomendação (SR) de requisitos não funcionais baseado no histórico de projetos para prover suporte à equipe de desenvolvimento Scrum durante o processo de engenharia de requisitos.

O sistema de recomendação proposto é baseado em vizinhança. De forma geral, para um RF de um projeto corrente, buscam-se RFs semelhantes em uma base histórica. Em seguida, verificam-se quais RNFs foram associados aos RFs retornados na busca anterior, sendo recomendados aqueles mais frequentes. Para viabilizar a solução, propõe-se um modelo de descrição de requisitos no Scrum baseado em categorias e *tags*, que possibilita a recuperação de dados de projetos de forma estruturada.

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

- definir modelos estruturados para representação de RFs e RNFs em conformidade com os princípios Scrum que viabilizem a recuperação de informações necessárias para a geração das recomendações;
- propor um sistema de recomendação de requisitos não funcionais baseado em dados

históricos;

- desenvolver uma ferramenta de software que implementa a solução proposta;
- validar a solução proposta por meio de experimentos;
- avaliar a solução proposta por meio de estudo de caso.

Para validar o sistema de recomendação proposto, foram executados experimentos com dados reais de 13 projetos de desenvolvimento de software que foram executados no Laboratório de Sistemas Embarcados e Computação Pervasiva [1] (Embedded) e no Núcleo de Pesquisa, Desenvolvimento e Inovação em Tecnologia da Informação, Comunicação e Automação [2] (VIRTUS) da Universidade Federal de Campina Grande (UFCG), em Campina Grande, Paraíba. O VIRTUS conta atualmente com 200 profissionais atuando em mais de 40 projetos de software por ano de diferentes clientes, com diversos domínios de aplicação em diferentes plataformas.

Por sua vez, a avaliação da utilidade prática da solução proposta foi realizada por meio de um estudo de caso embarcado com quatro projetos de desenvolvimento de software reais que estão sendo executados no Embedded.

1.3 Metodologia

Este trabalho está inserido no contexto do grupo *Intelligent Software Engineering Group* [3] (ISE), que é uma iniciativa do VIRTUS da UFCG. Desta forma, todas as pesquisas do ISE baseiam-se em problemas reais identificados em projetos de software do VIRTUS.

De acordo com a classificação proposta por Wieringa et al. [124], esta pesquisa caracteriza-se como uma proposta de solução, validação e avaliação. Portanto, seguindo o ciclo de engenharia proposto pelos autores, inicialmente, investigou-se a situação corrente da área de engenharia de requisitos ágil para identificar possíveis problemas. As atividades de investigação foram realizadas em dois contextos, na literatura e no VIRTUS. Como resultado, verificou-se que o problema relacionado à negligência com requisitos não funcionais

[1] Embedded: http://www.embeddedlab.org/

[2] VIRTUS: http://virtus.ufcg.edu.br/

[3] ISE: http://isegroup.org

proveniente da adoção de práticas da ER ágil foi observado em ambos os contextos, sendo o mesmo definido como o problema de pesquisa investigado. Posteriormente, por meio de uma revisão bibliográfica, investigou-se na literatura a existência de trabalhos que propõem soluções para resolver o problema estudado por meio da análise de dados históricos, mas não foi obtido sucesso nesse passo. Só então, a solução proposta foi projetada e, em seguida, validada com experimentos. Como foi obtido sucesso na etapa de validação, a solução foi integrada a uma ferramenta de gerenciamento de projetos e, posteriormente, avaliada em um contexto real por meio de um estudo caso.

As questões de pesquisa (RQs) consideradas neste trabalho e suas respectivas hipóteses são as seguintes:

RQ$_1$: como estruturar informações de recursos de software de projetos baseados em Scrum, de modo que seja possível o reúso, a rastreabilidade e a recuperação das informações?

- H_{1-1}: é possível estruturar as informações de recursos de software de projetos baseados em Scrum com *tags* e categorias para viabilizar o reúso, a rastreabilidade e a recuperação de suas informações.

RQ$_2$: como construir um sistema de recomendação baseado em dados históricos (memória) para prover suporte à tomada de decisão da equipe de desenvolvimento Scrum durante o processo de engenharia de requisitos, com foco na definição de requisitos não funcionais?

- H_{2-1}: é possível utilizar dados históricos de projetos de software baseados em Scrum estruturados com *tags* e categorias para construir um sistema de recomendação de RNFs.

RQ$_3$: qual a utilidade prática da solução proposta?

- H_{3-1}: a ferramenta que integra a solução proposta é útil para auxiliar na definição de requisitos não funcionais em projetos de software baseados em Scrum.
- H_{3-2}: é fácil usar a ferramenta que integra a solução proposta.

RQ$_4$: qual a precisão das recomendações do sistema de recomendação proposto?

- H_{4-1}: é possível obter precisão aceitável na recomendação de RFs com a ferramenta que integra a solução proposta.

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

1. identificar as principais características de projetos de software que incidem na definição de RNFs ($\mathbf{RQ}_1$);
2. definir modelos estruturados de RFs, RNFs e tarefas com base nas características identificadas na atividade anterior que possibilitem a rastreabilidade, reúso e recuperação de informações ($\mathbf{RQ}_1$);
3. definir processo de adaptação do Scrum com base nos modelos criados na atividade anterior para possibilitar a coleta de dados estruturados de projetos ($\mathbf{RQ}_1$);
4. gerar conjunto de dados históricos com informações reais de projetos de software baseados em Scrum ($\mathbf{RQ}_2$);
5. construir sistema de recomendação de requisitos não funcionais baseado em dados históricos ($\mathbf{RQ}_2$);
6. validar a solução proposta com experimentos ($\mathbf{RQ}_4$);
7. avaliar a solução proposta com estudo de caso ($\mathbf{RQ}_3$);
8. escrever o documento de tese.

1.4 Contribuições e Relevância

As principais contribuições desta tese são apresentadas a seguir.

- São definidos modelos estruturados para representação de RFs, RNFs e tarefas em conformidade com os princípios Scrum, que possibilitam a rastreabilidade, reúso e recuperação de dados de projetos de software;
- É apresentado um conjunto de dados históricos com informações de 13 projetos de software reais baseados em Scrum, que foi estruturado com base nos modelos citados anteriormente;

- É apresentado um sistema de recomendação de requisitos não funcionais baseado em memória para prover suporte à equipe de desenvolvimento Scrum durante o processo de engenharia de requisitos [97];
- É apresentada uma ferramenta que integra a solução proposta.

O trabalho colabora na área de engenharia de requisitos no contexto de Scrum, que é considerado o método ágil mais usado em projetos de desenvolvimento de software no mundo [116]. Portanto, este trabalho pode impactar positivamente na indústria de software, uma vez que a engenharia de requisitos é um processo crucial para o sucesso de projetos ágeis [129].

Com o objetivo principal alcançado, espera-se que as equipes de desenvolvimento Scrum sejam capazes de tomar decisões mais embasadas e precoces quanto à definição de requisitos não funcionais durante as negociação de requisitos com o *Product Owner*. Portanto, a solução proposta mostra-se relevante pois visa mitigar o problema de negligência com requisitos não funcionais em projetos ágeis, que é considerado um dos grandes desafios da engenharia de requisitos ágil por vários trabalhos da literatura [24, 36, 47, 61, 65, 66, 70, 92, 96, 106, 107, 109, 118].

Adicionalmente, dado que o campo de ER ágil é imaturo e o número de estudos com evidências empíricas de casos reais da indústria é escasso [66], a avaliação da solução em projetos de software reais apresenta relevância, pois pode incentivar outros pesquisadores a realizarem o mesmo.

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

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 e sistemas de recomendação.

Capítulo 3: Trabalhos Relacionados. Discutem-se os trabalhos relacionados na área de engenharia de requisitos ágil, com foco em requisitos não funcionais.

Capítulo 4: Modelo Estruturado de *User Stories (USs)*. Apresenta-se o Modelo Estruturado de *User Stories*, bem como detalhes das etapas de sua construção.

Capítulo 5: Sistema de Recomendação de Requisitos Não Funcionais. Apresenta-se o sistema de recomendação de requisitos não funcionais.

Capítulo 6: Validação da Solução. Apresentam-se as tarefas realizadas na validação da solução proposta.

Capítulo 7: Avaliação da Solução. Apresentam-se as tarefas realizadas na avaliação da solução proposta.

Capítulo 8: . Apresentam-se as considerações finais do trabalho bem como sugestões para trabalhos futuros.

Capítulo 2

Fundamentação Teórica

Neste capítulo são apresentados conceitos necessários ao entendimento deste documento. Nas próximas seções são apresentadas considerações acerca de sistemas de recomendação e Scrum.

2.1 Scrum

Scrum é o método ágil mais comum entre projetos de desenvolvimento de software ao redor do mundo. De acordo com uma pesquisa que contou com 1492 respondentes publicada em 2018 pela *VersionOne* [116], 70% dos projetos ágeis utilizam Scrum ou alguma variante (*Scrumban*, híbrido de Scrum e *XP*, etc).

Segundo Ken Schwaber e Jeff Sutherland [108], Scrum é um arcabouço para gerenciar e organizar trabalhos complexos que apresenta papéis, eventos, artefatos e regras. Em Scrum, o gerenciamento de requisitos é feito por meio da manutenção do *backlog* do produto, que consiste em uma lista priorizada de todos os itens imaginados para o software [62]. O profissional exercendo o papel de *Product Owner* é o responsável por gerenciar o *backlog* do produto, ou seja, é responsável pela elicitação e priorização dos requisitos, que são consideradas tarefas difíceis e decisivas para o sucesso dos projetos [129].

Scrum vem sendo usado para resolver problemas complexos desde o início dos anos 90 [108]. O Scrum é fundamentado no empirismo e apresenta uma abordagem iterativa e incremental, que visa aperfeiçoar o controle de riscos dos projetos. O arcabouço é fundamentado em três pilares para o controle do processo empírico, são eles: transparência,

inspeção e adaptação [108]:

- **Transparência**. Todos os responsáveis pelos resultados dos projetos devem estar cientes sobre os aspectos significativos do processo, ou seja, deve haver um padrão comum que facilite o entendimento sobre o que está sendo desenvolvido. Portanto, tanto as pessoas que realizam o trabalho quanto as que aceitam o resultado do que foi gerado devem ter uma definição comum de "pronto".

- **Inspeção**. Artefatos devem ser frequentemente inspecionados. As inspeções são importantes para identificar variações do que está sendo desenvolvido. Porém, estas não devem atrapalhar a execução das tarefas que estão em andamento.

- **Adaptação**. Caso sejam identificados desvios no produto em desenvolvimento durante a inspeção, que tornem o resultado do projeto inaceitável, ajustes no processo devem ocorrer o mais rápido possível, a fim de minimizar os desvios detectados.

Na Figura 2.1 [102] pode ser visto o arcabouço Scrum com os seguintes elementos, os quais são detalhados nas próximas subseções:

- Equipe Scrum: composto por *Product Owner*, Scrum *Master* e Equipe de Desenvolvimento;

- Eventos Scrum: representados por *Sprint*, Reunião de Planejamento da *Sprint* (do inglês, *Sprint Planning*), Reunião Diária (do inglês, *Daily* Scrum), Reunião de Revisão da *Sprint* (do inglês, *Sprint Review*), Reunião de Retrospectiva da *Sprint* (do inglês, *Sprint Retrospective*);

- Artefatos Scrum: com destaque para *Backlog* do Produto (do inglês, *Product Backlog*), *Backlog* da *Sprint* (do inglês, *Sprint Backlog*) e Incremento (do inglês, *Potentially shippable product increment*).

2.1.1 Equipe Scrum

A Equipe Scrum é formada pelo *Product Owner*, a Equipe de Desenvolvimento e o Scrum *Master*. Essa formação visa aumentar a flexibilidade, criatividade e, consequentemente, a

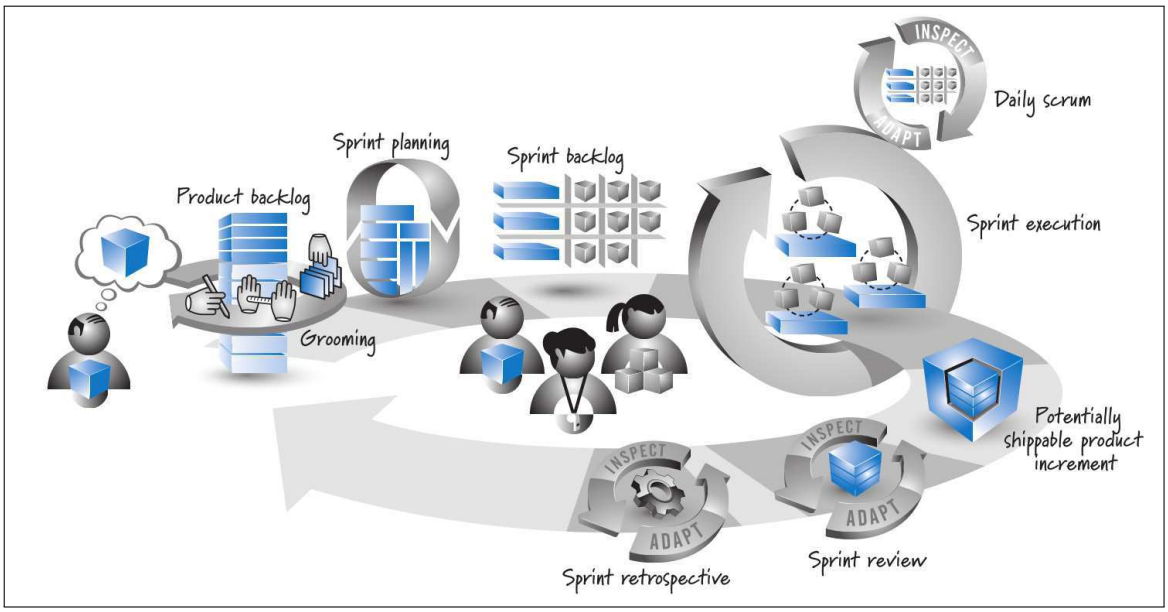


Figura 2.1: Visão geral do arcabouço Scrum [102]

produtividade do time [108]. Equipes Scrum devem ser multifuncionais e auto-organizáveis, ou seja, os indivíduos devem apresentar, em conjunto, as diversas competências necessárias ao desenvolvimento do produto e são responsáveis por definir a melhor forma de completar seu trabalho [102].

Product Owner

O *Product Owner* (em português, dono do produto) é o representante do cliente no projeto e é responsável por expressar claramente e ordenar os itens do produto que devem ser desenvolvidos pela Equipe de Desenvolvimento (itens do *Backlog* do Produto) [102]. Além disso, é responsável por:

- garantir que o trabalho da equipe agregue valor ao produto (negócio);
- garantir a visibilidade e transparência do *Backlog* do Produto;
- apresentar claramente o que a Equipe de Desenvolvimento irá trabalhar a seguir;
- garantir que a Equipe de Desenvolvimento entenda os itens do *Backlog* do Produto para minimizar desvios no processo de desenvolvimento.

Equipe de Desenvolvimento

A Equipe de Desenvolvimento é responsável por realizar o trabalho de desenvolvimento do produto. Além disso, é responsável por entregar de forma iterativa incrementos que agreguem valor de negócio ao produto [108]. Dentre as principais características da Equipe de Desenvolvimento, destacam-se:

- **auto-organizados:** integrantes do time que escolhem a melhor forma para completarem o seu trabalho;
- **multifuncionais:** o time é composto por integrantes que, como equipe, possuem todas as habilidades necessárias para desenvolver o produto;
- **sem diferenciação de títulos:** independente da tarefa que esteja executando, um integrante do time terá sempre o título de Desenvolvedor.

Scrum *Master*

O Scrum *Master* é responsável por gerenciar o processo de Scrum e assegurar que todos entendam e apliquem suas regras e práticas [108]. O Scrum *Master* atua de diversas formas no processo, incluindo:

- **treinador (em inglês, *coach*):** fornece liderança de processo e ajuda a equipe Scrum e a organização como um todo a desenvolver sua abordagem Scrum, levando em consideração suas características. Lidera e treina a organização na adoção do Scrum;
- **facilitador:** auxilia a equipe de desenvolvimento removendo impedimentos que possam afetar seu progresso e protege o time de interferências externas.

Apesar de exercer funções de liderança, o Scrum *Master* não atua como gerente, ou seja, não tem autoridade para controlar a Equipe Scrum [102].

2.1.2 Eventos Scrum

O Scrum apresenta eventos predefinidos que geram uma rotina e minimizam a necessidade de reuniões não planejadas. Durante a execução de um projeto, todos os eventos Scrum

devem ter uma duração predefinida (*time-boxed*), com duração máxima clara. Os eventos conhecidos de Scrum são *Sprint*, Reunião de Planejamento da *Sprint*, Reunião Diária, Reunião de Revisão da *Sprint* e Retrospectiva da *Sprint* [108]. A seguir, são detalhados cada um desses eventos.

Sprint

Em Scrum, o trabalho é realizado em iterações ou ciclos de duas a quatro semanas, chamados de *Sprints*. *Sprints* apresentam duração fixa, ou seja, possuem datas de início e fim predefinidas [108]. Além disso, são contínuas, ou seja, ao termino de uma *Sprint*, uma outra é iniciada até que o desenvolvimento do produto seja finalizado [102]. Adicionalmente, devem apresentar objetivo claro e não devem ser feitas mudanças de escopo e pessoal após sua inicialização que possam colocar em risco esse objetivo definido [102].

Por serem limitadas a um período curto de tempo (no máximo um mês), possibilitam a inspeção e adaptação do progresso do trabalho em direção ao objetivo final de forma contínua (no mínimo mensalmente) [102]. Ao final de cada *Sprint*, deve ser entregue um incremento potencialmente utilizável que gere valor de negócio ao produto em desenvolvimento [108].

Reunião de Planejamento da *Sprint*

O trabalho que será realizado na *Sprint* é planejado durante a Reunião de Planejamento da *Sprint* [108]. Nesta reunião, o *Product Owner* e a Equipe de Desenvolvimento entram em acordo sobre qual será o objetivo da *Sprint* [102]. Em contrapartida, o Scrum *Master* garante que o evento ocorra cumprindo o seu período de duração e que seu propósito seja entendido pelos participantes da reunião [108].

A Equipe de Desenvolvimento analisa o objeto almejado pelo PO e define de forma realista quais os itens de maior prioridade do *Backlog* do Produto podem ser desenvolvidos dentro do período da *Sprint* [108]. Também é comum que a Equipe de Desenvolvimento determine quais as atividades necessárias para finalizar o desenvolvimento dos itens selecionados e realize uma estimativa do esforço para completar todas essas atividades [102]. Para que as estimativas reflitam a realidade, é necessário que os itens do *Backlog* do Produto indicados pelo PO como de maior prioridade estejam bem detalhados e que haja um entendimento comum entre PO e Equipe de Desenvolvimento acerca do que deve ser feito para

definir que o objetivo ou meta da *Sprint* foi alcançada (Definição de Pronto [108]).

Reunião Diária

A Reunião Diária é o evento Scrum de no máximo 15 minutos em que a Equipe de Desenvolvimento sincroniza as atividades que foram desenvolvidas e planeja as tarefas que serão realizadas nas próximas 24 horas [108]. Como o próprio nome indica, a reunião ocorre diariamente, no mesmo local e horário. De forma geral, a Reunião Diária visa responder três perguntas [102]:

- o que eu fiz desde a última reunião diária?
- o que eu pretendo fazer até a próxima reunião diária?
- quais são os obstáculos ou impedimentos que estão atrapalhando o progresso das minhas atividades?

A reunião diária melhora a comunicação entre os membros da Equipe de Desenvolvimento, que com as respostas das questões indicadas, têm uma noção melhor sobre o andamento das atividades da equipe e uma visão do que pode ser melhorado para garantir que o objetivo da *Sprint* seja alcançado [102]. Por isso, este evento é considerado como chave para a inspeção e adaptação da *Sprint* [108].

Reunião de Revisão da *Sprint*

Ao final de cada *Sprint* ocorre a Reunião de Revisão da *Sprint*, que tem duração média de quatro horas [108]. O objetivo dessa reunião é inspecionar e adaptar o produto que está sendo desenvolvido [102]. No evento, que conta com a participação de toda a Equipe Scrum e demais partes interessadas, os itens que foram selecionados para a *Sprint* são analisados e revisados, e aqueles que foram finalizados (seguindo uma definição de "pronto" estabelecida) são apresentados pela Equipe de Desenvolvimento.

Após a análise do PO sobre os itens restantes do *Backlog* do Produto, o grupo discute sobre quais próximos passos podem ser seguidos. O resultado dessa discussão serve como entrada para a Reunião de Planejamento de *Sprint* seguinte [108].

Ao final da Reunião de Revisão da *Sprint*, o *Backlog* do Produto deve estar revisado e ajustado, caso necessário.

Reunião de Retrospectiva da *Sprint*

Enquanto a Reunião de Revisão da *Sprint* visa inspecionar e adaptar o produto em desenvolvimento, a Reunião de Retrospectiva da *Sprint* tem como principal objetivo inspecionar e adaptar o processo [102]. Este evento Scrum, que ocorre entre a Reunião de Revisão e próxima Reunião de Planejamento, dura em média três horas e conta com a participação de todo a Equipe Scrum [108].

A Equipe Scrum analisa como pode melhorar o processo seguindo o arcabouço do Scrum e, consequentemente, aumentar a qualidade do produto. Ao final da reunião, a Equipe Scrum deve ter identificado melhorias que serão implementadas na *Sprint* seguinte [102].

2.1.3 Artefatos do Scrum

Em Scrum são necessários alguns documentos para garantir o sucesso da aplicabilidade do arcabouço. Esses documentos são denominados de artefatos Scrum e devem ser visíveis aos participantes do Scrum de forma a garantir um entendimento comum para todos [108]. Dentre os artefatos que podem ser utilizados em Scrum, destacam-se: o *Backlog* do Produto, o *Backlog* da *Sprint* e o Incremento [108].

Backlog do Produto

O *Backlog* do Produto consiste em uma lista ordenada de todos os itens necessários ao produto, ou seja, todos os requisitos do produto em desenvolvimento [108]. O *Product Owner* é o responsável por manter o *Backlog* do Produto atualizado, disponível e ordenado [108]. Em geral, a ordem dos itens é definida por seu valor de negócio para o produto final.

O *Backlog* do Produto é um artefato "vivo", ou seja, está constantemente evoluindo e itens podem ser adicionados, removidos e modificados pelo PO [102]. Portanto, é dito que o *Backlog* do Produto nunca está completo [108].

Neste artefato Scrum, estão listados todos os requisitos, funções, melhorias e correções que devem ser feitas no produto [102]. Portanto, é importante que o documento seja con-

tinuamente refinado pelo PO. Os itens do topo da lista devem ser os mais detalhados, de modo que seja possível o Time de Desenvolvimento realizar estimativas acuradas acerca do seu desenvolvimento. Quando um item está detalhado o suficiente para ser desenvolvido em uma *Sprint*, é dito que este item está "preparado" para ser selecionado na Reunião de Planejamento da *Sprint* [108].

Em projetos Scrum, itens do *backlog* do produto podem ser descritos na forma de *user stories* [29], que especificam uma funcionalidade com valor agregado para um usuário ou para o comprador do produto de software. Segundo Mike Cohn [29], *USs* 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 *US* que servem para especificar os detalhes da funcionalidade;
- confirmação: testes que documentam detalhes e que podem ser usados para determinar quando uma *US* é concluída.

Além disso, *USs* 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*.

Backlog da *Sprint*

O *Backlog* da *Sprint* representa os itens do *Backlog* do Produto selecionados para uma *Sprint* e o detalhamento do conjunto de tarefas necessário para entregar esses itens em forma de incremento do produto [102]. Este documento detalha o trabalho da Equipe de Desenvolvimento que deve ser realizado para que o objetivo da *Sprint* seja alcançado [108].

O *Backlog* da *Sprint* evolui à medida que a Equipe de Desenvolvimento adquire mais conhecimento acerca do trabalho necessário para atingir o objetivo da Sprint corrente. Portanto, elementos podem ser adicionados, removidos ou modificados pela equipe [108]. À

medida que o trabalho é realizado, as estimativas do trabalho restante são atualizadas e podem ser gerados gráficos para representar o progresso dos trabalhos na *Sprint*, como gráficos de *Sprint Burndown*.

Incremento

O resultado de uma *Sprint* é um potencial incremento ao produto, ou seja, a soma dos itens selecionados na Reunião de Planejamento da *Sprint*, que devem estar "prontos" de forma que representem um incremento utilizável ao produto em desenvolvimento. O PO é o responsável por liberar ou não o incremento do produto [108].

2.2 Sistemas de Recomendação (SRs)

Sistemas de Recomendação são ferramentas que visam auxiliar usuários no processo de tomada de decisão por meio de sugestões de itens que podem ser de seu interesse [99]. O tema é de interesse tanto da academia quanto da indústria, pois SRs podem ser aplicados em diversos domínios de problema para ajudar usuários a lidar com sobrecarga de informação e/ou tornar suas decisões mais embasadas [2]. Alguns exemplos conhecidos da aplicação de SRs são: recomendação de produtos a comprar [77]; recomendação de filmes [17]; e recomendação de músicas [26]. Além desses domínios conhecidos, técnicas de recomendação podem ser aplicadas também na área de Engenharia de Software, por exemplo, na recomendação de requisitos de projetos de software [55, 87].

De acordo com Adomavicius e Tuzhilin [2], o problema de recomendação pode ser definido como: seja U o conjunto de todos os usuários e I o conjunto de todos os itens que podem ser recomendados. Seja f a função de utilidade de um item i para um usuário u (f: U x $I \rightarrow T$, onde T é o conjunto total ordenado). Então, para cada usuário $u \in U$, pretende-se recomendar os itens $i' \in I$ que maximizem a utilidade do usuário u. Mais formalmente, o problema pode ser representado pela Equação 2.1 [2].

$$\forall u \in U, i_u{}' = \arg\max_{i \in I} f(u, i) \tag{2.1}$$

Sistemas de recomendação podem ser classificados em três tipos de acordo com a abordagem utilizada para a geração das recomendações [2], são eles: filtragem baseada em con-

teúdo, filtragem colaborativa e abordagens híbridas.

Na filtragem baseada em conteúdo, as recomendações são geradas analisando-se o conteúdo dos itens que foram avaliados anteriormente pelo usuário e dos demais itens disponíveis para recomendação [78]. Por isso, sistemas de recomendação desse tipo têm suas raízes em abordagens de recuperação de informação e filtragem de informação [2]. Na Figura 2.2 [78] pode ser vista uma arquitetura de alto nível de um sistema de recomendação baseado em conteúdo, com destaque para os seguintes componentes:

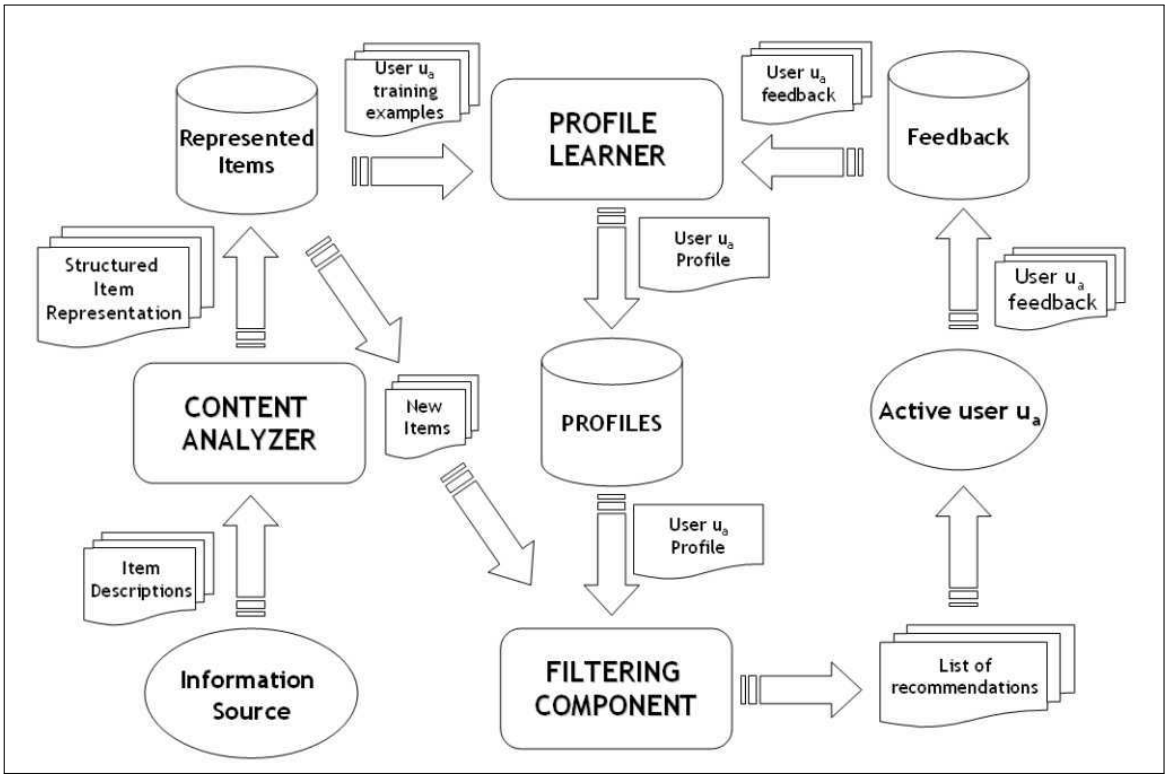


Figura 2.2: Arquitetura de alto nível de um sistema de recomendação baseado em conteúdo [78]

- **Analisador de Conteúdo (do inglês, *Content Analyzer*)**: utiliza técnicas de recuperação de informação para extrair características e gerar representações dos itens baseadas em seus conteúdos. Por exemplo, a geração de vetores de características;
- **Aprendiz de Perfil (do inglês, *Profile Learner*)**: aplica técnicas de aprendizagem de máquina para gerar os perfis de usuários, que em geral são obtidos por meio da análise de dados históricos que representam as avaliações atribuídas por usuários aos itens;

- **Componente de Filtragem (do inglês, *Filtering Component*)**: explora os perfis dos usuários para gerar listas de recomendações de itens. A utilidade de um item *i* para um usuário *u* pode ser calculada a partir da comparação do vetor de característica que representa o item *i* com o vetor de característica que representa o perfil do usuário.

Apesar de apresentar vantagens como independência de usuário e a geração de recomendações transparentes, alguns problemas podem ser identificados com o uso isolado de filtragem baseada em conteúdo, incluindo:

- **superespecialização**: como os sistemas de recomendação baseado em conteúdo recomendam itens que são similares ao perfil do usuário, dificilmente serão geradas recomendações de itens relevantes com conteúdo diferente do perfil do usuário;

- **novo usuário**: é necessário que o usuário avalie um número considerável de itens até que o sistema de recomendação consiga entender o seu perfil para gerar recomendações confiáveis;

- **análise de conteúdo limitada**: a abordagem depende das características que são extraídas dos conteúdos dos itens. Assim, alguns aspectos do item podem não ser capturados dependendo de como o conteúdo foi representado.

Em contrapartida, nos sistemas de recomendação baseados em filtragem colaborativa as recomendações são geradas a partir das avaliações de múltiplos usuários para itens iguais. A ideia principal desta abordagem é que a avaliação de um usuário u para um novo item i tende a ser igual a avaliação dada por um usuário v, desde que u e v tenham avaliado outros itens de forma semelhante [38].

Segundo Adomavicius e Tuzhilin [2], na filtragem colaborativa o problema de recomendação pode ser descrito mais formalmente da seguinte forma: a utilidade $f(u, i)$ atribuída ao item i para o usuário u é calculada com base nas utilidades $f(u_j, i)$ atribuídas ao item i pelos usuários $u_j \in U$ que foram considerados similares a u.

Algoritmos usados na filtragem colaborativa podem ser agrupados em duas classes [2], são elas:

- **baseados em memória**: a geração das recomendações é feita por meio de heurísticas com base nas avaliações atribuídas aos itens pelos usuários no passado. De forma

simples, a avaliação $r_{u,i}$ para o usuário u e o item i pode ser estimada a partir da média das avaliações atribuídas ao item i pelos n usuários mais similares a u [2]. Por isso, essa abordagem também é comumente chamada de abordagem baseada em vizinhança. A similaridade entre usuários pode ser calculada com base em diferentes métricas de similaridade e correlação como similaridade do *Cosseno* [2], distância *Euclidiana* [1], distância de *Manhattan* [1], correlação de *Pearson* [38], correlação de *Spearman* [38], etc.;

- **baseados em modelo**: usam as avaliações atribuídas para gerar modelos preditivos. De forma geral, as interações entre usuários e itens são modeladas com fatores representando características latentes dos usuários e dos itens. o modelo é então treinado com os dados disponíveis para ser usado posteriormente na predição das avaliações dos novos itens [38].

Apesar de sistemas de recomendação baseados em filtragem colaborativa apresentarem vantagens como simplicidade de desenvolvimento e eficiência na aplicação [38], alguns problemas podem ser identificados com o uso isolado dessa abordagem, incluindo:

- **esparsidade**: caso o número de avaliações dos usuários da base seja pequeno em relação ao número de avaliações que devem ser preditas, a acurácia das recomendações tende a ser menor [38];

- **novo usuário**: para gerar recomendações acuradas, os sistemas de recomendação baseados em filtragem colaborativa precisam conhecer primeiro as preferências dos usuários com base em suas avaliações atribuídas previamente [2]. Assim, para um novo usuário com poucas avaliações, as recomendações podem ser menos acuradas até que o sistema "aprenda" o seu perfil.

Por fim, os sistemas de recomendação híbridos combinam a filtragem baseada em conteúdo com a filtragem colaborativa a fim de mitigar as limitações que ambas as abordagens apresentam quando aplicadas individualmente [2]. A combinação pode ocorrer de diferentes formas [2], por exemplo: gerar ambos os sistemas separadamente e combinar suas predições; incorporar características de filtragem baseada em conteúdo em abordagens colaborativas; incorporar características de filtragem colaborativa em abordagens baseadas em conteúdo; e construir um sistema unificando características de ambas as abordagens.

Capítulo 3

Trabalhos Relacionados

Requisitos não funcionais desempenham um papel fundamental no desenvolvimento bem-sucedido de um produto de software [12]. Além disso, são considerados um fator de diferenciação entre produtos de software que apresentam funcionalidades semelhantes [4]. Apesar disso, devido às características iterativa e incremental da ER ágil, requisitos não funcionais são comumente negligenciados até as fases finais do desenvolvimento do software, resultando em problemas como aumento de custo do projeto e de esforço no seu desenvolvimento [81]. Por isso, é crescente o interesse no tema de RNFs aplicados ao contexto de desenvolvimento ágil de software. Os trabalhos encontrados na literatura que abordam esse tema são apresentados a seguir.

Villamizar et al. [117] apresentaram os resultados de um mapeamento sistemático sobre engenharia de requisitos de segurança no contexto de métodos ágeis. Como resultado, os autores identificaram 21 estudos relacionados ao tema e, após analisá-los, concluíram que as abordagens propostas no tema geralmente envolvem a modificação de métodos ágeis com a introdução de novos artefatos ou a proposição de diretrizes para auxiliar a elicitação de RNFs de segurança. As principais lacunas dos estudos identificadas pelos autores foram: falta de pesquisas sobre verificação e validação de requisitos de segurança, suporte limitado das ferramentas propostas e ausência de avaliação empírica. Por sua vez, em Curcio et al. [31], um estudo terciário para categorizar estudos secundários relacionados à integração de usabilidade e desenvolvimento ágil de software, foram identificados 14 trabalhos no tema. Os autores identificaram seis formas de integrar usabilidade a métodos ágeis e sete desafios relacionados a essa tarefa. Segundo os autores, a maioria dos estudos analisados negligen-

ciou os critérios de qualidade e apresentou dificuldades em utilizar métodos para sintetizar os resultados da pesquisa.

Suryawanshi e Rao [114] dissertaram sobre o tema de RNFs no desenvolvimento ágil, enfatizando a necessidade de considerar RNFs como segurança e desempenho em projetos ágeis. Moyon et al. [89] propuseram um método para dar suporte ao desenvolvimento contínuo considerando segurança a partir do mapeamento de requisitos padrões de segurança em um modelo de processo ágil. Em contrapartida, Camacho et al. [22] investigaram empiricamente como os membros de equipes ágeis lidam com testes não funcionais em seus projetos, visando identificar fatores preliminares que influenciam o teste de requisitos não funcionais de segurança e desempenho. Por sua vez, Krishna e Abraham [75] realizaram uma análise dos RNFs de desempenho e memória no desenvolvimento de sistemas embarcados de tempo real seguindo a filosofia incremental do desenvolvimento ágil. Como resultado, os autores descreveram formas de rastrear esses parâmetros ao longo do desenvolvimento.

Em Knauss et al. [72], um artigo de posicionamento, os autores argumentaram que uma engenharia *just-in-time* efetiva de requisitos não funcionais em projetos ágeis depende de uma base sólida de conhecimento a longo prazo sobre todos os requisitos de qualidade relevantes. Com base em suas experiências prévias, os autores afirmaram que a engenharia de requisitos ágil pode ser considerada como um problema de gerenciamento de conhecimento, abordando a aquisição de conhecimento, transformação de conhecimento tácito em explícito e vice-versa, armazenamento, divulgação e avaliação sistemática do conhecimento armazenado e, posteriormente, a aplicação em novas situações.

Firdaus et al. [8, 18, 53] apresentaram um modelo de processo de rastreabilidade para dar suporte a rastreabilidade de RNFs de segurança e desempenho no desenvolvimento ágil de software. A solução proposta baseou-se na integração de metamodelos para auxiliar a rastreabilidade de requisitos [18, 53] como também para traçar o impacto da mudança em RNFs no DAS [8]. Adicionalmente, em Younas et al. [128], um estudo complementar do mesmo grupo de pesquisa, foram apresentadas diretrizes para guiar desenvolvedores e partes interessadas na elicitação de RNFs no contexto de métodos ágeis.

Agra et al. [3] propuseram uma transformação do modelo *i** em *user stories* a fim de suprir as lacunas encontradas na documentação ágil, considerando razões, intenções e requisitos não funcionais. Em contrapartida, Aziz et al. [11] realizaram um estudo das abordagens

e métodos conhecidos para a documentação de RNFs em projetos ágeis. Por sua vez, em Behutiye et al. [16], foram apresentados resultados encontrados sobre práticas e desafios da documentação de RNFs em empresas que aplicam métodos ágeis. Os autores propuseram diretrizes para melhorar a documentação de RNFs no contexto de DAS.

Sachdeva and Chung [105] propuseram uma nova abordagem para lidar com RNFs de segurança e desempenho em projetos de software baseados em Scrum nos contextos de Internet das Coisas (IoT) e *Big Data*. No trabalho, foi proposta a consideração de RNFs de segurança como funcionalidades do sistema (conjunto de *user stories*) e RNFs de desempenho como critérios de aceitação de *user stories*.

Chugh e Mishra [28] propuseram uma abordagem que integra um método em quatro camadas para RNFs com a ER ágil (com base em quatro passos e quatro regras), visando auxiliar o processo de análise de RNFs e a priorização de *user stories*. A abordagem é centrada na identificação de partes interessadas chave para a definição de RNFs. Aljallabi e Mansour [4], por outro lado, combinaram a abordagem proposta em [28] com uma avaliação quantitativa de requisitos não funcionais para propor um processo com seis fases, que em síntese, baseia-se na interação entre partes interessadas e desenvolvedores para a definição de RNFs.

Na Tabela 3.1, pode ser vista uma síntese dos 31 trabalhos encontrados que fazem parte de 17 estudos diferentes. Estão destacados com "X" os trabalhos que apresentam soluções para projetos ágeis baseados em Scrum e trabalhos que apresentam abordagens para dar suporte a tomada de decisão na definição de RNFs.

Nas próximas seções, alguns trabalhos relacionados diretamente com o tema pesquisado neste trabalho de tese são discutidos em mais detalhes.

3.1 Suporte à Definição de Requisitos Não Funcionais em Projetos Ágeis

Em um estudo que envolve uma série de trabalhos, Farid [52], Farid e Mitropoulos [48–51], Domah e Mitropoulos [40], Maiti e Mitropoulos [81, 82, 84] e Farid et al. [83, 85] apresentaram soluções para captura, definição, priorização e predição de RNFs no contexto de DAS.

De forma geral, os autores propuseram um arcabouço de modelagem de RNFs para o contexto de DAS, chamado Modelagem de Requisitos Não Funcionais para Processos Ágeis (NORMAP) [52]. NORMAP é baseado em três blocos, são eles: i) proposta de um novo cartão de *user stories* baseado em oito questões (W^8); ii) proposta de uma taxonomia para requisitos em processo ágeis; e iii) associação entre RFs e RNFs por operadores de ponto de corte orientados a aspectos. Além disso, foi proposta uma ferramenta semiautomática de simulação para modelagem de requisitos não funcionais em processos ágeis (NORMATIC) [48], que apoia-se nos três blocos de construção do NORMAP.

Em Farid e Mitropoulos [50], foi proposto um componente para planejamento de requisitos não funcionais para processos ágeis (NORPLAN) como parte integral do arcabouço do NORMAP, usado especificamente para calcular riscos usando métricas de gerenciamento de projetos e de qualidade de requisitos e seus impactos. Como parte do NORMAP, foram propostos trabalhos para melhorar a visualização de requisitos [49, 51]. Por sua vez, em Domah e Mitropoulos [40], foi proposta uma metodologia para elicitação, raciocínio e validação de RNFs no contexto de DAS (NERV), que alcançou melhores resultados em comparação com o arcabouço NORMAP.

Como uma extensão do NORMAP [52] e do NERV [40], em Maiti e Mitropoulos [81, 82], foi proposta uma metodologia para coletar metadados de RNFs de artefatos de requisitos de software como documentos e imagens, disponibilizados nas fases iniciais dos projetos. O objetivo dos trabalhos foi possibilitar a captura, elicitação e priorização de requisitos não funcionais aplicando Reconhecimento Ótico de Caracteres (OCR). Por sua vez, em Maiti e Mitropoulo [84], foi investigado o tema de priorização de requisitos com base no arcabouço proposto em Maiti e Mitropoulos [82]. Por fim, em Farid et al. [83, 85], foi apresentada uma solução que utiliza árvores de decisão para prever RNFs de próximas iterações de projetos ágeis com base nos metadados coletados com a metodologia proposta em Maiti e Mitropoulos [81, 82].

Apesar dos trabalhos realizados nesse estudo apresentarem relevância considerável para o tema em questão, algumas lacunas podem ser observadas. Por exemplo, a base da captura de RNFs nos trabalhos foram documentos fornecidos pelos integrantes dos projetos como documentos textuais e imagens. Porém, nem sempre esses documentos estarão disponíveis, pois um dos valores ágeis é "software em funcionamento mais que documentação abran-

gente".

3.2 Requisitos Não Funcionais no Contexto de Scrum

Alguns trabalhos encontrados abordam o tema de RNFs no contexto de *Scrum* [19, 104]. Por exemplo, Bourimi et al. [19] propuseram um arcabouço ágil para a integração de engenharia de requisitos não funcionais, denominado de AFFINE. AFFINE atende simultaneamente a três necessidades, são elas: necessidade de reforçar conceitualmente a consideração de RNFs em etapas iniciais do desenvolvimento; equilíbrio explicito entre os usuários finais e as necessidades dos desenvolvedores; e necessidade de um tipo de arquitetura de referência que forneça suporte para RNFs. De forma geral, foi apresentada uma extensão do Scrum para reforçar a necessidade de considerar precocemente RNFs. Para tanto, os autores introduziram o papel de *NFR stakeholder* (partes interessadas de RNFs), que se preocupa com a realização e consideração de um respectivo RNF. Adicionalmente, os autores sugeriram uma arquitetura de apoio que fornece uma arquitetura genérica de referência para desenvolvimento de sistemas no contexto ágil. A solução proposta baseia-se apenas em um reforço conceitual e a adição de um novo papel em Scrum e não é proposta uma solução automatizada para prover suporte à tomada de decisão de integrantes de projetos de software baseados em Scrum na definição de RNFs, como proposto neste trabalho de tese.

Em contrapartida, Sabry e El-Rabbat [104] propuseram um arcabouço de refatoração arquitetural e técnicas para garantir o alcance dos níveis exigidos de RNFs por meio da formalização de *spikes* e definição de pronto dentro das práticas do Scrum. Os autores recomendaram que a incorporação de RNFs deve ser explicitamente definida no DoD dos projetos. Porém, assim como no trabalho anterior, não foi proposta uma solução automatizada para prover suporte à elicitação de requisitos não funcionais.

Tabela 3.1: Sumário dos trabalhos relacionados ao tema de pesquisa

Est.	Autores/Referência	Suporte.	Scrum	Proposta
E1	Suryawanshi e Rao [114]			Survey sobre RNFs no contexto de DAS.
E2	Villamizar et al. [117]			Mapeamento sistemático sobre segurança no contexto de DAS
E3	Curcio et al. [31]			Estudo terciário sobre usabilidade no contexto de DAS
E4	Moyon et al. [89]			Método para obter conformidade de segurança no contexto de DAS
E5	Knauss et al. [72]			Artigo de posicionamento sobre RNFs no contexto de DAS.
E6	Camacho et al. [22]			Estudo empírico sobre testes para RNFs no contexto de DAS.
E7	Behutiye et al. [16]			Práticas e desafios da documentação de RNFs no contexto de DAS.
E8	Krishna e Abraham [75]			Avaliação de RNFs no contexto de DAS.
E9	Firdaus et al. [8, 18, 53] Younas et al. [128]			Rastreabilidade de RNFs no contexto de DAS
E10	Agra et al. [3]			Documentação de RNFs no contexto de DAS.
E11	Aziz et al. [11]			Documentação de RNFs no contexto de DAS.
E12	Sachdeva and Chung [105]		X	Gerência de RNFs no contexto de DAS.
E13	Chugh e Mishra [28]			Abordagem conceitual para definição de RNFs no contexto de DAS.
E14	Aljallabi e Mansour [4]			Abordagem para definição de RNFs no contexto de DAS.
E15	Farid [52] Farid e Mitropoulos [48–51] Domah e Mitropoulos [40] Maiti e Mitropoulos [81, 82, 84] Farid et al. [83, 85]	X	X	Soluções para captura, definição, priorização e predição de RNFs no contexto de DAS.
E16	Bourimi et al. [19, 20]		X	Arcabouço conceitual para definição de RNFs no contexto de DAS.
E17	Sabry e El-Rabbat [104]		X	Arcabouço para manuseio de RNFs no contexto de DAS.

Capítulo 4

Modelo Estruturado de *User Stories*

Conforme mencionado anteriormente, o sistema de recomendação proposto baseia-se na análise de dados históricos de projetos para gerar recomendações de RNFs. Assim, é preciso estruturar as informações dos recursos de software a fim de possibilitar a coleta de informações para a operacionalização da solução. Neste capítulo, é apresentado o Modelo Estruturado de *User Stories*, bem como detalhes das etapas de sua construção. O trabalho abordado nas seções seguintes foi realizado em conjunto pelos integrantes do grupo *Intelligent Software Engineering* do VIRTUS.

4.1 Motivação

O manifesto ágil [15] destaca, entre outras coisas, que projetos ágeis devem valorizar o software em funcionamento mais que documentação abrangente. Apesar disso, a documentação de requisitos também é uma atividade importante desses projetos. Neste contexto, requisitos de usuários são comumente documentados na forma de *user stories*, que descrevem as características do software em desenvolvimento [59]. Segundo Nils C. Haugen [59], *user stories* são formuladas em uma ou duas sentenças – em linguagem cujo o conteúdo pode ser entendido pelo cliente – escritas normalmente em cartões indexados. Além disso, ao invés de transmitirem todos os detalhes sobre as características do software, os cartões servem como lembretes para conversas sobre tais características.

Porém, apesar de ser a notação de requisitos mais utilizada em projetos ágeis [129], o modelo tradicional de *user story* é escrito em linguagem natural (isto é, texto não estrutu-

rado), que dificulta a rastreabilidade de suas informações e, consequentemente, o reúso do seu conteúdo por integrantes de projetos diferentes e a recuperação de suas informações por abordagens inteligentes baseadas em dados históricos, por exemplo, sistemas baseados em aprendizagem de máquina e inteligência artificial [100].

Além do modelo de *user story* tradicional, outras abordagens de documentação de requisitos podem ser utilizadas, por exemplo, Linha de Produto de Software (LPS) [7], que é uma abordagem que possibilita o desenvolvimento de software baseado em partes reutilizáveis. Os produtos de uma LPS compartilham um conjunto de características comuns e apresentam variabilidades que os distinguem [94]. No entanto, LPS concentra-se em famílias de produtos de domínios específicos [58], por exemplo, sistemas financeiros e sistemas governamentais, que reduz sua aplicabilidade para representações mais genéricas, ou seja, independentes de domínio específico (por exemplo, sistemas de informação Web).

Logo, visando mitigar as limitações do modelo tradicional de *user stories* e de LPSs, mostrou-se necessária a investigação de uma abordagem de documentação de requisitos ágeis com foco em Scrum que permita a rastreabilidade, o reúso e a recuperação das informações, mesmo em projetos de domínios diferentes.

4.2 Metodologia de Pesquisa

Reúso de software não inclui apenas a reutilização de componentes desenvolvidos anteriormente, mas também o reaproveitamento de qualquer artefato relacionado ao desenvolvimento de um recurso de software [125]. Por exemplo, antes de iniciar o desenvolvimento de uma nova funcionalidade, desenvolvedores reusam – mesmo que implicitamente – informações de experiências em projetos passados a fim de minimizar o esforço de desenvolvimento e melhorar a velocidade e a qualidade do trabalho.

No contexto de Scrum, *user stories* são usadas para registrar os recursos de software e as necessidades do usuário [66]. Portanto, itens de um projeto tais como requisitos funcionais e não funcionais, tarefas, código fonte, casos de teste e estimativas de esforço estão relacionados ao recurso descrito por uma *user story*. Logo, ao desenvolverem novas funcionalidades, é comum que os desenvolvedores procurem por *user stories* similares prontas, visando reutilizar direta ou indiretamente seu conteúdo.

Neste contexto, o principal objetivo do estudo apresentado neste capítulo é definir um modelo estruturado de documentação de recursos de software que, diferentemente da abordagem tradicional de *user story*, viabilize o reúso de itens de software por integrantes dos projetos. Ainda, consequentemente, permita a rastreabilidade e a recuperação das informações por abordagens que se baseiam em dados históricos como sistemas de recomendação baseados em memória. Para atingir este objetivo, investigou-se a questão de pesquisa $\mathbf{RQ}_1$.

$\mathbf{RQ}_1$: como estruturar informações de recursos de software de projetos baseados em Scrum, de modo que seja possível o reúso, a rastreabilidade e a recuperação das informações?

Para responder $\mathbf{RQ}_1$, foram realizados os seguintes passos (ver Figura 4.1):

- **Levantamento bibliográfico:** investigação na literatura para buscar por trabalhos existentes que respondam a questão de pesquisa;
- **Proposta de um novo modelo de documentação:** como não foram encontrados trabalhos diretamente relevantes para responder a questão de pesquisa no passo anterior, foi proposto um modelo de documentação de recursos de software, chamado de Modelo Estruturado de *User Stories*;
- **Estudo empírico:** estudo para avaliar empiricamente o modelo proposto.

As seções seguintes detalham os passos realizados para a criação e validação do modelo proposto.

4.3 Levantamento Bibliográfico

As principais pesquisas relacionadas à reutilização de informações históricas de projetos de software baseiam-se em medidas de similaridade textual, principalmente usando LPS [76, 119]. Como mencionado anteriormente, essa abordagem concentra-se no desenvolvimento e manutenção de famílias específicas de produtos de software, aproveitando os seus aspectos comuns e observando as variabilidades previstas.

Nas últimas duas décadas, pesquisas e práticas têm sido feitas no campo de LPS [63]. Alguns trabalhos propuseram a extração automática ou semiautomática considerando diferentes tipos de fontes de informação, por exemplo, diagramas de *design*, código fonte,

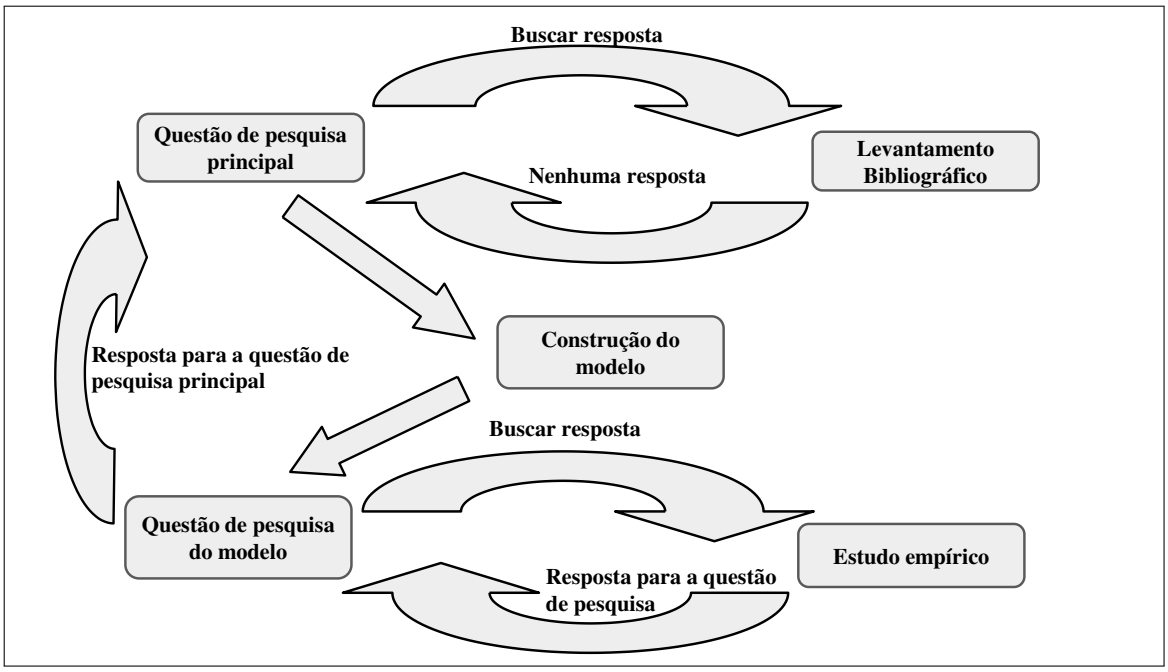


Figura 4.1: Visão geral da metodologia utilizada

descrições informais de produtos, etc. Em contrapartida, outras abordagens foram propostas para extrair informações de sistemas e seus diferentes artefatos para linhas de produtos de software. Nesta seção, o foco são os trabalhos que extraem modelos de recursos de software (do inglês, *feature model*) de documentos escritos em linguagem natural.

Bakar et al. [13] estudaram sistematicamente as diferentes abordagens de extração de características de requisitos escritos em linguagem natural. Em contrapartida, Dumitru et al. [41] e Hamza et al. [58] apresentaram sistemas de recomendação que modelam e recomendam recursos de produtos de software para um determinado domínio.

A principal vantagem de uma LPS é a reutilização sistemática da infraestrutura comum (artefatos e recursos) que é compartilhada para criar diferentes variantes de produtos de software [10]. Porém, LPSs exigem o desenvolvimento e a manutenção de documentação rigorosa e abrangente [90], o que contraria um dos valores propostos no manifesto ágil: "Software em funcionamento mais que documentação abrangente" [15]. Ao contrário da engenharia associada às LPSs, métodos ágeis envolvem um processo com planejamento e projeto inicial reduzidos. No entanto, LPSs e métodos ágeis podem ser combinados para atender às metas de negócio [32].

Robeer et al. [100] propuseram uma ferramenta que gera automaticamente um modelo

conceitual a partir de uma coleção de requisitos ágeis expressos como *USs*. Além disso, Soares et al. [112] identificaram as principais dificuldades ao trabalhar com requisitos ágeis a partir dos modelos criados em [100].

Por outro lado, alguns trabalhos apresentam modificações no modelo tradicional de *US* para melhorar sua capacidade de representação, seja visual [121], seja escrita [79, 120, 122].

Wautelet et al. [122] realizaram um estudo sobre modelos apresentados na literatura a fim de alcançar unificação de sintaxe e acordo semântico na documentação de requisitos com *USs*. Lucassen et al. [79] propuseram um arcabouço de qualidade de *US*, que baseia-se em um conjunto de 13 critérios de qualidade que devem ser levados em consideração por participantes de projetos ao escreverem *USs*. Com base nestes critérios, os autores apresentaram uma ferramenta que detecta defeitos de qualidade na escrita de *USs* e sugere possíveis soluções. A fim de melhorar a representação visual do modelo tradicional de *US*, Wautelet et al. [121] propuseram transformar elementos específicos de conjuntos de *USs* em casos de uso com base no modelo unificado proposto em [122]. Por outro lado, em [120], os autores apresentaram formas de construir um diagrama de lógica personalizada com base no modelo apresentado em [122].

Apesar de contribuírem na área engenharia de requisitos, as solução apresentadas anteriormente não atendem à demanda indicada na Seção 4.1, pois busca-se uma proposta que possibilite o reúso, a rastreabilidade e a recuperação das informações de *USs* em projetos ágeis baseados em *Scrum*.

4.4 Construção do Modelo Estruturado de *User Stories*

Como não foram encontrados trabalhos na literatura para responder a questão de pesquisa principal (ver Seção 4.2), foi proposto um modelo estruturado de descrição de *USs*. O modelo baseia-se na categorização de *USs* de acordo com uma taxonomia em dois níveis.

Para a criação do modelo, foram consideradas apenas informações de projetos Web no contexto de sistemas de informação, ou seja, o modelo apresentado neste capítulo é direcionado apenas para estes tipos de projetos. Porém, os passos apresentados a seguir podem ser realizados em demais tipos de projetos para a obtenção dos seus respectivos modelos estruturados de *user stories*.

Tabela 4.1: Exemplos de classificação de *USs* de acordo com suas características

#	Descrição da US	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

Para definir a taxonomia, foram analisados cinco *backlogs* de produto de diferentes projetos reais de uma empresa de software brasileira, contendo um total de 118 *USs* e 336 tarefas. Na Tabela 4.1 são apresentados exemplos de *USs* classificadas de acordo com suas características. Inicialmente, verificou-se o contexto da funcionalidade representada por cada *US* a fim de identificar similaridades entre elas e agrupá-las. Em seguida, foi definida uma nomenclatura para cada grupo formado. Este primeiro nível de classificação foi chamado de **Módulo**. Por exemplo, as *USs* 1, 2 e 3 do exemplo foram consideradas similares e classificadas com o módulo de "Autenticação". Por outro lado, as *USs* 4, 5 e 6 foram classificadas com o módulo "Cadastro". Adicionalmente, as *USs* de um mesmo módulo que apresentavam propósito semelhante foram agrupadas entre si e uma nomenclatura foi definida para cada subgrupo. Este segundo nível de classificação foi chamado de **Operação**. Por exemplo, as *USs* 1, 2 e 3 foram classificadas com as operações "Fazer *login* com usuário e senha", "Recuperar senha" e "Acessar pela primeira vez o sistema", respectivamente. Portanto, todas as *USs* provenientes dos *backlogs* analisados foram classificadas com uma categoria e uma subcategoria, ou seja, um módulo e uma operação.

Na Tabela 4.2 pode ser vista a taxonomia proposta com os três módulos e suas respectivas operações. A descrição de cada módulo é a seguinte: **Cadastro**, consiste em operações relacionadas a criar, ler, atualizar e excluir dados (i.e., *CRUD*); **Autenticação**, consiste em operações relacionadas à autenticação e autorização; e **Gerencial**, consiste em operações relacionadas à visualização de painéis, notificação e geração de relatórios.

Na Figura 4.2 podem ser vistas as etapas necessárias para classificar uma nova *US* usando a taxonomia proposta. O processo de classificação inicia-se quando uma nova *US* é cadastrada para um projeto. Além das informações básicas (título, descrição, etc.), deve-se selecionar um módulo e uma operação que representa a *US*, caso estes já tenham sido previamente definidos. Caso contrário, deve-se propor a criação de nova classificação com a nomenclatura do novo módulo e da nova operação. No exemplo da Figura 4.2, uma nova *US* é cadastrada e classificada com o módulo "Autenticação" e a operação "Fazer *login* com usuário e senha". O processo de criação de novos módulos e operações deve ser auditado por um profissional especialista na área para evitar que novas categorias sejam criadas sem necessidade real.

Com as *USs* categorizadas, é possível recuperar suas informações e seus itens relacionados, isto é, requisitos não funcionais, tarefas, casos de teste, etc. Na Figura 4.3 pode ser visto

Tabela 4.2: Modelo para classificar *User Stories*

Módulo	Operação
Autenticação	Fazer *login* com usuário e senha
	Fazer *login* com OAuth
	Recuperar senha
	Acessar pela primeira vez o sistema
	Validar permissão do usuário
	Atualizar perfil
	Criar conta
	Remover conta
Cadastro	Recuperação de dados
	Atualizar dados
	Inserir dados
	Remover dados
	Modificar inserção de dados
Gerencial	Visualizar painel
	Exportar relatório em PDF
	Exportar relatório em XLS
	Notificar via e-mail
	Notificar via aplicação

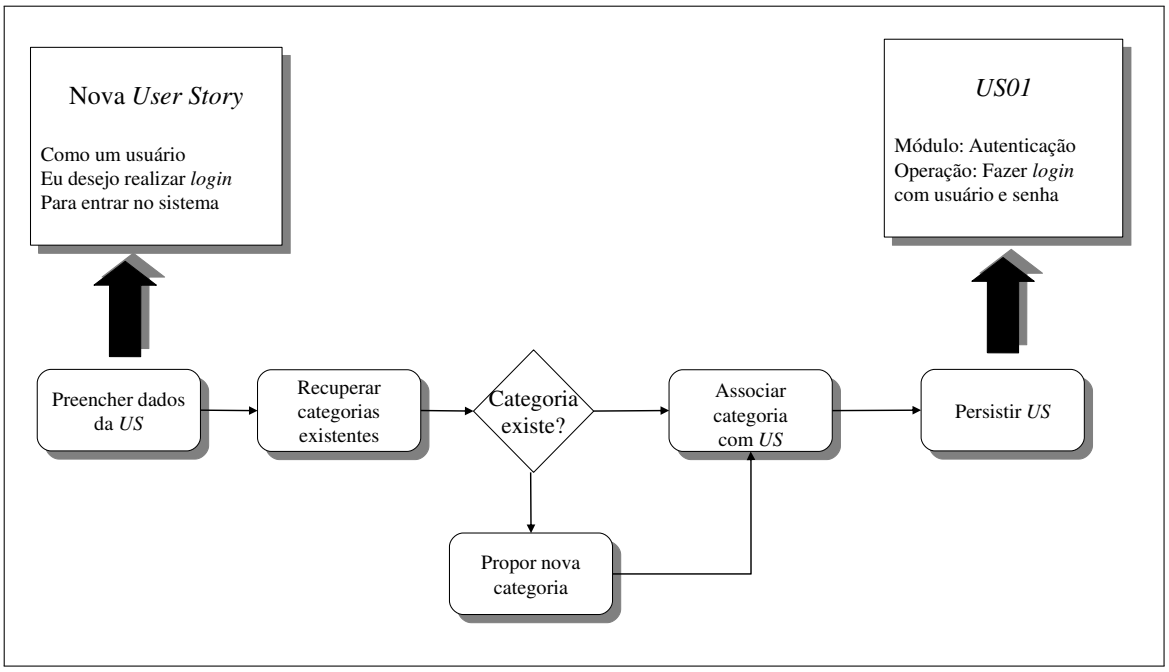


Figura 4.2: Processo para classificar *USs*

um exemplo da recuperação em um banco de dados de informações de *USs* classificadas com o módulo "Autenticação" e com a operação "Fazer *login* com usuário e senha". Estas informações poderiam ser reusadas por integrantes de projetos em execução para auxiliar na tomada de decisão. Adicionalmente, com a taxonomia proposta, é possível rastrear *USs* entre projetos diferentes, que podem ser utilizadas como entrada de abordagens baseadas na recuperação de dados históricos, como o sistema de recomendação de requisitos não funcionais apresentado no Capítulo 5.

4.5 Extensão do Modelo para Mapeamento de Tarefas

Conforme descrito anteriormente, um dos objetivos do modelo proposto é possibilitar a reutilização de itens de software por integrantes dos projetos. Para validar essa intenção, foi realizada uma extensão do Modelo Estruturado de *USs* para prover o mapeamento de tarefas técnicas e, consequentemente, possibilitar a reutilização desses itens.

Em Scrum, *USs* são decompostas em tarefas técnicas durante a 2ª fase da Reunião de Planejamento. Porém, assim como ocorre com a representação tradicional de *USs*, tarefas técnicas são comumente escritas em linguagem natural (não estruturada), que dificulta seu

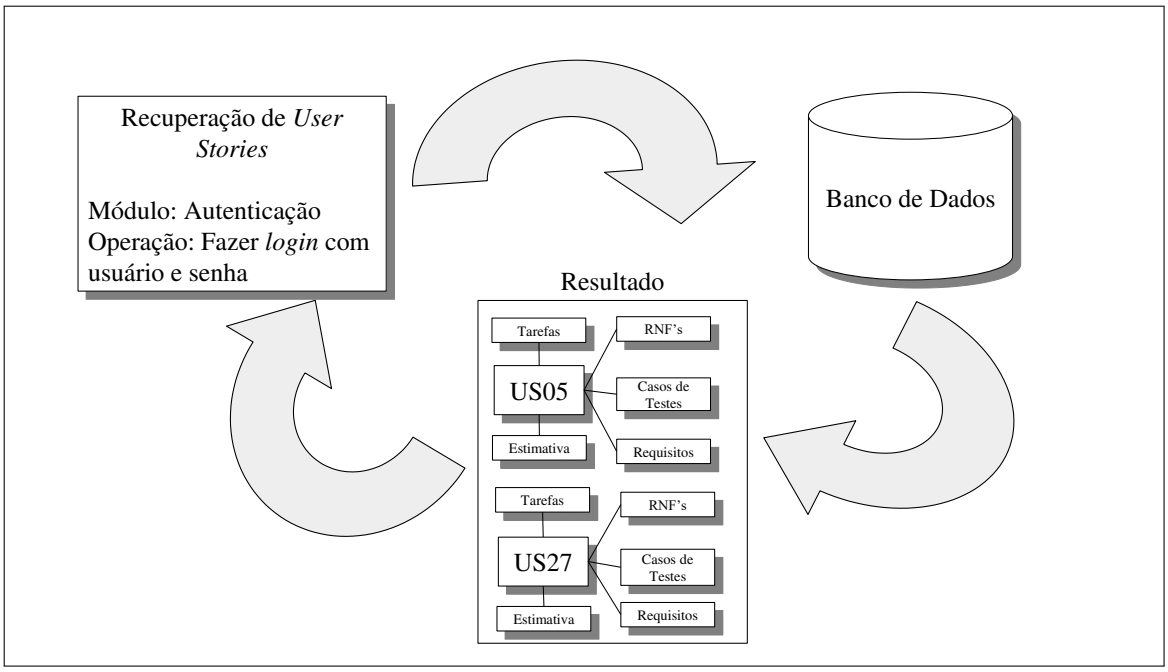


Figura 4.3: Exemplo de recuperação de informações de uma *US* por meio do módulo e da operação

reúso, mesmo com as *USs* mapeadas.

Portanto, para cada combinação de módulo e operação, foi definido um conjunto de tarefas associadas, que são descritas de acordo com uma taxonomia de tarefas. O processo para a criação dessa taxonomia é similar ao empregado na criação do modelo de *USs*, ou seja, foram analisados os descritivos das 336 tarefas dos cinco projetos citados na seção anterior. Complementarmente, foi elicitado o conhecimento técnico de desenvolvedores que auxiliaram os pesquisadores no processo de adição de novas tarefas comuns a cada operação, mas que não foram identificadas a partir da análise das tarefas dos projetos. Como resultado, obteve-se um conjunto de 131 categorias de tarefas, cada uma relacionada a um módulo e uma operação do modelo estruturado de *USs*.

Na Tabela 4.3 pode ser vista a distribuição de tarefas por módulo e operação. Por exemplo, para a operação "Fazer *login* com usuário e senha", foram definidas 13 tarefas. Na Tabela 4.4, pode-se ver a taxonomia de tarefas definida para o módulo "Cadastro" e a operação "Inserir dados". Por sua vez, o modelo estruturado de *US* completo e expandido com informações de tarefas pode ser visto no Apêndice A.

Portanto, ao cadastrar um nova *US*, é possível recuperar o conjunto de tarefas associado a

Tabela 4.3: Número de tarefas por módulo e operação

Módulo	**Operação**	**Tarefas**
Autenticação	Fazer *login* com usuário e senha	13
	Fazer *login* com *OAuth*	3
	Recuperar senha	6
	Acessar pela primeira vez o sistema	8
	Validar permissão do usuário	5
	Atualizar perfil	7
	Criar conta	8
	Remover conta	2
Total		52
Cadastro	Recuperação de dados	14
	Atualizar dados	8
	Inserir dados	9
	Remover dados	6
	Modificar inserção de dados	9
Total		46
Gerencial	Visualizar painel	4
	Exportar relatório em PDF	9
	Exportar relatório em XLS	6
	Notificar via e-mail	7
	Notificar via aplicação	7
Total		33
Total		131

Tabela 4.4: Taxonomia de tarefas definidas para o módulo “Cadastro” e a operação “Inserir dados”

Módulo	**Operação**	**Tarefas**
Cadastro	Inserir dados	Criar tela inserção
		Criar entidade no banco de dados
		Criar rota de inserção de dados
		Fazer validação de dados no cliente
		Fazer chamada para tela de inserção
		Criar método de inserção
		Validar dados inserção no servidor
		Consumir serviço para inserção
		Persistir dados banco de dados

partir da busca por módulo e operação. Na Figura 4.4 pode ser visto um exemplo de recuperação do conjunto de tarefas associado ao módulo “Cadastro” e a operação “Inserir dados”. Desta forma, a partir do conjunto retornado, é possível reusar tarefas por meio da seleção daquelas que são adequadas para o desenvolvimento da *US* em questão. Também é possível que uma tarefa decomposta da *US* não tenha sido especificada no modelo. Neste caso, uma nova categoria de tarefa deve ser criada e associada a um modelo e uma operação para posterior reúso. O processo de criação de novas tarefas deve ser auditado por um profissional especialista na área para evitar que novas categorias sejam criadas sem necessidade real.

4.6 Validação do Modelo

Para validar o modelo proposto, foi conduzido um estudo empírico, visando investigar a viabilidade de documentar requisitos de projetos baseados em Scrum com o modelo de *user story* proposto.

4.6.1 *Design* do Estudo

O objetivo do estudo empírico é avaliar a eficiência do modelo proposto com relação à cobertura e representatividade (mais detalhes na Tabela 4.5). Por isso, foram formuladas duas

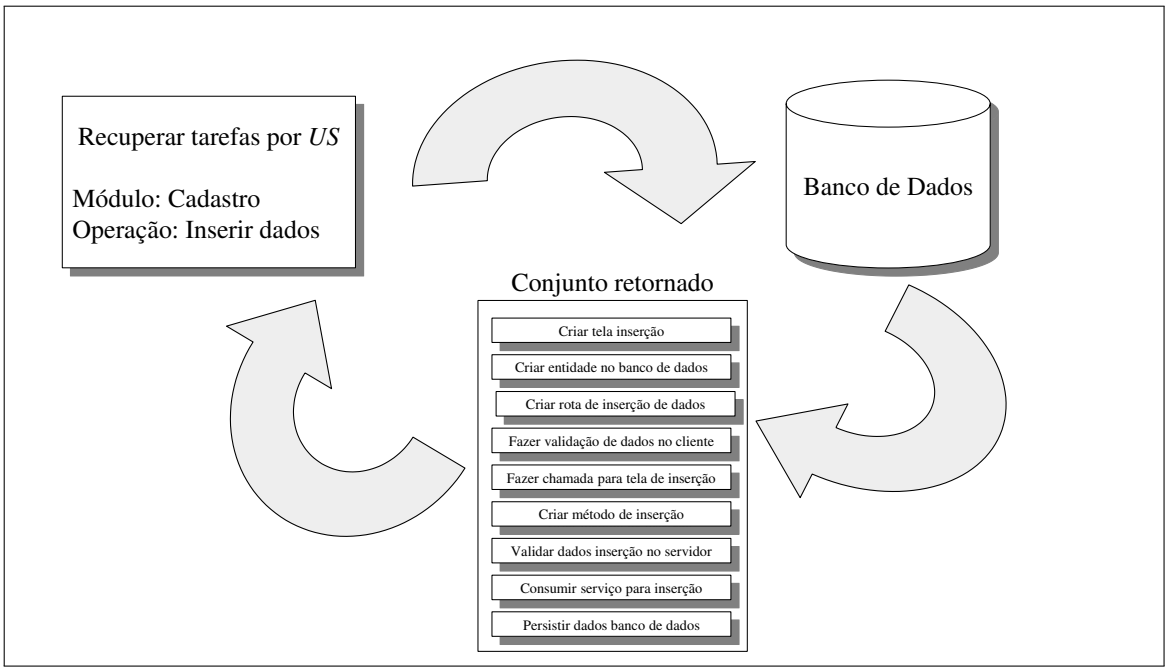


Figura 4.4: Exemplo de recuperação de tarefas associadas ao módulo "Cadastro" e a operação "Inserir dados"

questões de pesquisa secundárias que, caso respondidas positivamente, trazem evidências para responder a questão de pesquisa principal $\mathbf{RQ}_1$. São elas:

- $\mathbf{RQ}_{1.1}$: qual o nível de cobertura do modelo estruturado de *User Stories* expandido?
- $\mathbf{RQ}_{1.2}$: qual o nível de representatividade do modelo estruturado de *User Stories* expandido?

Tabela 4.5: Descrição do objetivo do estudo empírico

Objetivo do Estudo Empírico	
Objetivo	Modelo de *US*
Propósito	Caracterizar
Foco de Qualidade	Eficiência
Perspectiva	*Backlogs* de projetos de software baseados em Scrum
Contexto	Pequenas empresas de software que estão executando o *Scrum*

O estudo empírico foi executado após a coleta de *backlogs* de produto de quatro empresas de software que trabalham com processos ágeis baseados em *Scrum*. O objeto de estudo é composto por 26 projetos que possuem 530 *user stories* e 1879 tarefas. Originalmente, todas as *USs* dos projetos foram descritas de forma não estruturada, ou seja, sem nenhum tipo de classificação prévia.

De forma geral, o estudo empírico consiste em analisar as documentações dos projetos e classificar cada *US* de acordo com a taxonomia definida na Seção 4.4. É importante destacar que *USs* destinadas à especificação de atividades técnicas (*USs* Técnicas), por exemplo, configuração de ambiente de desenvolvimento, não são consideradas no modelo proposto e, consequentemente, na sua avaliação. Portanto, das 530 *USs* coletadas inicialmente, 407 foram consideradas para a avaliação, pois são *USs* que especificam funcionalidades em si, chamadas de *USs* de Negócio.

Conforme descrito na Seção 4.4, a taxonomia proposta apresenta três módulos: Autenticação, Cadastro e Gerencial. Do montante de *USs* de negócio, foram classificadas como *Não mapeada* aquelas em que não foi possível atribuir um dos módulos. Desta forma, a medida que define a cobertura do modelo ($\mathbf{RQ}_{1.1}$) é dada pelo número de *USs* classificadas pelo modelo sobre o número total de *USs* de negócio (Equação 4.1), em que a cobertura ideal é 1 (100% de cobertura).

$$Cobertura = \frac{\mid Número\ de\ USs\ de\ negócio\ classificadas \mid}{\mid Número\ total\ de\ USs\ de\ negócio \mid}. \tag{4.1}$$

Além da cobertura, foi analisada também a representatividade do modelo, ou seja, a capacidade de representar de forma coerente o maior número possível de *USs* dos projetos com o menor número possível de módulos e operações. A representatividade do modelo ($\mathbf{RQ_{1.2}}$) é dada pelo número total de operações propostas na taxonomia (Seção 4.4) sobre o número total de *USs* de negócio classificadas (Equação 4.2). Assim, quanto mais próxima de 1, pior é a representatividade do modelo.

$$Representatividade = \frac{\mid Número\ total\ de\ operações\ do\ modelo \mid}{\mid Número\ total\ de\ USs\ de\ negócio\ classificadas \mid}. \tag{4.2}$$

De forma análoga, foram avaliadas a cobertura e a representatividade do modelo quanto ao mapeamento de tarefas. Desta forma, a medida que define a cobertura do modelo para tarefas ($\mathbf{RQ}_{1.1}$) é dada pelo número de tarefas classificadas sobre o número total de tarefas

provenientes de *USs* de negócio (Equação 4.3), em que a cobertura ideal é 1 (100% de cobertura). Por sua vez, a representatividade do modelo para tarefas ($\mathbf{RQ}_{1.2}$) é dada pelo número total de categorias de tarefas propostas na taxonomia (Seção 4.4) sobre o número total de tarefas provenientes de *USs* de negócio (Equação 4.4). Assim, quanto mais próxima de 1, pior é a representatividade do modelo.

$$Cobertura = \frac{\mid Número\ de\ tarefas\ de\ USs\ de\ negócio\ classificadas \mid}{\mid Número\ total\ de\ tarefas\ de\ USs\ de\ negócio \mid}. \tag{4.3}$$

$$Representatividade = \frac{\mid Número\ total\ de\ categorias\ de\ tarefas\ do\ modelo \mid}{\mid Número\ total\ de\ tarefas\ de\ USs\ de\ negócio\ classificadas \mid}. \tag{4.4}$$

4.6.2 Resultados do Estudo Empírico

Nesta seção, são apresentados os resultados obtidos por meio do estudo empírico com relação às medidas de cobertura e representatividade do modelo de *USs* e tarefas.

Como informado anteriormente, os resultados apresentados nessa seção foram obtidos por meio da análise de 26 *backlogs* de produto de diferentes projetos. Na Tabela 4.6 podem ser vistos exemplos de *USs* de três projetos diferentes rotulados como *Projeto 1*, *Projeto 2* e *Projeto 3*. Observando os exemplos apresentados, pode-se notar, por exemplo, que as *user stories US1* e *US4* podem ser consideradas semelhantes, pois apresentam o mesmo módulo e a mesma operação (ou seja, "Autenticação" e "Fazer *login* com usuário e senha"), mesmo sendo *USs* de projetos diferentes. Além disso, pode-se observar na tabela alguns exemplos de *USs* de negócio que não foram mapeadas (*US8*) e *USs* técnicas (US9).

Como pode ser visto na Figura 4.5, foi obtido um valor de 90,17% de cobertura considerando *USs* de negócio. Além disso, na Figura 4.6 pode ser vista a proporção de *USs* de negócio por módulo. A maioria das *USs* foram classificadas com o módulo "Cadastro" (80,11%). Uma vez que os projetos avaliados são sistemas de informação Web, este resultado já era esperado. Por outro lado, *USs* de "Autenticação" e "Gerencial" representaram 10,35% e 9,54% das *USs* de negócio mapeadas, respectivamente. Este resultado também é justificável, pois funcionalidades de autenticação e geração de relatórios são menos recorrentes nesses tipos de projetos do que funcionalidades relacionadas a cadastro de dados.

Quanto à medida de representatividade, foi obtido um valor de aproximadamente $0,049$,

Tabela 4.6: Exemplo de classificação de *USs*

Projeto	#	Descrição da *US*	Módulo	Operação
1	US1	Realizar o *login*	Autenticação	Fazer *login* com usuário e senha
1	US2	Criar uma seleção de projeto	Cadastro	Inserir dados
1	US3	Visualizar lista de contratos	Cadastro	Recuperação de dados
2	US4	*Login* dos terapeutas	Autenticação	Fazer *login* com usuário e senha
2	US5	Cadastro de Beneficiários	Cadastro	Inserir dados
2	US6	Relatório de frequência	Gerencial	Exportar relatório em PDF
3	US7	Visualizar os critérios de validação associados a cada vantagem escolhida	Cadastro	Recuperação de dados
3	US8	Definição de *thresholds*	*Não mapeada*	*Não mapeada*
3	US9	Fazer *deploy* da aplicação	N/A	N/A

pois foram necessárias 18 operações para classificar 367 das 407 *USs* de negócio no contexto de sistemas de informação Web.

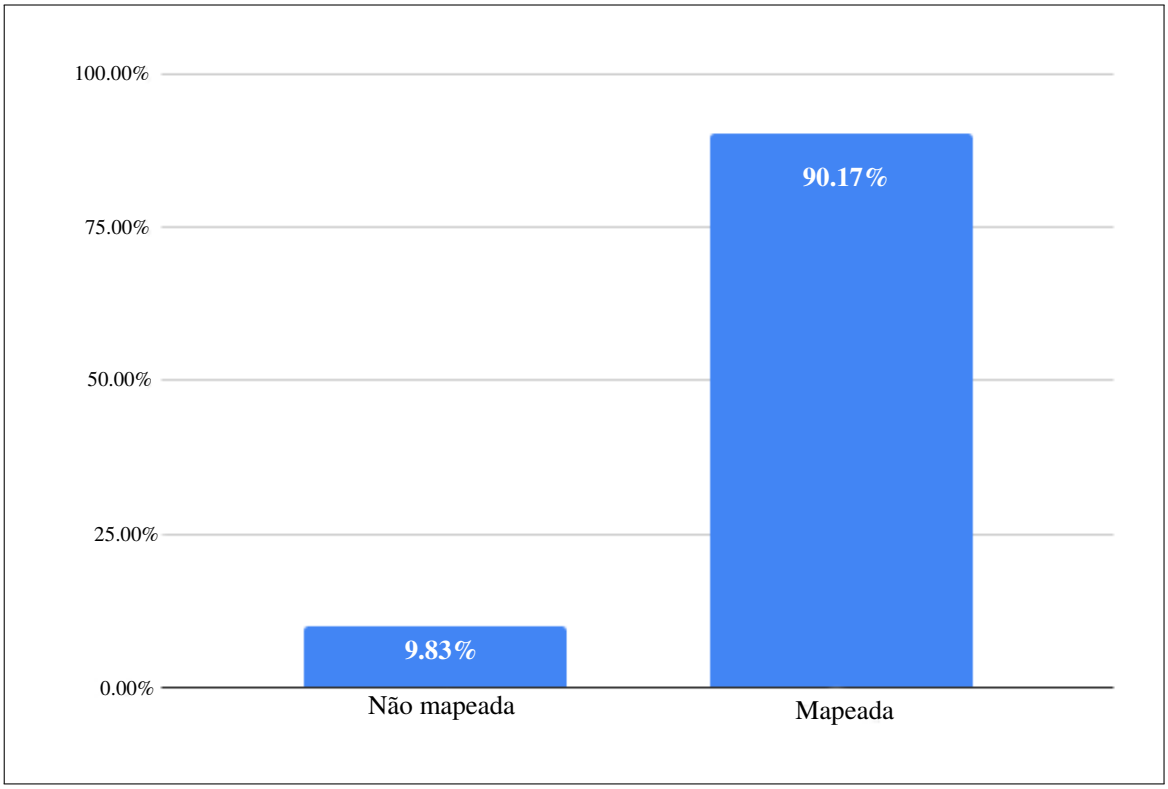


Figura 4.5: Cobertura do modelo de *US* proposto

Após a análise dos dados das *USs*, realizou-se a análise das métricas quanto à representação de tarefas. É importante destacar que tarefas provenientes de *USs* técnicas foram desconsideradas para os cálculos das métricas de cobertura e representatividade. Desta forma, das 1879 tarefas, foram analisadas 1206 tarefas de *USs* de negócio.

Na Tabela 4.7 podem ser vistos exemplos de tarefas de três projetos diferentes, *Projeto 1*, *Projeto 2* e *Projeto 3*. Ao observar os exemplos, pode-se notar, por exemplo, que as tarefas das *user stories US1* e *US4* podem ser consideradas semelhantes, pois ambas foram classificadas como "Criar campo *Login* e Senha", mesmo sendo provenientes de *USs* de projetos diferentes.

Na Figura 4.7 são apresentados os resultados de cobertura do modelo para a representação de tarefas. Um total de 976 das 1206 tarefas provenientes de *USs* de negócio foram mapeadas, ou seja, 80,93%. Em contrapartida, não foi possível mapear 230 tarefas, ou seja, um percentual de 19,07%. Quanto à medida de representatividade, foi obtido um valor de

Tabela 4.7: Exemplo de classificação de tarefas provenientes de *USs* de negócio

Projeto	#	Descrição da US	Descrição da tarefa	Módulo	Operação	Tarefa do modelo
A	US1	Realizar o login	Criar a tela de login (Front-End)	Autenticação	Fazer login com usuário e senha	Criar campos Login e Senha
A	US2	Criar uma seleção de projeto	Criar uma tela para cadastro de uma seleção (Front-End)	Cadastro	Inserir dados	Criar tela inserção
A	US3	Visualizar lista de contratos	Criar service para listagem de Contratos	Cadastro	Recuperação de dados	Criar serviço de listagem
B	US4	Login dos terapeutas	criar campos no frontend	Autenticação	Fazer login com usuário e senha	Criar campos Login e Senha
B	US5	Cadastro de Beneficiários	Criar front com campos e botões	Cadastro	Inserir dados	Criar tela inserção
B	US6	Relatório de frequência	Criar serviço	Gerencial	Exportar relatório em PDF	Criar serviço para exportação de relatórios
C	US7	Desejo visualizar e gerenciar as informações da minha equipe	Criar tela de listagem de equipes de um determinado coach	Cadastro	Recuperação de dados	Criar tela listagem
C	US8	Desejo editar perfil de aluno	Criar serviço para editar perfil de aluno	Cadastro	Atualizar dados	Criar serviço alteração
C	US9	Criar um contrato para associar às contas criadas	Criar tela de criação de contrato	Cadastro	Inserir dados	Criar tela inserção

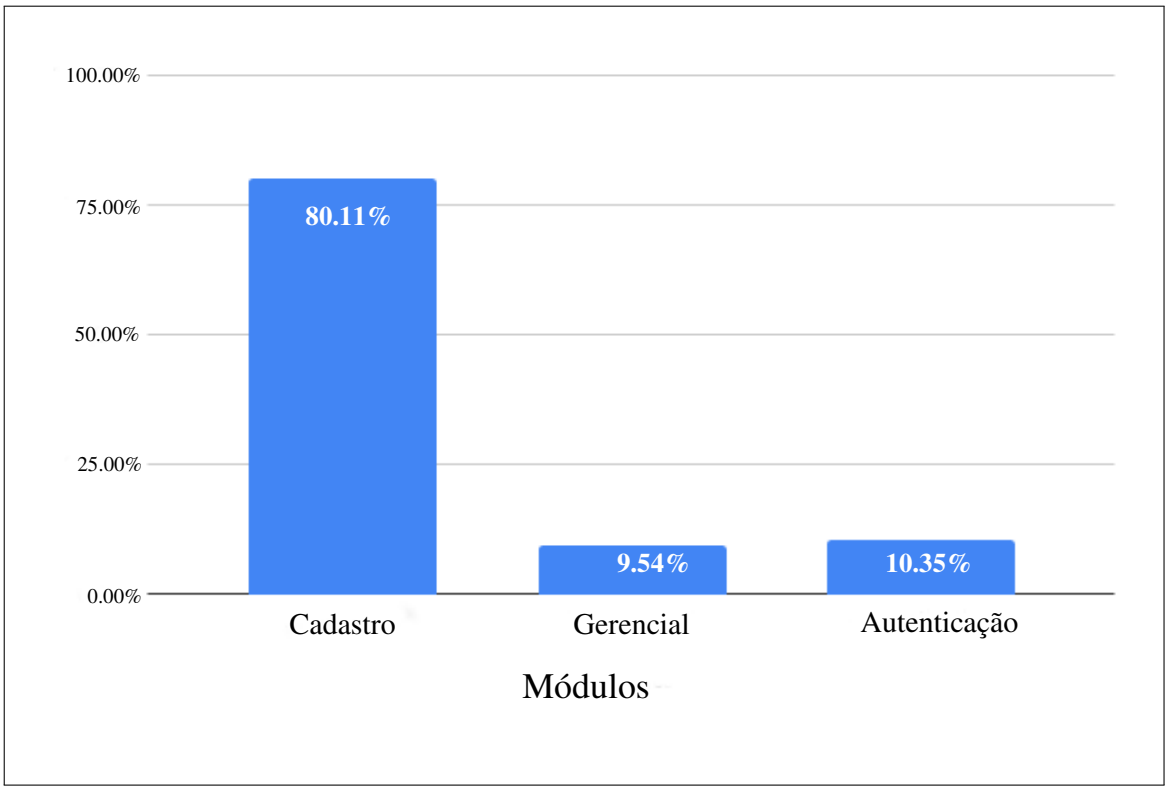


Figura 4.6: Proporção de *USs* de negócio mapeadas por módulo

aproximadamente $0,13$, pois foram necessárias 131 categorias de tarefas para classificar 976 das 1206 tarefas provenientes de *USs* de negócio no contexto de sistemas de informação Web.

4.6.3 Discussão dos Resultados

Nesta seção, os resultados obtidos com o estudo empírico são discutidos e as questões de pesquisa secundárias e principal são respondidas.

RQ$_{1.1}$: cobertura do modelo. Ao analisar os resultados, é possível observar que 367 das 407 *USs* de negócio foram classificadas, ou seja, uma medida de cobertura de 90,17%. Embora não haja dados empíricos para avaliar essa métrica, o valor pode ser considerado positivo, uma vez que a maioria das *USs* foram mapeadas. Este resultado indica que 90,17% das funcionalidades descritas por *USs* em diferentes projetos podem ser rastreadas e, consequentemente, suas informações podem ser retornadas com uma simples recuperação de dados por módulo e operação, que possibilita diferentes tipos de aplicações como a recomendação de requisitos não funcionais apresentada no Capítulo 5. Adicionalmente, foi obtida uma co-

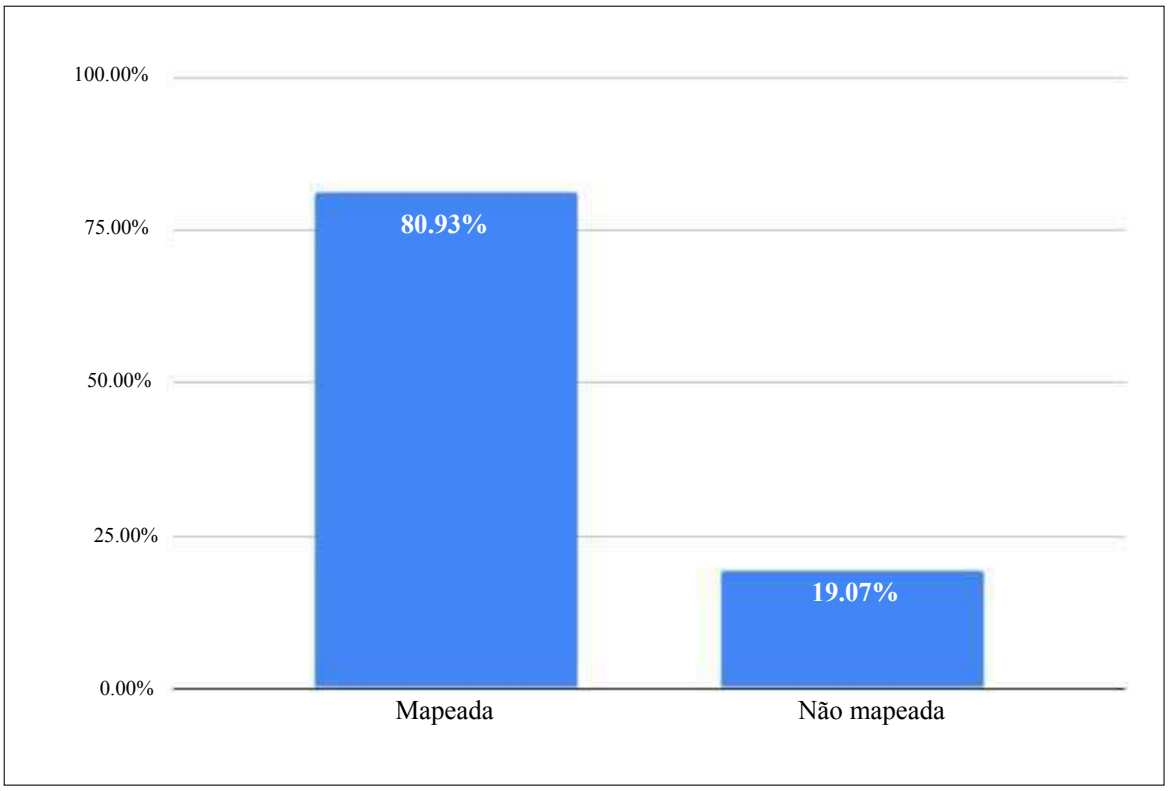


Figura 4.7: Cobertura do modelo de *US* expandido para a representação de tarefas

bertura para tarefas de 80,93%. Portanto, teoricamente, 8 em cada 10 tarefas dos projetos poderiam ser reusadas.

RQ$_{1.2}$: representatividade do modelo. Ao analisar os resultados, é possível observar que foram necessárias apenas 18 operações para mapear 367 *USs* de negócio (90,17%) de diferentes projetos de domínios distintos no contexto de sistemas de informação Web, ou seja, apenas 9,83% das *USs* de 26 projetos diferentes não foram mapeadas pelo modelo. Porém, como foi apresentado na Seção 4.4, o modelo pode ser atualizado caso necessário. Embora não haja dados empíricos para avaliar essa métrica, de forma geral, pode-se concluir que o modelo apresenta uma medida de representatividade positiva para o contexto estudado. Adicionalmente, com 131 categorias de tarefas, foi possível representar 976 de 1206 tarefas dos 26 projetos.

Portanto, existem evidências de que é possível classificar *USs* e tarefas de um determinado contexto com o modelo de *US* expandido com informações de tarefas. Além disso, o estudo empírico comprovou a viabilidade de usar o modelo proposto para modelar informações de recursos de software de projetos baseados em Scrum, de modo que seja possível a

rastreabilidade de informações entre projetos e, consequentemente, a recuperação e o reúso dessas informações como é apresentado com mais detalhes no Capítulo 5, respondendo positivamente a questão de pesquisa principal ($\mathbf{RQ}_1$).

4.7 Ameaças à Validade

Ameaças à validade dizem respeito a fatores externos que podem afetar o resultado de um estudo. Neste trabalho, considerou-se a classificação de ameaças à validade proposta por Wohlin et al. em [126]. A seguir, são apresentadas as ameaças à validade do estudo empírico.

Ameaça à validade interna. O processo de classificação das *USs* foi realizado em conjunto pelos pesquisadores do grupo ISE, ou seja, cada pesquisador ficou responsável pela classificação de um conjunto de *USs*. Desta forma, é possível que *USs* semelhantes tenham sido classificadas com categorias diferentes, resultando em inconsistências no conjunto de dados. Porém, para mitigar este problema e reduzir o viés, cada *US* foi classificada por um pesquisador e revisada por outro, ou seja, a tarefa foi realizada em pares. Além disso, outra ameaça à validade interna tem relação com a busca que foi feita na literatura por abordagens de documentação de requisitos. Como foi feito apenas um levantamento bibliográfico, trabalhos relevantes na área podem ter sido desconsiderados. Porém, para mitigar este problema, a busca na literatura foi realizada por quatro pesquisadores, visando evitar o viés.

Ameaça à validade de construção. Os resultados do estudo empírico foram obtidos com base no cálculo das medidas de cobertura e representatividade. Com base nessas medidas, concluiu-se que o modelo proposto pode ser usado para classificar e recuperar dados históricos, pois foi obtida uma medida de cobertura de 90,17% e uma medida de representatividade de aproximadamente $0,049$, ou seja, foi possível classificar grande parte das *USs* de negócio com uma proporção pequena entre a quantidade de operações e de *USs*. Porém, não há suporte empírico que indique a consistência desses números, que podem ser coletados em trabalhos futuros.

Ameaça à validade externa. As categorias propostas na taxonomia foram definidas com base em um conjunto de dados de projetos de uma única empresa. Além disso, para o estudo empírico, foram coletados *backlogs* de produto de quatro empresas. Ambos os conjuntos de *backlogs* de produto – os usados para definir as categorias utilizadas no estudo empírico,

bem como os objetos de estudo - são provenientes de projetos no contexto de sistemas de informação Web, o que explica o grande número de *USs* relacionadas a cadastro de dados. Portanto, considerando outro contexto, podem ser obtidos resultados diferentes quanto às categorias e à proporção entre operações da taxonomia e *USs* de negócio, representando uma ameaça à validade de conclusão.

4.8 Considerações Finais do Capítulo

Neste capítulo, apresentou-se um modelo de documentação de recursos de software, chamado de Modelo Estruturado de *User Stories*. Adicionalmente, foi apresentada uma expansão do modelo para prover a representação de tarefas.

Para validar do modelo proposto, foi realizado um estudo empírico com informações de *USs* e tarefas de 26 projetos de software de quatro empresas de software brasileiras. Como resultado, foi obtida uma taxa de cobertura de 90,17% das *USs* de negócio avaliadas, ou seja, foi possível mapear com módulo e operação 9 em cada 10 *USs* do projetos. Quanto à cobertura obtida na representação de tarefas, obteve-se uma taxa de 80,11%, ou seja, foi possível mapear 8 em cada 10 tarefas com o conjunto de categorias de tarefas proposto. Quanto à representatividade, foram necessárias apenas 18 operações para representar 367 *USs* de negócio e 131 categorias de tarefas para representar 976 tarefas. Portanto, concluiu-se que é possível usar o modelo proposto para estruturar informações de recursos de software de projetos baseados em Scrum, viabilizando a rastreabilidade de informações entre projetos e, consequentemente, a recuperação e o reúso dessas informações como apresentado no Capítulo 5, dando suporte para responder positivamente a questão de pesquisa principal ($\mathbf{RQ}_1$).

Capítulo 5

Sistema de Recomendação de Requisitos Não Funcionais

Neste capítulo, é apresentado o sistema de recomendação de requisitos não funcionais, que visa dar suporte aos integrantes de projetos de software baseados em Scrum na elicitação de RNFs.

5.1 Visão Geral da Solução Proposta

Como já mencionado ao longo deste documento, o processo de análise de requisitos iterativo e incremental do Scrum é eficiente na entrega rápida de requisitos funcionais de qualidade. Porém, é comum que requisitos não funcionais sejam negligenciados até os últimos estágios do desenvolvimento do software. Por isso, neste trabalho, propõe-se uma solução para auxiliar integrantes de equipes de desenvolvimento Scrum na definição de requisitos não funcionais durante o processo de análise de requisitos, de forma que sejam capazes de tomar decisões mais embasadas e precoces. A solução proposta é baseada em um sistema de recomendação que utiliza o modelo estruturado de *USs* descrito no Capítulo 4.

Na Figura 5.1 é apresentada uma visão geral da solução proposta. Nas reuniões de planejamento de *Sprint* (1), a equipe de desenvolvimento estrutura as informações dos requisitos funcionais com o Modelo Estruturado de *USs* (2). O conteúdo estruturado é coletado e processado pelo sistema de recomendação baseado em memória (3), que gera recomendações de RNFs para cada *US* de entrada a partir da análise de dados históricos (4). Por fim, o

PO e a equipe de desenvolvimento avaliam as recomendações retornadas, podendo aceitá-las ou rejeitá-las. Caso sejam aceitas, os RNFs recomendados são adicionados ao documento de requisitos do projeto. Caso contrário, o *feedback* negativo é armazenado a fim de realimentar o sistema de recomendação.

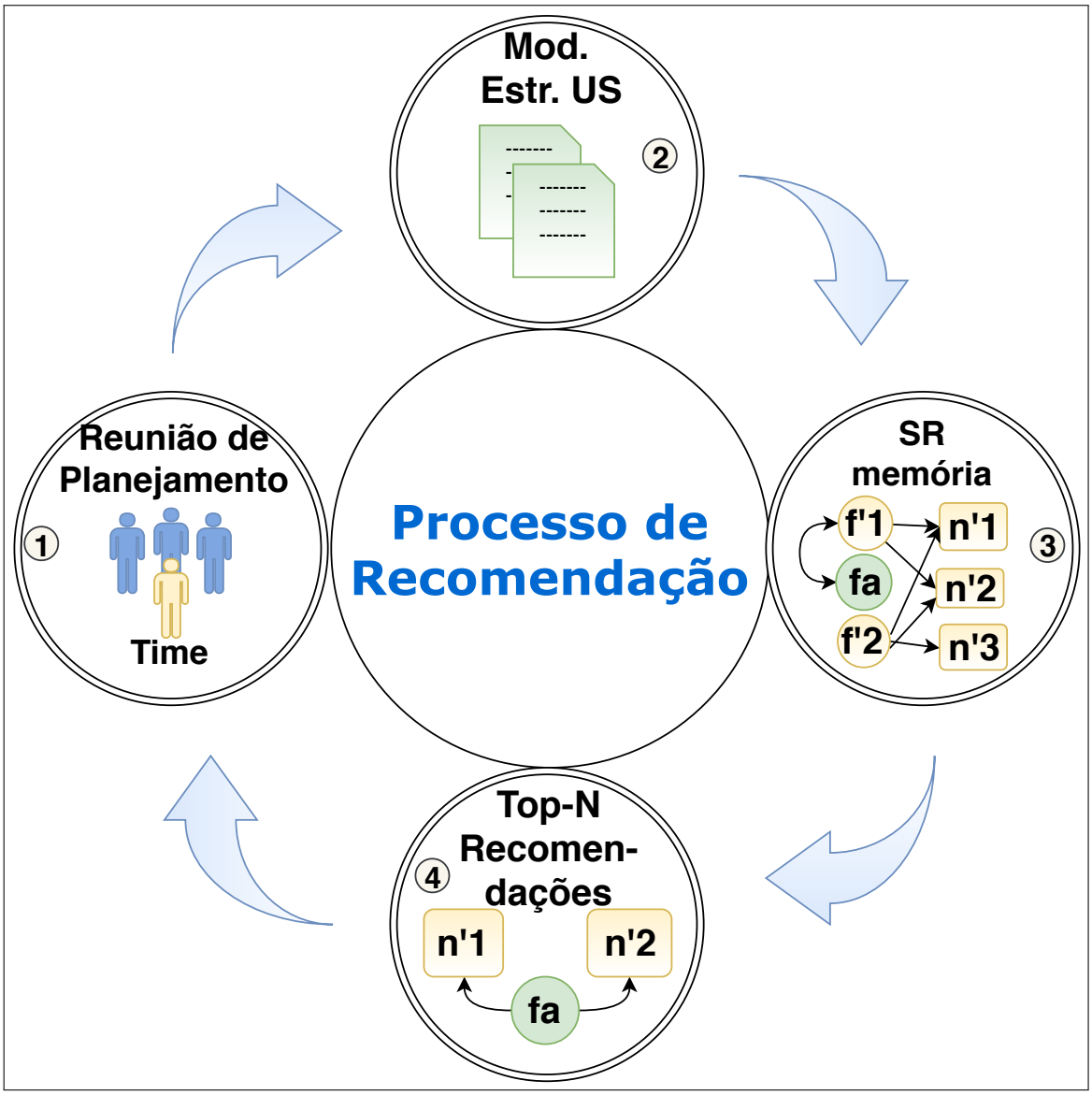


Figura 5.1: Visão geral da solução proposta

A solução proposta foi integrada a uma ferramenta Web de gerenciamento de projetos baseados em Scrum, chamada de *NFRec*. Alguns exemplos apresentados neste documento são casos reais extraídos dessa ferramenta.

O processo de adaptação do Scrum e o sistema de recomendação são detalhados nas subseções seguintes. A adaptação foi realizada em conjunto com Antônio Alexandre Moura

Costa, aluno de doutorado integrante do grupo de pesquisa *ISE* da UFCG, cujo trabalho está relacionado à formação de múltiplas equipes em projetos ágeis baseados no Scrum.

5.2 Adaptação do Scrum

A adaptação do Scrum consiste em adicionar novos elementos ao seu arcabouço, modificando sua estrutura original. Neste trabalho, as mudanças propostas estão relacionadas à inserção de elementos para estruturar as informações de projetos e viabilizar a operacionalização do sistema de recomendação.

Para realizar as atividades descritas nas seções subsequentes, elicitou-se o conhecimento de três gerentes de projetos de software com experiência em Scrum de quatro a 10 anos. Adicionalmente, dois questionários foram utilizados como instrumento de coletada de dados de profissionais da área, visando complementar o conhecimento dos especialistas supracitados. Ambos os questionários foram desenvolvidos por meio do serviço *Google Forms* [1], onde foi gerado um *link* e disponibilizado para os entrevistados.

O primeiro questionário aplicado corresponde a uma pesquisa espontânea, ou seja, não foram disponibilizadas opções de respostas previamente definidas para evitar viés (ver Apêndice B). Ao todo, 26 profissionais o responderam, dos quais: 14 analistas de qualidade, seis desenvolvedores, dois Scrum *Masters*, dois líderes técnicos, um *coach* técnico e um gerente de projeto. Em contrapartida, o segundo questionário compreende uma pesquisa estimulada, no qual os participantes puderam escolher entre as opções de respostas elicitadas anteriormente pelos três especialistas (ver Apêndice C). Ao todo, 32 profissionais o responderam, dentre eles: oito Scrum *Masters*, oito desenvolvedores, quatro líderes técnicos, quatro analistas de qualidade, três *coaches* técnicos, dois *Product Owners*, dois gerentes de projeto e um professor de nível superior com experiência em Scrum.

As modificações propostas na adaptação do Scrum são apresentadas a seguir.

5.2.1 Estruturação dos Perfis de Projetos

Ao iniciar um novo projeto, seguindo práticas ágeis ou tradicionais, é comum que suas informações gerais sejam documentadas na forma de um termo de abertura de projeto (do inglês,

[1] *Google Forms*: https://www.google.com/forms/about/

Project Charter) [60]. Porém, o conteúdo desse tipo de documento é comumente descrito em linguagem natural não estruturada. Por isso, a primeira atividade realizada na adaptação do Scrum refere-se à estruturação dos perfis dos projetos. A abordagem de estruturação proposta é baseada na atribuição de *tags* aos perfis, visando possibilitar a rastreabilidade das informações. Porém, antes de iniciar o processo, investigaram-se quais características/fatores de projetos de software podem influenciar na definição de RNFs. Essa etapa teve como finalidade decompor os perfis dos projetos em diferentes características para tornar possível a atribuição de *tags* por característica específica.

Uma visão geral da metodologia de pesquisa aplicada nesta etapa do trabalho é apresentada na Figura 5.2.

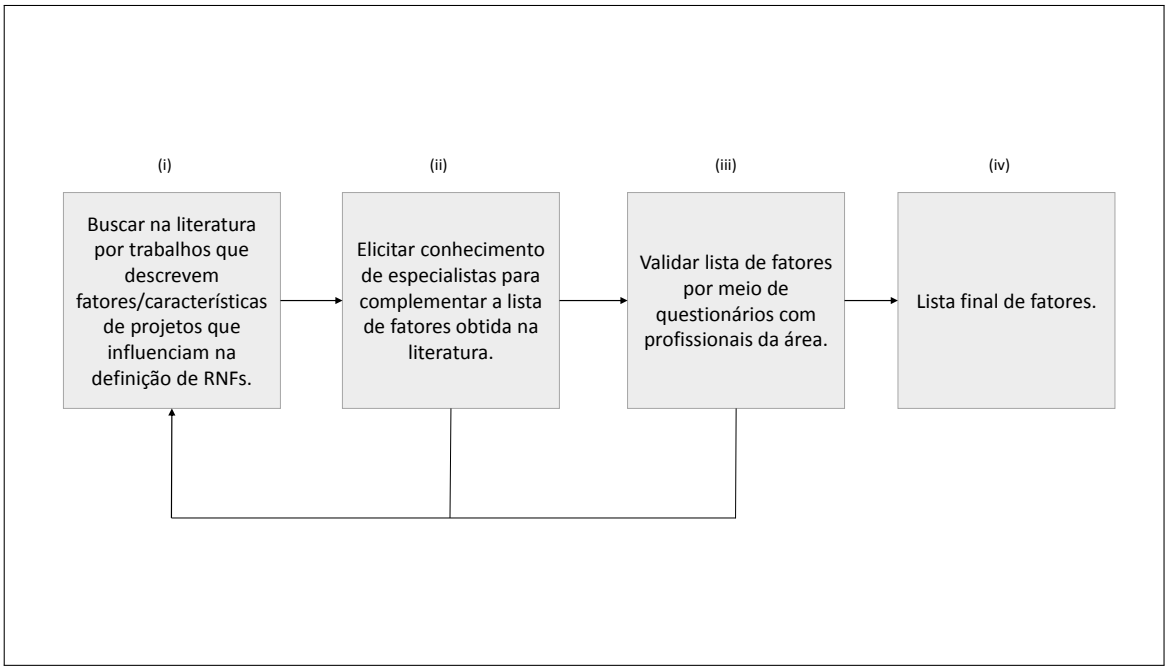


Figura 5.2: Visão geral da metodologia utilizada para definição de fatores/características de projetos que incidem na definição de RNFs

Primeiramente, após levantamento bibliográfico de trabalhos que abordam o tema, foram identificados dois fatores relevantes, são eles: domínio de aplicação (i.e., saúde, bancário, etc.) [35, 46, 80] e categoria do projeto ou plataforma (i.e., web, móvel, embarcado, *desktop*, etc.) [35, 80]. Posteriormente, com o auxílio dos três gerentes de projeto, os dois fatores identificados na literatura foram confirmados e mais três foram elicitados, são eles: objetivo do projeto (i.e., produto ou protótipo), arquitetura de software (i.e., cliente-servidor, *MVC*,

camadas, etc.) e tecnologias utilizadas no desenvolvimento (i.e., *java*, *mongo*, etc.).

Por fim, avaliou-se a escolha dos cinco fatores com os dois questionários mencionados anteriormente. Por se tratar de uma pesquisa espontânea, o questionário 1 não dispunha de alternativas de respostas previamente definidas. Portanto, de acordo com suas experiências prévias, cada participante respondeu em texto livre a respeito de quais fatores de projetos incidem na definição de RNFs.

Os resultados da pesquisa são apresentados na Figura 5.3. Para gerar o gráfico mostrado, realizou-se um mapeamento entre as respostas textuais e os fatores previamente elicitados [30]. Por exemplo, uma das respostas concedida foi a seguinte: "*Deve-se levar em consideração características como a definição da plataforma usada pelo software, qual a finalidade do software, se será usado apenas pelo cliente internamente ou externamente, público alvo do software, entre outros*". A mesma foi contabilizada para os fatores "domínio de aplicação" e "plataforma".

Após o mapeamento, o fator "plataforma", com 15 respostas de 26 possíveis, obteve o maior número de indicações, seguido por "domínio de aplicação" com oito, "arquitetura de software" com sete e "objetivo do projeto" e "tecnologias utilizadas" com três indicações cada. Além disso, os respondentes indicaram que comumente são consideradas experiências passadas em projetos similares para auxiliar a elicitação de RNFs de projetos correntes, que é uma prática recomendada pelo PMBOK® [93], conhecida como utilização de lições aprendidas (do inglês *lessons learned*). Ao todo, sete participantes deram respostas nesse contexto. Por exemplo, um dos respondentes afirmou: "*Verificando os requisitos funcionais e as características esperadas do sistema, é possível buscar sistemas similares para tomar como base para listar os RNFs esperados e típicos as funcionalidades principais.*".

Por outro lado, no questionário 2 (pesquisa estimulada), os respondentes puderam escolher entre os fatores elicitados anteriormente. Cada participante pôde escolher mais de um fator dos cinco elicitados pelos gerentes como também informar opcionalmente novos fatores não listados. Os resultados da pesquisa são mostrados na Figura 5.4, onde observa-se que o fator plataforma, com 27 respostas de 32 possíveis, obteve o maior número de indicações, seguido por domínio de aplicação com 22, objetivo com 18, tecnologias utilizadas com 13 e arquitetura de software com 12. Além disso, quatro outros fatores foram citados pelos participantes. Porém, não foram considerados válidos, pois representam requisitos não

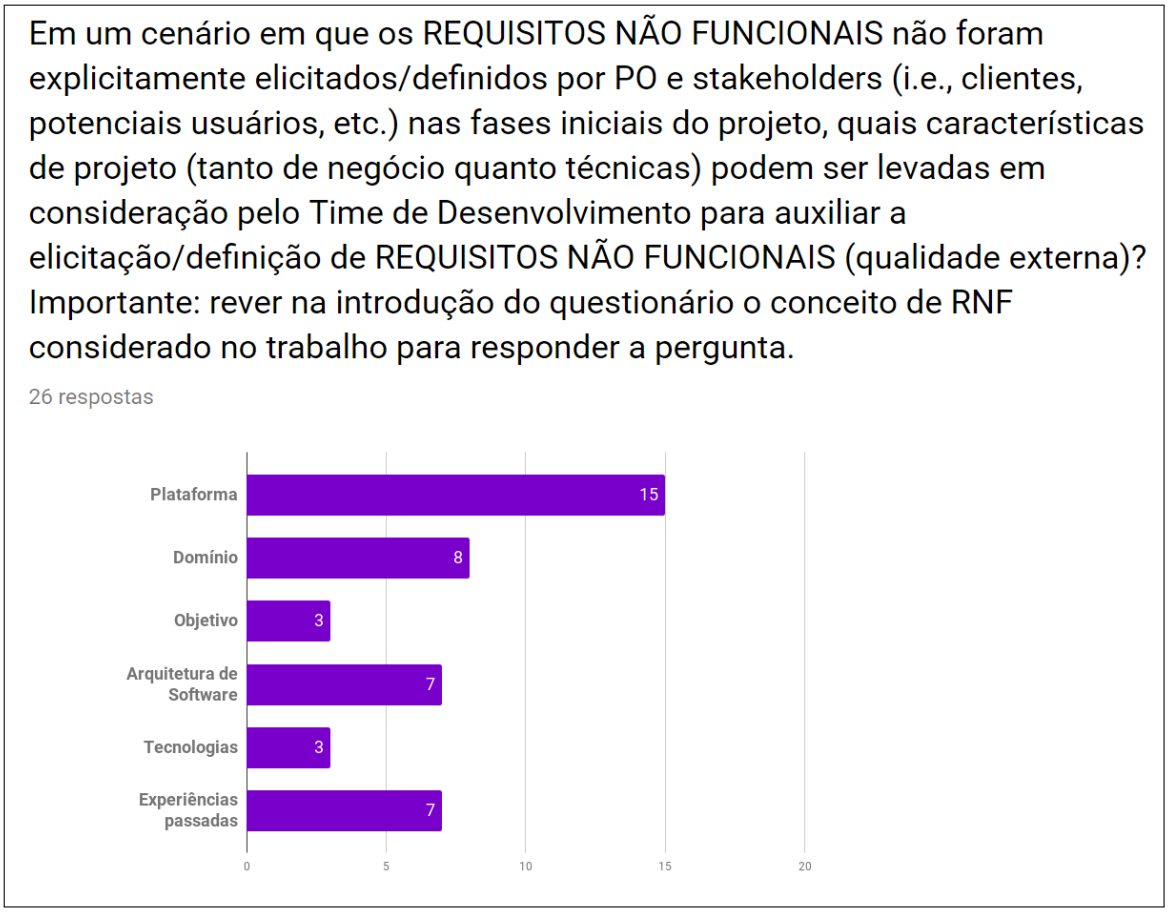


Figura 5.3: Resultado de questionário espontâneo sobre fatores chave na definição de requisitos não funcionais

funcionais em si, ao invés de características de projetos. Por exemplo, um dos respondentes citou *"segurança dos dados"* como fator que incide na definição de RNFs.

Figura 5.4: Resultado de questionário estimulado sobre fatores chave na definição de requisitos não funcionais

Diante do exposto, os cinco fatores elicitados pelos especialistas foram considerados adequados para a representação estruturada dos perfis de projetos. Adicionalmente, para auxiliar o processo de atribuição de *tags*, o fator "tecnologias utilizadas" foi decomposto em linguagens de programação, *APIs*, *frameworks* e persistência de dados, que consiste no conjunto de tecnologias comumente usado no desenvolvimento de software. Para evitar inconsistências no processo, foi criado um repositório central de *tags*. Portanto, cada fator está relacionado a um conjunto de *tags* predefinido que deve ser atualizado continuamente. A criação de novas *tags* é encorajada, mas deve ser auditada por um especialista responsável.

Na Figura 5.5 é apresentado um exemplo de um perfil estruturado de projeto criado com o auxílio da ferramenta *NFRec*. O projeto é do domínio (do inglês, *Aplication Domain*) "Ferramenta de Desenvolvimento" (do inglês, *Development Tools*), o objetivo do projeto (do inglês *Objective*) é "Produto" (do inglês *Product*), a arquitetura de software (do inglês, *Software Architecture*) é cliente-servidor (do inglês, *client-server*), a plataforma de desenvolvimento (do inglês, *Platform*) é Web e as tecnologias utilizadas são as linguagens de programação (do inglês, *Languages*) *Java* e *JavaScript* e os *frameworks Angular* e *Spring Boot*.

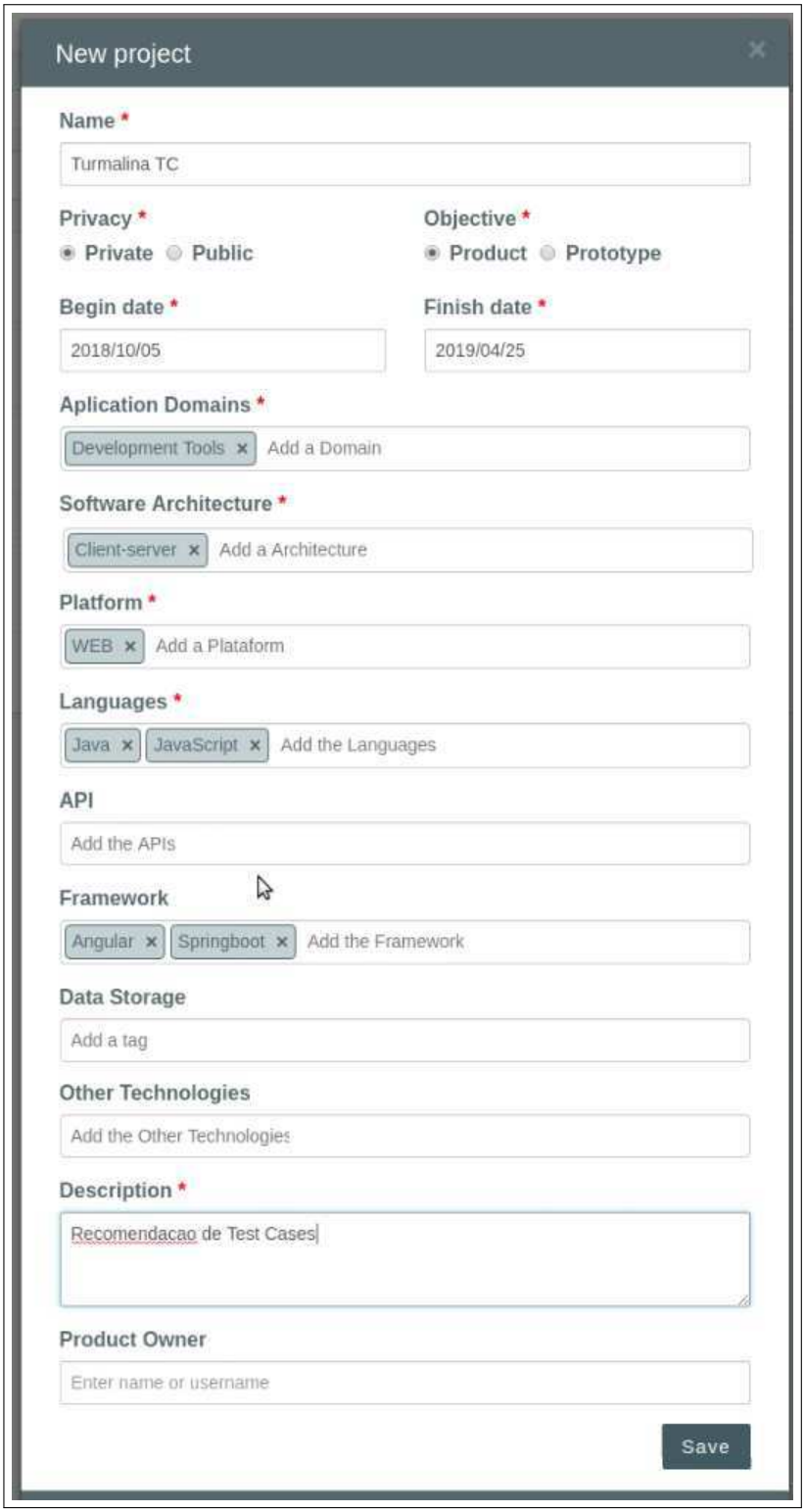


Figura 5.5: Exemplo de criação de perfil de projeto com a ferramenta *NFRec*

5.2.2 Estruturação de *USs* e Tarefas

A estruturação de *USs* é baseada no Modelo Estruturado de *US* (mais detalhes no Capítulo 4), consistindo em classificar *USs* com módulo e operação. Na Figura 5.6 é apresentado um exemplo de criação de uma US com a ferramenta *NFRec* durante a 1ª fase da Reunião de Planejamento Scrum. No exemplo em questão, a *US* foi classificada com o módulo "Cadastro" e a operação "Recuperação de Dados".

Em Scrum, *USs* são decompostas em tarefas técnicas durante a 2ª fase da Reunião de Planejamento. Porém, da mesma maneira como ocorre com *USs*, tarefas também são geralmente descritas em linguagem natural não estruturada, dificultando seu reúso e recuperação. Por isso, propõe-se uma estruturação de tarefas, que é baseada no mapeamento proposto no modelo estruturado apresentado no Capítulo 4 e na atribuição de *tags* de tecnologia, ou seja, *tags* relacionadas a linguagens de programação, *APIs*, *frameworks* e persistência de dados. Na Figura 5.7 é apresentado um exemplo em que uma tarefa é criada na ferramenta *NFRec*, sendo a mesma mapeada como "Associar entidades no banco de dados". Adicionalmente, foram-lhe atribuídas as *tags Java*, *Spring Boot* e *MySQL* para linguagem de programação, *framework* e persistência de dados, respectivamente. Este processo visa atualizar de forma iterativa e incremental o perfil estruturado dos projetos quanto ao fator "tecnologias utilizadas".

5.2.3 Estruturação de RNFs

Assim como *USs* e tarefas, RNFs são comumente representados em linguagem natural, ou seja, não são descritos de forma estruturada. Por isso, propõe-se o Modelo Estruturado de RNFs que permite a rastreabilidade, reúso e recuperação destes elementos.

Porém, como RNFs podem estar relacionados a características diferentes do software em construção, é preciso que o escopo da pesquisa seja melhor definido. Segundo Freeman e Pryce [54], a qualidade de um sistema pode ser avaliada de duas perspectivas diferentes, pela verificação da qualidade interna ou pela análise da qualidade externa. Neste contexto, requisitos de qualidade interna especificam o nível de qualidade requerido na visão interna do produto e avaliam quão bem ele atende às necessidades de seus desenvolvedores e administradores [54]. Alguns exemplos de requisitos não funcionais que influenciam a qualidade

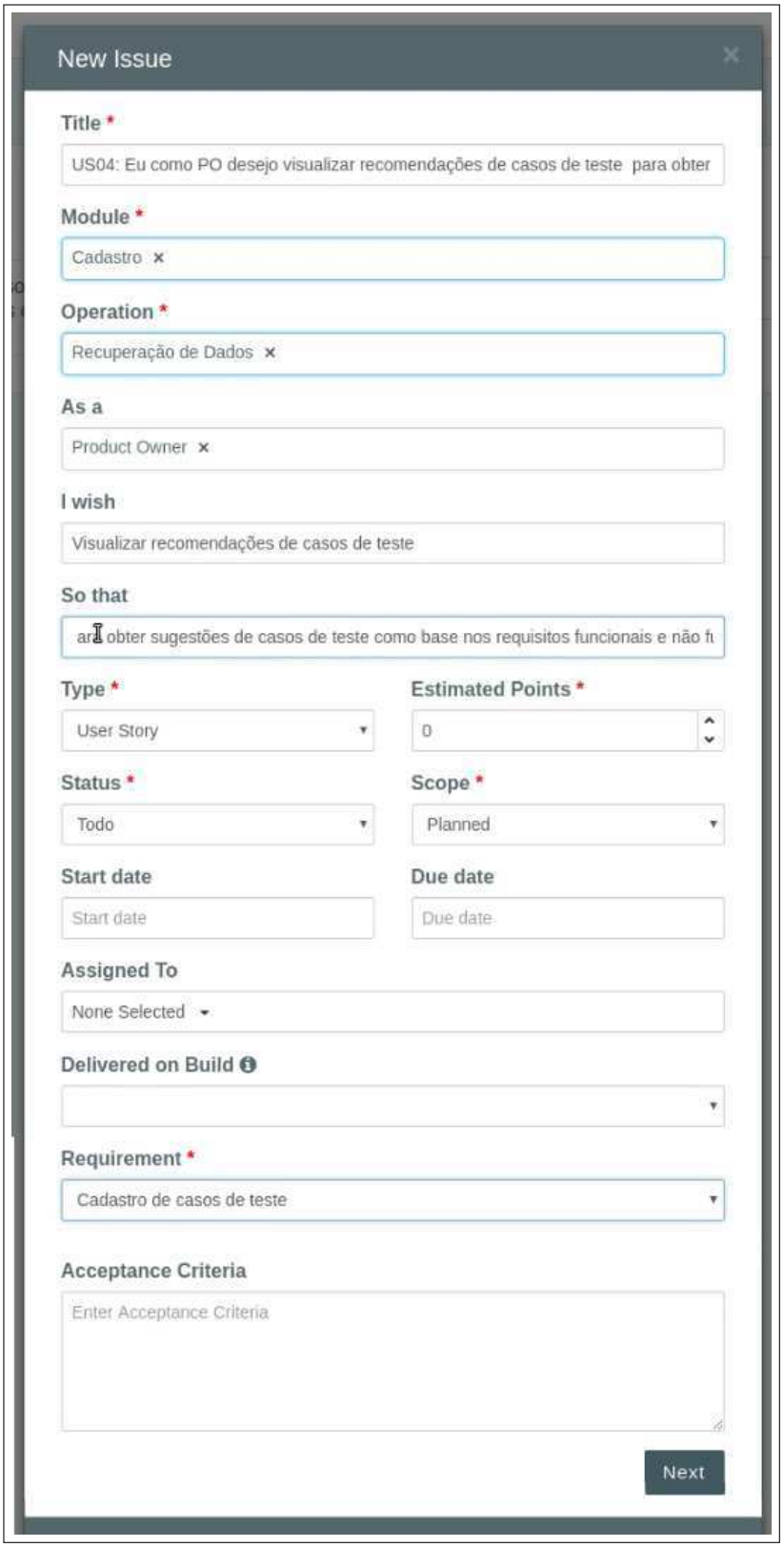


Figura 5.6: Exemplo de criação de *US* com a ferramenta *NFRec*

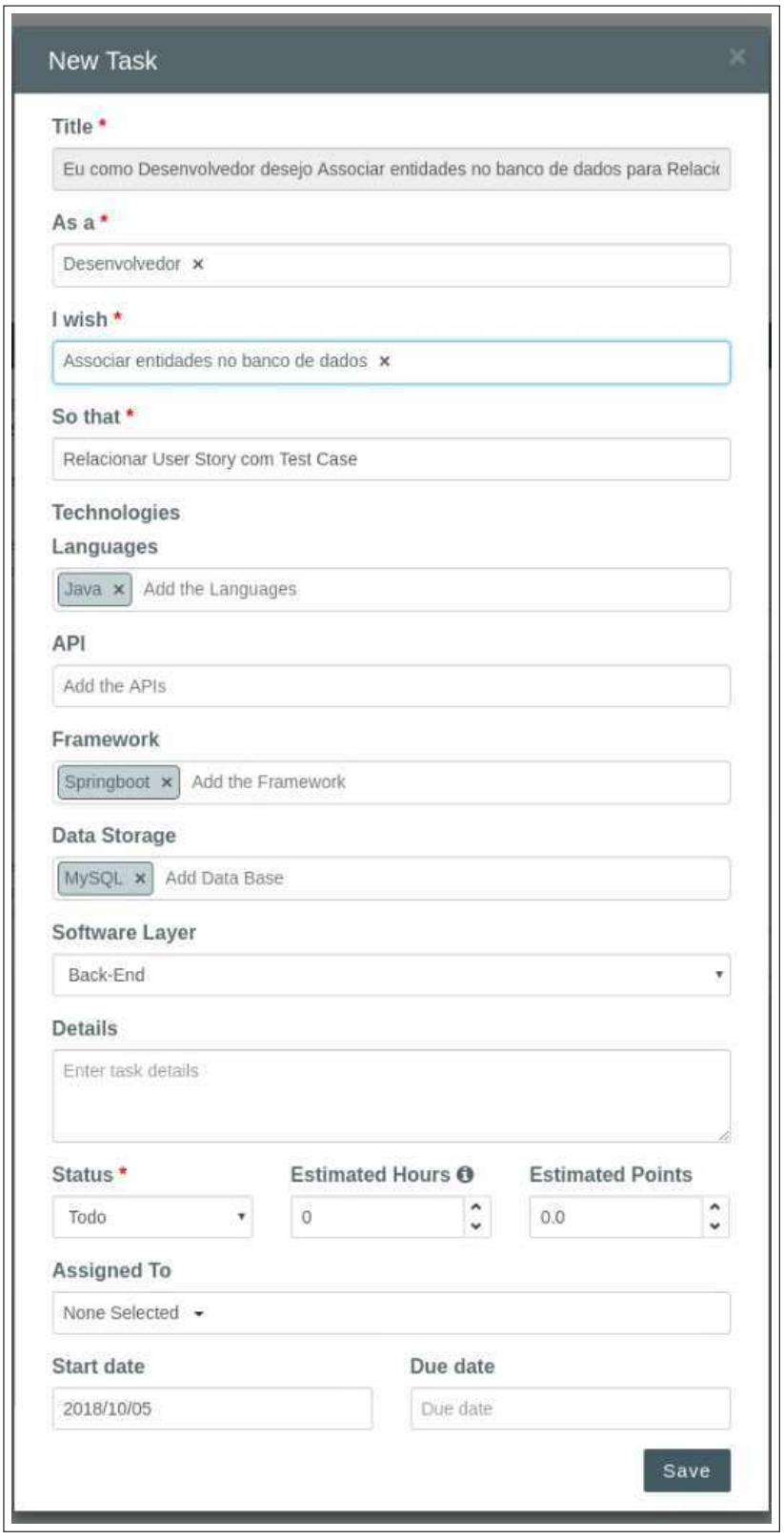


Figura 5.7: Exemplo de criação de tarefa com a ferramenta *NFRec*

interna são: manutenibilidade, portabilidade e testabilidade. Em contrapartida, requisitos de qualidade externa especificam o nível de qualidade do sistema durante sua execução e avaliam quão bem ele atende às necessidades de seus clientes e usuários [54]. Alguns exemplos de requisitos não funcionais que influenciam a qualidade externa são: desempenho, segurança e confiabilidade.

Neste trabalho de tese são considerados apenas requisitos não funcionais relacionados à qualidade externa. Portanto, requisitos como manutenibilidade e testabilidade estão fora do escopo deste trabalho, pois são atributos importantes principalmente para os desenvolvedores [45], gerando menor risco de negligência. Adicionalmente, RNFs podem estar relacionados a um RF específico ou a todo o sistema simultaneamente, que são conhecidos como atributos de qualidade transversais [88]. Neste trabalho, o foco está na associação entre RNFs e RFs. Portanto, atributos de qualidade transversais não fazem parte do escopo deste trabalho. Logo, ao longo deste documento, o termo requisito não funcional refere-se a RNFs que influenciam na qualidade externa associados a RFs específicos. As atividades realizadas na construção do modelo estruturado de RNFs de qualidade externa são apresentadas a seguir.

Tabela 5.1: Exemplo de tipos e atributos do modelo estruturado de RNFs

Tipo	Atributo
Performance	*Response time*
	Capacity
	Transit delay
	Efficiency compliance
Reliability	*Availability*
	Integrity
	Fault tolerance
	Recoverability
Security	*Confidentiality*
	Access control
	Authentication

Análogo ao modelo de *USs*, o modelo estruturado de RNFs é baseado em uma classificação por categorias e subcategorias. Neste caso, os RNFs são categorizados por **tipo** e

atributo. Na Tabela 5.1 pode ser visto um esboço do modelo proposto [2]. Por sua vez, o modelo completo pode ser visualizado no Apêndice D. Para definição das categorias e subcategorias, foram considerados os resultados apresentados por Mairiza et al. em [80], que realizaram uma investigação extensa na literatura acerca do conceito de RNFs e, como resultado, especificaram os cinco tipos de RNFs mais estudados na literatura e seus respectivos atributos, são eles: desempenho, confiabilidade, usabilidade, segurança e manutenibilidade. Como na construção do modelo proposto foram considerados apenas fatores relacionados à qualidade externa, o tipo "manutenibilidade" foi desconsiderado. Adicionalmente, em um segundo momento, o tipo "usabilidade" também foi desconsiderado por limitações na aquisição de informações para validá-lo.

Além da categorização com tipo e atributo, para cada RNF é especificada uma sentença, que descreve uma restrição de qualidade requerida. Para evitar inconsistências e incompletude de representação, as sentenças devem ser escritas de acordo com as indicações apresentadas por Eckhardt et al. em [43] e [44], que propuseram uma abordagem para criação de sentenças de RNFs.

Com o auxílio dos três gerentes de projeto, um conjunto inicial de sentenças do modelo foi definido com base em dados de 13 projetos Scrum reais. Foram extraídas sentenças a partir de critérios de aceitação de *USs*, documentações de casos de testes e relatórios de *bugs*. Durante a análise dos documentos dos projetos não foram identificados RNFs do tipo "usabilidade".

Na Figura 5.8 é ilustrado um exemplo de RNF estruturado que foi criado com a ferramenta *NFRec*. No exemplo em questão, o RNF foi mapeado com o tipo "Confiabilidade" (do inglês, *Reliability*) e o atributo "Tolerância a falha" (do inglês, "Fault tolerance").

5.3 Recomendação de Requisitos Não Funcionais

Com a possibilidade de recuperar as informações do histórico de projetos, o próximo passo foi conceber o sistema de recomendação baseado em memória. Neste trabalho, o problema relacionado à negligência com requisitos não funcionais característico da engenharia de re-

[2] O modelo estruturado de RNFs foi descrito no idioma inglês, pois os trabalhos relacionados que dão suporte à esta etapa do trabalho de tese apresentam soluções com base na gramática inglesa estruturada.

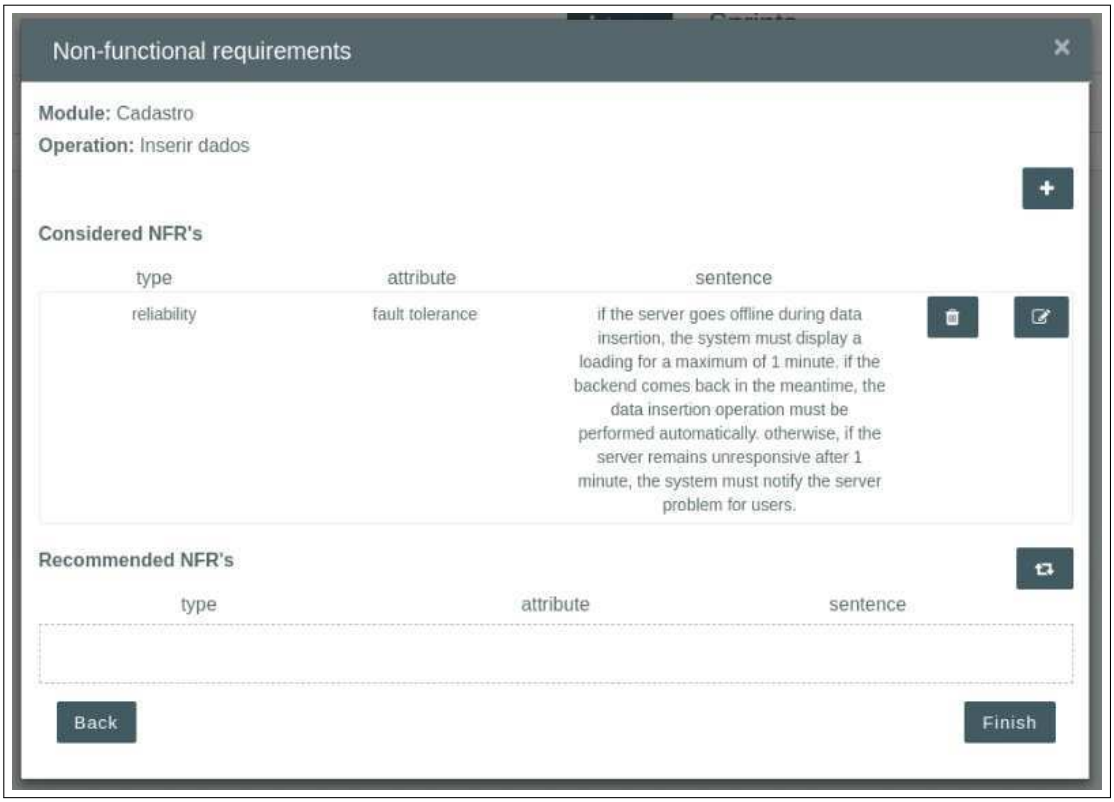


Figura 5.8: Exemplo de criação de RNF com a ferramenta *NFRec*

quisitos ágil é abordado como um problema de recomendação. Assim, adaptando a definição genérica de problemas de recomendação apresentada por Adomavicius e Tuzhilin [2] para o contexto deste trabalho, tem-se: seja F o conjunto de todos os RFs e N o conjunto de todos os RNFs que podem ser recomendados. Seja u a função de utilidade de um RNF n para um RF f, ou seja, u: $F \times N \rightarrow T$, onde T é o conjunto total ordenado. Então, para cada RF $f \in F$, pretende-se recomendar os RNFs $n' \in N$ que maximizem a utilidade de f. Formalmente, o problema pode ser representado pela Equação 5.1 [2].

$$\forall f \in F, \quad n_f^{'} = \arg\max_{n \in N} u(f, n). \tag{5.1}$$

Desta forma, o sistema de recomendação proposto é composto pelos seguintes componentes:

- **Coletor de Dados:** coleta informações dos artefatos estruturados (isto é, perfis de projetos, *USs*, tarefas e RNFs) a fim de criar os perfis de RFs e RNFs, que são os elementos principais do SR proposto;

- **Gerenciador de Perfis**: gera os perfis dos RFs (*USs* estruturadas) e RNFs com base nas informações coletadas pelo Coletor de Dados;

- **Recomendador**: analisa os perfis dos RFs para gerar recomendações personalizadas de RNFs.

Cada um dos componentes do SR proposto é detalhado nas subseções seguintes.

5.3.1 Coletor de Dados

A coleta de dados consiste em extrair informações de fontes de dados para representar os elementos do sistema de recomendação a fim de possibilitar a criação de seus perfis. Neste trabalho, as fontes de dados são os artefatos estruturados com a adaptação do Scrum e os elementos do SR são representados por RFs e RNFs.

No sistema de recomendação proposto, a tarefa de coleta de informações é realizada pelo componente Coletor de Dados. As informações utilizadas para a representação de RFs são extraídas das *USs*, tarefas e perfis dos projetos estruturados. Mais especificamente, para cada RF, são coletadas informações sobre módulo e operação especificados com o modelo estruturado de *USs*, bem como o conjunto de tarefas técnicas associadas à *US*. Complementarmente, são coletadas *tags* referentes aos cinco fatores que representam o perfil estruturado do projeto ao qual o RF pertence, são eles: plataforma, domínio de aplicação, objetivo do projeto, arquitetura de software e *tags* de tecnologias divididas em linguagens de programação, *APIs*, *frameworks* e persistência de dados. Em contrapartida, as informações extraídas de RNFs compreendem os dados provenientes do modelo estruturado de RNFs, ou seja, tipo, atributo e sentença especificados para cada RNF. Por fim, são coletadas informações acerca das associações entre RFs e RNFs, ou seja, a coocorrência das combinações módulo/operação com tipo/atributo/sentença.

Novas informações adicionadas diretamente aos perfis dos projetos ou aos artefatos estruturados durante os eventos de Scrum (ou seja, Reunião de Planejamento e Reunião de Revisão) são extraídas pelo Coletor de Dados e utilizadas para atualizar os perfis de RFs e RNFs.

5.3.2 Gerenciador de Perfis

A partir dos dados coletados pelo Coletor de Dados, os perfis de RFs e RNFs são gerados pelo componente Gerenciador de Perfis. O perfil de um requisito funcional $f \in F$ é formado pelas *tags* referentes aos cinco fatores extraídos do projeto P ao qual pertence e pelas informações provenientes do modelo estruturado de *USs*.

Na Tabela 5.2 podem ser vistos exemplos de perfis de RFs. Por exemplo, o RF f_1 foi mapeado com o módulo "Autenticação" e a operação "Criar conta" e, ademais, as tarefas t_1 e t_2 foram definidas para o seu desenvolvimento. Adicionalmente, é proveniente de um projeto cuja plataforma é "Web", o domínio de aplicação é "Automação Residencial", o objetivo final é "Produto", a arquitetura de software é "*MVC*" e as tecnologias utilizadas foram especificadas com as *tags Java*, *Node*, *Facebook* e *MongoDB* para Linguagens de programação, *frameworks*, *APIs* e persistência de dados, respectivamente.

Tabela 5.2: Exemplos de perfis de requisitos funcionais

ID	**Mód.**	**Ope.**	**Plat.**	**Arq. soft.**	**Dom.**	**Obj.**	**Ling. prog.**	***Fram.***	***APIs***	**Pers. dados**	**Tarefas**
f_1	Auten.	Criar conta	Web	*MVC*	Auto. Resi.	Prot.	*Java*	*Node*	*Facebook*	*MongoDB*	t_1, t_2
f_2	Cadas.	Inserir dados	Web	Clien -serv	Estud.	Prod.	*JavaScript*	*Node*	*Mongoose*	*MongoDB*	t_3, t_4
f_3	Cadas.	Inserir dados	Web	Clien -serv	Estud.	Prod.	*Java*	*Springboot*	*JPA*	*MySQL*	t_3, t_5
f_4	Cadas.	Recup. dados	Web	Clien -serv	Educ.	Prod.	*TypeScript*	*Angular*	*Mongoose*	*MongoDB*	t_6, t_7
f_5	Cadas.	Recup. dados	Web	Cama.	Rec. info.	Prod.	*Python*	*Django*	*Firebase*	*MySQL*	t_6, t_7

Os perfis de RNFs são gerados a partir das informações do modelo estruturado de RNFs em que "tipo" refere-se à classe do RNF, "atributo" a uma propriedade específica do tipo e a "sentença" a uma descrição da propriedade. Na Tabela 5.3 podem ser vistos exemplos de perfis de RNFs. Por exemplo, o RNF n_1 foi mapeado com o tipo "Desempenho" (do inglês, *Performance*), o atributo "tempo de resposta" (do inglês, *response time*) e a sentença especificada descreve uma restrição relacionada ao tempo de resposta na execução de uma funcionalidade.

Tabela 5.3: Exemplos de perfis de requisitos não funcionais

ID	Tipo	Atributo	Sentença
n_1	*Performance*	*response time*	*In 100% of all cases, the system must have a mean response time of <= 3s between event <search items> and event <display text information of listed items>. It is assumed that there are 200 concurrent users.*
n_2	*Performance*	*response time*	*In 100% of all cases, the system must have a mean response time of < 5s between event <view chart> and event <first page view>.*
n_3	*Performance*	*transit delay*	*The system must provide pagination when the number of returned search results exceeds the number of records to be displayed per page.*
n_4	*Reliability*	*integrity*	*The system must validate all data when entered based on the data validation rules. The system must use client-side data validation to improve performance for ordinary execution, i.e., no attacks. Client-side validation is faster since no request is sent to server.*
n_5	*Security*	*authentication*	*The system must verify newly registered users by sending them a verification token via e-mail before allowing them to log in.*

Por fim, são avaliadas as coocorrências entre RFs e RNFs para gerar uma matriz binária de coocorrência, que complementa os perfis dos elementos do SR. Na Tabela 5.4 pode ser visto um exemplo dessa matriz em que o valor atribuído é 1 (um) caso um RF $f_i \in F$ seja associado a um RNF $n_j \in N$ ou 0, caso contrário.

Com os perfis dos requisitos funcionais e requisitos não funcionais conhecidos, é possível gerar recomendações personalizadas de RNFs para RFs.

5.3.3 Recomendador

A geração das recomendações é realizada no componente Recomendador. Segundo Koren e Bell [73], abordagens colaborativas de recomendação (ou seja, filtragem colaborativa) pro-

Tabela 5.4: Exemplo de matriz binária de coocorrência entre requisitos funcionais e requisitos não funcionais.

	n_1	n_2	n_3	n_4	n_5
f_1	0	0	0	0	1
f_2	0	0	0	1	0
f_3	0	0	0	1	0
f_4	0	1	0	0	0
f_5	1	0	1	0	0

duzem recomendações personalizadas para usuários com base em padrões de avaliação ou uso, sem a necessidade de informações externas sobre itens ou usuários, ganhando bastante interesse e progresso a partir de 2009, principalmente, por causa do prêmio Netflix [3], o qual concedeu uma recompensa de 1 milhão de dólares para a equipe ganhadora, que propôs uma abordagem de recomendação deste tipo.

Neste contexto, existem duas abordagens principais de filtragem colaborativa, são elas [73]: baseada em estimativa de vizinhança (por exemplo, *k-NN* [38]) e baseada em fatores latentes (por exemplo, fatoração de matriz [74]). Apesar de abordagens baseadas em fatores latentes comumente apresentarem melhor precisão e escalabilidade [73], as baseadas em estimativa de vizinhança ainda são recorrentes na literatura e em aplicações comerciais [73]. Segundo Koren e Bell [73], este fato explica-se pois abordagens baseadas em estimativa de vizinhança são simples de desenvolver e fornecem explicações intuitivas do raciocínio por trás das recomendações, que em ambientes reais podem melhorar a experiência do usuário além do que a precisão aprimorada pode alcançar.

Diante do exposto anteriormente, neste trabalho é proposta uma abordagem de recomendação baseada em vizinhança. A solução apresentada é baseada no classificador k-ésimo Vizinho mais Próximo (*k-NN*), que apresenta vantagens para a construção de SRs como [5]: é considerado um dos algoritmos mais simples de aprendizado de máquina; não requer o aprendizado e a manutenção de modelos; pode se adaptar a mudanças rápidas na matriz de classificação dos elementos do SR; permite a explicação das recomendações em termos de elementos previamente relacionados; e apesar de sua simplicidade, apresenta bons resultados

[3] Prêmio Netflix: https://www.netflixprize.com/

de precisão e é receptivo a melhorias.

As características gerais do SR proposto são as seguintes:

- **filtragem colaborativa baseada em memória (vizinhança) [38]**: as recomendações são geradas a partir da análise de dados históricos referentes à coocorrência entre RFs e RNFs. Portanto, para calcular a utilidade u de um RNF n para um RF f, avalia-se a relação de n com os k RFs de projetos passados mais similares à f;
- **recomendação de bons itens [57]**: o objetivo final do sistema de recomendação é sugerir um conjunto com os RNFs mais indicados para um determinado RF.

Neste contexto, a geração das recomendações é dividida em duas etapas, são elas: (i) estimativa de vizinhança usando o método *k-NN* e (ii) geração da recomendação de itens.

Estimativa de vizinhança usando o método *k-NN*

O objetivo desta etapa é identificar o conjunto $\hat{F}$ de RFs de uma base histórica mais similares a um RF f_a, em que f_a é um requisito funcional de um projeto corrente ao qual pretende-se gerar as recomendações, denominado de requisito funcional alvo. Para realizar esta tarefa, foi usado o método *k-NN*, que retorna os k vizinhos mais próximos de um elemento de entrada, ou seja, os k elementos mais similares. A fim de alcançar o objetivo definido, a etapa de estimativa de vizinhança foi dividida em três tarefas, são elas: (i) realizar uma pré-filtragem na base histórica para recuperar apenas RFs que apresentam módulo e operação correspondentes aos de f_a e que não são provenientes de projetos cujo objetivo principal é o desenvolvimento de um protótipo; (ii) calcular a similaridade entre f_a e cada um dos requisitos recuperados na pré-filtragem; e (iii) retornar os k RFs mais similares a f_a, ou seja, os k vizinhos mais próximos.

Na Tabela 5.5 é apresentado um exemplo no qual pretende-se recomendar RNFs para um RF alvo f_a que foi mapeado com o módulo "Cadastro" e a operação "Recuperação de dados". Os RFs da base histórica são representados por f_1, f_2, f_3, f_4 e f_5. Esse exemplo é usado como base para as explanações das três tarefas apresentadas a seguir.

A tarefa (i) consiste em uma pré-filtragem na base histórica para recuperar apenas RFs que apresentam módulo e operação correspondentes aos do RF alvo e que não pertencem

Tabela 5.5: Exemplo de recomendação de RNFs para um RF alvo

ID	Mód.	Ope.	Plat.	Arq. soft.	Dom.	Obj.	Ling. prog.	*Fram.*	*APIs*	Pers. dad.	*Tarefas*
f_a	Cadas.	Recup. dad.	Web	Clien-serv	Educ.	Prod.	*TypeScript*	*Angular*	*Mongoose*	*MongoDB*	t_6, t_7
f_1	Auten.	Criar conta	Web	*MVC*	Auto. Resi.	Prot.	*Java*	*Node*	*Facebook*	*MongoDB*	t_1, t_2
f_2	Cadas.	Inserir dados	Web	Clien -serv	Estud.	Prod.	*JavaScript*	*Node*	*Mongoose*	*MongoDB*	t_3, t_4
f_3	Cadas.	Inserir dados	Web	Clien -serv	Estud.	Prod.	*Java*	*Springboot*	*JPA*	*MySQL*	t_3, t_5
f_4	Cadas.	Recup. dados	Web	Clien -serv	Educ.	Prod.	*TypeScript*	*Angular*	*Mongoose*	*MongoDB*	t_6, t_7
f_5	Cadas.	Recup. dados	Web	Cama.	Rec. info.	Prod.	*Python*	*Django*	*Firebase*	*MySQL*	t_6, t_7

a projetos nos quais desenvolveu-se um protótipo, pois esses projetos em geral não consideram RNFs devidamente. Para tanto, realiza-se uma busca na base pela 2-tupla <módulo, operação> do RF alvo, excluindo-se RFs cujo objetivo do projeto foi definido com a *tag* "Protótipo". Na Tabela 5.6 pode ser visto o resultado dessa tarefa aplicada ao exemplo exposto anteriormente na Tabela 5.5, onde foram filtrados os RFs com módulo "Cadastro" e operação "Recuperação de Dados". Como resultado da busca, foram retornados os RFs f_4 e f_5. Portanto, f_1, f_2 e f_3 foram desconsiderados. O requisito f_1 seria descartado mesmo apresentando o mesmo módulo e operação de f_a, pois é proveniente de um projeto que desenvolveu um protótipo. Os RFs retornados são considerados no cálculo de similaridade (tarefa (ii)).

Para realizar a tarefa (ii), é preciso primeiramente representar os perfis dos RFs em um formato que possibilite a aplicação de uma métrica de similaridade. Para tanto, optou-se pela representação com vetores binários *n*-dimensionais, que auxiliam no processamento e análise dos dados. Na Tabela 5.7 podem ser vistos os vetores gerados com base nas características extraídas do perfil do RF alvo f_a, em que cada posição refere-se à uma característica. Os vetores foram preenchidos de acordo com a seguinte condição: atribui-se valor 1 (um) caso a característica correspondente esteja presente no perfil do RF, ou 0 (zero), caso contrário. Portanto, o vetor do RF alvo é sempre preenchido apenas com valores iguais a 1. Em contrapartida, os vetores dos RFs pré-filtrados na tarefa (i) podem apresentar valores iguais

Tabela 5.6: Exemplo de recomendação de RNFs para um RF alvo após a pré-filtragem

ID	Mód.	Ope.	Plat.	Arq. soft.	Dom.	Obj.	Ling. prog.	*Fram.*	*APIs*	Pers. dad.	*Tarefas*
f_a	Cadas.	Recup. dad.	Web	Clien-serv	Educ.	Prod.	*TypeScript*	*Angular*	*Mongoose*	*MongoDB*	t_6, t_7
~~f_1~~	~~Auten.~~	Criar conta	~~Web~~	~~*MVC*~~	Auto. Resi.	Prot.	~~*Java*~~	~~*Node*~~	~~*Facebook*~~	~~*MongoDB*~~	~~t_1, t_2~~
~~f_2~~	~~Cadas.~~	Inserir dados	~~Web~~	Clien -serv	~~Estud.~~	~~Prod.~~	~~*JavaScript*~~	~~*Node*~~	~~*Mongoose*~~	~~*MongoDB*~~	~~t_3, t_4~~
~~f_3~~	~~Cadas.~~	Inserir dados	~~Web~~	Clien -serv	~~Estud.~~	~~Prod.~~	~~*Java*~~	~~*Springboot*~~	~~*JPA*~~	~~*MySQL*~~	~~t_3, t_5~~
f_4	Cadas.	Recup. dados	Web	Clien -serv	Educ.	Prod.	*TypeScript*	*Angular*	*Mongoose*	*MongoDB*	t_6, t_7
f_5	Cadas.	Recup. dados	Web	Cama.	Rec. info.	Prod.	*Python*	*Django*	*Firebase*	*MySQL*	t_6, t_7

a 0 (zero) ou 1 (um) seguindo a condição indicada anteriormente.

No exemplo apresentado na Tabela 5.7, pode-se concluir que o perfil do requisito f_4 é idêntico ao perfil do requisito f_a, pois seu vetor de características foi preenchido apenas com valores iguais a 1. Por outro lado, o requisito funcional f_5 assemelha-se à f_a quanto à plataforma e às tarefas, pois ambos são provenientes de projetos Web e estão associados às tarefas t_6 e t_7. Com os vetores gerados, é possível calcular a similaridade entre eles, ou seja, executar a tarefa (ii).

Tabela 5.7: Exemplos de vetores de características gerados com base nas características extraídas de um RF alvo f_a

ID	Web	Clien-serv	Educ.	TypeScript	*Angular*	*Mongoose*	*MongoDB*	t_6	t_7
f_a	1	1	1	1	1	1	1	1	1
f_4	1	1	1	1	1	1	1	1	1
f_5	1	0	0	0	0	0	0	1	1

A similaridade entre vetores pode ser calculada com base em diferentes métricas de similaridade e correlação como similaridade do cosseno [2], distância euclidiana [1], distância de *Manhattan* [1], correlação de *Pearson* [38], correlação de *Spearman* [38], etc. Na solução proposta, a similaridade entre requisitos funcionais é calculada com base na distância de *Manhattan* (Equação 5.2), que é uma medida de distância multidimensional e obteve melhores resultados em experimentos realizados (mais detalhes no Capítulo 6).

$$d(f_a, f') = \sum_{i=1}^{n} |f_{ai} - {f'}_i| \,, \tag{5.2}$$

De forma geral, o cálculo da distância de *Manhattan* é dado pela soma das diferenças entre os valores de dois vetores de entrada, ambos com mesma dimensão *d*. Portanto, dado que os vetores de características considerados na solução proposta só podem assumir valores iguais a 0 (zero) ou 1 (um), consequentemente, a maior distância possível entre dois vetores é igual a *d*. Em contrapartida, a menor distância possível é 0 (zero), independente da dimensão dos vetores. Por exemplo, considerando os vetores da Tabela 5.7, a distância entre f_a e f_5 é 6, pois os dois requisitos funcionais só apresentam três características em comum de nove consideradas na construção dos vetores. Por outro lado, a distância entre f_a e f_4 é 0, pois possuem vetores de características idênticos. Adicionalmente, pode-se normalizar os valores obtidos com a distância de *Manhattan* em um intervalo fechado de $[0, 1]$ pelo cálculo da similaridade de *Manhattan* (Equação 5.3), em que 1 é a maior similaridade possível entre dois vetores. Nesse caso, a similaridade entre f_a e f_5 é 0,33 e entre f_a e f_4 é 1.

$$sim(f_a, f') = 1 - \frac{\sum_{i=1}^{n} |f_{ai} - {f'}_i|}{n}. \tag{5.3}$$

Por fim, para realizar a tarefa (iii), é preciso apenas ordenar de forma decrescente o conjunto de requisitos pré-filtrados na tarefa (i) pela similaridade calculada na tarefa (ii) e selecionar apenas os *k* primeiros RFs desse conjunto. Para definir o valor de *k* ideal, foram executados experimentos (mais detalhes no Capítulo 6), resultando na escolha de *k* igual a 1. Portanto, considera-se apenas um vizinho para a geração das recomendações de RNFs. No exemplo estudado, o vizinho mais próximo de f_a é f_4, com similaridade 1.

Recomendação de Itens

Esta etapa compreende a geração da recomendação de requisitos não funcionais para o RF alvo com base no conjunto $\hat{F}$, que denota os *k* vizinhos mais próximos identificados na etapa anterior. Formalmente, pretende-se recomendar os RNFs $n' \in N$ que maximizem a utilidade do RF alvo f_a. Desta forma, a utilidade $u_{f_a,n'}$ de um RNF n' para o RF alvo f_a é computada pela Equação 5.4. Nessa equação, calcula-se o somatório da quantidade de coocorrências entre os k vizinhos de f_a e o RNF n' ponderada pela similaridade de f_a e seus vizinhos, em

que $u_{f',n'}$ retorna o valor 1, caso o vizinho f' apresente o RNF n' em sua lista de RNFs, ou 0, caso contrário.

$$u_{f_a,n'} = \sum_{f' \in \hat{F}} sim(f_a, f') \times u_{f',n'}. \tag{5.4}$$

No exemplo estudado, foi considerado apenas um vizinho de f_a, ou seja, o RF f_4. Portanto, para gerar as recomendações para f_a, basta apenas recuperar os RNFs associados a f_4 (ver Tabela 5.4). Logo, foi recomendado para f_a o RNF n_2 (ver Tabela 5.3), pois está associado a f_4 (ver Tabela 5.4).

Na Figura 5.9 pode ser visto um exemplo de recomendação de RNFs com a ferramenta *NFRec*.

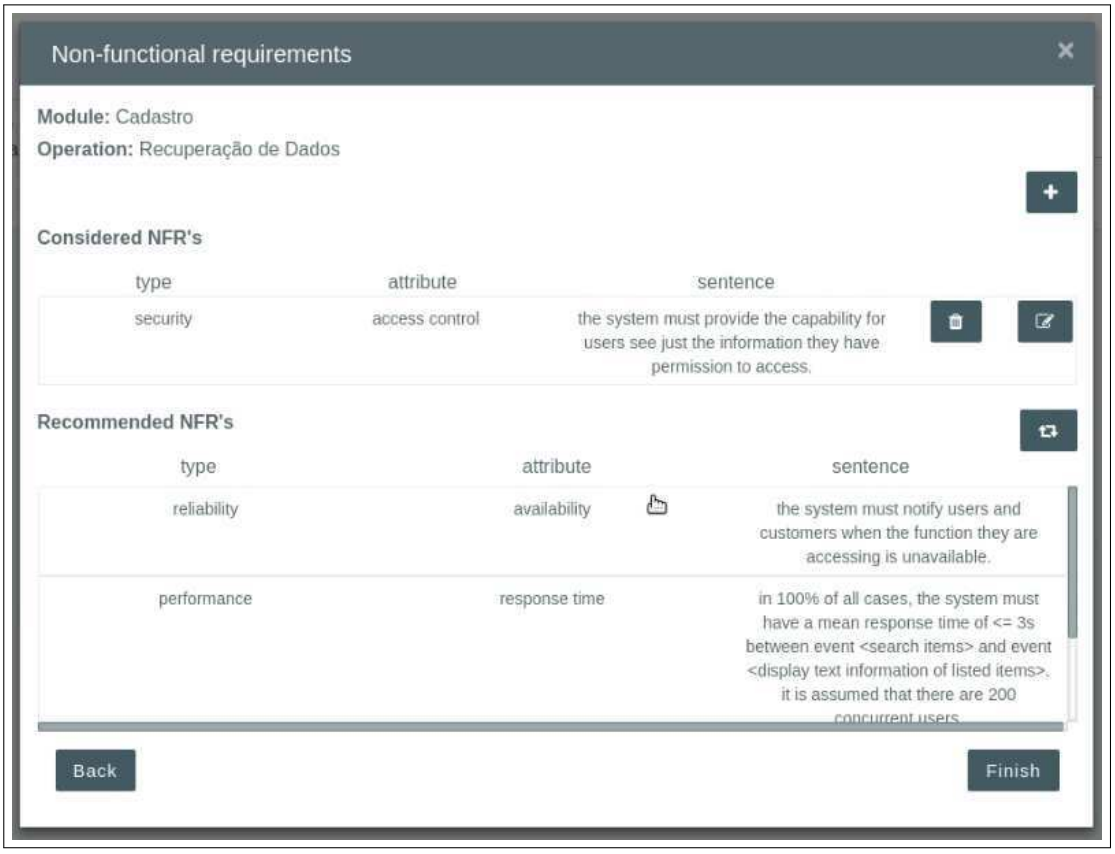


Figura 5.9: Exemplo de recomendação de RNFs com a ferramenta *NFRec*

5.4 Considerações Finais do Capítulo

Neste capítulo, foi descrito o sistema de recomendação de RNFs que visa dar suporte a integrantes de projetos de software baseados em Scrum na elicitação de RNFs durante o processo de análise de requisitos. Para viabilizar a operacionalização do SR, foi proposta uma adaptação do Scrum, que é baseada na estruturação de informações de projetos, RFs, tarefas e RNFs com *tags* e categorias.

O sistema de recomendação proposto é composto por três componentes, são eles: Coletor de Dados, que coleta informações estruturadas de RFs, tarefas e RNFs; Gerenciador de Perfil, que gera perfis de RFs e RNFs; e Recomendador, que processa as recomendações. O SR é baseado em filtragem colaborativa baseada em vizinhança e visa gerar recomendações de itens. A etapa de estimativa de vizinhança é realizada com o auxílio do classificador *k-NN*, configurado com a medida de similaridade de *Manhattan* e k igual a 1. Para recomendar RNFs para um RF alvo, são verificadas as informações dos seus vizinhos quanto aos RNFs associados.

Por fim, como parte do trabalho, a solução proposta foi implementada em uma ferramenta Web de gerenciamento de projetos, chamada de *NFRec*.

Capítulo 6

Validação da Solução

Neste capítulo, são apresentadas as tarefas realizadas na validação da solução proposta, mais especificamente, são apresentados detalhes da configuração experimental *off-line* realizada, dos resultados alcançados e das conclusões obtidas por meio de análise estatística.

6.1 *Design* do Experimento

O experimento apresentado nesta seção visa identificar qual configuração do SR apresenta melhor eficácia na geração das recomendações de requisitos não funcionais. Porém, antes de executá-lo, é preciso analisar as características do sistema de recomendação para identificar a forma mais adequada de avaliá-lo.

As atividades de validação descritas nesta seção baseiam-se principalmente nas indicações apresentadas por Gunawardana e Shani em [57], que discorrem sobre os principais tipos de tarefas de recomendação e classificam uma série de métricas de avaliação apropriadas para cada tarefa. Segundo os autores, sistemas de recomendação podem ser divididos em três classes de acordo com a tarefa de recomendação que realizam, são elas: 1) recomendação de bons itens, que consiste em recomendar os melhores itens para usuários; 2) otimização de utilidade, que pretende maximizar uma variável, por exemplo, maximizar os lucros de *websites*; e 3) predição de avaliações, que consiste em predizer as avaliações de usuários para um determinado conjunto de itens. Esta classificação visa agrupar também os SRs quanto à forma de avaliação, pois para cada classe existem formas de avaliação que se adéquam mais as suas características. Neste trabalho, o sistema de recomendação proposto é colaborativo,

baseado em vizinhança e tem como objetivo principal recomendar os "melhores" requisitos não funcionais para cada um dos requisitos funcionais alvos. Portanto, pode ser classificado como um SR para a recomendação de bons itens, isto é, classe 1).

Neste contexto, dois protocolos de avaliação podem ser usados, Avaliação *On-line* e Configuração Experimental *Off-line* [57], em que o primeiro analisa diferentes configurações do sistema de recomendação aplicado em um ambiente real enquanto que o segundo avalia o SR com base em dados históricos que representam possíveis cenários reais. Embora seja indicado avaliar o SR com dados reais em uma avaliação *on-line*, esta tarefa pode ser custosa e, ademais, o SR deve ser pré-avaliado antes de ser apresentado para usuários potenciais. Por isso, a configuração *off-line* se mostra importante, pois pode evitar uma experiência de uso negativa para os usuários de teste. Nesta seção, são apresentados os resultados de uma configuração experimental *off-line*, com base em dados de 13 projetos reais.

Após a seleção do protocolo de validação, outro passo importante é a seleção das métricas de avaliação, que também devem ser escolhidas de acordo com a classe do SR. Segundo Gunawardana e Shani [57], as métricas mais indicadas para a avaliação de um sistema de recomendação de bons itens em uma configuração experimental *off-line* estão relacionadas com precisão (Equação 6.1) e *recall* (Equação 6.2). Porém, tipicamente é observada uma relação inversa (do inglês, *trade-off*) entre essas duas métricas, pois listas de recomendação mais longas normalmente melhoram o *recall*, mas reduzem a precisão. Por isso, existem medidas que sumarizam as duas métricas anteriores como a *F-measure* (Equação 6.3), que calcula a média harmônica da precisão e do *recall* [111]. Tendo em vista os motivos expostos anteriormente, a métrica *F-measure* foi considerada como medida de eficácia do SR na configuração experimental *off-line* realizada.

$$precisão = \frac{\mid \{recomendações\ relevantes\} \cap \{recomendações\ retornadas\} \mid}{\mid \{recomendações\ retornadas\} \mid}, \tag{6.1}$$

$$recall = \frac{\mid \{recomendações\ relevantes\} \cap \{recomendações\ retornadas\} \mid}{\mid \{recomendações\ relevantes\} \mid}, \tag{6.2}$$

$$F\text{-}measure = 2 \cdot \frac{precisão \cdot recall}{precisão + recall}. \tag{6.3}$$

Portanto, em síntese, a validação da solução proposta baseia-se na seguinte configuração:

- **protocolo**: configuração experimental *off-line*;
- **métrica avaliada**: *F-measure* calculada a partir da precisão e do *recall* das recomendações geradas.

As seções a seguir apresentam detalhes sobre o experimento realizado de acordo com as diretrizes de documentação de experimentos apresentadas por Jedlitschka e Pfahl em [67].

6.1.1 Objetivos e Hipóteses

Como o sistema de recomendação proposto é baseado no algoritmo de classificação *k-NN*, o objetivo principal deste experimento é examinar qual configuração do algoritmo – em termos de número de vizinhos e medida de similaridade utilizados para realizar o cálculo de vizinhança – apresenta melhor eficácia na recomendação de requisitos não funcionais. De acordo com o método *GQM* (do inglês, *Goal/Question/Metric*), o objetivo do experimento pode ser descrito da seguinte forma:

Analisar diferentes configurações do SR de RNFs
para o propósito de sua avaliação
com respeito à sua eficácia com base na métrica *F-measure*
do ponto de vista de um conjunto de dados históricos de projetos reais baseados em *Scrum*
no contexto de Engenharia de Requisitos Ágil.

Desta forma, pretende-se responder a questão de pesquisa $\mathbf{RQ}_2$ (ver Seção 1.3). Para tanto, foi formulada a seguinte questão de pesquisa secundária:

$\mathbf{RQ}_{2.1}$: Qual configuração apresenta melhor resultado de *F-measure* na recomendação de RNFs do ponto de vista do conjunto de dados histórico apresentado na Subseção 6.1.3?

A fim de responder a questão de pesquisa secundária, as seguintes hipóteses foram formuladas:

$H_{0-2.1}$**:** não há diferença entre as diferentes configurações do SR de RNFs em relação à métrica *F-measure* das recomendações.

$H_{1-2.1}$**:** há diferença entre as diferentes configurações do SR de RNFs em relação à métrica *F-measure* das recomendações.

6.1.2 Variáveis

O *design* de experimento aplicado considera dois 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 RNFs, com 11 níveis, ou seja, *k* variando de 1 a 11;
- **medida de similaridade**: medida de distância/similaridade usada para o cálculo de vizinhança com cinco níveis, são eles: similaridade do *Cosseno* [2], distância *Euclidiana* [1], distância de *Manhattan* [1], distância de *Jaccard* [5] e distância de *Canberra* [69].

A variável resposta considerada no experimento é a seguinte:

- ***F-measure***: média harmônica da precisão e do *recall* das recomendações de RNFs.

6.1.3 Geração da Base de Dados

Como informado anteriormente, uma configuração experimental *off-line* requer um conjunto de dados que represente cenários reais do domínio do problema [57]. Por isso, uma base de dados foi gerada a partir de informações de projetos de software reais. O conjunto de dados gerado contém informações de 13 projetos de software baseados em Scrum. Mais especificamente, foram coletadas informações gerais dos perfis dos projetos, suas *user stories* com tarefas e requisitos não funcionais associados.

Os perfis dos projetos foram construídos a partir da análise dos seus documentos de abertura e planejamento (do inglês *project charter* e *project plan*, respectivamente), de onde foram extraídas informações relacionadas aos fatores que incidem na elicitação de requisitos não funcionais (mais detalhes na Seção 5.2.1), ou seja, plataforma do projeto, domínio do projeto, objetivo do projeto, arquitetura de software e tecnologias. Em contrapartida, as informações de *USs* e tarefas foram coletadas a partir dos documentos de *backlog* de produto e *backlog* de *Sprint*, respectivamente. Para garantir a rastreabilidade e recuperação das informações pelo SR, todas as *USs* e tarefas foram estruturadas com o modelo de *US* apresentado

no Capítulo 4. Por fim, os RNFs foram obtidos e associados às *USs* por meio da análise de critérios de aceitação de *USs*, documentações de casos de testes e relatórios de *bugs*. Analogamente ao que foi realizado com *USs* e tarefas, os RNFs foram estruturados com base no modelo estruturado de RNFs (ver Subseção 5.2.3). Ao final do processo, o conjunto de dados foi validado pelos gerentes dos respectivos projetos.

Na Tabela 6.1 pode ser visto um sumário dos dados coletados dos 13 projetos. Por exemplo, o projeto $P5$ é do domínio "Educação" e foram extraídas informações de 60 requisitos funcionais (*USs*), 81 RNFs e 216 tarefas. Além disso, é possível observar que o projeto apresentou uma média de $1,4$ requisitos não funcionais por *US*.

6.1.4 Execução

Segundo Gunawardana e Shani [57], para executar experimentos *off-line* em SRs de bons itens, é preciso simular o processo *on-line* de geração das recomendações e as ações dos usuários que ocorrem a partir das sugestões geradas. Normalmente, este processo é realizado por meio da coleta de dados históricos, em que parcela dos dados é ocultada e o SR tenta prever as interações reais entre usuários e itens. Posteriormente, a lista de itens recomendada é comparada com os dados reais para fins de cálculo de métricas de eficácia. No contexto deste trabalho, avaliam-se as interações entre RFs e RNFs em projetos de software. Portanto, ao executar o experimento, parcela do conjunto de dados históricos apresentado na subseção anterior é ocultada e o SR tenta prever quais RNFs apresentam interações com os RFs, ou seja, RNFs são recomendados para os RFs da parcela ocultada. Então, a métrica *F-measure* é calculada comparando-se as recomendações de RNFs geradas e as interações reais observadas.

Para evitar viés na escolha de qual parcela deve ser ocultada, diferentes abordagens de divisão dos dados podem ser usadas [41]. Por exemplo, *Akaike's Information Criterion* [21], *Bayesian Information Criterion* [56] e Validação Cruzada [9]. Segundo Gunawardana e Shani [57], validação cruzada é uma das abordagens mais conhecidas para validar sistemas de recomendações, que consiste em dividir o conjunto de dados original em dois conjuntos disjuntos: treinamento e teste. O conjunto de treinamento representa uma parcela maior dos dados e é usado para prover conhecimento ao SR. Em contrapartida, o conjunto de teste é usado para verificar as métricas que avaliam a eficácia do SR [57].

Tabela 6.1: Sumário das informações coletadas dos 13 projetos de software

ID Proj.	**Domínio**	**Nº RFs**	**Nº RNFs**	**Nº Tarefas**	**RNFs por RFs**
P1	Entretenimento	9	22	31	2,4
P2	Utilitários	6	10	15	1,7
P3	Utilitários	16	22	36	1,4
P4	Educação	30	61	100	2,0
P5	Educação	60	81	216	1,4
P6	Gestão e Negócios	44	93	126	2,1
P7	Gestão e Negócios	39	90	136	2,3
P8	Recursos de Informação	49	80	131	1,6
P9	Recursos de Informação	13	27	53	2,1
P10	Recursos de Informação	27	29	64	1,1
P11	Comunicações	36	85	91	2,4
P12	Recursos de Informação	23	37	53	1,6
P13	Escritório	31	56	75	1,8
Total	-	383	693	1127	1,8

Segundo Celma [25], o protocolo de validação cruzada mais comum para avaliar a eficácia de SRs ao preverem interações reais que ocorreram anteriormente é o *leave-p-out* [101], que consiste em dividir o conjunto original em n parcelas, e então, utilizar iterativamente p parcelas de cada vez como conjunto de teste ao passo que $n - p$ formam o conjunto de treinamento, onde $p > 0$ e $n > p$. Neste trabalho, usou-se o protocolo *leave-p-out* com p igual a 2, ou seja, para cada execução do SR, os dados de dois projetos foram utilizados como conjunto de teste enquanto que os dados dos 11 projetos restantes formaram o conjunto de treinamento. O procedimento foi repetido de modo que a amostra foi dividida de todas as formas possíveis com dois projetos como conjunto de teste e 11 como conjunto de treinamento.

Desta forma, para cada combinação de variáveis independentes, o cálculo do número de execuções do SR é dado por C_p^n, onde *n* é o número de projetos do conjunto de dados histórico. Portanto, com $n = 13$ e $p = 2$, tem-se um total de 78 execuções por configuração (ou seja, C_2^{13}). Adicionalmente, dado que foram avaliadas 55 configurações diferentes do SR (11 níveis de *k* e cinco níveis de medida de similaridade), tem-se um total de 4290 execuções ao final do experimento. Por fim, como são geradas recomendações de RNFs para RFs de dois projetos diferentes por execução, consequentemente, são calculadas métricas de eficácia para ambos os projetos, decorrendo em um total de 8580 resultados de *F-measure*, ou seja, 156 por configuração do SR.

Na seção seguinte, os resultados de precisão, *recall* e *F-measure* são apresentados de forma sumarizada, ou seja, são mostradas as médias dos 156 valores obtidos para cada configuração com a validação cruzada *leave-2-out*.

6.2 Resultados

Nesta seção, são apresentados os resultados de precisão, *recall* e *F-measure* obtidos a partir da execução do SR usando o protocolo de validação cruzada *leave-2-out*.

Na Tabela 6.2, é possível observar as médias de precisão para diferentes configurações do SR. Por exemplo, a média obtida com k igual a 1 e distância de *Manhattan* é de 78%. Em contrapartida, na Tabela 6.3, são apresentadas as médias de *recall* para diferentes combinações de variáveis independentes. Por exemplo, a média obtida com k igual a 1 e distância

de *Manhattan* é de 82%. Por fim, na Tabela 6.4, pode-se observar as médias de *F-measure* obtidas por meio do cálculo da média harmônica das duas métricas anteriores. Por exemplo, o resultado médio de *F-measure* para a execução com k igual a 1 e distância de *Manhattan* é de 79%.

Tabela 6.2: Resultados de precisão obtidos a partir da execução dos experimentos com as diferentes configurações do SR

	JACCARD	**MANHATTAN**	**CANBERRA**	**COSSENO**	**EUCLIDIANA**
1	75,9%	78,5%	78,5%	78,5%	78,5%
2	74,2%	73,8%	73,8%	73,8%	73,8%
3	72,3%	71,1%	71,1%	71,1%	71,1%
4	71,8%	67,9%	67,9%	67,9%	67,9%
5	70,4%	66,0%	66,0%	66,0%	66,0%
6	68,6%	64,5%	64,5%	64,5%	64,5%
7	67,2%	62,6%	62,6%	62,6%	62,6%
8	66,3%	61,5%	61,5%	61,5%	61,5%
9	64,6%	60,8%	60,8%	60,8%	60,8%
10	63,4%	60,3%	60,3%	60,3%	60,3%
11	62,2%	59,7%	59,7%	59,7%	59,7%

Adicionalmente, os resultados de precisão, *recall* e *F-measure* podem ser visualizados nas Figuras 6.1, 6.2 e 6.3, respectivamente. Pela análise dos gráficos, observa-se um aumento nos valores de *recall* e uma redução nos valores de precisão à medida que a quantidade de vizinhos *k* aumenta, independentemente da medida de similaridade utilizada. Além disso, é possível observar que não houve variação nos resultados das três métricas de eficácia para as medidas de similaridade/distância de *Manhattan*, *Canberra*, *Euclidiana* e *Cosseno*. É possível que esta situação ocorra pelo fato destas medidas apresentarem comportamentos semelhantes com vetores binários [5, 27]. Assim, apesar de retornarem valores de similaridade distintos em seus cálculos, a ordem dos vizinhos é mantida. Desta forma, não há variação nas listas de RNFs recomendadas e nem nas métricas de eficácia calculadas. Por outro lado, com a medida de *Jaccard*, que tem base nas diferenças entre os vetores de características (calcula as dissimilaridades) [27], os resultados mostraram-se levemente diferentes dos obtidos com

Tabela 6.3: Resultados de *recall* obtidos a partir da execução dos experimentos com as diferentes configurações do SR

	JACCARD	**MANHATTAN**	**CANBERRA**	**COSSENO**	**EUCLIDIANA**
1	75,0%	81,6%	81,6%	81,6%	81,6%
2	81,6%	84,5%	84,5%	84,5%	84,5%
3	83,9%	87,0%	87,0%	87,0%	87,0%
4	87,0%	88,4%	88,4%	88,4%	88,4%
5	89,1%	88,7%	88,7%	88,7%	88,7%
6	90,5%	89,3%	89,3%	89,3%	89,3%
7	90,7%	90,6%	90,6%	90,6%	90,6%
8	91,0%	90,8%	90,8%	90,8%	90,8%
9	91,2%	91,1%	91,1%	91,1%	91,1%
10	91,9%	91,2%	91,2%	91,2%	91,2%
11	91,9%	91,3%	91,3%	91,3%	91,3%

Tabela 6.4: Resultados de *F-measure* obtidos a partir da execução dos experimentos com as diferentes configurações do SR

	JACCARD	**MANHATTAN**	**CANBERRA**	**COSSENO**	**EUCLIDIANA**
1	74,9%	79,0%	79,0%	79,0%	79,0%
2	77,1%	77,7%	77,7%	77,7%	77,7%
3	76,8%	76,9%	76,9%	76,9%	76,9%
4	77,8%	75,8%	75,8%	75,8%	75,8%
5	77,4%	74,6%	74,6%	74,6%	74,6%
6	76,8%	73,9%	73,9%	73,9%	73,9%
7	76,0%	73,1%	73,1%	73,1%	73,1%
8	75,6%	72,5%	72,5%	72,5%	72,5%
9	74,5%	72,2%	72,2%	72,2%	72,2%
10	74,0%	71,8%	71,8%	71,8%	71,8%
11	73,1%	71,4%	71,4%	71,4%	71,4%

as demais medidas.

Adicionalmente, ao observar a Figura 6.3, é possível identificar que houve uma queda praticamente constante nos resultados de *F-measure* para configurações com distância de *Jaccard* e valores de *k* maiores que 4. Por outro lado, com as demais medidas de similaridade, os resultados de *F-measure* decaíram com *k* maiores que 1. Por fim, observa-se que com 79%, as configurações com *k* igual a 1 e similaridades de *Manhattan*, *Canberra*, *Euclidiana* ou *Cosseno* alcançaram a maior média de *F-measure* na recomendação de RNFs. Porém, para identificar qual tratamento obteve melhor resultado de fato, mostrou-se necessária a realização de uma comparação pareada (análise estatística) entre os tratamentos.

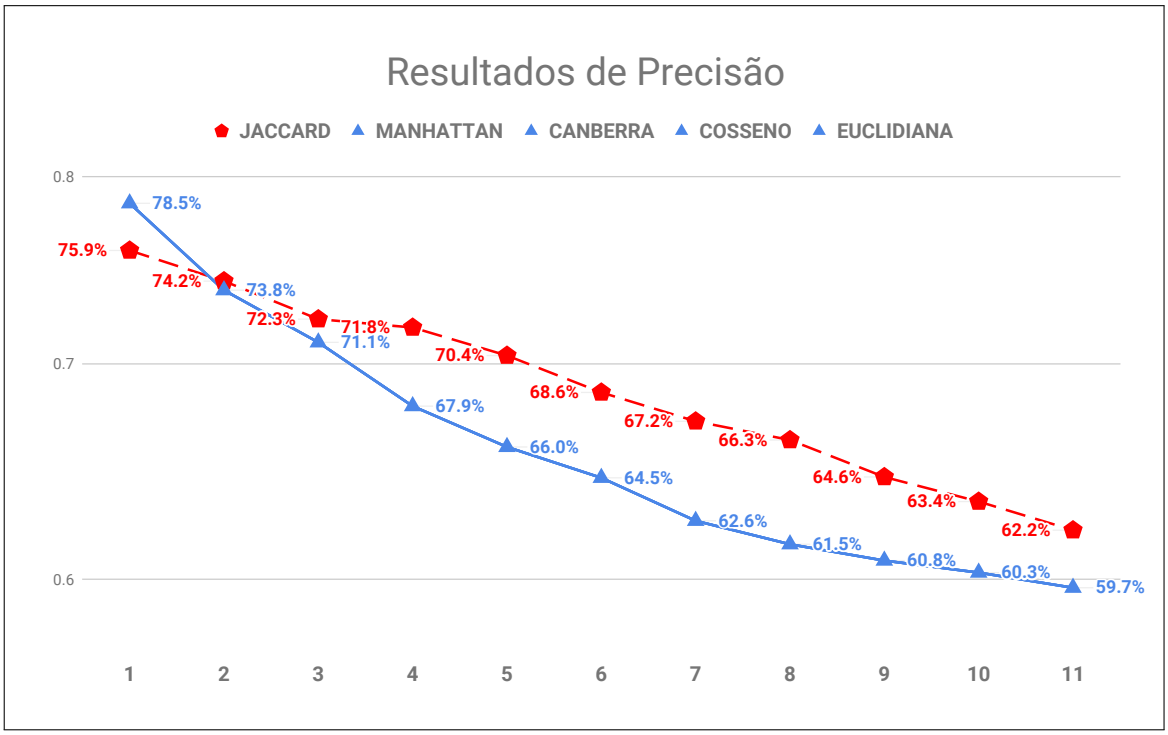


Figura 6.1: Resultados para a métrica precisão

6.3 Análise Estatística

Para responder a questão de pesquisa declarada na Subseção 6.1.1, é necessário comparar estatisticamente os múltiplos resultados de *F-measure* obtidos com a validação cruzada. Para tanto, pretende-se aceitar ou refutar a hipótese nula $H_{0-2.1}$ (ver Subseção 6.1.1). No contexto de Análise Estatística, alguns testes são apropriados para comparações múltiplas, por

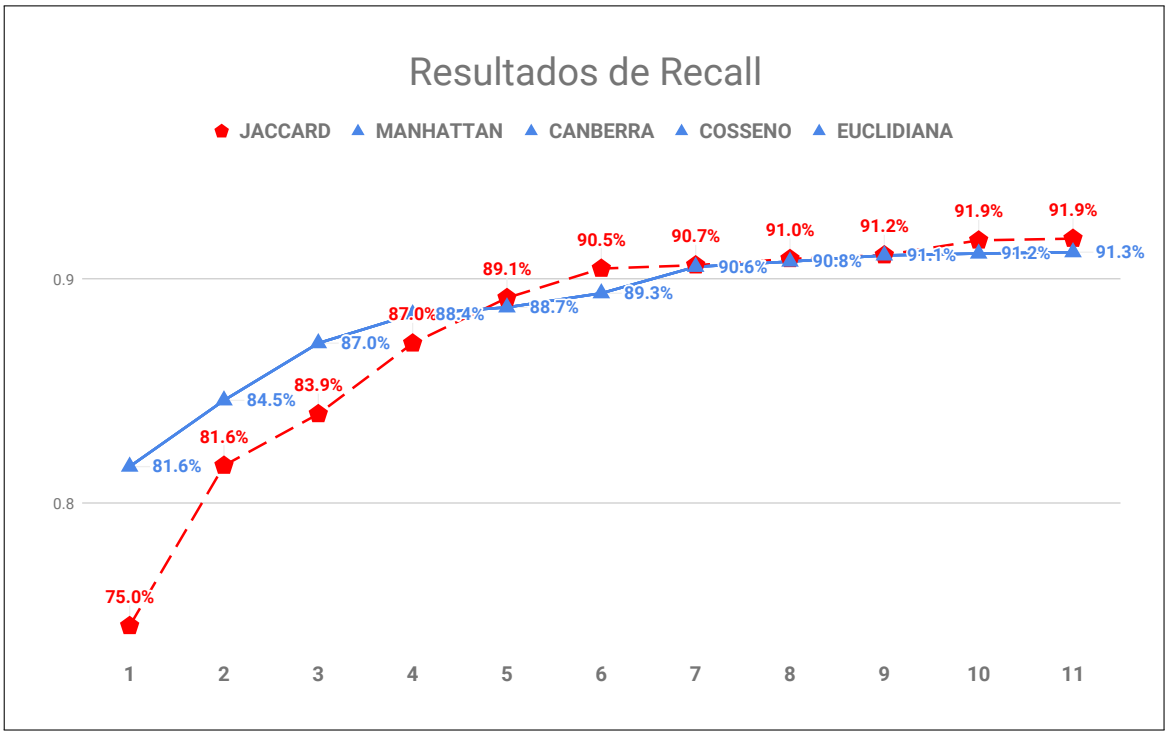


Figura 6.2: Resultados para a métrica *recall*

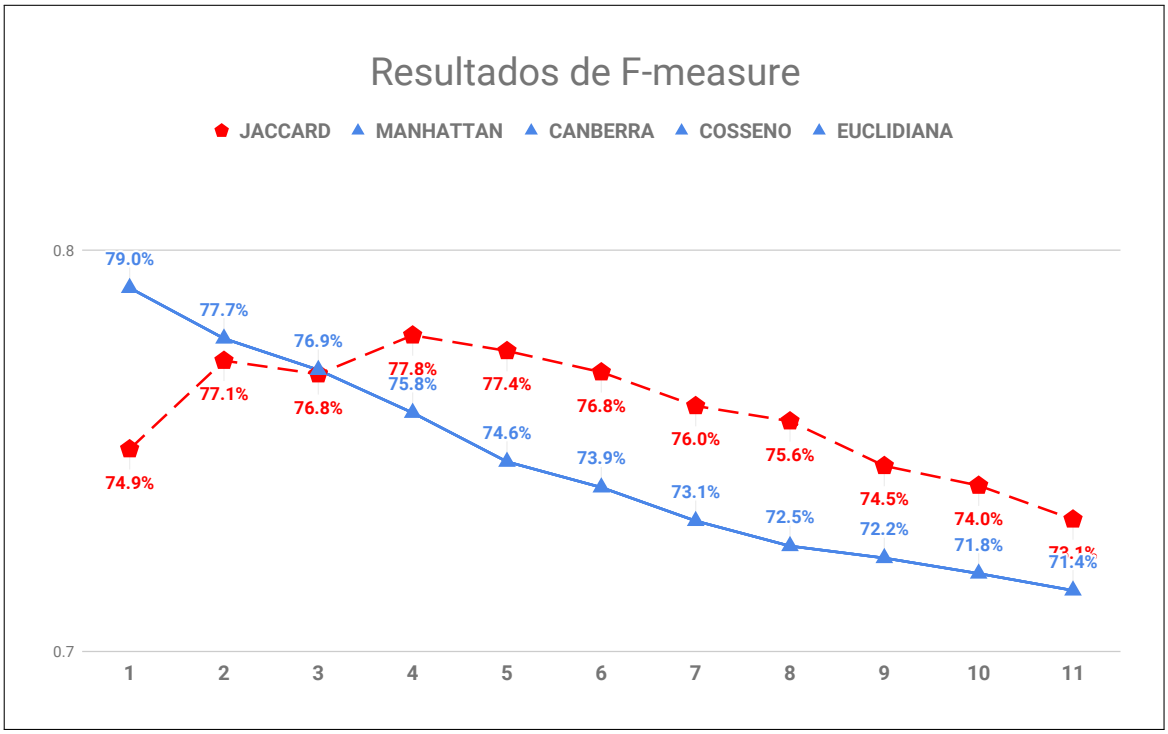


Figura 6.3: Resultados para a métrica *F-measure*

exemplo, ANOVA [37] e *Friedman* [37].

Por tratar-se de um teste paramétrico, o teste de ANOVA requer que alguns requisitos sejam atendidos, como: normalidade e homoscedasticidade dos dados. Em contrapartida, quando esses requisitos são violados, pode-se usar testes não paramétricos como o de *Friedman*. Portanto, primeiramente, avaliaram-se os perfis dos dados com os resultados de *F-measure* provenientes das 55 configurações do SR para definir o tipo de teste apropriado, paramétrico ou não paramétrico. Posteriormente, o teste escolhido foi aplicado com a finalidade de comparar as amostras e refutar ou aceitar a hipótese nula levantada. A seguir, são apresentados detalhes da análise dos perfis dos dados que resultou na escolha do teste de *Friedman*.

6.3.1 Conhecendo o Perfil dos Dados

Neste trabalho, verificou-se a normalidade dos dados com o teste de *Shapiro-Wilk*, que é considerado o teste de maior poder estatístico para essa finalidade [98]. O teste foi executado com nível de significância de 5% (ou seja, $\alpha = 0,05$), em que avaliaram-se as seguintes hipóteses:

- H_0: A amostra provém de uma população normalmente distribuída.
- H_1: A amostra não provém de uma população normalmente distribuída.

Na Tabela 6.5 podem ser vistos os resultados dos testes em termos de *p*-valor para as diferentes amostras de *F-measure*. Caso um dos valores apresentados na tabela seja menor que $0,05$, a hipótese nula H_0 é refutada com 95% de confiança para a amostra específica, caso contrário, a hipótese nula é aceita. Ao observar os resultados, percebe-se que todos os testes retornaram *p*-valores inferiores a $0,05$. Portanto, nenhuma das amostras analisadas provém de uma população com distribuição normal. Logo, o teste de ANOVA não é indicado para realizar as comparações, pois o requisito de normalidade foi violado. Por isso, optou-se pela alternativa não paramétrica, ou seja, o teste de *Friedman*.

Tabela 6.5: 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

	JACCARD	**MANHATTAN**	**CANBERRA**	**COSSENO**	**EUCLIDIANA**
1	7,32E-05	9,18E-07	9,18E-07	9,18E-07	9,18E-07
2	4,15E-09	2,44E-06	2,44E-06	2,44E-06	2,44E-06
3	5,10E-08	6,95E-08	6,95E-08	6,95E-08	6,95E-08
4	4,28E-06	1,08E-08	1,08E-08	1,08E-08	1,08E-08
5	1,06E-07	4,12E-08	4,12E-08	4,12E-08	4,12E-08
6	2,39E-07	7,25E-09	7,25E-09	7,25E-09	7,25E-09
7	2,83E-08	2,32E-09	2,32E-09	2,32E-09	2,32E-09
8	1,79E-07	4,20E-08	4,20E-08	4,20E-08	4,20E-08
9	4,97E-07	3,19E-08	3,19E-08	3,19E-08	3,19E-08
10	2,98E-07	2,12E-08	2,12E-08	2,12E-08	2,12E-08
11	4,59E-08	8,03E-08	8,03E-08	8,03E-08	8,03E-08

6.3.2 Análise dos Dados

Nesta parte do trabalho, foi usado o teste de *Friedman*, que é um equivalente não paramétrico da ANOVA de medidas repetidas [37]. Apesar de apresentar teoricamente menor poder estatístico que seu correspondente paramétrico quando os requisitos de normalidade e homoscedasticidade são atendidos, pesquisas apontam que ambos os testes apresentam resultados similares em geral [37].

Na Figura 6.4 pode ser visto o resultado do teste de *Friedman* para a comparação dos resultados de *F-measure* provenientes das 55 configurações do SR, com nível de significância de 5%, ou seja, $\alpha = 0,05$. Da mesma forma que o teste de normalidade apresentado anteriormente, a hipótese nula $H_{0-2.1}$ é rejeitada caso o *p*-valor do teste seja inferior a $0,05$ e aceita, caso contrário. Como o *p*-valor retornado foi de *2,2E-16*, rejeitou-se a hipótese nula $H_{0-2.1}$ (ver Subseção 6.1.1). Portanto, com 95% de confiança, é possível concluir que os resultados de *F-measure* são diferentes entre as configurações do SR.

Para identificar qual das configurações do SR apresenta melhor resultado de *F-measure* e responder a questão de pesquisa $\mathbf{RQ}_{2.1}$ apresentada na Subseção 6.1.1, realizou-se um teste

```
> friedman.test(fmeasure ~ m_k | ID, data= mydata)

        Friedman rank sum test

data:  fmeasure and m_k and ID
Friedman chi-squared = 3890.8, df = 54, p-value < 2.2e-16
```

Figura 6.4: Resultado do teste de *Friedman* para verificar a hipótese H_{0-1} apresentada na Subseção 6.1.1

de *post-hoc*, que compara os diferentes tratamentos.

```
Post Hoc Analysis

Alpha: 0.05 ; DF Error: 8370
t-Student: 1.960247
LSD: 395.7245

Treatments with the same letter are not significantly different.

                  Sum of ranks groups
JACCARD_CINCO           6485.0      a
CANBERRA_UM             6466.5      a
COSINE_UM               6466.5      a
EUCLIDEAN_UM            6466.5      a
MANHATTAN_UM            6466.5      a
JACCARD_QUATRO          6434.5      a
CANBERRA_DOIS           6306.5     ab
COSINE_DOIS             6306.5     ab
EUCLIDEAN_DOIS          6306.5     ab
MANHATTAN_DOIS          6306.5     ab
CANBERRA_TRES           6269.5     ab
COSINE_TRES             6269.5     ab
EUCLIDEAN_TRES          6269.5     ab
MANHATTAN_TRES          6269.5     ab
JACCARD_SEIS            6005.5     bc
CANBERRA_QUATRO         5831.5     cd
COSINE_QUATRO           5831.5     cd
EUCLIDEAN_QUATRO        5831.5     cd
MANHATTAN_QUATRO        5831.5     cd
JACCARD_DOIS            5651.5     cd
JACCARD_TRES            5506.5     de
JACCARD_SETE            5449.5     de
JACCARD_OITO            5158.0      e
CANBERRA_CINCO          4509.0      f
COSINE_CINCO            4509.0      f
EUCLIDEAN_CINCO         4509.0      f
MANHATTAN_CINCO         4509.0      f
JACCARD_NOVE            4246.5     fg
      ⋮                                 ⋮
JACCARD_DEZ             4199.5     fg
JACCARD_UM              4136.0     fg
CANBERRA_SEIS           3865.0     gh
COSINE_SEIS             3865.0     gh
EUCLIDEAN_SEIS          3865.0     gh
MANHATTAN_SEIS          3865.0     gh
JACCARD_ONZE            3495.5     hi
CANBERRA_SETE           3282.0      i
COSINE_SETE             3282.0      i
EUCLIDEAN_SETE          3282.0      i
MANHATTAN_SETE          3282.0      i
CANBERRA_OITO           2722.0      j
COSINE_OITO             2722.0      j
EUCLIDEAN_OITO          2722.0      j
MANHATTAN_OITO          2722.0      j
CANBERRA_NOVE           2537.5     jk
COSINE_NOVE             2537.5     jk
EUCLIDEAN_NOVE          2537.5     jk
MANHATTAN_NOVE          2537.5     jk
CANBERRA_DEZ            2214.0     kl
COSINE_DEZ              2214.0     kl
EUCLIDEAN_DEZ           2214.0     kl
MANHATTAN_DEZ           2214.0     kl
CANBERRA_ONZE           1864.5      l
COSINE_ONZE             1864.5      l
EUCLIDEAN_ONZE          1864.5      l
MANHATTAN_ONZE          1864.5      l
```

Figura 6.5: Resultado do teste de *post-hoc*

Na Figura 6.5 são apresentados os resultados desse teste, na qual pode-se observar o *ranking* das configurações por resultado de *F-measure* e a informação do grupo ao qual cada configuração foi classificada. Desta forma, quanto melhor ranqueada uma configuração, melhor é seu resultado de *F-measure* obtido na validação cruzada. Porém, caso duas configurações sejam classificadas em um mesmo grupo, implica que ambas não apresentam resultados significantemente diferentes. Por exemplo, as configurações com $k = 5$ e *Jaccard* e com $k = 1$ e *Canberra* (1ª e 2ª posições do *ranking*, respectivamente) foram classificadas no mesmo grupo "a". Por isso, não é possível afirmar que ambas as configurações apresentam resultados de *F-measure* significantemente diferentes. Portanto, como as 14 primeiras configurações do *ranking* foram classificadas no grupo "a", conclui-se que elas apresentam os maiores resultados de *F-measure* dentre as 55 configurações estudadas neste experimento. Porém, não é possível afirmar qual delas apresenta melhor resultado entre si. Por isso, foram avaliados critérios de desempate, resultando na escolha da configuração com $k = 1$ e medida de *Manhattan*.

O primeiro critério diz respeito ao número de vizinhos *k* considerado no cálculo de vizinhança. Teoricamente, quanto menor for o valor de *k*, menor é o número de comparações e, consequentemente, menor é o tempo de resposta do algoritmo *k-NN*. Por isso, optou-se por configurações com menor valor de *k*, ou seja, $k = 1$. Assim, dentre as 14 configurações com melhor *F-measure*, foram selecionadas quatro que consideraram apenas um vizinho para gerar as recomendações de RNFs, são elas: configuração com $k = 1$ e medida de *Canberra*; configuração com $k = 1$ e medida do *Cosseno*; configuração com $k = 1$ e medida *Euclidiana*; e configuração com $k = 1$ e medida de *Manhattan*.

O segundo critério de desempate refere-se ao tipo de medida de similaridade/distância. Analisaram-se as características das quatro medidas (ou seja, *Cosseno*, *Manhattan*, *Canberra* e *Euclidiana*) a fim de identificar a que melhor se adéqua ao domínio de problema do sistema de recomendação proposto.

Neste contexto, a similaridade do *Cosseno* é uma medida de similaridade entre dois vetores não nulos que mede o cosseno do ângulo entre eles. Desta forma, é uma medida de orientação e não magnitude. Portanto, não se adéqua totalmente ao domínio do problema estudado neste trabalho, pois pretende-se avaliar a magnitude das diferenças entre os vetores de características dos RFs. Adicionalmente, é possível que o vetor do RF candidato a vizinho

seja nulo, caso o mesmo não apresente nenhuma característica do RF alvo, sendo necessárias pequenas modificações na medida do *Cosseno* para sua aplicação. Um exemplo desse caso pode ser visto na Tabela 6.6, onde o candidato a vizinho f'_1 não apresenta nenhuma característica em comum com o RF alvo f_a.

Tabela 6.6: Exemplo de vetor de característica nulo

	Web	Educação	Cliente-servidor	*TypeScript*	Criar ord. para list.	Expandir info. de item da list.
f_a	1	1	1	1	1	1
f'_1	0	0	0	0	0	0

Por outro lado, as medidas de *Manhattan*, *Canberra* e *Euclidiana* são medidas de distância multidimensionais que derivam-se da distância de *Minkowski* [5, 69] e podem ser aplicadas diretamente na solução proposta. Neste trabalho, por questões de simplicidade, optou-se pela distância de *Manhattan*, que calcula a distância entre dois vetores de tamanhos *d* por meio da soma das diferenças entre eles. Assim, como os vetores de características considerados na solução proposta só podem assumir valores iguais a 0 (zero) ou 1 (um), o cálculo da distância entre eles é dado pela contagem das características do RF alvo que não estão presentes no perfil do RF candidato a vizinho. Por exemplo, na situação exemplificada na Tabela 6.6, a distância entre o RF alvo f_a e o RF candidato a vizinho f'_1 é 6.

Portanto, a configuração escolhida juntamente com seus resultados obtidos no experimento são os seguintes ($\mathbf{RQ}_2$):

- medida de similaridade: similaridade de Manhattan;
- número de vizinhos (*k*): 1;
- precisão: $78,5\%$;
- *recall*: $84,5\%$;
- *F-measure*: 79%.

6.4 Ameaças à Validade

A seguir, são apresentadas as ameaças à validade do experimento bem como as ações tomadas para mitigá-las.

Ameaça à validade da conclusão. A base de dados histórica é composta apenas por 13 projetos de software. Porém, é recomendável ter um conjunto de dados representativo para validar sistemas de recomendação baseados em memória, para possibilitar a representação de diferentes casos do domínio do problema. Para mitigar o problema, foi reduzido o escopo da validação apenas para projetos no contexto de sistemas de informação Web, tornando a base mais representativa para esse contexto.

Ameaça à validade interna. A coleta de dados não foi feita continuamente durante as *Sprints* dos projetos, conforme indicado na adaptação do Scrum. Por isso, tem-se uma ameaça à validade interna. Porém, a base histórica foi formada apenas por informações reais dos projetos disponibilizadas por seus gerentes em ferramentas de gerenciamento, planilhas, artefatos como *backlogs* de projeto e de *Sprint*, etc. Desta forma, as informações de *USs*, tarefas e RNFs foram apenas estruturadas nos modelos propostos de *USs* e RNFs, ou seja, nenhuma informação foi adicionada pelo pesquisador para evitar viés. Além disso, os gerentes responsáveis por cada projeto fizeram a revisão das informações após o processo de estruturação.

Ameaça à validade de construção. Uma das variáveis independentes do experimento é a medida de similaridade/distância utilizada no cálculo de vizinhança pelo algoritmo *k-NN*. Neste experimento, avaliaram-se os resultados de *F-measure* provenientes de cinco medidas diferentes. Porém, sabe-se que outras medidas de similaridade e correlação poderiam ter sido avaliadas também. Desta forma, tem-se uma ameaça à validade de construção do experimento. Para mitigar esse problema, a escolha das medidas se deu por meio de uma pesquisa em trabalhos na área que abordam o tema como em [5], que analisa diferentes medidas de similaridade ou distância apropriadas para o classificador *k-NN*. Neste trabalho, os experimentos foram executados com base no protocolo de validação cruzada *leave-p-out*. Porém, sabe-se que outras abordagens poderiam ter sido utilizadas como *Akaike's Information Criterion* [21] e *Bayesian Information Criterion* [56].

Ameaça à validade externa. O experimento foi realizado com um conjunto de dados

histórico contendo apenas informações de projetos no contexto de sistemas de informação Web. Portanto, não é possível generalizar os resultados dos experimentos para outros contextos diferentes. Para mitigar esse problema, pode-se continuar a coleta de dados de projetos de diferentes contextos em trabalhos futuros.

6.5 Considerações Finais do Capítulo

Neste capítulo, foram apresentadas as atividades realizadas para a validação da solução proposta. Mais especificamente, foram apresentados detalhes sobre a escolha do protocolo de Configuração Experimental *Off-line*, o *design* do experimento com informações sobre variáveis e hipóteses, a geração da base de dados histórica utilizada no experimento, a escolha do protocolo de validação cruzada *leave-p-out*, os resultados alcançados com a execução do experimento e as conclusões obtidas com a análise estatística.

De forma geral, após a análise estatística, observou-se que existem diferenças significativas nos resultados de *F-measure* das diferentes configurações do SR, ou seja, rejeitou-se a hipótese nula $H_{0-2.1}$ (ver Subseção 6.1.1). Desta forma, foi realizado um teste de *post-hoc* que identificou 14 configurações com maior eficácia na recomendação de RNFs para o conjunto de dados histórico coletado (ver Subseção 6.1.3), respondendo a questão de pesquisa $\mathbf{RQ}_{2.1}$ (ver Subseção 6.1.1). Posteriormente, foram analisados 2 critérios de desempate, resultando na escolha da configuração com $k = 1$ e medida de *Manhattan*, respondendo a questão de pesquisa principal $\mathbf{RQ}_2$.

Capítulo 7

Avaliação da Solução

Neste capítulo, são apresentados detalhes acerca do processo de avaliação da solução proposta, que foi realizada por meio da avaliação da utilidade prática da solução e da precisão das recomendações.

7.1 Avaliação da Utilidade e Precisão da Solução Proposta

A utilidade prática da solução foi avaliada por meio de um estudo de caso em quatro projetos reais baseados em Scrum. Para tanto, avaliou-se o uso da ferramenta *NFRec*, que implementa o SR proposto. Como resultado, concluiu-se que o custo-benefício de utilizar a ferramenta é positivo e, consequentemente, a solução proposta é útil para o gerenciamento de requisitos não funcionais. Adicionalmente, foi avaliada a precisão das recomendações geradas durante o estudo. No geral, o SR obteve uma precisão de 81,8%.

O estudo de caso executado é do tipo embarcado [103], no qual cada projeto representa uma unidade de análise e cada time de desenvolvimento o sujeito. O estudo foi realizado simultaneamente durante duas *Sprints* de 15 dias dos quatro projetos, tendo assim 30 dias de duração. Nesse intervalo, foi possível coletar informações de *USs* dos três módulos do modelo estruturado de *USs* e, portanto, julgou-se como um período suficiente para responder as questões de pesquisa.

O objetivo do estudo de caso é avaliar o custo-benefício da solução proposta na perspectiva da equipe de desenvolvimento no âmbito de suporte ao gerenciamento de requisitos não funcionais. Portanto, pretende-se responder as questões de pesquisa principais $\mathbf{RQ}_3$ e $\mathbf{RQ}_4$

(Ver Seção 1.3). Para responder $\mathbf{RQ}_4$, foi calculada a precisão das recomendações geradas no estudo. Por sua vez, para $\mathbf{RQ}_3$, foram formuladas três questões de pesquisa secundárias que, caso respondidas positivamente, trazem evidências para respondê-la. São elas:

1. $\mathbf{RQ}_{3.1}$: a ferramenta *NFRec* é útil para auxiliar na definição de requisitos não funcionais em projetos de software baseados em Scrum?

2. $\mathbf{RQ}_{3.2}$: o custo para usar a ferramenta *NFRec* em projetos de software baseados em Scrum é aceitável?

3. $\mathbf{RQ}_{3.3}$: as recomendações geradas pela ferramenta *NFRec* podem auxiliar a equipe a mitigar problemas relacionados à negligência com RNFs em projetos de software baseados em Scrum?

Portanto, o custo-benefício da ferramenta é considerado positivo caso as perguntas sejam respondidas positivamente.

Como mencionado anteriormente, quatro projetos de software representaram as unidades de análise do estudo que, nesta seção, são referidos como Projeto A, Projeto B, Projeto C e Projeto D. Todos eles consistem em sistemas de informação Web e seus escopos são definidos como:

1. **Projeto A**: visa desenvolver um sistema com um serviço na nuvem e um cliente Web que possibilita a escrita individual ou compartilhada de cordéis e poesias. É composto por dois desenvolvedores;

2. **Projeto B**: visa desenvolver um sistema com um cliente Web para geração de recursos gráficos como crachás, cartões de visita, etc. É composto por três desenvolvedores;

3. **Projeto C**: visa desenvolver um sistema com um serviço na nuvem e um cliente Web para o gerenciamento de projetos de capacitação, possibilitando a gestão de atividades e horários de alunos, seleções de participantes, etc. É composto por cinco desenvolvedores;

4. **Projeto D**: visa desenvolver um sistema com um serviço na nuvem e um cliente Web para construção e execução de *Redes Bayesianas*. É composto por cinco desenvolvedores.

Antes do estudo de caso, todos os projetos realizavam o gerenciamento de requisitos durante as Reuniões de Planejamento de *Sprint* com auxílio de uma versão anterior da mesma ferramenta sem a integração da solução proposta, ou seja, sem a adaptação do Scrum e a recomendação de requisitos não funcionais. Portanto, *User Stories* e tarefas técnicas eram escritas em linguagem natural e não era realizada diretamente nenhuma atividade na ferramenta visando a definição de RNFs, que eram descritos de variadas formas, por exemplo, como critérios de aceitação, itens da definição de pronto, requisitos funcionais, falhas identificadas por casos de testes, etc.

Para responder as questões de pesquisa secundárias referentes à $\mathbf{RQ}_3$, foram coletados dados por meio de um questionário (Apêndice E), que foi gerado com base no Modelo de Aceitação de Tecnologia (TAM) [33, 34]. De forma geral, o TAM visa explicar por que indivíduos optam por adotar ou não adotar uma tecnologia específica ao realizar uma tarefa e baseia-se em duas variáveis: Percepção de Utilidade (PU) e Percepção de Facilidade de Uso (PFU). A percepção de utilidade está relacionada ao grau em que um indivíduo acredita que o uso de uma determinada tecnologia aumentaria seu desempenho no trabalho. Em contrapartida, a percepção de facilidade de uso refere-se ao grau em que um indivíduo acredita que usar uma determinada tecnologia estaria livre de esforço físico e mental. Portanto, adaptando o TAM para o contexto deste trabalho, foram formuladas 14 perguntas para avaliar a percepção de utilidade da ferramenta *NFRec* ($\mathbf{RQ}_{3.1}$) e 14 para avaliar a percepção de facilidade de uso da mesma ($\mathbf{RQ}_{3.2}$). Complementarmente, foram formuladas quatro perguntas para avaliar a relevância das recomendações ($\mathbf{RQ}_{3.3}$). Para cada uma das perguntas, optou-se por utilizar uma escala Likert de cinco níveis para coletar as respostas dos participantes. A escala adotou os valores: (1) Discordo Totalmente, (2) Discordo, (3) Neutro, (4) Concordo e (5) Concordo Totalmente. Por fim, coletou-se em resposta textual simples a opinião dos sujeitos sobre a ferramenta.

O questionário foi respondido por cada equipe de desenvolvimento em conjunto, ou seja, foram coletadas quatro respostas ao todo. As equipes dos projetos B e C tinham em média um ano de experiência em projetos de software baseados em Scrum. Por outro lado, as equipes dos projetos A e D apresentavam em média 3 anos de experiência nesse mesmo tipo de projeto.

Durante a execução do estudo, a ferramenta foi disponibilizada e executada diretamente

na máquina de um dos integrantes de cada equipe. As recomendações foram geradas com base na configuração definida na validação da solução proposta (mais detalhes no Capítulo 6), ou seja, similaridade de *Manhattan* e *k* igual a 1.

A execução do estudo compreendeu a realização de duas etapas, são elas: (i) treinamento e (ii) execução.

Fase de Treinamento. Nesta fase, os participantes do estudo receberam um treinamento que teve como objetivo apresentar os conceitos relacionados à solução proposta necessários à sua devida aplicação. O treinamento ocorreu por meio de apresentação audiovisual, em que primeiramente foram apresentados conceitos sobre a adaptação do Scrum e sobre requisitos não funcionais. Em seguida, foi feita uma demonstração da ferramenta na qual foram apresentadas as funcionalidades de criação de projeto, criação de *US*, criação de tarefa e gerenciamento de recomendações de RNFs. O treinamento teve duração de 1 hora.

Fase de Execução. Após o treinamento, a ferramenta foi utilizada durante a 2ª fase das Reuniões de Planejamento de *Sprint* em cada unidade de análise. Após o planejamento de cada *Sprint*, os resultados de precisão das recomendações foram calculados para cada projeto.

7.1.1 Execução do Estudo de Caso

Para a execução do estudo de caso, o banco de dados da ferramenta *NFRec* foi populado com dados históricos de 13 projetos da mesma empresa (mais detalhes na Seção 6.1.3).

Em um primeiro momento, as equipes criaram na ferramenta os perfis dos seus respectivos projetos com base na atribuição de *tags* referentes aos cinco fatores que incidem na definição de RNFs, ou seja, domínio de aplicação, plataforma, objetivo do projeto, arquitetura de software e tecnologias (mais detalhes na Seção 5.2.1). Nenhuma das equipes apresentou dificuldades nesse passo.

Com os perfis dos projetos criados, o estudo foi iniciado com a *Sprint* corrente de cada projeto. A ferramenta foi utilizada durante a 2ª fase das Reuniões de Planejamento dos mesmos, nas quais as equipes criaram *USs* e tarefas estruturadas. A duração das reuniões não sofreu modificações em nenhum dos projetos, ou seja, continuou dentro do intervalo planejado de 1 hora. Para cada *US* criada, as equipes receberam recomendações de RNFs e puderam interagir livremente com a ferramenta, aceitando ou rejeitando as sugestões. Adi-

cionalmente, as equipes puderam criar novos RNFs e associá-los às *USs*. As atividades realizadas no primeiro momento do estudo de caso são apresentadas a seguir.

No Projeto A, avaliou-se sua primeira *Sprint*, para a qual definiu-se uma *US* no contexto de criação de conta de usuário, que foi mapeada com o módulo “Autenticação” e a operação “Criar conta”. Para essa *US*, foram recomendados dois RNFs do tipo “Segurança”, mas apenas um deles foi considerado relevante pela equipe. O RNF aceito refere-se à criptografia das informações de acesso. Em contrapartida, o RNF desconsiderado refere-se à geração de chave de acesso automática. Segundo os desenvolvedores, esse RNF não se adéqua ao escopo do projeto em questão, mas é apropriado para o módulo e operação definidos. No geral, o SR obteve uma precisão de 50% na primeira *Sprint* avaliada do Projeto A.

Para o Projeto B, avaliou-se a segunda *Sprint*, para a qual foram definidas duas *USs*, uma no contexto de inserção de usuário por um administrador e outra no contexto de exportação de informações para arquivo *PDF*. A primeira *US* foi mapeada com o módulo "Cadastro"e a operação “Inserir dados”. Por sua vez, a segunda *US* foi mapeada com o módulo “Gerencial” e a operação “Exportar relatório em PDF”. Para a primeira *US*, foram recomendados seis RNFs do tipo “Confiabilidade”, dos quais cinco foram aceitos. O RNF desconsiderado refere-se à avaliação de integridade de dados no servidor. Porém, como no projeto está sendo desenvolvido apenas um cliente Web (*front-end*), o mesmo não foi considerado adequado ao escopo do projeto. Em contrapartida, para a segunda *US*, foram recomendados dois RNFs dos tipo “Segurança” e “Desempenho”, respectivamente. Porém, o RNF de desempenho não foi aceito, pois refere-se à capacidade máxima de itens que é possível exportar com a funcionalidade. Segundo os desenvolvedores, como o sistema é voltado apenas para o uso interno de uma empresa, essa restrição não é tratada no escopo do projeto. No geral, o SR obteve uma precisão de 75% na primeira *Sprint* avaliada do Projeto B.

Para o Projeto C, avaliou-se a primeira *Sprint* em execução, para a qual foram especificadas três *USs*. A primeira *US* foi mapeada com o módulo “Autenticação” e a operação “Acessar pela primeira vez o sistema”. Para essa *US*, foram recomendados e aceitos três RNFs do tipo “Segurança” referentes à envio de *token* de verificação por e-mail para permitir o primeiro acesso ao sistema, criptografia dos dados de acesso e pressupostos requeridos após o primeiro acesso. Por outro lado, a segunda *US* foi mapeada com o módulo “Cadastro” e a operação “Inserir dados”. Para essa *US*, foram recomendados e aceitos três RNFs do

tipo “Confiabilidade” relacionadas à capacidade de tolerância à falhas e validação de dados no cliente e no servidor. Por fim, a terceira *US* foi mapeada com o módulo “Cadastro” e a operação “Atualizar dados”. Para essa *US*, foram recomendados dois RNFs do tipo “Confiabilidade” e um do tipo “Segurança”, dos quais dois foram aceitos e um rejeitado. Os RNFs aceitos são relacionados à verificação de integridade de dados (“Confiabilidade”). Em contrapartida, o RNF desconsiderado refere-se a captura de identificação do usuário ao editar informações no sistema. Segundo os desenvolvedores, apesar de ser relacionado diretamente com a operação da *US* em questão, o mesmo não está no escopo do projeto. No geral, o SR obteve 88,8% de precisão na primeira *Sprint* avaliada do Projeto C.

Para o Projeto D, avaliou-se a segunda *Sprint*, para a qual foram definidas três *USs* nos contextos de inserção de dados e visualização de painel de resultados. Dentre elas, duas foram mapeadas com o módulo “Cadastro” e a operação “Inserir dados”. Para cada uma delas, foram recomendados e aceitos três RNFs do tipo “Confiabilidade”, dos quais dois referiam-se à verificação de integridade de dados de inserção e um à tolerância a falhas no envio de dados ao servidor. A terceira *US* especificada na *Sprint* foi mapeada com o módulo “Gerencial” e a operação “Visualizar painel”. Para a mesma, foram recomendados dois RNFs, um do tipo “Segurança” e outro do tipo “Desempenho”. O requisito de segurança refere-se à definição de restrições de controle de acesso de usuários às informações do painel e foi aceito. Por outro lado, o requisito de desempenho refere-se ao tempo de resposta para execução da funcionalidade e foi desconsiderado. No geral, o SR obteve uma precisão de 87,5% na primeira *Sprint* avaliada do Projeto D.

É importante destacar que as recomendações foram validadas pelos respectivos *POs* dos projetos. Porém, não foi avaliado se as premissas e restrições especificadas pelos RNFs foram consideradas no desenvolvimento das *USs* dentro das *Sprints* correntes, pois essa questão não faz parte do escopo deste trabalho de tese.

Após a finalização das primeiras *Sprints* avaliadas, foram executadas as atividades referentes ao segundo momento do estudo de caso, que são apresentadas a seguir.

No Projeto A, avaliou-se a segunda *Sprint* planejada, na qual definiram-se duas *USs*, uma no contexto de autenticação de usuário e outra no contexto de inserção de publicações. A primeira *US* foi mapeada com o módulo “Autenticação” e a operação “Fazer *login* com usuário de senha”. Por sua vez, a segunda *US* foi mapeada com o módulo “Cadastro” e

a operação “Inserir dados”. Para a primeira *US*, foi recomendado e aceito um RNF do tipo “Segurança”. Em contrapartida, para a segunda *US*, foram recomendados e aceitos três RNFs do tipo “Confiabilidade”, dos quais dois referiam-se à integridade dos dados inseridos e um à tolerância a falhas no envio de dados ao servidor. No geral, o SR obteve uma precisão de 100% na segunda *Sprint* avaliada do Projeto A.

Para o Projeto B, avaliou-se a terceira *Sprint* planejada, na qual foram definidas três *US* nos contextos de visualização de painéis, modificação de formulário para inserção de dados e atualização de informações, respectivamente. A primeira *US* foi mapeada com o módulo “Gerencial” e a operação “Visualizar painel”. Por sua vez, a segunda *US* foi mapeada com o módulo “Cadastro” e a operação “Modificar inserção de dados”. Por fim, a terceira *US* foi mapeada com o módulo “Cadastro” e a operação “Atualizar dados”. Para a primeira *US*, foi recomendado e aceito um RNF do tipo “Desempenho”. Para a segunda *US*, recomendou-se um RNF do tipo “Confiabilidade” relacionada à validação de dados de inserção no servidor, que foi desconsiderada por não se adequar ao escopo do projeto. Finalmente, para a terceira *US*, foram recomendados dois RNFs do tipo “Confiabilidade”, dos quais um foi aceito e outro desconsiderado. Como no caso da *US* anterior, a equipe alegou que uma das recomendações não se adequava ao escopo do projeto. No geral, o SR obteve uma precisão de 50% na segunda *Sprint* do projeto B.

Para o Projeto C, foram definidas duas *USs* do módulo “Cadastro” referentes à segunda *Sprint* planejada. A primeira *US* foi mapeada com a operação “Modificar inserção de dados”, a qual recomendou-se um RNF relacionado à capacidade de recuperação de falhas da aplicação (“Confiabilidade”), sendo o mesmo aceito pelos desenvolvedores. Por sua vez, a segunda *US* foi mapeada com a operação “Recuperação de Dados”. Para a mesma, foram retornadas duas recomendações, uma relacionada à restrição de acesso a informações apresentadas na listagem de itens (“Segurança”) e outra relacionada à disponibilidade da aplicação (“Confiabilidade”), ambas foram aceitas pelos desenvolvedores. No geral, o SR obteve 100% de precisão na segunda *Sprint* avaliada do Projeto C.

Por fim, para o Projeto D, avaliou-se a terceira *Sprint* planejada, na qual foram definidas três *USs* relacionadas à modificação de formulário para inserção de dados, visualização de painel de resultados e atualização de dados da rede Bayesiana. A primeira *US* foi mapeada com o módulo “Cadastro” e a operação “Modificar inserção de dados” e recebeu duas

recomendações de RNFs do tipo "Confiabilidade", ambas aceitas. A segunda *US* foi mapeada com o módulo "Gerencial" e a operação "Visualizar painel". Foram recomendados dois RNFs para a mesma, um do tipo "Segurança" e outro do tipo "Desempenho". O requisito de segurança refere-se à definição de restrições de controle de acesso de usuários às informações do painel e foi aceito. Por outro lado, o requisito de desempenho refere-se ao tempo de resposta para execução da funcionalidade e foi desconsiderado. Por fim, a terceira *US* foi mapeada com o módulo "Cadastro" e a operação "Atualizar dados". Para essa *US*, foram recomendados dois RNFs do tipo "Confiabilidade", ambos aceitos. No geral, o SR obteve 83,3% de precisão na segunda *Sprint* avaliada do Projeto D.

7.1.2 Resultados e Discussão

Após o estudo de caso, os valores de precisão total e por projeto foram calculados (Tabela 7.1). Como pode ser visto na tabela, ao todo foram recomendados 44 RNFs, dos quais 36 foram aceitos, resultando em uma taxa de precisão de 81,8% das recomendações para as 20 *USs* especificadas, respondendo a questão de pesquisa principal $\mathbf{RQ}_4$. Dentre os quatro projetos avaliados, obteve-se maior taxa de precisão no Projeto C (91,7%), o qual é composto por profissionais com experiência no uso do modelo estruturado de *US*. Por outro lado, a menor taxa de precisão foi observada no Projeto B (66,7%), que foi o único dos quatro projetos a desenvolver apenas um cliente Web (*front-end*). Portanto, algumas recomendações foram rejeitadas por sugerirem premissas ou restrições de *back-end* como a validação de integridade de dados enviados ao servidor.

Tabela 7.1: Valores de precisão calculados a partir dos dados obtidos no estudo de caso

Projeto	**Nº de *USs***	**Nº de RNFs aceitos**	**Nº de RNFs Recom.**	**Precisão**
Projeto A	3	5	6	83,3%
Projeto B	5	8	12	66,7%
Projeto C	5	11	12	91,7%
Projeto D	7	12	14	85,7%
Total	20	36	44	81,8%

Adicionalmente, cada equipe respondeu um questionário para avaliar a utilidade prática

da solução. Para avaliar as respostas, os valores da escala Likert representados por (1) e (2) foram considerados como indicativos de discordância (Discordo), (3) como indicativo de neutralidade (Neutro) e (4) e (5) como indicativos de concordância (Concordo).

Os dados coletados são sumarizados nas Tabelas 7.2, 7.3 e 7.4 para PU, PFU e relevância das recomendações, respectivamente. Estão destacadas em negrito as respostas que são positivas para as variáveis investigadas. O número máximo de respostas por questão é quatro. Portanto, como foram definidas 14 questões para cada variável do TAM, o número máximo de respostas por variável é 56. Por outro lado, como foram definidas quatro questões para a variável que mede a relevância das recomendações, o número máximo de respostas para a mesma é 16. O resultado final para cada variável analisada é dado pelo somatório dos itens sobre o número máximo de respostas para o questionário correspondente.

Ao analisar os dados da Tabela 7.2, verifica-se que os participantes da pesquisa demonstraram boa aceitação para 13 dos 14 itens analisados. Apenas os itens PU9 e PU12 não receberam avaliações majoritariamente positivas. Para PU9, foram três neutros e um positivo. Em contrapartida, para PU12 foram dois positivos e dois neutros. De forma geral, a PU foi avaliada positivamente em 48 das 56 respostas possíveis. Portanto, a avaliação demonstrou uma aceitação da percepção de utilidade de 85,7%. Logo, é possível afirmar que os participantes da pesquisa consideram a ferramenta útil ($\mathbf{RQ}_{3.1}$).

Ao analisar os dados da Tabela 7.3, verifica-se que os respondentes demonstraram boa aceitação para 12 dos 14 itens de PFU analisados. Apenas os itens PFU1, PFU11, PFU12, PFU13 e PFU14 não receberam avaliações majoritariamente positivas. Para PFU1 e PFU13, foram duas respostas positivas e duas negativas. Em contrapartida, para PFU11, PFU12 e PFU14, foram duas respostas positivas e duas neutras. De forma geral, a PFU foi avaliada positivamente em 41 das 56 respostas possíveis. Portanto, a avaliação demonstrou uma aceitação de PFU de 73,2%. Inicialmente, alguns desenvolvedores tiveram dificuldades para utilizar o modelo estruturados de *USs* e definir módulos e operações. Esse fato pode explicar porque os valores de aceitação de PFU foram menores do que os de PU. Mesmo assim, é possível afirmar que os participantes da pesquisa consideraram que a ferramenta é fácil de usar ($\mathbf{RQ}_{3.2}$).

Por fim, ao analisar os dados da Tabela 7.4, verifica-se que os respondentes demonstraram boa aceitação para todos os itens referentes à avaliação da relevância das recomendações

Tabela 7.2: Dados coletados para a variável percepção de utilidade do TAM

ID	Questão	Concordo	Neutro	Discordo
PU1	O trabalho é difícil sem	**4**	0	0
PU2	Melhora o controle sobre o trabalho	**4**	0	0
PU3	Melhora o desempenho no trabalho	**4**	0	0
PU4	Atende as minhas necessidades	**4**	0	0
PU5	Economiza tempo	**3**	1	0
PU6	Trabalho mais rapidamente	**4**	0	0
PU7	Suporta aspectos crítico do meu trabalho	**4**	0	0
PU8	Realiza mais trabalho	**3**	1	0
PU9	Reduz o tempo improdutivo	**1**	3	0
PU10	Melhora efetividade	**4**	0	0
PU11	Melhora a qualidade do trabalho	**3**	1	0
PU12	Aumenta a produtividade	**2**	2	0
PU13	Torna o trabalho mais fácil	**4**	0	0
PU14	Útil	**4**	0	0

Tabela 7.3: Dados coletados para a variável percepção de facilidade de uso do TAM

ID	Questão	Concordo	Neutro	Discordo
PFU1	Confuso	2	0	**2**
PFU2	Propenso a erros	0	1	**3**
PFU3	Frustrante	0	0	**4**
PFU4	Dependência de manual	0	1	**3**
PFU5	Requer esforço mental	0	0	**4**
PFU6	Fácil recuperação de Erros	**4**	0	0
PFU7	Rígido e Inflexível	1	1	**2**
PFU8	Controlável	**4**	0	0
PFU9	Comportamento Inesperado	0	0	**4**
PFU10	Incômodo	0	1	**3**
PFU11	Interação compreensível	**2**	2	0
PFU12	Facilidade de lembrar	**2**	2	0
PFU13	Fornece orientação	**2**	0	2
PFU14	Fácil de usar	**2**	2	0

($\mathbf{RQ}_{3.3}$). Todas as equipes consideraram que o uso da ferramenta que integra a solução proposta pode auxiliar na definição precoce de RNFs e na redução de retrabalho em fases tardias do desenvolvimento. Além disso, três das quatro equipes consideraram que o uso da ferramenta *NFRec* pode melhorar a documentação de RNFs nos projetos. Por fim, duas equipes consideram que as recomendações geradas podem auxiliar em decisões sobre a arquitetura do software em desenvolvimento. No geral, a avaliação demonstrou uma aceitação para a relevância das recomendações de 81,3%.

Tabela 7.4: Dados coletados para avaliar a relevância das recomendações

ID	Questão	Concordo	Neutro	Discordo
QR1	Definição precoce de RNFs	**4**	0	0
QR2	Auxílio na definição da arquitetura	**2**	1	1
QR3	Melhor documentação de RNFs	**3**	1	0
QR4	Redução de retrabalho em fases tardias	**4**	0	0

7.2 Ameaças à Validade

As ameaças à validade relacionadas ao processo de avaliação da solução proposta são apresentadas a seguir.

Ameaça à validade da conclusão. O estudo de caso foi executado em duas *Sprints* apenas, representando uma ameaça à validade de conclusão. Pois, com mais *Sprints*, as respostas dadas poderiam sofrer modificações. Porém, para mitigar essa ameaça, foram avaliadas quatro equipes simultaneamente, que retornaram resultados semelhantes. Ademais, foram coletados dados referentes a todos os módulos do modelo estruturado de *USs* no período do estudo, ou seja, diferentes cenários de recomendação foram avaliados, mesmo com a curta duração da avaliação.

Ameaça à validade interna. Os desenvolvedores de duas das quatro equipes tinham em média um ano de experiência, o que pode acarretar em uma ameaça à validade interna, pois podem não ter maturidade suficiente para responderem o questionário. Para mitigar essa ameaça, o questionário foi respondido em conjunto por cada equipe, de forma que os desenvolvedores tiveram a oportunidade de discutir entre si suas respostas, agregando seus conhecimentos. Outra ameaça à validade interna refere-se à construção do questionário. Porém, para mitigar esse problema, o mesmo foi gerado com base no TAM, que é um modelo validado e extensivamente utilizado na literatura para avaliar novas tecnologias.

Ameaça à validade externa. O estudo de caso foi realizado com quatro projetos de uma mesma instituição e, consequentemente, não é possível generalizar os resultados do mesmo para outras empresas que utilizam o Scrum. Porém, como trabalho futuro, pretende-se continuar a avaliação da solução proposta em mais projetos.

7.3 Considerações Finais do Capítulo

Neste capítulo, foram apresentadas as atividades realizadas durante o processo de avaliação da solução proposta, na qual foram verificadas a utilidade prática da solução e a precisão das recomendações obtidas em um contexto real.

Para tanto, foi realizado um estudo de caso embarcado com quatro projetos de software baseados em Scrum durante duas *Sprints* com 15 dias de duração cada. No estudo, as equi-

pes dos quatro projetos utilizaram a ferramenta *NFRec* que implementa a solução proposta. Quanto à utilidade prática da solução, avaliou-se o custo-benefício de usar a ferramenta no âmbito de suporte ao gerenciamento de RNFs por meio de um questionário baseado no TAM, que avaliou dois atributos, percepção de utilidade e percepção de facilidade de uso. Adicionalmente, avaliou-se se a ferramenta pode auxiliar equipes de desenvolvimento de software à mitigar problemas relacionados à negligência de requisitos não funcionais (relevância das recomendações). Como resultado, a avaliação demonstrou uma aceitação de 85,7% para PU, 73,2% para PFU e 81,3% para relevância das recomendações.

Quanto à precisão das recomendações, foi obtida uma taxa máxima de 91,7% para o Projeto C e uma taxa de precisão mínima de 66,7% para o Projeto B. Por fim, a taxa média de precisão obtida no estudo de caso para os quatro projetos foi de 81,8%.

Capítulo 8

Considerações Finais

Nesta pesquisa, foi apresentada uma solução para prover suporte automatizado à equipe de desenvolvimento em projetos ágeis baseados em Scrum na elicitação de requisitos não funcionais, visando auxiliar o gerenciamento contínuo de requisitos. A solução proposta é baseada em duas atividades principais, adaptação do Scrum e recomendação de RNFs. A adaptação visa estruturar informações de projetos de software com base em *tags* e categorias e contribui para o processo de coleta de dados pelo sistema de recomendação de RNFs, podendo ser usada para diferentes domínios de aplicação. A recomendação de RNFs é gerada a partir da análise de dados históricos estruturados dos projetos por meio de filtragem colaborativa baseada em vizinhança. A solução foi integrada a uma ferramenta Web de gerenciamento de projetos baseados em Scrum, chamada de *NFRec*. A adoção da ferramenta visa auxiliar o gerenciamento contínuo de requisitos em projetos de software baseados em Scrum, com foco na redução do risco de negligência dos requisitos não funcionais.

A validação da solução proposta foi realizada por meio de uma configuração experimental *off-line* com uma base de dados histórica formada a partir de informações reais de 13 projetos de software. O objetivo desta etapa foi definir, com base na métrica de eficácia *F-measure*, a melhor configuração do algoritmo *k-NN*, que é a base do SR proposto. Para tanto, foram investigadas duas variáveis independentes, medida de similaridade e quantidade de vizinhos considerados na estimativa de vizinhança. Como resultado, das 55 configurações avaliadas, verificou-se que 14 obtiveram os melhores resultados de *F-measure* para a recomendação de RNFs. Assim, após uma análise de critérios de desempate, escolheu-se a configuração com medida de similaridade de *Manhattan* e número de vizinhos igual a 1, que

obteve uma taxa de *F-measure* de 79%. Com estes resultados, concluiu-se que é possível automatizar a elicitação de RNFs por meio de dados históricos.

A avaliação da solução proposta foi realizada por meio da análise da utilidade prática da solução e da precisão das recomendações. A utilidade prática refere-se ao grau de utilidade e facilidade de uso da ferramenta que implementa a solução para a tarefa de elicitação de requisitos não funcionais. Por outro lado, a precisão das recomendações está relacionada à eficácia das recomendações obtida em um contexto real. Para tanto, realizou-se um estudo de caso embarcado durante duas *Sprints* com duração de 15 dias em quatro projetos reais. Como resultado, as equipes demonstraram boa aceitação para a ferramenta quanto à utilidade (85,7%) e facilidade de uso (73,2%). Além disso, observou-se uma taxa de precisão de 81,8% das recomendações geradas durante o estudo. Portanto, concluiu-se que o custo-benefício de utilizar a ferramenta é positivo e, consequentemente, a solução proposta é útil para o gerenciamento de requisitos não funcionais.

Desta forma, os resultados obtidos de *F-measure* (validação) e precisão (avaliação) demonstram que é possível utilizar dados históricos para gerar recomendações de RNFs para prover suporte automatizado à equipe de desenvolvimento em projetos ágeis baseados em Scrum, dando suporte às hipóteses H_{2-1} e H_{4-1} (ver hipóteses na Seção 1.3). Além disso, como as recomendações são geradas com base nas informações estruturadas de projetos, *USs*, tarefas e RNFs, concluiu-se que é possível estruturar as informações de recursos de software de projetos baseados em Scrum com *tags* e categorias, dando suporte à hipótese H_{1-1}. Por fim, os resultados coletados na avaliação quanto à percepção da utilidade e da facilidade de uso da ferramenta *NFRec* demonstram que a mesma teve boa aceitação pelas equipes participantes do estudo de caso, dando suporte às hipóteses H_{3-1} e H_{3-2}.

Como perspectivas futuras, alguns pontos do trabalho podem ser melhorados. Para a criação do modelo estruturado de *USs* e tarefas, foram consideradas apenas informações de projetos Web no contexto de sistemas de informação. Por isso, não é possível representar diretamente com o modelo proposto projetos de outros contextos. Desta forma, pode-se investigar a criação de modelos específicos para outros contextos ou a generalização do modelo proposto para aumentar sua capacidade de representação.

Para validação da solução, foram coletadas informações históricas de 13 projetos de uma mesma instituição. Porém, para garantir maior diversidade e representatividade da base, é

importante continuar a coleta e estruturação de informações de mais projetos e, de preferência, de outras empresas.

Além disso, para montagem da base histórica, as informações dos 13 projetos foram estruturadas manualmente por meio da atribuição de categorias e *tags*, que representou uma limitação na execução do trabalho dado o esforço despendido no processo. Desta forma, pode-se investigar formas de automatizar o processo de estruturação de perfis de projetos, *USs*, tarefas e RNFs históricos.

Para a construção do SR, o método *k-NN* foi utilizado para a etapa de estimativa de vizinhança, que se baseia na escolha da medida de similaridade e no número de vizinhos (k). Na solução apresentada neste trabalho, considerou-se um k fixo, definido no processo de validação. Porém, o número de vizinhos ideal para gerar as recomendações pode variar conforme a base histórica cresce. Por isso, alguns trabalhos investigam soluções dinâmicas do *k-NN* [68, 91, 127], onde o número de vizinhos pode variar de acordo com condições específicas. Desta forma, pode-se investigar a utilização de uma abordagem dinâmica do *k-NN* no contexto de recomendação de RNFs.

Por fim, como a avaliação da solução foi realizada com quatro equipes por um período de 30 dias, pode-se continuar a avaliação da solução com mais equipes e por períodos maiores para minimizar a ameaça à validade externa da solução.

Bibliografia

[1] Gediminas Adomavicius and YoungOk Kwon. Multi-criteria recommender systems. In *Recommender systems handbook*, pages 769–803. Springer, 2011.

[2] 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.

[3] Celso Agra, Denise Assis, Aêda Sousa, Aline Jaqueira, Márcia Lucena, Teresa Maciel, and Fernanda Alencar. Transformação do modelo i * em user stories : razões , intenções e NFR na documentação ágil. *Citeseer*, page 99, 2015.

[4] B. M. Aljallabi and A. Mansour. Enhancement approach for non-functional requirements analysis in agile environment. In *2015 International Conference on Computing, Control, Networking, Electronics and Embedded Systems Engineering (ICCNEEE)*, pages 428–433, Sept 2015.

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

[6] D. Ameller, C. Ayala, J. Cabot, and X. Franch. How do software architects consider non-functional requirements: An exploratory study. In *2012 20th IEEE International Requirements Engineering Conference (RE)*, pages 41–50, Sept 2012.

[7] Sven Apel, Don Batory, Christian Kstner, and Gunter Saake. *Feature-Oriented Software Product Lines: Concepts and Implementation*. Springer Publishing Company, Incorporated, 1st edition, 2016.

[8] Adila Firdaus Arbain, Dayang Norhayati Abang Jawawi, Imran Ghani, and Wan MN Wan Kadir. Non-functional requirement traceability process model for agile software development. *Journal of Telecommunication, Electronic and Computer Engineering (JTEC)*, 9(3-5):203–211, 2017.

[9] Sylvain Arlot and Alain Celisse. A survey of cross-validation procedures for model selection. *Statist. Surv.*, 4:40–79, 2010.

[10] Wesley K. Assunção, Roberto E. Lopez-Herrejon, Lukas Linsbauer, Silvia R. Vergilio, and Alexander Egyed. Reengineering legacy applications into software product lines: A systematic mapping. *Empirical Softw. Engg.*, 22(6):2972–3016, December 2017.

[11] Y Aziz, T Aziz, MI Malik, MK Baig, MZ Ali, and M Baqer. Non functional requirement in agile software development. *University of Engineering and Technology Taxila. Technical Journal*, 22(1):107, 2017.

[12] V. Bajpai and R. P. Gorthi. On non-functional requirements: A survey. In *Electrical, Electronics and Computer Science (SCEECS), 2012 IEEE Students' Conference on*, pages 1–4, March 2012.

[13] Noor Hasrina Bakar, Zarinah M. Kasirun, and Norsaremah Salleh. Feature extraction approaches from natural language requirements for reuse in software product lines. *J. Syst. Softw.*, 106(C):132–149, August 2015.

[14] Kent Beck and Cynthia Andres. *Extreme Programming Explained: Embrace Change (2Nd Edition)*. Addison-Wesley Professional, 2004.

[15] Kent Beck, Mike Beedle, Arie Van Bennekum, Alistair Cockburn, Ward Cunningham, Martin Fowler, James Grenning, Jim Highsmith, Andrew Hunt, Ron Jeffries, et al. Manifesto for agile software development. *Agile Manifesto*, 2001.

[16] Woubshet Behutiye, Pertti Karhapää, Dolors Costal, Markku Oivo, and Xavier Franch. Non-functional requirements documentation in agile software development: Challenges and solution proposal. In Michael Felderer, Daniel Méndez Fernández, Burak Turhan, Marcos Kalinowski, Federica Sarro, and Dietmar Winkler, editors,

Product-Focused Software Process Improvement, pages 515–522, Cham, 2017. Springer International Publishing.

[17] James Bennett, Stan Lanning, et al. The netflix prize. In *Proceedings of KDD cup and workshop*, volume 2007, page 35. New York, NY, USA, 2007.

[18] Adila Firdaus Binti Arbain, Imran Ghani, and Wan Mohd Nasir Wan Kadir. Agile non functional requiremnents (NFR) traceability metamodel. *2014 8th Malaysian Software Engineering Conference, MySEC 2014*, pages 228–233, 2014.

[19] Mohamed Bourimi, Thomas Barth, Joerg M. Haake, Bernd Ueberschär, and Dogan Kesdogan. AFFINE for enforcing earlier consideration of NFRs and human factors when building socio-technical systems following agile methodologies. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 6409 LNCS:182–189, 2010.

[20] Mohamed Bourimi and Ricardo Tesoriero. Non-Functional Requirements for Distributable User Interfaces in Agile Processes. *DUI '14 Proceedings of the 2014 Workshop on Distributed User Interfaces and Multimodal Interaction*, pages 1–13, 2014.

[21] Hamparsum Bozdogan. Model selection and akaike's information criterion (aic): The general theory and its analytical extensions. *Psychometrika*, 52(3):345–370, Sep 1987.

[22] C. R. Camacho, S. Marczak, and D. S. Cruzes. Agile team members perceptions on non-functional testing: Influencing factors from an empirical study. In *2016 11th International Conference on Availability, Reliability and Security (ARES)*, pages 582–589, Aug 2016.

[23] Amadeu Silveira Campanelli and Fernando Silva Parreiras. Agile methods tailoring – a systematic literature review. *Journal of Systems and Software*, 110:85 – 100, 2015.

[24] L. Cao and B. Ramesh. Agile requirements engineering practices: An empirical study. *IEEE Software*, 25(1):60–67, Jan 2008.

[25] Òscar Celma. *Evaluation Metrics*, pages 109–128. Springer Berlin Heidelberg, Berlin, Heidelberg, 2010.

[26] Òscar Celma. Music recommendation. In *Music Recommendation and Discovery*, pages 43–85. Springer, 2010.

[27] Seung-Seok Choi, Sung-Hyuk Cha, and Charles C Tappert. A survey of binary similarity and distance measures. *Journal of Systemics, Cybernetics and Informatics*, 8(1):43–48, 2010.

[28] Nupur Chugh and Aditya Dev Mishra. Assimilation of Four Layered Approach to NFR in Agile Requirement Engineering. *International Journal of Computer Applications*, 78(5):25–28, 2013.

[29] Mike Cohn. *User stories applied: For agile software development*. Addison-Wesley Professional, 2004.

[30] D. S. Cruzes and T. Dyba. Recommended steps for thematic synthesis in software engineering. In *2011 International Symposium on Empirical Software Engineering and Measurement*, pages 275–284, Sep. 2011.

[31] Karina Curcio, Rodolfo Santana, Sheila Reinehr, and Andreia Malucelli. Usability in agile software development: A tertiary study. *Computer Standards & Interfaces*, 2019.

[32] Ivonei Freitas Da Silva, Paulo Anselmo da Mota Silveira Neto, Pádraig O'Leary, Eduardo Santana de Almeida, and Silvio Romero de Lemos Meira. Using a multi-method approach to understand agile software product lines. *Information and Software Technology*, 57:527–542, 2015.

[33] Fred D Davis. *A technology acceptance model for empirically testing new end-user information systems: Theory and results*. PhD thesis, Massachusetts Institute of Technology, 1985.

[34] Fred D Davis. Perceived usefulness, perceived ease of use, and user acceptance of information technology. *MIS quarterly*, pages 319–340, 1989.

[35] Jose Luis De La Vara, Krzysztof Wnuk, Richard Berntsson-Svensson, Juan Sánchez, and Björn Regnell. An empirical study on the importance of quality requirements in industry. In *SEKE*, pages 438–443, 2011.

[36] Andrea De Lucia and Abdallah Qusef. Requirements engineering in agile software development. *Journal of Emerging Technologies in Web Intelligence*, 2(3):212–220, 2010.

[37] Janez Demšar. Statistical comparisons of classifiers over multiple data sets. *J. Mach. Learn. Res.*, 7:1–30, December 2006.

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

[39] Torgeir Dingsøyr, Sridhar Nerur, VenuGopal Balijepally, and Nils Brede Moe. A decade of agile methodologies: Towards explaining agile software development. *Journal of Systems and Software*, 85(6):1213 – 1221, 2012. Special Issue: Agile Development.

[40] D. Domah and F. J. Mitropoulos. The nerv methodology: A lightweight process for addressing non-functional requirements in agile software development. In *Southeast-Con 2015*, pages 1–7, April 2015.

[41] Horatiu Dumitru, Marek Gibiec, Negar Hariri, Jane Cleland-Huang, Bamshad Mobasher, Carlos Castro-Herrera, and Mehdi Mirakhorli. On-demand feature recommendations derived from mining public product descriptions. In *Proceedings of the 33rd International Conference on Software Engineering*, ICSE '11, pages 181–190, New York, NY, USA, 2011. ACM.

[42] Tore Dybå and Torgeir Dingsøyr. Empirical studies of agile software development: A systematic review. *Information and Software Technology*, 50(9):833 – 859, 2008.

[43] J. Eckhardt, A. Vogelsang, and H. Femmer. An approach for creating sentence patterns for quality requirements. In *2016 IEEE 24th International Requirements Engineering Conference Workshops (REW)*, pages 308–315, Sep. 2016.

[44] J. Eckhardt, A. Vogelsang, H. Femmer, and P. Mager. Challenging incompleteness of performance requirements by sentence patterns. In *2016 IEEE 24th International Requirements Engineering Conference (RE)*, pages 46–55, Sep. 2016.

[45] Jonas Eckhardt. *Categorizations of Product-related Requirements in Practice*. PhD thesis, Technische Universität München, 2017.

[46] Jonas Eckhardt, Andreas Vogelsang, and Daniel Méndez Fernández. Are "non-functional"requirements really non-functional?: An investigation of non-functional requirements in practice. In *Proceedings of the 38th International Conference on Software Engineering*, ICSE '16, pages 832–842, New York, NY, USA, 2016. ACM.

[47] K. Elghariani and N. Kama. Review on agile requirements engineering challenges. In *2016 3rd International Conference on Computer and Information Sciences (ICCOINS)*, pages 507–512, Aug 2016.

[48] W. M. Farid and F. J. Mitropoulos. Normatic: A visual tool for modeling non-functional requirements in agile processes. In *2012 Proceedings of IEEE Southeastcon*, pages 1–8, March 2012.

[49] W. M. Farid and F. J. Mitropoulos. Novel lightweight engineering artifacts for modeling non-functional requirements in agile processes. In *2012 Proceedings of IEEE Southeastcon*, pages 1–7, March 2012.

[50] W. M. Farid and F. J. Mitropoulos. Norplan: Non-functional requirements planning for agile processes. In *2013 Proceedings of IEEE Southeastcon*, pages 1–8, April 2013.

[51] W. M. Farid and F. J. Mitropoulos. Visualization and scheduling of non-functional requirements for agile processes. In *2013 Proceedings of IEEE Southeastcon*, pages 1–8, April 2013.

[52] Weam M. Farid. The normap methodology: Lightweight engineering of non-functional requirements for agile processes. In *Proceedings of the 2012 19th Asia-Pacific Software Engineering Conference - Volume 01*, APSEC '12, pages 322–325, Washington, DC, USA, 2012. IEEE Computer Society.

[53] Adila Firdaus, Imran Ghani, Dayang Norhayati Abg Jawawi, and Wan Mohd Nasir Wan Kadir. Non functional requirements (NFRs) traceability metamodel for agile development. *Jurnal Teknologi*, 77(9):115–125, 2015.

[54] Steve Freeman and Nat Pryce. *Growing object-oriented software, guided by tests*. Pearson Education, 2009.

[55] 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.

[56] Jayanta K Ghosh, Mohan Delampady, and Tapas Samanta. *An introduction to Bayesian analysis: theory and methods*. Springer Science & Business Media, 2007.

[57] Asela Gunawardana and Guy Shani. A Survey of Accuracy Evaluation Metrics of Recommendation Tasks. *The Journal of Machine Learning Research*, 10:2935–2962, 2009.

[58] M. Hamza and R. J. Walker. Recommending features and feature relationships from requirements documents for software product lines. In *2015 IEEE/ACM 4th International Workshop on Realizing Artificial Intelligence Synergies in Software Engineering*, pages 25–31, May 2015.

[59] N. C. Haugen. An empirical study of using planning poker for user story estimation. In *AGILE 2006 (AGILE'06)*, pages 9 pp.–34, July 2006.

[60] Diane S Hayes. Evaluation and application of a project charter template to improve the project planning process. In *Project Management Journal*. Project Management Institute, 2000.

[61] Petra Heck and Andy Zaidman. A systematic literature review on quality criteria for agile requirements specifications. *Software Quality Journal*, Sep 2016.

[62] Ville T. Heikkila, Daniela Damian, Casper Lassenius, and Maria Paasivaara. A Mapping Study on Requirements Engineering in Agile Software Development. *Proceedings - 41st Euromicro Conference on Software Engineering and Advanced Applications, SEAA 2015*, pages 199–207, 2015.

[63] Ruben Heradio, Hector Perez-Morago, David Fernandez-Amoros, Francisco Javier Cabrerizo, and Enrique Herrera-Viedma. A bibliometric analysis of 20 years of research on software product lines. *Information and Software Technology*, 72:1 – 15, 2016.

[64] Rashina Hoda, Norsaremah Salleh, John Grundy, and Hui Mien Tee. Systematic literature reviews in agile software development: A tertiary study. *Information and Software Technology*, 85:60 – 70, 2017.

[65] Irum Inayat, Lauriane Moraes, Maya Daneva, and Siti Salwah Salim. A reflection on agile requirements engineering: Solutions brought and challenges posed. In *Scientific Workshop Proceedings of the XP2015*, XP '15 workshops, pages 6:1–6:7, New York, NY, USA, 2015. ACM.

[66] 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(Part B):915 – 929, 2015. Computing for Human Learning, Behaviour and Collaboration in the Social and Mobile Networks Era.

[67] A. Jedlitschka and D. Pfahl. Reporting guidelines for controlled experiments in software engineering. In *2005 International Symposium on Empirical Software Engineering, 2005.*, pages 10 pp.–, Nov 2005.

[68] L. Jiang, Z. Cai, D. Wang, and S. Jiang. Survey of improving k-nearest-neighbor for classification. In *Fourth International Conference on Fuzzy Systems and Knowledge Discovery (FSKD 2007)*, volume 1, pages 679–683, Aug 2007.

[69] 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.

[70] R. Kasauli, G. Liebel, E. Knauss, S. Gopakumar, and B. Kanagwa. Requirements engineering challenges in large-scale agile system development. In *2017 IEEE 25th International Requirements Engineering Conference (RE)*, pages 352–361, Sept 2017.

[71] M. Kassab. An empirical study on the requirements engineering practices for agile software development. In *2014 40th EUROMICRO Conference on Software Engineering and Advanced Applications*, pages 254–261, Aug 2014.

[72] E. Knauss, G. Liebel, K. Schneider, J. Horkoff, and R. Kasauli. Quality requirements in agile as a knowledge management problem: More than just-in-time. In *2017 IEEE 25th International Requirements Engineering Conference Workshops (REW)*, pages 427–430, Sep. 2017.

[73] Yehuda Koren and Robert Bell. *Advances in Collaborative Filtering*, pages 77–118. Springer US, Boston, MA, 2015.

[74] Yehuda Koren, Robert Bell, and Chris Volinsky. Matrix factorization techniques for recommender systems. *Computer*, 42(8):30–37, August 2009.

[75] Arun B. Krishna and Sunil E. Abraham. Statistical analysis of memory and performance non functional requirements in real time embedded system development for agile methodology. *2015 International Conference on Industrial Instrumentation and Control, ICIC 2015*, pages 300–305, 2015.

[76] Charles W. Krueger and Paul C. Clements. Feature-based systems and software product line engineering: Ple for the enterprise. In *Proceedings of the 21st International Systems and Software Product Line Conference - Volume A*, SPLC '17, pages 253–253, New York, NY, USA, 2017. ACM.

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

[78] Pasquale Lops, Marco de Gemmis, and Giovanni Semeraro. *Content-based Recommender Systems: State of the Art and Trends*, pages 73–105. Springer US, Boston, MA, 2011.

[79] Garm Lucassen, Fabiano Dalpiaz, Jan Martijn E. M. van der Werf, and Sjaak Brinkkemper. Improving agile requirements: the quality user story framework and tool. *Requirements Engineering*, 21(3):383–403, Sep 2016.

[80] Dewi Mairiza, Didar Zowghi, and Nurie Nurmuliani. An investigation into the notion of non-functional requirements. In *Proceedings of the 2010 ACM Symposium on Applied Computing*, SAC '10, pages 311–317, New York, NY, USA, 2010. ACM.

[81] R. R. Maiti and F. J. Mitropoulos. Capturing, eliciting, predicting and prioritizing (cepp) non-functional requirements metadata during the early stages of agile software development. In *SoutheastCon 2015*, pages 1–8, April 2015.

[82] R. R. Maiti and F. J. Mitropoulos. Capturing, eliciting, and prioritizing (cep) nfrs in agile software engineering. In *SoutheastCon 2017*, pages 1–7, March 2017.

[83] Richard R. Maiti, Aleksandr Krasnov, and Marvin Wilborne. Predicting nfrs in agile software engineering. In *Proceedings of the 19th Annual SIG Conference on Information Technology Education*, SIGITE '18, pages 161–161, New York, NY, USA, october 2018. ACM.

[84] Richard R. Maiti and Frank J. Mitropoulos. Prioritizing Non-Functional Requirements in Agile Software Engineering. *Proceedings of the SouthEast Conference on - ACM SE '17*, pages 212–214, 2017.

[85] Richard Rabin Maiti, Aleksandr Krasnov, and Deanna Marie Wilborne. Agile software engineering & the future of non-functional requirements. *Journal of Software Engineering Practice*, 2(1):1–8, december 2018.

[86] Robert C. Martin and Micah Martin. Agile practices. In *Agile Principles, Patterns, and Practices in C# (Robert C. Martin)*, chapter 1, pages 27–37. Prentice Hall PTR, Upper Saddle River, NJ, USA, 2006.

[87] Jamshaid G Mohebzada, Guenther Ruhe, and Armin Eberlein. Systematic mapping of recommendation systems for requirements engineering. In *Software and System Process (ICSSP), 2012 International Conference on*, pages 200–209. IEEE, 2012.

[88] Ana Moreira, João Araújo, and Isabel Brito. Crosscutting quality attributes for requirements engineering. In *Proceedings of the 14th international conference on Software engineering and knowledge engineering*, pages 167–174. ACM, 2002.

[89] Fabiola Moyon, Kristian Beckers, Sebastian Klepper, Philipp Lachberger, and Bernd Bruegge. Towards continuous security compliance in agile software development at scale. In *Proceedings of the 4th International Workshop on Rapid Continuous Software Engineering*, RCoSE '18, pages 31–34, New York, NY, USA, 2018. ACM.

[90] Padraig O'Leary, Fergal McCaffery, Ita Richardson, and Steffen Thiel. Towards agile product derivation in software product line engineering. *International Workshop on Rapid Integration of Software Engineering techniques*, 2007.

[91] Stefanos Ougiaroglou, Alexandros Nanopoulos, Apostolos N. Papadopoulos, Yannis Manolopoulos, and Tatjana Welzer-Druzovec. Adaptive k-nearest-neighbor classification using a dynamic number of nearest neighbors. In Yannis Ioannidis, Boris Novikov, and Boris Rachev, editors, *Advances in Databases and Information Systems*, pages 66–82, Berlin, Heidelberg, 2007. Springer Berlin Heidelberg.

[92] F. Paetsch, A. Eberlein, and F. Maurer. Requirements engineering and agile software development. In *WET ICE 2003. Proceedings. Twelfth IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises, 2003.*, pages 308–313, June 2003.

[93] PMI. *A Guide to the Project Management Body of Knowledge (PMBOK® Guide), 6th Edition.* Project Management Institute, 2017.

[94] Klaus Pohl, Günter Böckle, and Frank J van Der Linden. *Software product line engineering: foundations, principles and techniques*. Springer Science & Business Media, 2005.

[95] Nagy Ramadan and Salwa Megahed. Requirements Engineering in Scrum Framework. *International Journal of Computer Applications*, 149(8):24–29, 2016.

[96] Balasubramaniam Ramesh, Lan Cao, and Richard Baskerville. Agile requirements engineering practices and challenges: an empirical study. *Information Systems Journal*, 20(5):449–480, 2010.

[97] Felipe Ramos, Alexandre Costa, Mirko Perkusich, Hyggo Almeida, and Angelo Perkusich. A non-functional requirements recommendation system for scrum-based pro-

jects. *In The 30th International Conference on Software Engineering and Knowledge Engineering*, 2018.

[98] Nornadiah Mohd Razali, Yap Bee Wah, et al. Power comparisons of shapiro-wilk, kolmogorov-smirnov, lilliefors and anderson-darling tests. *Journal of statistical modeling and analytics*, 2(1):21–33, 2011.

[99] Francesco Ricci, Lior Rokach, and Bracha Shapira. *Introduction to Recommender Systems Handbook*, pages 1–35. Springer US, Boston, MA, 2011.

[100] M. Robeer, G. Lucassen, J. M. E. M. v. d. Werf, F. Dalpiaz, and S. Brinkkemper. Automated extraction of conceptual models from user stories via nlp. In *2016 IEEE 24th International Requirements Engineering Conference (RE)*, pages 196–205, Sept 2016.

[101] F. Roda. An experimental study on collaborative filtering for requirements engineering. In *2012 IEEE 17th International Conference on Engineering of Complex Computer Systems*, pages 23–28, July 2012.

[102] Kenneth S. Rubin. Scrum framework. In *Essential Scrum: A practical guide to the most popular Agile process*, chapter 2, pages 13–28. Addison-Wesley, 2012.

[103] Per Runeson and Martin Höst. Guidelines for conducting and reporting case study research in software engineering. *Empirical Software Engineering*, 14(2):131, Dec 2008.

[104] Ahmed E Sabry and Sherif S El-Rabbat. Proposed framework for handling architectural nfr's within scrum methodology. In *Proceedings of the International Conference on Software Engineering Research and Practice (SERP)*, page 238. The Steering Committee of The World Congress in Computer Science, Computer Engineering and Applied Computing (WorldComp), 2015.

[105] V. Sachdeva and L. Chung. Handling non-functional requirements for big data and iot projects in scrum. In *2017 7th International Conference on Cloud Computing, Data Science Engineering - Confluence*, pages 216–221, Jan 2017.

[106] Eva-Maria Schon, Jörg Thomaschewski, and María José Escalona. Agile requirements engineering: A systematic literature review. *Computer Standards & Interfaces*, 49(Supplement C):79 – 91, 2017.

[107] Eva-Maria Schön, Dominique Winter, María José Escalona, and Jörg Thomaschewski. *Key Challenges in Agile Requirements Engineering*, pages 37–51. Springer International Publishing, Cham, 2017.

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

[109] S. Asim Ali Shah Sehrish Alam, Shahid Nazir Bhatti and Dr. Amr Mohsen Jadi. Impact and challenges of requirement engineering in agile methodologies: A systematic review. *International Journal of Advanced Computer Science and Applications(IJACSA)*, 2017.

[110] Ana Silva, André Silva, Thalles Araújo, Renan Willamy, Felipe Ramos, Alexandre Costa, Mirko Perkusich, and Ednaldo Dilorenzo. Ordering the product backlog in agile software development projects: A systematic literature review. In *International Conference on Software Engineering & Knowledge Engineering*, pages 1–7, July 2017.

[111] J. Slankas and L. Williams. Automated extraction of non-functional requirements in available documentation. In *2013 1st International Workshop on Natural Language Analysis in Software Engineering (NaturaLiSE)*, pages 9–16, May 2013.

[112] H. F. Soares, N. S. R. Alves, T. S. Mendes, M. Mendonça, and R. O. Spínola. Investigating the link between user stories and documentation debt on software projects. In *2015 12th International Conference on Information Technology - New Generations*, pages 385–390, April 2015.

[113] Stavros Stavru. A critical examination of recent industrial surveys on agile method usage. *Journal of Systems and Software*, 94:87 – 97, 2014.

[114] Trupti Suryawanshi and Gauri Rao. A survey to support nfrs in agile software development process. *International Journal of Computer Science and Information Technologies*, 6(6):5487–5489, 2015.

[115] Vishvadeep Tripathi and Arvind Kumar Goyal. Agile Requirement Engineer: Roles and Responsibilities. *IJISET - International Journal of Innovative Science, Engineering & Technology*, 1(3):213–219, 2014.

[116] VersionOne. *12th Annual State of Agile Survey*, 2018. Acessado em: 19 de dezembro de 2018. https://explore.versionone.com/state-of-agile/versionone-12th-annual-state-of-agile-report.

[117] Hugo Villamizar, M Kalinowski, M Viana, and D Méndez Fernández. A systematic mapping study on security in agile requirements engineering. *CoRR*, 2018.

[118] VN Vithana. Scrum requirements engineering practices and challenges in offshore software development. *International Journal of Computer Applications*, 116(22), 2015.

[119] Shuai Wang, David Buchmann, Shaukat Ali, Arnaud Gotlieb, Dipesh Pradhan, and Marius Liaaen. Multi-objective test prioritization in software product line testing: An industrial case study. In *Proceedings of the 18th International Software Product Line Conference - Volume 1*, SPLC '14, pages 32–41, New York, NY, USA, 2014. ACM.

[120] Y. Wautelet, S. Heng, M. Kolp, I. Mirbel, and S. Poelmans. Building a rationale diagram for evaluating user story sets. In *2016 IEEE Tenth International Conference on Research Challenges in Information Science (RCIS)*, pages 1–12, June 2016.

[121] Yves Wautelet, Samedi Heng, Diana Hintea, Manuel Kolp, and Stephan Poelmans. Bridging user story sets with the use case model. In Sebastian Link and Juan C. Trujillo, editors, *Advances in Conceptual Modeling*, pages 127–138, Cham, 2016. Springer International Publishing.

[122] Yves Wautelet, Samedi Heng, Manuel Kolp, and Isabelle Mirbel. Unifying and extending user story models. In Matthias Jarke, John Mylopoulos, Christoph Quix, Colette Rolland, Yannis Manolopoulos, Haralambos Mouratidis, and Jennifer Horkoff, editors, *Advanced Information Systems Engineering*, pages 211–225, Cham, 2014. Springer International Publishing.

[123] Dave West and Tom Grant. Agile Development: Mainstream Adoption Has Changed Agility – Trends In Real-World Adoption Of Agile Methods. Technical report, Forrester Research, January 2010.

[124] Roel Wieringa, Neil Maiden, Nancy Mead, and Colette Rolland. Requirements engineering paper classification and evaluation criteria: a proposal and a discussion. *Requirements Engineering*, 11(1):102–107, 2006.

[125] James Withey. Investment analysis of software assets for product lines. Technical report, CARNEGIE-MELLON UNIV PITTSBURGH PA SOFTWARE ENGINEERING INST, 1996.

[126] Claes Wohlin, Per Runeson, Martin Hst, Magnus C. Ohlsson, Bjrn Regnell, and Anders Wessln. *Experimentation in Software Engineering*. Springer Publishing Company, Incorporated, 2012.

[127] J. Wu, Z. Cai, and Z. Gao. Dynamic k-nearest-neighbor with distance and attribute weighted for classification. In *2010 International Conference on Electronics and Information Engineering*, volume 1, pages V1–356–V1–360, Aug 2010.

[128] M Younas, DNA Jawawi, I Ghani, and R Kazmi. Non-functional requirements elicitation guideline for agile methods. *Journal of Telecommunication, Electronic and Computer Engineering (JTEC)*, 9(3-4):137–142, 2017.

[129] Didar Zowghi and Zhi Jin. The Role of Requirements Engineering Practices in Agile Development: An Empirical Study. *Communications in Computer and Information Science*, 432 CCIS(January), 2014.

Apêndice A

Modelo Estruturado de USs e Tarefas

O modelo estruturado de *USs* expandido com informações de tarefas é apresentado nas páginas a seguir.

Módulo	Operação	Tarefas
Autenticação	Fazer login com usuário e senha	Criar campos Login e Senha
Autenticação	Fazer login com usuário e senha	Armazenar dados temporários (Função "Remember me")
Autenticação	Fazer login com usuário e senha	Criar serviço Auth (Server)
Autenticação	Fazer login com usuário e senha	Criar serviço Auth (Client)
Autenticação	Fazer login com usuário e senha	Declarar o provider
Autenticação	Fazer login com usuário e senha	Integrar back com front
Autenticação	Fazer login com usuário e senha	Criptografar senha no banco
Autenticação	Fazer login com usuário e senha	Configurar API
Autenticação	Fazer login com usuário e senha	Fazer logout
Autenticação	Fazer login com usuário e senha	Verificação de senha expirada
Autenticação	Fazer login com usuário e senha	Implementar timeout de sessão
Autenticação	Fazer login com usuário e senha	Implementar Captcha
Autenticação	Fazer login com usuário e senha	Fazer o roteamento da URL
Autenticação	Fazer login com OAuth	Criar serviço
Autenticação	Fazer login com OAuth	Fazer roteamento de URL
Autenticação	Fazer login com OAuth	Consumir API externa
Autenticação	Recuperar senha	Alterar tela de login
Autenticação	Recuperar senha	Validar e-mail e senha informado
Autenticação	Recuperar senha	Criar rota de recuperação de senha
Autenticação	Recuperar senha	Gerar senha padrão
Autenticação	Recuperar senha	Criar serviço de notificação
Autenticação	Recuperar senha	Criar tela para cadastro de nova senha
Autenticação	Acessar pela primeira vez o sistema	Criar tela para primeiro acesso
Autenticação	Acessar pela primeira vez o sistema	Comparar senhas informadas
Autenticação	Acessar pela primeira vez o sistema	Validar e-mail e senha informado
Autenticação	Acessar pela primeira vez o sistema	Integrar back com front
Autenticação	Acessar pela primeira vez o sistema	Criar rota de cadastro
Autenticação	Acessar pela primeira vez o sistema	Encriptar senha do novo usuário
Autenticação	Acessar pela primeira vez o sistema	Persistir no banco os dados de acesso
Autenticação	Acessar pela primeira vez o sistema	Fazer o roteamento da URL
Autenticação	Validar Permissão do Usuário	Adicionar atributo de proteção no provider
Autenticação	Validar Permissão do Usuário	Recuperar dados do usuário logado
Autenticação	Validar Permissão do Usuário	Validar permissão do usuário na aplicação
Autenticação	Validar Permissão do Usuário	Criar middlewares para validação do token
Autenticação	Validar Permissão do Usuário	Bloquear rotas
Autenticação	Atualizar perfil	Criar tela para atualizar perfil
Autenticação	Atualizar perfil	Integrar back com front
Autenticação	Atualizar perfil	Criar serviço user
Autenticação	Atualizar perfil	Declarar o provider
Autenticação	Atualizar perfil	Validar senha com confirmar senha
Autenticação	Atualizar perfil	Criar rota de cadastro
Autenticação	Atualizar perfil	Persistir no banco os dados de acesso
Autenticação	Criar conta	Criar tela para nova conta
Autenticação	Criar conta	Criar entidade no banco de dados
Autenticação	Criar conta	Integrar back com front
Autenticação	Criar conta	Atualizar serviço para nova conta
Autenticação	Criar conta	Declarar o provider

Módulo	Operação	Tarefas
Autenticação	Criar conta	Criar rota de cadastro
Autenticação	Criar conta	Encriptar senha do usuário
Autenticação	Criar conta	Persistir no banco os dados de acesso
Autenticação	Remover conta	Integrar back com front
Autenticação	Remover conta	Criar serviço
Cadastro	Recuperação de Dados	Criar tela listagem
Cadastro	Recuperação de Dados	Criar paginação para listagem
Cadastro	Recuperação de Dados	Criar ordenação para listagem
Cadastro	Recuperação de Dados	Criar filtro para listagem
Cadastro	Recuperação de Dados	Criar Gráfico
Cadastro	Recuperação de Dados	Criar tela para visualizar detalhes
Cadastro	Recuperação de Dados	Alterar menu do sistema
Cadastro	Recuperação de Dados	Expandir informações de item da listagem
Cadastro	Recuperação de Dados	Fazer chamada para tela de listagem
Cadastro	Recuperação de Dados	Criar serviço de listagem
Cadastro	Recuperação de Dados	Criar rotas para listagem
Cadastro	Recuperação de Dados	Consumir serviço para listagem
Cadastro	Recuperação de Dados	Criar tela para visualizar detalhes
Cadastro	Recuperação de Dados	Recuperar dados do banco de dados
Cadastro	Atualizar dados	Criar tela alteração
Cadastro	Atualizar dados	Fazer validação de dados no cliente.
Cadastro	Atualizar dados	Fazer chamada para tela de alteração
Cadastro	Atualizar dados	Criar serviço alteração
Cadastro	Atualizar dados	Validar Dados no Servidor
Cadastro	Atualizar dados	Consumir serviço para alteração
Cadastro	Atualizar dados	Associar entidades no banco de dados
Cadastro	Atualizar dados	Atualizar dados banco de dados
Cadastro	Inserir dados	Criar tela inserção
Cadastro	Inserir dados	Criar entidade no banco de dados
Cadastro	Inserir dados	Criar rota de inserção de dados
Cadastro	Inserir dados	Fazer validação de dados no cliente
Cadastro	Inserir dados	Fazer chamada para tela de inserção
Cadastro	Inserir dados	Criar método de inserção
Cadastro	Inserir dados	Validar dados inserção no servidor
Cadastro	Inserir dados	Consumir serviço para inserção
Cadastro	Inserir dados	Persistir dados banco de dados
Cadastro	Modificar inserção de dados	Modificar tela inserção
Cadastro	Modificar inserção de dados	Modificar entidade no banco de dados
Cadastro	Modificar inserção de dados	Modificar rota de inserção de dados
Cadastro	Modificar inserção de dados	Modificar validação de dados no cliente
Cadastro	Modificar inserção de dados	Modificar chamada para tela de inserção
Cadastro	Modificar inserção de dados	Modificar método de inserção
Cadastro	Modificar inserção de dados	Modificar validação dados inserção no servidor
Cadastro	Modificar inserção de dados	Consumir serviço para inserção
Cadastro	Modificar inserção de dados	Persistir dados banco de dados
Cadastro	Remover dados	Alterar tela para exclusão
Cadastro	Remover dados	Criar serviço exclusão

Módulo	Operação	Tarefas
Cadastro	Remover dados	Criar rotas para remoção de dados
Cadastro	Remover dados	Consumir serviço para exclusão
Cadastro	Remover dados	Criar modal de confirmação e toast de feedback
Cadastro	Remover dados	Remover dados banco de dados
Gerencial	Visualizar painel	Criar tela do painel
Gerencial	Visualizar painel	Declarar no provider
Gerencial	Visualizar painel	Adicionar verificação de autorização
Gerencial	Visualizar painel	Adicionar menus no painel
Gerencial	Exportar relatório em PDF	Adicionar botão para exportação do relatório
Gerencial	Exportar relatório em PDF	Criar Relatório
Gerencial	Exportar relatório em PDF	Modificar relatório
Gerencial	Exportar relatório em PDF	Criar dialog para pegar o titulo do PDF
Gerencial	Exportar relatório em PDF	Adicionar campo de titulo no dialog
Gerencial	Exportar relatório em PDF	Integrar back com front
Gerencial	Exportar relatório em PDF	Criar serviço para exportação de relatórios
Gerencial	Exportar relatório em PDF	Exportar Gráfico
Gerencial	Exportar relatório em PDF	Implementar metodo para exportação de relatório em PDF
Gerencial	Exportar relatório em XLS	Adicionar botão para exportação do relatório
Gerencial	Exportar relatório em XLS	Criar dialog para pegar o título do relatório
Gerencial	Exportar relatório em XLS	Adicionar campo de titulo no dialog
Gerencial	Exportar relatório em XLS	Integrar back com front
Gerencial	Exportar relatório em XLS	Criar serviço para exportação de relatórios
Gerencial	Exportar relatório em XLS	Implementar método para exportação de relatório
Gerencial	Notificar via E-mail	Criar serviço de envio de e-mail
Gerencial	Notificar via E-mail	Implementar metodo para envio do e-mail
Gerencial	Notificar via E-mail	Agendar notificação
Gerencial	Notificar via E-mail	Recuperar dados para envio do e-mail
Gerencial	Notificar via E-mail	Consumir API externa
Gerencial	Notificar via E-mail	Criar Layout do E-mail
Gerencial	Notificar via E-mail	Recuperar dados referentes ao e-mail
Gerencial	Notificar via aplicação	Adicionar no painel uma aba de notificações
Gerencial	Notificar via aplicação	Criar view do card de notificação
Gerencial	Notificar via aplicação	Criar serviço de notificação via Firebase no backend
Gerencial	Notificar via aplicação	Recuperar dados do usuário que receberá a notificação
Gerencial	Notificar via aplicação	Recuperar dados da notificação
Gerencial	Notificar via aplicação	Criar serviço para conexão com firebase no frontend
Gerencial	Notificar via aplicação	Atualizar aba notificações com os novos cards de notificação

Apêndice B

Perguntas do Questionário 1

As perguntas do Questionário 1 (pesquisa espontânea) e informações sobre os respondentes são apresentadas nas páginas a seguir.

Questionário sobre Requisitos Não Funcionais (RNFs) em Projetos Scrum

Este questionário faz parte da pesquisa de doutorado de Felipe Barbosa Araújo Ramos do Programa de Pós-Graduação em Ciência da Computação (PPGCC) da Universidade Federal de Campina Grande (UFCG).

O formulário visa coletar informações da indústria sobre o gerenciamento de requisitos não funcionais em projetos Scrum. O participante deve responder as questões com base nas experiências vivenciadas em projetos Scrum que trabalhou ou que está trabalhando atualmente.

Nesta pesquisa é adotada a classificação de RNFs proposta na ISO/IEC 25030. Mais especificamente, o termo REQUISITO NÃO FUNCIONAL refere-se a requisitos de qualidade de stakeholders, também conhecidos como atributos de qualidade externa, que impactam diretamente a experiência do usuário ao utilizar o software. Assim, não são considerados RNFs relacionados à qualidade interna (i.e., relacionados à construção do software em si) como manutenibilidade, flexibilidade, portabilidade, reutilização, legibilidade, testabilidade e compreensão. Mais detalhes em: https://goo.gl/HS72Vc.

Leia atentamente as instruções contidas em cada item deste questionário.

Ao enviar sua resposta neste questionário, você estará de acordo com os termos de consentimento que se encontram em https://goo.gl/mX9YPK.

* Required

1. **Nome** *

2. **E-mail** *

3. **Experiência em projetos de software baseados em Scrum** *
   Mark only one oval.

   - ◯ Menos de 1 ano
   - ◯ Entre 1 e 3 anos
   - ◯ Entre 4 e 6 anos
   - ◯ Entre 7 e 10 anos
   - ◯ Acima de 10 anos

4. **Cargo atual** *
   Check all that apply.

   - [ ] Desenvolvedor
   - [ ] Desenvolvedor (Líder Técnico)
   - [ ] Coach
   - [ ] Scrum Master
   - [ ] Product Owner
   - [ ] Gerente de Projeto
   - [ ] Analista de Qualidade
   - [ ] Other: _____

5. **Em um cenário em que os REQUISITOS NÃO FUNCIONAIS não foram explicitamente elicitados/definidos por PO e stakeholders (i.e., clientes, potenciais usuários, etc.) nas fases iniciais do projeto, quais características de projeto (tanto de negócio quanto técnicas) podem ser levadas em consideração pelo Time de Desenvolvimento para auxiliar a elicitação/definição de REQUISITOS NÃO FUNCIONAIS (qualidade externa)? Importante: rever na introdução do questionário o conceito de RNF considerado no trabalho para responder a pergunta.** *

   _____

   _____

   _____

   _____

   _____

6. **Quais tipos de REQUISITOS NÃO FUNCIONAIS (qualidade externa) são mais comuns/recorrentes em projetos que você trabalhou/trabalha? Importante: rever na introdução do questionário o conceito de RNF considerado no trabalho para responder a pergunta.** *

   _____

   _____

   _____

   _____

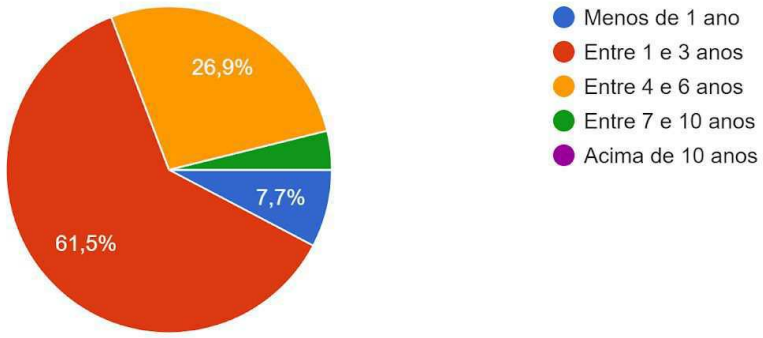
   _____

7. **Quais tipos de REQUISITOS NÃO FUNCIONAIS (qualidade externa) são mais críticos ao sucesso do projeto caso não sejam atendidos até o final do desenvolvimento do software? Importante: rever o conceito de RNF apresentado na introdução do questionário para responder a pergunta.** *

8. **Comentários gerais (opcional)**
   Espaço para fazer comentários com informações extras que não foram abordados no formulário.

Experiência em projetos de software baseados em Scrum

26 respostas



Apêndice C

Perguntas do Questionário 2

As perguntas do Questionário 2 (pesquisa estimulada) e informações sobre os respondentes são apresentadas nas páginas a seguir.

Questionário sobre Requisitos Não Funcionais (RNFs) em Projetos Scrum

Este questionário faz parte da pesquisa de doutorado de Felipe Barbosa Araújo Ramos do Programa de Pós-Graduação em Ciência da Computação (PPGCC) da Universidade Federal de Campina Grande (UFCG).

O formulário visa adquirir informações sobre o gerenciamento de requisitos não funcionais em projetos Scrum. O participante deve responder as questões com base na sua experiência em projetos Scrum que participou anteriormente ou que está participando na atualidade.

Leia atentamente as instruções contidas neste questionário.

Ao enviar sua resposta neste questionário, você estará de acordo com os termos de consentimento que se encontram em https://goo.gl/mX9YPK.

* Required

1. **Nome** *

2. **E-mail** *

3. **Experiência em Projetos Ágeis (Scrum)** *
   Mark only one oval.
   - ◯ Menos de 1 ano
   - ◯ Entre 1 e 3 anos
   - ◯ Entre 4 e 6 anos
   - ◯ Entre 7 e 10 anos
   - ◯ Mais de 10 anos

4. **Cargo atual** *
Mark only one oval.

- ◯ Desenvolvedor
- ◯ Desenvolvedor (Líder Técnico)
- ◯ Coach
- ◯ Scrum Master
- ◯ Product Owner
- ◯ Gerente de Projeto
- ◯ Other: _____

5. **Como os RNFs são registrados nos projetos?** *
Check all that apply.

- ☐ Como itens do Product Backlog (user stories)
- ☐ Como critérios de aceitação de user stories
- ☐ Como itens da Definição de Pronto
- ☐ Não são considerados
- ☐ Não são registrados
- ☐ Outros (especifique abaixo)

6. Caso outras formas de registro tenham sido utilizadas, adicione abaixo. Siga o formato do exemplo, com as informações entre parênteses e separadas por vírgula. Ex.: (forma 01), (forma 02).

_____

_____

_____

_____

_____

7. **Em qual fase dos projetos os RNFs são considerados comumente?** *
Mark only one oval.

- ◯ Montagem inicial do Product Backlog (pré-desenvolvimento)
- ◯ Fase inicial do desenvolvimento (i.e., primeiras Sprints)
- ◯ Continuamente ao longo do projeto
- ◯ Fase final do desenvolvimento (i.e., últimas Sprints)
- ◯ Em nenhuma fase do projeto
- ◯ Outros (especifique abaixo)

8. Caso os RNFs tenham sido considerados em outras fases do projeto, cite abaixo. Siga o formato do exemplo, com as informações entre parênteses e separadas por vírgula. Ex.: (fase 01), (fase 02).

9. **Quais fatores/características dos projetos você considera como chave na definição de requisitos não funcionais?** *

Check all that apply.

- [ ] Categoria do projeto/Plataforma (i.e., Web, Mobile, Desktop, Embedded)
- [ ] Domínio do Projeto (i.e., e-commerce, saúde, bancário, música, TV, etc)
- [ ] Objetivo do Projeto (i.e., Produto ou Protótipo)
- [ ] Arquitetura do Projeto (i.e., camadas, cliente-servidor, etc)
- [ ] Tecnologias utilizadas (i.e., Java, Nodejs, SQL, etc)
- [ ] Outros (especifique abaixo)

10. Caso considere outros fatores/características de projetos relevantes para a definição de RNFs, cite abaixo. Siga o formato do exemplo, com as informações entre parênteses e separadas por vírgula. Ex.: (fator 01), (fator 02).

11. **Quais requisitos NÃO FUNCIONAIS você considera que são mais comuns/recorrentes em projetos que você trabalhou?** *

Check all that apply.

- [ ] Baixo Tempo de Resposta em Requisições de Busca (Web: 2-5 segundos = rápido)
- [ ] Baixo Tempo de Resposta (Web: 1 - 3 segundos = rápido)
- [ ] Baixo Consumo de Memória (Percentage, ou um limite da infra, relacionado a carregamento de dados da base.)
- [ ] Baixo Consumo de Energia (Mobile: associado a consumo dos sensores / radio, Bluetooth, ou CPU)
- [ ] Baixo tempo de resposta para requisições de login (Web: 1 - 2 segundos = rápido)
- [ ] Fazer o lougout da aplicação após tempo máximo de inatividade (por exemplo, tempo máximo = 7 min)
- [ ] Criptografia SSL em login
- [ ] Internacionalização ou localização
- [ ] Suportar x usuários acessando simultaneamente (x é a capacidade máxima de usuários on-line)
- [ ] Outros (especifique abaixo)

12. Informe outros requisitos não funcionais que você considera que são recorrentes em projetos de software em geral. Siga o formato do exemplo, com as informações entre parênteses e separadas por vírgula. Ex.: (requisito 01), (requisito 02).

_____

_____

_____

_____

_____

13. **Quais requisitos FUNCIONAIS você considera que são mais comuns/recorrentes em projetos que você trabalhou?** *

Check all that apply.

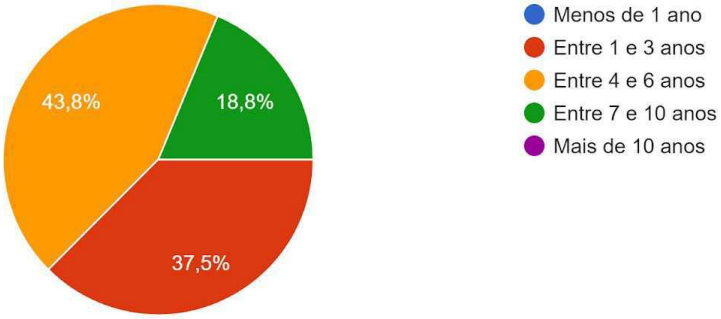
- [ ] Controle de acesso (login)
- [ ] Visualização de home/dashboard
- [ ] Visualização de relatórios
- [ ] Recuperação de Informação
- [ ] Visualização de Status
- [ ] Gerenciamento de Usuários
- [ ] Busca por informação
- [ ] Outros (especifique abaixo)

14. Informe outros requisitos funcionais que você considera que são recorrentes em projetos de software em geral. Siga o formato do exemplo, com as informações entre parênteses e separadas por vírgula. Ex.: (requisito 01), (requisito 02).

15. **Comentários gerais (opcional)**
    Espaço para fazer comentários com informações extras que não foram abordados no formulário.

Experiência em Projetos Ágeis (Scrum)

32 respostas



Apêndice D

Modelo Estruturado de RNFs

O modelo estruturado de RNFs é apresentado nas páginas a seguir.

Tipo	Atributo	Sentença
Performance	response time	In 100% of all cases, the system must have a mean response time of < 3s between event <user login> and event <first page view>. It is assumed that there are 50 concurrent users, during 120 seconds with 30-seconds of ramp-up.
Performance	response time	In 99% of all cases, the system must have a mean response time of <= 1s between event <view dashboard> and event <time to first byte>.
Performance	response time	In 100% of all cases, the system must have a mean response time of < 5s between event <view chart> and event <first page view>.
Performance	response time	In 100% of all cases, the system must have a mean response time of <= 3s between event <search items> and event <display text information of listed items>. It is assumed that there are 200 concurrent users.
Performance	capacity	The system must be able to export a maximum of 10000 items at a time.
Performance	capacity	The system must be able to store a minimum of 10000 user accounts.
Performance	transit delay	The system must provide pagination when the number of returned search results exceeds the number of records to be displayed per page.
Performance	efficiency compliance	The system shall be able to process a value of 1 notification per second or less, and up to and including 100 notifications in 15 seconds or less.
Reliability	availability	The system must notify users and customers when the function they are accessing is unavailable.
Reliability	integrity	The system must validate all data when entered based on the data validation rules. The system must use client-side data validation to improve performance for ordinary execution, i.e., no attacks. Client-side validation is faster since no request is sent to server.
Reliability	integrity	The system must store and display a four digit year for all dates.
Reliability	integrity	To be strict, all methods should always validate all parameters. Do not assume that data, e.g., HTTP parameters, comes from your client code. It could come from an attacker. Server-side validation is necessary, whether there is client-side validation or not. This not only improves security, but also reduces the risk of corrupt data, makes it easier to find bugs and facilitates error handling.
Reliability	integrity	The system must identify data that did not pass edit checks, and present such data to the customer/user for correction. Edit checking will continue until all data submitted is correct.
Reliability	integrity	For entities that are bound by effective and end dates, the system shall verify that there is no other conflicting or overlapping record as defined by business rules.
Reliability	integrity	The system must present data submitted back to the customer for verification.
Reliability	integrity	The system must support business rules for validation of data (formatting).
Reliability	fault tolerance	If the server goes offline during data insertion, the system must display a loading for a maximum of 1 minute. If the backend comes back in the meantime, the data insertion operation must be performed automatically. Otherwise, if the server remains unresponsive after 1 minute, the system must notify the server problem for users.
Reliability	fault tolerance	If the server goes offline during data removal, the system must display a loading for a maximum of 1 minute. If the backend comes back in the meantime, the data removal operation must be performed automatically. Otherwise, if the server remains unresponsive after 1 minute, the system must notify the server problem for users.
Reliability	recoverability	The system must provide a mechanism that allows the system to backup and recover data.
Reliability	recoverability	The system must have the ability for point in time recovery.
Reliability	recoverability	The system must be able to perform incremental and full system backups.

Reliability	recoverability	The system must allow for redundant data stores and application servers to support business continuity.
Security	confidentiality	Use Secure Sockets Layer (SSL) connections to encrypt information that travels between the device and the server.
Security	confidentiality	If users replace their security information, the system must send them a verification token via e-mail before allowing them to log in.
Security	confidentiality	The system must capture the user identification and timestamp when information is exported.
Security	confidentiality	The system must capture the user identification and timestamp when information is created, or deleted.
Security	confidentiality	The system must use parameterized queries, implemented with prepared statements to prohibit SQL injection. The prepared statement execution consists of two stages: prepare and execute. At the prepare stage a statement template is sent to the database server. The server performs a syntax check and initializes server internal resources for later use. During execute stage the client binds parameter values and sends them to the server. The server creates a statement from the statement template and the bound values and executes it.
Security	access control	The system must dynamically display menus, pages, and functionality based on the security access of the user.
Security	access control	The system must automatically disable customer and user accounts for inactivity after a designated period of time. This should be configurable by user type.
Security	access control	The system must provide the capability for an administrator to reset PINs/passwords enabling users and customers to sign-on with a default, whereupon the system will require that the PIN/password be changed before granting access.
Security	access control	The system must provide the capability to create permission policy levels for users who will add, edit, or delete items from a list, open a list, and view items, lists, and pages.
Security	access control	Access key must be generated automatically in the entity's registry.
Security	access control	The system must enable authorized users to export data using standard file formats.
Security	access control	Users shall be forced to change their password the next time they log in if they have not changed it within the length of time established as “password expiration duration.
Security	access control	Users must change the initially assigned login authentication information (password) immediately after the first successful login. The initial password may never be reused.
Security	access control	The system must provide the capability for users see just the information they have permission to access.
Security	access control	The system must provide the capability for an administrator to enable and disable customer and user access.
Security	authentication	The system must support a unique user-id and password of configurable length to access the system for all system users based on user role.
Security	authentication	The system must require a first time user to login to the system using a combination of a unique identifier and a shared secret.
Security	authentication	When the password has changed, the system must automatically notify a customer via e-mail.
Security	authentication	The system must expire passwords within a configurable time, to be determined by the administrator.
Security	authentication	The system must give the customer a designated number of attempts to establish a valid PIN/password, according to business rules, before requesting the customer contact support help desk.
Security	authentication	The system must allow customers and users to establish and change their own PIN/password.

Security	authentication	The system must verify newly registered users by sending them a verification token via e-mail before allowing them to log in.
Security	authentication	The password must be at least 7 characters long and contain at least: 1 Uppercase letter, 1 Lowercase letter, 1 digit, 1 special character.
Security	authentication	The system must terminate a user session after a specified configurable period of inactivity by the user. This timeout should end the session and require authenticated logon to return to session.

Apêndice E

Perguntas do Questionário 3

As perguntas do Questionário 3 para avaliar a utilidade prática da solução proposta são apresentadas nas páginas a seguir.

Avaliação da Ferramenta de Recomendação de Requisitos Não Funcionais (NFRec)

Este questionário faz parte da pesquisa de doutorado de Felipe Barbosa Araújo Ramos do Programa de Pós-Graduação em Ciência da Computação (PPGCC) da Universidade Federal de Campina Grande (UFCG).

Leia atentamente as instruções contidas em cada item deste questionário.

Ao enviar sua resposta, você estará de acordo com os termos de consentimento que se encontram em https://goo.gl/mX9YPK.

*Obrigatório

Informações Gerais

1. **Experiência em projetos de software baseados em Scrum** *
   Marcar apenas uma oval.

   - ◯ Menos de 1 ano
   - ◯ Entre 1 e 3 anos
   - ◯ Entre 4 e 6 anos
   - ◯ Entre 7 e 10 anos
   - ◯ Acima de 10 anos

Utilidade da ferramenta NFRec

2. **Meu trabalho seria mais difícil sem a ferramenta NFRec quanto ao gerenciamento de requisitos não funcionais** *
   Marcar apenas uma oval.

   - ◯ Concordo Totalmente
   - ◯ Concordo
   - ◯ Neutro
   - ◯ Discordo
   - ◯ Discordo Totalmente

3. **O uso da ferramenta NFRec me dá maior controle sobre meu trabalho quanto ao gerenciamento de requisitos não funcionais?** *
Marcar apenas uma oval.

- ◯ Concordo Totalmente
- ◯ Concordo
- ◯ Neutro
- ◯ Discordo
- ◯ Discordo Totalmente

4. **O uso da ferramenta NFRec melhora meu desempenho no trabalho quanto ao gerenciamento de requisitos não funcionais?** *
Marcar apenas uma oval.

- ◯ Concordo Totalmente
- ◯ Concordo
- ◯ Neutro
- ◯ Discordo
- ◯ Discordo Totalmente

5. **A ferramenta NFRec atende às minhas necessidades relacionadas do trabalho.** *
Marcar apenas uma oval.

- ◯ Concordo Totalmente
- ◯ Concordo
- ◯ Neutro
- ◯ Discordo
- ◯ Discordo Totalmente

6. **O uso da ferramenta NFRec me ajuda a economizar tempo?** *
Marcar apenas uma oval.

- ◯ Concordo Totalmente
- ◯ Concordo
- ◯ Neutro
- ◯ Discordo
- ◯ Discordo Totalmente

7. **O uso da ferramenta NFRec me permite finalizar as tarefas mais rapidamente quanto ao gerenciamento de requisitos não funcionais?** *
   Marcar apenas uma oval.

   - ◯ Concordo Totalmente
   - ◯ Concordo
   - ◯ Neutro
   - ◯ Discordo
   - ◯ Discordo Totalmente

8. **A ferramenta NFRec suporta aspectos críticos do meu trabalho?** *
   Marcar apenas uma oval.

   - ◯ Concordo Totalmente
   - ◯ Concordo
   - ◯ Neutro
   - ◯ Discordo
   - ◯ Discordo Totalmente

9. **A ferramenta NFRec me permite finalizar mais tarefas relacionadas ao gerenciamento de requisitos não funcionais do que seria possível sem ela?** *
   Marcar apenas uma oval.

   - ◯ Concordo Totalmente
   - ◯ Concordo
   - ◯ Neutro
   - ◯ Discordo
   - ◯ Discordo Totalmente

10. **A ferramenta NFRec reduz o tempo que gasto em atividades improdutivas?** *
    Marcar apenas uma oval.

    - ◯ Concordo Totalmente
    - ◯ Concordo
    - ◯ Neutro
    - ◯ Discordo
    - ◯ Discordo Totalmente

11. **A ferramenta NFRec melhora minha efetividade no trabalho quanto ao gerenciamento de requisitos não funcionais?** *
Marcar apenas uma oval.

- ◯ Concordo Totalmente
- ◯ Concordo
- ◯ Neutro
- ◯ Discordo
- ◯ Discordo Totalmente

12. **A ferramenta NFRec melhora a qualidade do meu trabalho?** *
Marcar apenas uma oval.

- ◯ Concordo Totalmente
- ◯ Concordo
- ◯ Neutro
- ◯ Discordo
- ◯ Discordo Totalmente

13. **O uso da ferramenta NFRec melhora a minha produtividade?** *
Marcar apenas uma oval.

- ◯ Concordo Totalmente
- ◯ Concordo
- ◯ Neutro
- ◯ Discordo
- ◯ Discordo Totalmente

14. **O uso da ferramenta NFRec faz meu trabalho ficar mais fácil quanto ao gerenciamento de requisitos não funcionais?** *
Marcar apenas uma oval.

- ◯ Concordo Totalmente
- ◯ Concordo
- ◯ Neutro
- ◯ Discordo
- ◯ Discordo Totalmente

15. **No geral, acho o uso da ferramenta NFRec útil para meu trabalho?** *
Marcar apenas uma oval.

- ◯ Concordo Totalmente
- ◯ Concordo
- ◯ Neutro
- ◯ Discordo
- ◯ Discordo Totalmente

Facilidade de Uso da ferramenta NFRec

16. **Eu geralmente fico confuso ao utilizar a ferramenta NFRec** *
Marcar apenas uma oval.

- ◯ Concordo Totalmente
- ◯ Concordo
- ◯ Neutro
- ◯ Discordo
- ◯ Discordo Totalmente

17. **Frequentemente cometo erros utilizando a ferramenta NFRec** *
Marcar apenas uma oval.

- ◯ Concordo Totalmente
- ◯ Concordo
- ◯ Neutro
- ◯ Discordo
- ◯ Discordo Totalmente

18. **Interagir com a ferramenta NFRec é geralmente frustrante** *
Marcar apenas uma oval.

- ◯ Concordo Totalmente
- ◯ Concordo
- ◯ Neutro
- ◯ Discordo
- ◯ Discordo Totalmente

19. **Devo sempre consultar alguém quando uso a ferramenta NFRec** *
Marcar apenas uma oval.

- ◯ Concordo Totalmente
- ◯ Concordo
- ◯ Neutro
- ◯ Discordo
- ◯ Discordo Totalmente

20. **O uso da ferramenta NFRec requer muito esforço mental** *
Marcar apenas uma oval.

- ◯ Concordo Totalmente
- ◯ Concordo
- ◯ Neutro
- ◯ Discordo
- ◯ Discordo Totalmente

21. **Eu considero fácil me recuperar de erros encontrados ao utilizar a ferramenta NFRec** *
Marcar apenas uma oval.

- ◯ Concordo Totalmente
- ◯ Concordo
- ◯ Neutro
- ◯ Discordo
- ◯ Discordo Totalmente

22. **A ferramenta NFRec é rígida e inflexível de se interagir** *
Marcar apenas uma oval.

- ◯ Concordo Totalmente
- ◯ Concordo
- ◯ Neutro
- ◯ Discordo
- ◯ Discordo Totalmente

23. **Eu acho fácil fazer a ferramenta NFRec realizar o que eu quero que ele faça** *
Marcar apenas uma oval.

- ◯ Concordo Totalmente
- ◯ Concordo
- ◯ Neutro
- ◯ Discordo
- ◯ Discordo Totalmente

24. **A ferramenta NFRec geralmente se comporta de maneira inesperada** *
Marcar apenas uma oval.

- ◯ Concordo Totalmente
- ◯ Concordo
- ◯ Neutro
- ◯ Discordo
- ◯ Discordo Totalmente

25. **Eu acho muito complicado utilizar a ferramenta NFRec** *
Marcar apenas uma oval.

- ◯ Concordo Totalmente
- ◯ Concordo
- ◯ Neutro
- ◯ Discordo
- ◯ Discordo Totalmente

26. **Minha interação com a ferramenta NFRec é fácil de entender** *
Marcar apenas uma oval.

- ◯ Concordo Totalmente
- ◯ Concordo
- ◯ Neutro
- ◯ Discordo
- ◯ Discordo Totalmente

27. **É fácil para mim lembrar como executar tarefas utilizando a ferramenta NFRec** *
Marcar apenas uma oval.

- ◯ Concordo Totalmente
- ◯ Concordo
- ◯ Neutro
- ◯ Discordo
- ◯ Discordo Totalmente

28. **A ferramenta NFRec fornece orientação útil na execução de tarefas.** *
Marcar apenas uma oval.

- ◯ Concordo Totalmente
- ◯ Concordo
- ◯ Neutro
- ◯ Discordo
- ◯ Discordo Totalmente

29. **No geral, considero o uso da ferramenta NFRec fácil** *
Marcar apenas uma oval.

- ◯ Concordo Totalmente
- ◯ Concordo
- ◯ Neutro
- ◯ Discordo
- ◯ Discordo Totalmente

Avaliação da Qualidade das Recomendações

30. **As recomendações geradas pela ferramenta NFRec são relevantes mesmo nas fases iniciais do projeto (ou seja, Sprints 1 ou 2)?** *
Marcar apenas uma oval.

- ◯ Concordo Totalmente
- ◯ Concordo
- ◯ Neutro
- ◯ Discordo
- ◯ Discordo Totalmente

31. **As recomendações geradas pela ferramenta NFRec podem ser úteis na tomada de decisão quanto à arquitetura do software em desenvolvimento?** *
Marcar apenas uma oval.

- ◯ Concordo Totalmente
- ◯ Concordo
- ◯ Neutro
- ◯ Discordo
- ◯ Discordo Totalmente

32. **As recomendações geradas pela ferramenta NFRec podem contribuir para uma melhor documentação de NFRs no projeto?** *
Marcar apenas uma oval.

- ◯ Concordo Totalmente
- ◯ Concordo
- ◯ Neutro
- ◯ Discordo
- ◯ Discordo Totalmente

33. **As recomendações geradas pela ferramenta NFRec podem contribuir para uma redução de retrabalho da equipe em fases finais do projeto?** *
Marcar apenas uma oval.

- ◯ Concordo Totalmente
- ◯ Concordo
- ◯ Neutro
- ◯ Discordo
- ◯ Discordo Totalmente

34. **Comentários sobre a ferramenta de recomendação de requisitos não funcionais NFRec** *

_____

_____

_____

_____

_____