UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ

DANIEL CARLOS KUHN

ANÁLISE E PROJETO DE UMA FERRAMENTA PARA ESPECIFICAÇÃO DE REQUISITOS E MODELAGEM DE SOFTWARE

DOIS VIZINHOS

2021

DANIEL CARLOS KUHN

ANÁLISE E PROJETO DE UMA FERRAMENTA PARA ESPECIFICAÇÃO DE REQUISITOS E MODELAGEM DE SOFTWARE

ANALYSIS AND DESIGN OF A TOOL FOR SOFTWARE SPECIFICATION AND MODELING

Trabalho de Conclusão de Curso apresentado como requisito parcial à obtenção do título de Bacharel em Engenharia de Software, da Universidade Tecnológica Federal do Paraná.

Orientador: Profa. Dra. Alinne C. Corrêa de Souza

Coorientador: MSc. Diógenes Dias

DOIS VIZINHOS

2021

Ministério da Educação
UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ
COORD. DO CURSO DE ENG. DE SOFTWARE-DV

TERMO DE APROVAÇÃO

TRABALHO DE CONCLUSÃO DE CURSO - TCC

ANÁLISE E PROJETO DE UMA FERRAMENTA PARA ESPECIFICAÇÃO DE REQUISITOS E MODELAGEM DE SOFTWARE

Por

Daniel Carlos Kuhn

Monografia apresentada às 20 horas 30 min do dia 25 de agosto de 2021 como requisito parcial, para conclusão do Curso de Bacharelado em Engenharia de Software da Universidade Tecnológica Federal do Paraná, Câmpus Dois Vizinhos. O candidato foi arguido pela Banca Examinadora composta pelos professores abaixo assinados. Após deliberação e conferidas, bem como achadas conforme, as alterações indicadas pela Banca Examinadora, o trabalho de conclusão de curso foi considerado APROVADO.

Banca examinadora:

Prof. Gustavo Jansen de Souza Santos	Membro
Profa. Marisangela Pacheco Brittes	Membro
Profa. Alinne Cristinne Corrêa Souza	Orientadora
Prof. Francisco Carlos Monteiro Souza	Professor responsável TCCII

Documento assinado eletronicamente por (Document electronically signed by) **ALINNE CRISTINNE CORREA SOUZA**, **PROFESSOR DO MAGISTERIO SUPERIOR**, em (at) 31/08/2021, às 14:40, conforme horário oficial de Brasília (according to official Brasilia-Brazil time), com fundamento no (with legal based on) art. 4º, § 3º, do Decreto nº 10.543, de 13 de novembro de 2020.

Documento assinado eletronicamente por (Document electronically signed by) **GUSTAVO JANSEN DE SOUZA SANTOS**, **PROFESSOR DO MAGISTERIO SUPERIOR**, em (at) 31/08/2021, às 14:44, conforme horário oficial de Brasília (according to official Brasilia-Brazil time), com fundamento no (with legal based on) art. 4º, § 3º, do Decreto nº 10.543, de 13 de novembro de 2020.

Documento assinado eletronicamente por (Document electronically signed by) **MARISANGELA PACHECO BRITTES**, **PROFESSOR DO MAGISTERIO SUPERIOR**, em (at) 31/08/2021, às 17:03, conforme horário oficial de Brasília (according to official Brasilia-Brazil time), com fundamento no (with legal based on) art. 4º, § 3º, do Decreto nº 10.543, de 13 de novembro de 2020.

Documento assinado eletronicamente por (Document electronically signed by) **Daniel Carlos Kuhn**, **Usuário Externo**, em (at) 31/08/2021, às 18:15, conforme horário oficial de Brasília (according to official Brasilia-Brazil time), com fundamento no (with legal based on) art. 4º, § 3º, do Decreto nº 10.543, de 13 de novembro de 2020.

Documento assinado eletronicamente por (Document electronically signed by) **FRANCISCO CARLOS MONTEIRO SOUZA**, **PROFESSOR DO MAGISTERIO SUPERIOR**, em (at) 31/08/2021, às 18:19, conforme horário oficial de Brasília (according to official Brasilia-Brazil time), com fundamento no (with legal based on) art. 4º, § 3º, do Decreto nº 10.543, de 13 de novembro de 2020.

A autenticidade deste documento pode ser conferida no site (The authenticity of this document can be checked on the website) https://sei.utfpr.edu.br/sei/controlador_externo.php?acao=documento_conferir&id_orgao_acesso_externo=0, informando o código verificador (informing the verification code) **2217806** e o código CRC (and the CRC code) **1CC01B90**.

Referência: Processo nº 23064.037302/2021-27 SEI nº 2217806

AGRADECIMENTOS

Agradeço à minha querida orientadora, a qual, sempre atenciosa, esteve muito presente desde a idealização do trabalho.

Agradeço também a meus amigos e familiares, os quais me deram o apoio necessário a todo momento.

“Somos arquitetos do nosso próprio destino.”

EINSTEIN, Albert.

RESUMO

Com os avanços tecnológicos, o uso de sistemas baseados em computação está cada vez mais presente nas atividades da sociedade, aumentando a busca por softwares de qualidade, confiáveis e com custo reduzido. Nesse contexto, à medida que os sistemas requeridos crescem em complexidade, é necessária a utilização de ferramentas que auxiliem e integrem diferentes atividades do processo de desenvolvimento de software. No entanto, uma ferramenta que contemple a especificação e modelagem de software para o contexto ágil continua sendo um desafio para *stakeholders*, uma vez que realizam as suas respectivas tarefas em diferentes ferramentas, podendo acarretar incertezas e inconsistências entre os artefatos gerados. Diante disso, o objetivo desta monografia é análise e projeto de uma ferramenta web chamada *Requirements spEcification and system MOdeling* para auxiliar a especificação de requisitos e modelagem de software. A ferramenta é composta por três módulos: (i) Gerenciamento de Projetos, (ii) Especificação de Requisitos Ágil, e (iii) Modelagem de Software. Além disso, também foi desenvolvida uma arquitetura para garantir o design e a estruturação da ferramenta. Foi realizado um estudo de caso para avaliar a eficácia da arquitetura proposta para facilitar o desenvolvimento da ferramenta REMO. Portanto, os resultados alcançados mostraram que a ferramenta pode auxiliar analistas nas atividades de especificação dos requisitos e modelagem de software para o contexto ágil.

Palavras-chave: Especificação de software, Modelagem de software, Arquitetura de Software

ABSTRACT

With the technological advances, the use of systems based on data processing are more and more included in our society, increasing the search for high quality, trustworthy and low-cost software. In this context, as the required systems grow in complexity, it is necessary to use tools that assist and integrate different activities in the software development process. However, a tool that includes the software specification and modeling activities for the agile context remains a challenge for stakeholders, since they perform their tasks in different tools, which may lead to uncertainties and inconsistencies between the generated artifacts. Therefore, the objective of this monograph is the analysis and design of a web tool called Requirements spEcification and system MOdeling to assist the requirements specification and software modeling. The tool is composed of three modules: (i) Project management; (ii) agile requirements specification; and (iii) software modeling. Also, an architecture was developed to ensure the design and the structuring of the tool. A case study was conducted to evaluate the effectiveness of the proposed architecture to facilitate the development of REMO tool. The results showed that the tool can assist the analysts during the requirement specification and software modeling activities in agile context.

Keywords: Software especification, Software Modeling, Software Architecture

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE SIGLAS

ACM	Association for Computing Machinery
AR	Arquitetura de Referência
AS	Arquitetura de Software
ATL	ATLAS Transformation Language
CE	Critério de Exclusão
CI	Critério de Inclusão
CMMI	Capability Maturity Model Integration
CNL	Controlled Natural Language
DSL	Domain-Specific Languages
ES	Engenharia de Software
ER	Engenharia de Requisitos
ERA	Especificação de Requisitos Ágil
EU	Estória de Usuário
FDD	Feature-Driven Development
GPR	Gerenciamento de Projetos
LSD	Lean Software Development
MS	Mapeamento Sistemático
MOS	Modelagem de Software
NPL	Natural Language Processing
OCL	Object Constraint Language
OpenUP	Open Unified Process
IEEE	Institute of Electric and Electronic Engineers
QP	Questão de Pesquisa
SMS	Systematic Mapping Study
SOA	Service-Oriented Archtecture
SysML	Systems Modeling Language
REMO	Requirements spEcification and system MOdeling
RF	Requisito Funcional
RNF	Requisito Não-Funcional
RUP	Rational Unified Process
UFMG	Universidade Federal de Minas Gerais
UML	Unified Modeling Language
USP	Universidade de São Paulo
UTFPR	Universidade Tecnológica Federal do Paraná
XML	Extensible Markup Language
XP	Extreme Programing

SUMÁRIO

1 INTRODUÇÃO

Com os avanços tecnológicos, o uso de sistemas baseados em computação está cada vez mais presente nas atividades da sociedade, aumentando a busca por softwares de qualidade, confiáveis e com custo reduzido. Nesse contexto, à medida que os sistemas requeridos crescem em complexidade, é necessária a utilização de um processo de desenvolvimento de software que atenda as demandas do mercado.

O processo de desenvolvimento consiste em um conjunto de atividades que, apoiadas pela Engenharia de Software (ES), levam à produção de um produto de software. Essas atividades vão desde a identificação das necessidades do software a ser desenvolvido, até a fase de desenvolvimento, teste e entrega ao usuário final (SOMMERVILLE, 2016).

Esses processos são definidos, muitas vezes, especificamente para cada empresa, com base em modelos de processos, podendo ser tradicionais ou ágeis. No entanto, os modelos tradicionais têm tido baixa utilização nos últimos anos, ao contrário dos modelos ágeis, os quais estão presentes na grande maioria das empresas de tecnologia (RIVAS; SOUZA, 2014).

Com a finalidade de garantir essa flexibilidade no processo de desenvolvimento de software, os modelos de processos ágeis começaram a ganhar notoriedade. Essa flexibilidade é apoiada por características que fornecem suporte às atividades ágeis, como entregas parciais, melhoria contínua do produto em desenvolvimento, maior cooperação entre equipe de desenvolvimento e cliente, flexibilidade de escopo do projeto, além da criação de valor progressivo conforme as necessidades do cliente (BECK et al., 2001). Além disso, a necessidade de utilização processos ágeis vêm se tornando cada vez mais comum na indústria, devido à realização de atividades paralelas e entregas frequentes aos clientes.

Independente da metodologia utilizada para auxiliar o processo de desenvolvimento de software, uma das atividades fundamentais é a especificação do software. Essa atividade consiste em identificar as necessidades apresentadas pelos usuários ou clientes, também conhecidas como requisitos, que devem ser contempladas pelo sistema a ser desenvolvido.

O processo de especificação é iniciado com a descoberta do objetivo do sistema por meio da identificação dos *stakeholders* e de suas necessidades, e documentação dessas informações de forma que sejam concisas para análise, comunicação, e posterior implementação (NUSEIBEH; EASTERBOOK, 2000).

Em um sistema de software, os requisitos especificados consistem em descrições dos serviços ofertados pelo sistema assim como suas restrições operacionais (SOMMERVILLE, 2016). No entanto, para que esses requisitos escritos em linguagem natural sejam compreendidos pela equipe de desenvolvimento é interessante realizar a modelagem do software, a qual visa transformar as informações sobre o sistema em notações gráficas de fácil compreensão. Nesse contexto, a modelagem auxilia a idealização do software por meio de notações gráficas. Essas notações são chamadas de diagramas e permitem compreender desde a estrutura até o comportamento do sistema a ser desenvolvido (BOOCH; RUMBAUGH; JACOBSON, 2006).

Além das atividades de especificação e modelagem de software, um fator crítico está relacionado ao gerenciamento de requisitos, uma vez que o entendimento dos *stakeholders* a respeito do problema está em constante mudança. Segundo Sommerville (2016), o gerenciamento de requisitos é o processo de análise, decisão e compreensão com relação às mudanças nos requisitos do sistema. É importante destacar que todas as modificações devem ser atualizadas em todos os artefatos, para que não sejam geradas possíveis inconsistências.

Nesse contexto, a ausência de um único sistema que contemple as atividades de especificação e modelagem do software, leva os *stakeholders* a realizar as suas respectivas tarefas em diferentes ferramentas. Isso pode acarretar em inconsistências entre os artefatos de um projeto, como por exemplo, a inclusão de uma nova estória de usuário no documento de requisitos, sem ter sido atualizado um diagrama de caso de uso. Além de evitar problemas como este, utilizar uma única ferramenta para a manutenção de todo o projeto, pode reduzir o esforço e tempo empregado por um *stakeholder* na realização destas atividades.

1.1 Objetivos

Em torno da necessidade de uma única ferramenta que contemple todas as atividades anteriormente citadas, o objetivo deste trabalho consiste em realizar a análise e o projeto de uma ferramenta Web chamada ***R****equirements spEcification and Software*

***MO**deling (REMO)* para o contexto ágil. Essa ferramenta visa auxiliar as atividades de elicitação, especificação, validação, gerenciamento de requisitos, bem como modelagem de software por meio da geração automática de diagramas utilizando a Linguagem de Modelagem Unificada (do inglês, *Unified Modeling Language - UML*) a partir dos requisitos especificados.

Dentre os diferenciais relacionados à análise e ao projeto da ferramenta, é possível destacar a utilização da Arquitetura Orientada a Serviços (do inglês, *Service-Oriented Architecture - SOA*). Com a utilização desse estilo arquitetural, futuramente, a ferramenta poderá ser expandida para a plataforma *Mobile*, assim como a possibilidade de geração de novos diagramas, por exemplo.

Para alcançar o objetivo geral os seguintes objetivos específicos foram delineados:

- investigar e analisar ferramentas existentes para o contexto acadêmico para auxiliar a especificação e gerenciamento de requisitos, bem como a modelagem de software;
- realizar a especificação da ferramenta REMO;
- criar uma Arquitetura Orientada a Serviços;
- avaliar por meio de estudo de caso o projeto da ferramenta REMO.

1.2 Metodologia de Pesquisa

Para alcançar os objetivos definidos, foram realizadas seis atividades, as quais são apresentadas na Figura 1.

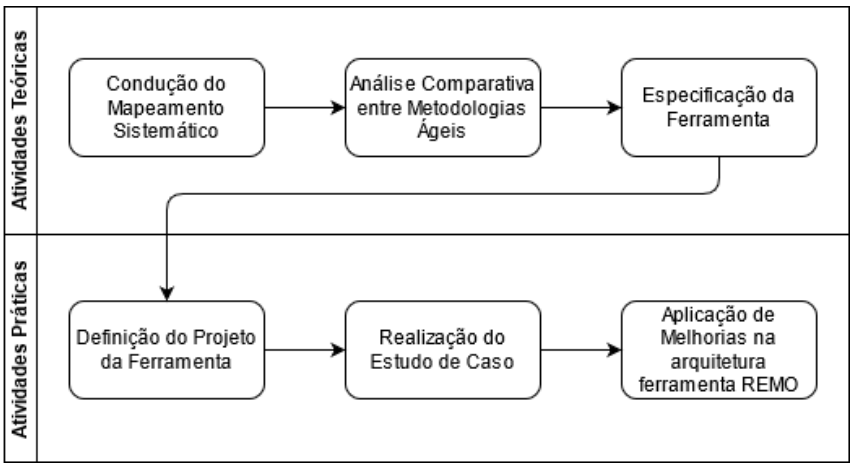

Figura 1 - Visão geral do processo metodológico
Fonte: Autoria própria.

1. **Condução do Mapeamento Sistemático**: esta atividade consistiu na identificação de evidências sobre ferramentas que automatizem as atividades de elicitação, especificação, validação, gerenciamento de requisitos e modelagem de software;
2. **Análise Comparativa entre Metodologias Ágeis**: nesta atividade foram analisadas diferentes metodologias ágeis a fim de extrair as principais destas características para auxiliar a especificação das funcionalidades da ferramenta;
3. **Especificação da Ferramenta**: nesta atividade foi realizado o levantamento dos requisitos funcionais e não funcionais da ferramenta, com base nas características extraídas na Atividade 2, bem como a criação dos diagramas de caso de uso, atividades e classes;
4. **Definição do Projeto da Ferramenta**: durante esta atividade foi criada a arquitetura da ferramenta utilizando SOA;
5. **Realização do Estudo de Caso**: nesta atividade a especificação e a arquitetura da ferramenta foram avaliadas por meio de um estudo de caso com especialistas em arquitetura a fim de verificar a eficácia da arquitetura proposta para a ferramenta REMO;
6. **Aplicação de Melhorias na arquitetura ferramenta REMO**: para aproximar a arquitetura criada ao máximo de uma versão definitiva para o futuro desenvolvimento da ferramenta REMO, as melhorias extraídas após a execução do estudo de caso foram realizadas originando uma versão final da arquitetura.

1.3 Estrutura da Monografia

Esta monografia está organizada em cinco capítulos. No primeiro capítulo, estão apresentados a contextualização, a motivação, os objetivos da presente monografia, bem como o processo metodológico conduzido.

No Capítulo 2 são definidos e apresentados os principais conceitos com relação aos processos de desenvolvimento de software, especificação de requisitos, modelagem e arquitetura de software, os quais irão auxiliar na compreensão das informações dispostas neste trabalho.

No Capítulo 3 é apresentado um Mapeamento Sistemático (MS) visando identificar ferramentas que vêm sido utilizadas para especificação, gerenciamento de requisitos e

geração de diagramas voltados para o contexto de modelos de processos tradicionais e ágeis.

No Capítulo 4 é apresentado o processo de análise e projeto da ferramenta para auxiliar a condução das atividades de elicitação, especificação, validação gerenciamento de requisitos e modelagem de software.

No Capítulo 5 são detalhados o planejamento e a execução do estudo de caso que visam avaliar a arquitetura proposta, bem como a discussão dos resultados obtidos. Por fim, no Capítulo 6 são descritas as conclusões do trabalho, bem como as principais contribuições, limitações e trabalhos futuros que poderão ser conduzidos.

2 ASPECTOS CONCEITUAIS

Ao longo deste capítulo, são apresentados os aspectos conceituais que permeiam o tema abordado a fim de introduzir e contextualizar o assunto deste trabalho. Este Capítulo está organizado da seguinte forma: Na Seção 2.1 é apresentada uma visão sobre processos de desenvolvimento de software. Na Seção 2.2, são abordadas as principais metodologias ágeis. Na Seção 2.3 é fornecida uma visão geral sobre a especificação de software e os conceitos sobre requisitos funcionais e não funcionais. Os conceitos de modelagem de software são apresentados na Seção 2.4, junto aos seus principais diagramas. Na Seção 2.5 é realizada a contextualização sobre arquitetura de software. Por fim, na Seção 2.6 são apresentadas as considerações finais do Capítulo.

2.1 Processo de Software

Independente do tipo de produto de software a ser desenvolvido, para que seja garantida a sua qualidade, é necessário que conceitos e técnicas utilizadas dentro da ES sejam aplicadas. Essa qualidade não é ditada somente por linguagens, técnicas ou habilidades de programação, mas também por meio de outros fatores, como por exemplo, o processo de desenvolvimento de software utilizado pela equipe envolvida em um projeto (PRESSMAN, 2016).

O processo de desenvolvimento de software consiste em um conjunto estruturado de atividades exigidas para desenvolver um produto de software (SOMMERVILLE, 2016). Neste contexto, um processo bem definido, com base nas necessidades do projeto, pode definir o sucesso ou decadência de uma ferramenta de software.

Apesar de existir diferentes processos de software, todos são compostos por quatro atividades fundamentais (SOMMERVILLE, 2016):

1. **especificação de software:** consiste na especificação das funcionalidades do software e suas restrições;

2. **projetos e implementação de software:** consiste na concepção e desenvolvimento do software para atender as especificações;

3. **validação de software:** visa validar o software por meio de testes a fim de garantir que o software atenda as necessidades do cliente;

4. **evolução de software:** consiste na evolução e manutenção do software para atender as mudanças solicitadas pelo cliente.

Essas atividades, muitas vezes com nomenclaturas diferentes, compõem o processo de desenvolvimento de software. Embora não exista um processo ideal de software, geralmente um processo é utilizado com base no cenário de cada organização, levando em consideração por exemplo, o contexto ao qual o processo se aplica e as suas necessidades específicas. Geralmente, um processo não é criado por completo pela organização, uma vez que a empresa normalmente utiliza como base modelos de processos existentes que tiveram resultados positivos com relação à sua aplicação.

Os modelos de processos de desenvolvimento de software não devem ser tratados como processos definitivos a serem aplicados, mas sim como abstrações que podem ser moldadas e aplicadas em diferentes abordagens do desenvolvimento de software (SOMMERVILLE, 2016). Esses modelos são divididos entre os modelos tradicionais e os modelos ágeis, os quais podem ser tanto sequenciais quanto iterativos.

Neste contexto, como o foco do trabalho é realizar a análise e o projeto de uma ferramenta voltada para o contexto ágil, na próxima seção serão abordadas as metodologias ágeis.

2.2 Metodologias Ágeis

As metodologias ágeis foram desenvolvidas a partir da necessidade da agilidade de interação entre as atividades existentes no processo de desenvolvimento de software. Com as metodologias ágeis espera-se atender os diferentes tipos de mudanças de modo rápido e adaptativo. Por meio do Manifesto Ágil (do inglês, *Agile Manifesto*), foram definidos princípios comuns os quais são seguidos por todos os modelos ágeis existentes (BECK et al., 2001).

O Manifesto Ágil pode ser definido como um guia do desenvolvimento ágil, no qual estão descritos os valores e princípios que permeiam uma metodologia ágil. As ideias propostas pelo manifesto visam gerar uma maior qualidade na entrega, colocando uma

prioridade em itens comparados a outros. O Beck et al. (2001) define os seguintes valores e 12 princípios:

- **Valores:**
  - Indivíduos e interações mais que processos e ferramentas;
  - Software em funcionamento mais que documentação abrangente;
  - Colaboração do cliente mais que negociação de contratos;
  - Respostas a mudanças mais que seguir um plano.

- **Princípios:**
  - Prioridade na satisfação do cliente com entregas contínuas e software de valor agregado.
  - Mudanças nos requisitos são bem vindas.
  - Entregas frequentes de softwares funcionais com o menor tempo possível a cada entrega.
  - Pessoas de negócios e desenvolvedores devem trabalhar em conjunto.
  - Construa projetos com indivíduos motivados.
  - Transmita informações face a face para e entre uma equipe de desenvolvimento.
  - Software funcional é a primeira medida de progresso.
  - Processos ágeis promovem desenvolvimento sustentável, mantendo um ritmo constante entre os envolvidos do projeto.
  - Sempre atento à excelência técnica e designe promovendo um aumento na agilidade.
  - Simplicidade é essencial.
  - Equipes auto-organizáveis geram melhores resultados.
  - Reflexões regulares e ajustes nos processos para se tornar mais eficaz.

Diferentes metodologias ágeis compartilham da filosofia do Manifesto Ágil, que apesar de parecidas, apresentam diferenças no ciclo de desenvolvimento, ou artefatos a serem usados. Dentre as principais metodologias ágeis, é possível citar:

- *Scrum*;

- Programação Extrema (do inglês, *Extreme Programing - XP*);
- Desenvolvimento Dirigido por Funcionalidades (do inglês, *Feature-Driven Development - FDD*);
- Desenvolvimento de Software Enxuto (do inglês, *Lean Software Development - LSD*);
- Processo Unificado Aberto (do inglês, *Open Unified Process - OpenUP*).

2.2.1 Scrum

O *Scrum* é um dos *frameworks* mais utilizados no processo de desenvolvimento de produtos considerados complexos. Com o *Scrum* é possível empregar técnicas e processos durante a construção do produto. Com o uso deste *framework*, é possível manter uma certa flexibilidade e controle sobre as atividades relacionadas ao processo (SCHWABER, 1997). A Figura 2 representa o processo de desenvolvimento da metodologia ágil *Scrum*.

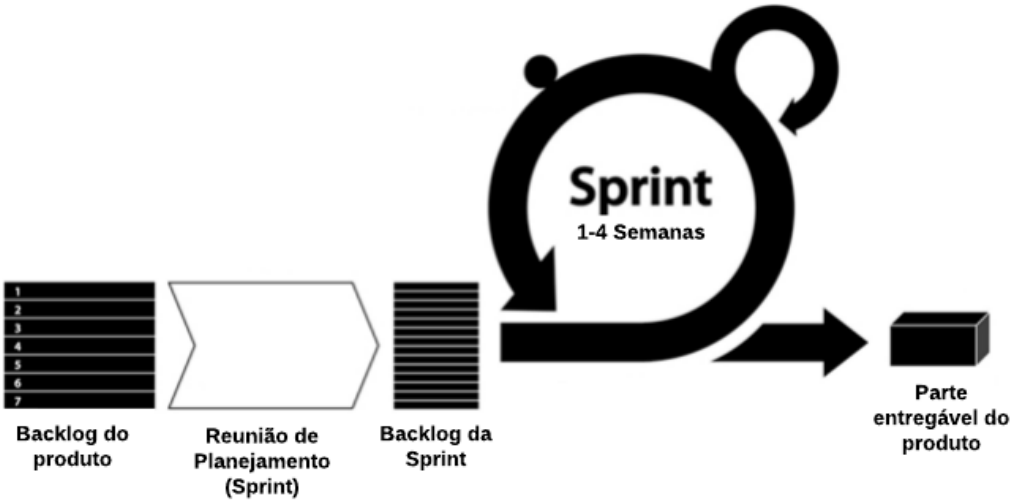

Figura 2 - Processo da metodologia ágil *Scrum*.
Fonte: Adaptado do modelo de Lin et al. (2011).

O *backlog* do produto contém todas as funcionalidade levantadas para um produto de software, as quais serão desenvolvidas durante as *Sprints* definidas para o projeto. Antes de toda *Sprint* é realizada a reunião de planejamento de *Sprint*, na qual são definidas as tarefas a serem realizadas para satisfazer uma porção do *Backlog* do produto, constituindo então o *Backlog* da *Sprint*.

Com isso, durante a execução da *Sprint*, a qual pode durar de uma a quatro semanas, variando de acordo a necessidade de cada projeto, as tarefas são realizadas e

discutidas nas reuniões diárias (do inglês, *daily meeting*), as quais ocorrem durante todo o andamento da *Sprint*. Assim, ao final de cada *Sprint*, uma parte funcional do produto é finalizada.

2.2.2 XP

O XP permite que sejam realizadas e disponibilizadas *releases* de um software frequentemente dentro de um curto ciclo de desenvolvimento (BECK K.; ANDRES, 2004). A Figura 3 apresenta o processo da metodologia XP.

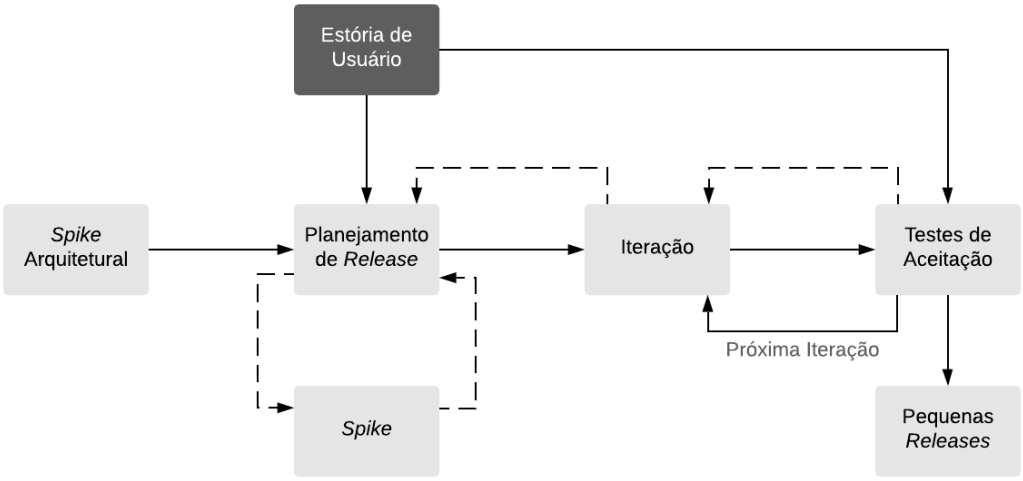

Figura 3 - Processo da metodologia ágil *XP*.
Fonte: Adaptado do modelo de Rech (2009).

Para entender o processo da metodologia ágil XP, primeiramente é necessário compreender seu contexto. Segundo Philippus (2009), o chamado *"spike"* é um experimento que permite que os desenvolvedores aprendam, por meio deste teste de implementação, o suficiente a respeito dos elementos em uma estória de usuário. Desse modo o *Spike Architetural*, também conhecida como *Sprint* 0, é um protótipo da arquitetura do projeto, o qual permite serem realizados estudos de viabilidade relacionados ao projeto a ser implementado.

Com isso, após ter sido finalizado o *Spike Arquitetural*, o Planejamento da *Sprint* é realizado em conjunto com a *Spike*, com base nas estórias de usuário levantadas. Após finalizado o planejamento é aplicada uma iteração e, em seguida, os testes de aceitação. Caso as implementações passem pelo teste de aceitação, uma *release* é entregue. É interessante destacar que alguns processos são cíclicos, permitindo que seja retornada para

a fase anterior, caso seja necessário.

2.2.3 FDD

Com o propósito de melhorar as atividades relacionadas à comunicação, complexidade e qualidade dos produtos, em 1997 foi proposto um modelo de processo guiado a funcionalidades. Este modelo auxilia na redução da complexidade das atividades, por meio da decomposição de um grande problema em pequenos pontos a serem tratados, de forma na qual o processo seja guiado por essas divisões do escopo principal (GOYAL, 2008). Além disso, a qualidade do software é analisada de vários pontos de vistas, tanto do ponto de vista de um usuário quanto do ponto de vista de um desenvolvedor. A representação desse modelo pode ser vista na Figura 4.

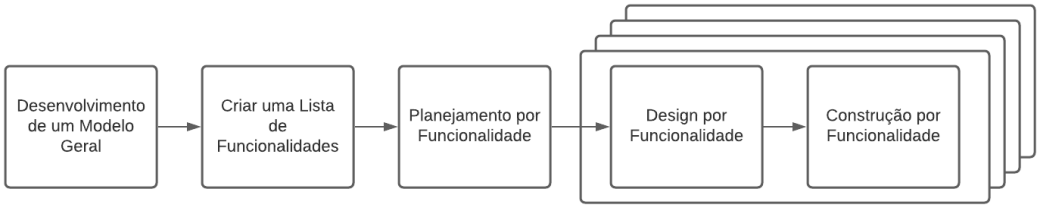

Figura 4 - Processo da metodologia ágil *FDD*.
Fonte: Adaptado de Fernandes et al. (2011).

Primeiramente é criado um modelo geral da proposta de desenvolvimento, no qual é definido o escopo principal do produto. Na fase seguinte, é especificada uma lista de funcionalidades a serem desenvolvidas durante o andamento do produto. Após isso, cada funcionalidade passa por um planejamento, com intenção de definir os pontos criticos, como prioridade e outras informações. Em seguida é realizado o design por funcionalidade, especificando diagramas de classes a serem utilizados na etapa seguinte, na qual é realizada a construção por funcionalidades.

2.2.4 LSD

Segundo Ebert, Abrahamsson e Oza (2012), o Desenvolvimento de Software Enxuto (do inglês, *Lean Software Development - LSD*) é um paradigma de desenvolvimento de produto com foco em criar valor para o cliente, eliminar desperdícios, no emponderamento de pessoas e na melhoria contínua. Essa metodologia segue um ciclo contínuo, o qual pode ser iniciado pela criação de valor para o cliente, visando a entrega rápida seguindo

os princípios mencionados anteriormente, priorizando-os durante todo o andamento do processo. A Figura 5 apresenta o processo LSD.

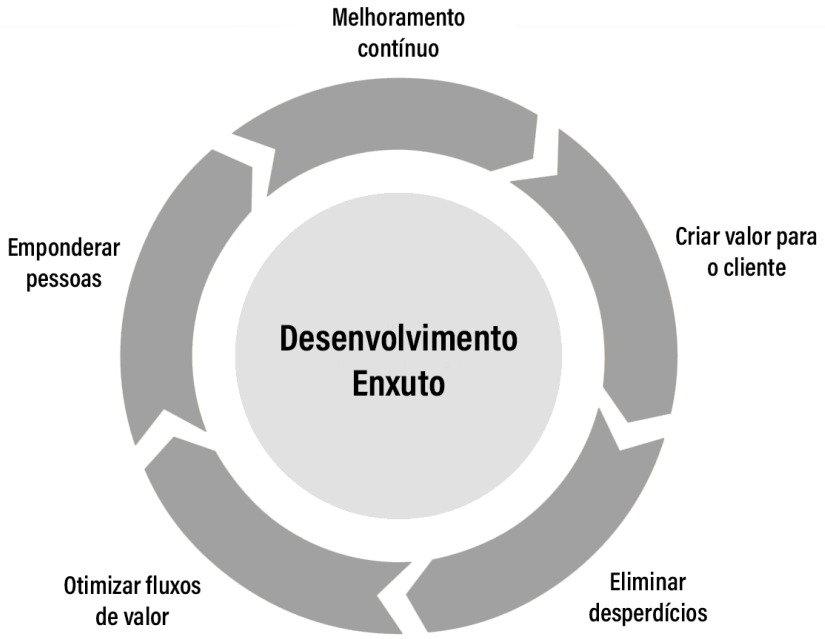

Figura 5 - Processo *LSD*.
Fonte: Adaptado do modelo de Ebert, Abrahamsson e Oza (2012).

Um dos motivadores para o surgimento do LSD foram os princípios do *Toyota Way*, os quais inspiraram não somente a indústria de software mas também vários outros setores. Como é dito pelos autores Poppendieck e Poppendieck (2003), não necessariamente é acelerado o processo de desenvolvimento de software com a adição de mais pessoas, mas sim melhorando implicando melhorias no mesmo, sempre com o foco nas características mais valorizadas pelo cliente.

2.2.5 OPENUP

O OpenUp é um processo integrado e ágil que estrutura o ciclo de vida do produto em quatro fases, sendo elas: *(i)* concepção, *(ii)* elaboração, *(iii)* construção e *(iv)* transição. Além disso, o OpenUP fornece práticas derivadas das opiniões de grandes líderes do desenvolvimento de software (COSSENTINO; HILAIRE; SEIDITA, 2013).

O modelo apresenta algumas características do Processo Racional Unificado (do inglês, *Rational Unified Process - RUP*), além de ser uma alternativa livre de ferramentas

auxiliares, fornecendo um baixo nível de formalismo durante a execução das tarefas. A representação desse modelo de processos é apresentada na Figura 6.

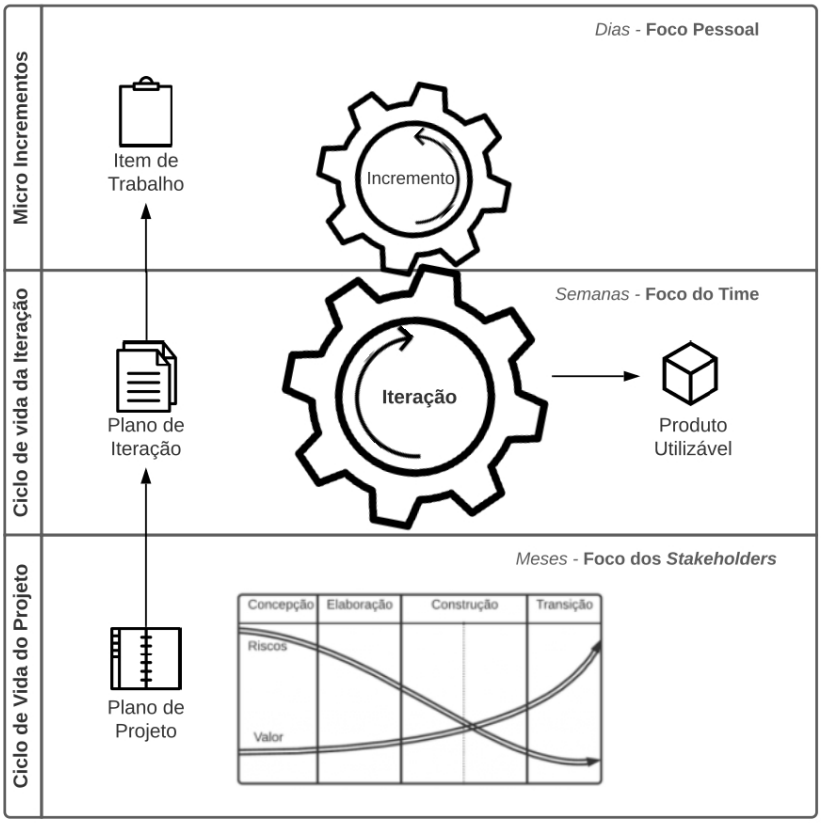

Figura 6 - Processo da metodologia ágil *OpenUP*.
Fonte: Baseado no modelo de Cossentino, Hilaire e Seidita (2013).

O OpenUP é dividido em três camadas: *(i)* Micro Incrementos; *(ii)* Ciclo de vida da iteração; e *(iii)* Ciclo de vida do projeto. Na primeira camada os Micro Incrementos são definidos como resultado de um curto período de tempo trabalhado (horas ou dias). Na segunda camada são criadas partes executáveis pelos micro incrementos realizados. Por fim, a terceira e última camada permite uma visibilidade ampla sobre o andamento do projeto. Está é subdividida nas fases de Concepção, Elaboração, Construção e Transição, com início e fim de cada uma demarcada pelos *milestones*.

Portanto, independente da metodologia ágil utilizada, as atividades fundamentais durante o processo de desenvolvimento de software estão relacionadas à fase de especificação de software. Nesta fase são elicitadas, especificadas e verificadas as necessidades do usuário.

2.3 Especificação de Software

A especificação de software é uma das fases fundamentais dentre as presentes no processo de desenvolvimento de software. Por meio da especificação de software busca-se aplicar as melhores práticas para a análise, documentação e verificação dos requisitos do software a ser desenvolvido.

Os requisitos de software representam as necessidades dos usuários ou clientes, que por sua vez devem ser contempladas pelo sistema a ser desenvolvido. Segundo Sommerville (2016), os requisitos de software são classificados como:

- **Requisitos Funcionais (RFs):** representam as funcionalidades/serviços que devem ser satisfeitos pelo sistema a ser desenvolvido, ou seja, descrevem o que o sistema deve fazer. Um exemplo é o cadastro de um produto;
- **Requisitos Não Funcionais (RNFs):** são as restrições relacionadas sobre como os requisitos funcionais deverão ser implementados. Um exemplo é a definição do tempo de resposta de uma tela.

Uma vez que os requisitos são especificados, esses devem ser priorizados conforme a demanda do projeto e de acordo com os valores de prioridade estimados. Para a priorização podem ser utilizados diversos métodos. Normalmente são atribuídos pesos para cada característica, como por exemplo a gravidade, a urgência e a tendência do requisito.

O processo relacionado à ER é composto por quatro atividades, no qual são aplicadas técnicas utilizadas para levantar, detalhar, documentar e validar os requisitos de um software (SOMMERVILLE, 2016). Dentre essas atividades, é possível citar: *(i)* estudo de viabilidade, *(ii)* elicitação e análise de requisitos, *(iii)* especificação de requisitos e *(iv)* validação de requisitos. Cada uma dessas atividades se relaciona com os respectivos artefatos que podem ser gerados, como pode ser visto na Figura 7.

As atividades definidas por Sommerville (2016), as quais compõem o processo de ER, são descritas a seguir.

- **Estudo de viabilidade:** consiste em um estudo preliminar referente às características do sistema a ser concebido. Nesse estudo são levados em consideração os requisitos de negócios, uma breve descrição do sistema em questão, assim como o modo com o qual o sistema irá solucionar alguma necessidade do usuário. Ainda nesta atividade, é

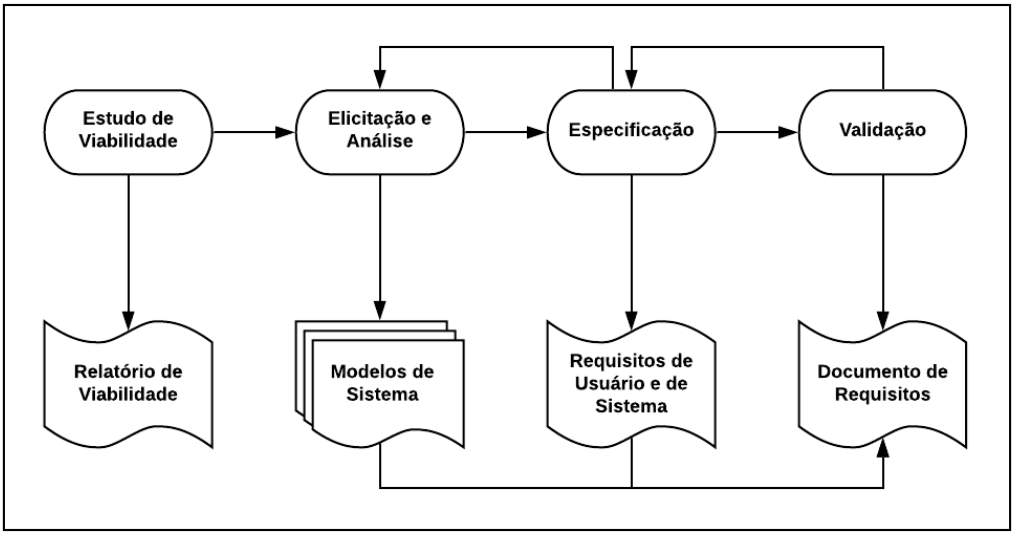

Figura 7 - Visão geral da ER
Fonte: Adaptado do modelo de (SOMMERVILLE, 2016)

verificada a viabilidade de implementação do sistema ou de funcionalidades específicas, analisando tanto aspectos tecnológicos quanto aspectos de negócio, assim como o impacto que essas implementações podem causar no contexto do desenvolvimento. Ao final da atividade, deve ser gerado o relatório de viabilidade, contendo todas as informações obtidas durante os estudos.

- **Elicitação e análise:** consiste na interação dos analistas com os demais *stakeholders* em busca de informações referentes aos requisitos do sistema, os quais incorporam as funcionalidades necessárias a serem desenvolvidas. O processo de elicitação envolve quatro principais sub-atividades, sendo elas: *(i)* descoberta de requisitos, *(ii)* classificação de requisitos, *(iii)* priorização e negociação de requisitos e *(iv)* especificação de requisitos. A partir das informações obtidas nessa atividade, podem ser gerados os modelos de sistema a serem utilizados no desenvolvimento.

- **Especificação:** consiste na descrição dos requisitos de usuário e do sistema de forma consistente e singular, conforme os padrões adotados pela organização. Nessa etapa são descritas todas as necessidades definidas na atividade de elicitação, de modo que a documentação seja compreensível para todos os *stakeholders*.

- **Validação:** é uma das fases mais importantes, por meio da qual pretende-se garantir que os requisitos especificados atendem as reais necessidades dos usuários. A partir dessa atividade, devem ser identificadas as inconsistências dos requisitos, as quais

podem ser verificadas utilizando diferentes tipos de técnicas, tais como: revisões, prototipação ou geração de casos de teste.

Os RFs, por se tratarem das funcionalidades que o sistema deve satisfazer e que serão utilizadas pelos usuários, precisam ser abstraídos a fim de facilitar a compreensão por parte dos *stakeholders*. Nesse contexto, a modelagem de software pode auxiliar nesta tarefa por meio da geração de diferentes diagramas, visando facilitar a compreensão dos diferentes *stakeholders* como usuários, clientes ou desenvolvedores, entre outros.

2.4 Modelagem de Software

A modelagem de software é a forma com a qual é comnunicada a estrutura e o comportamento de um sistema de software, visando facilitar ou melhorar a compreensão das necessidades do desenvolvimento de um software, por meio de modelos gráficos contendo as informações do projeto (BOOCH; RUMBAUGH; JACOBSON, 2006).

Para que esses modelos tenham um certo nível de semelhança e seguimento de padrões, deve ser adotado o uso de uma linguagem para modelagem. Neste contexto é importante destacar a Linguagem de Modelagem Unificada (do inglês, *Unified Modeling Language - UML*). A UML foi criada por Booch, Jacobson e Rumbaugh em 1994, com o objetivo de auxiliar as tarefas de especificação, documentação, desenvolvimento e visualização da arquitetura de sistema. Por meio da UML, é possível gerar modelos gráficos, os quais também podem ser chamados de diagramas (BOOCH; RUMBAUGH; JACOBSON, 2006).

Atualmente existem 14 diagramas diferentes, sendo eles: Diagrama de Classes; Diagrama de Componentes; Diagrama de Pacotes, Diagrama de Objetos; Diagrama de Estrutura Composta; Diagrama de Instalação; Diagrama de Perfil; Diagrama de Casos de Uso, Diagrama de Atividades; Diagrama de Transição de Estados; Diagrama de Sequência; Diagrama de Interatividade; Diagrama de Colaboração ou Comunicação; Diagrama de Tempo.

Esses diagramas são subdivididos em duas categorias sendo elas: *(i)* Estrutural e *(ii)* Comportamental. Na primeira categoria estão os diagramas que priorizam a descrição de estruturas de um sistema. Já na segunda, detalham o comportamento de partes de um sistema. Neste contexto, é importante destacar o diagrama de caso de uso classificado como diagrama comportamental e o diagrama de classes como estrutural.

2.4.1 Diagrama de caso de Uso

Segundo Booch, Rumbaugh e Jacobson (2006), um diagrama de caso de uso auxilia no levantamento dos RFs, descrevendo um conjunto de funcionalidades do sistema e suas interações com elementos externos e entre si. Esse diagrama auxilia na comunicação entre o cliente e os analistas, uma vez que apresenta as principais funcionalidades do sistema com foco no cliente.

Os principais elementos deste diagrama são: *(i)* ator, o qual está relacionado às ações realizadas dentro do sistema; *(ii)* caso de uso, que representa uma funcionalidade do sistema; e *(iii)* relacionamentos, os quais indicam a existência de uma interação entre atores e casos de uso, bem como entre casos de usos.

Na Figura 8 é apresentado um exemplo de diagrama de caso de uso para um cenário fictício de um sistema para o gerenciamento de festas. Neste cenário existem três atores os quais interagem com o sistema, sendo eles: *(i)* Administrador que se relaciona com quatro casos de uso relacionados ao gerenciamento da festa; *(ii)* Usuário que acessa o sistema para visualizar as informações e comprar os ingressos; e *(iii)* Adm. cartão que valida as informações do cartão utilizado para a compra.

Este tipo de diagrama é considerado muito importante, pois o mesmo é utilizado para apresentar uma demonstração do comportamento do sistema em um alto nível de abstração em relação: *(i)* às interações; *(ii)* às atividades a serem executadas dentro do sistema, de forma obrigatória ou opcional; *(iii)* aos tipos de usuários que irão interagir com o sistema; e *(iv)* às interações "herdadas" de um tipo de usuário para outro, determinadas por níveis de acesso;

2.4.2 Diagrama de Classes

Os diagramas de classes normalmente são utilizados para representar sistemas orientados a objetos, apresentando a estrutura do sistema de forma estática (BOOCH; RUMBAUGH; JACOBSON, 2006). Este diagrama visa apresentar as informações que estão sendo modeladas no sistema, tais como: *(i)* uma classe; *(ii)* uma interface; *(iii)* um tipo de dado; ou *(iv)* um componente. Portanto, o diagrama de classes ilustra graficamente como será a estrutura do software, como cada um dos componentes da sua estrutura estarão interligados, e como essas estruturas podem ser desenvolvidas separadamente.

Na Figura 9 é ilustrado o diagrama de classes para o mesmo cenário do sistema de gerenciamento de festas apresentado na Seção 2.4.1. A partir deste cenário foram obtidas

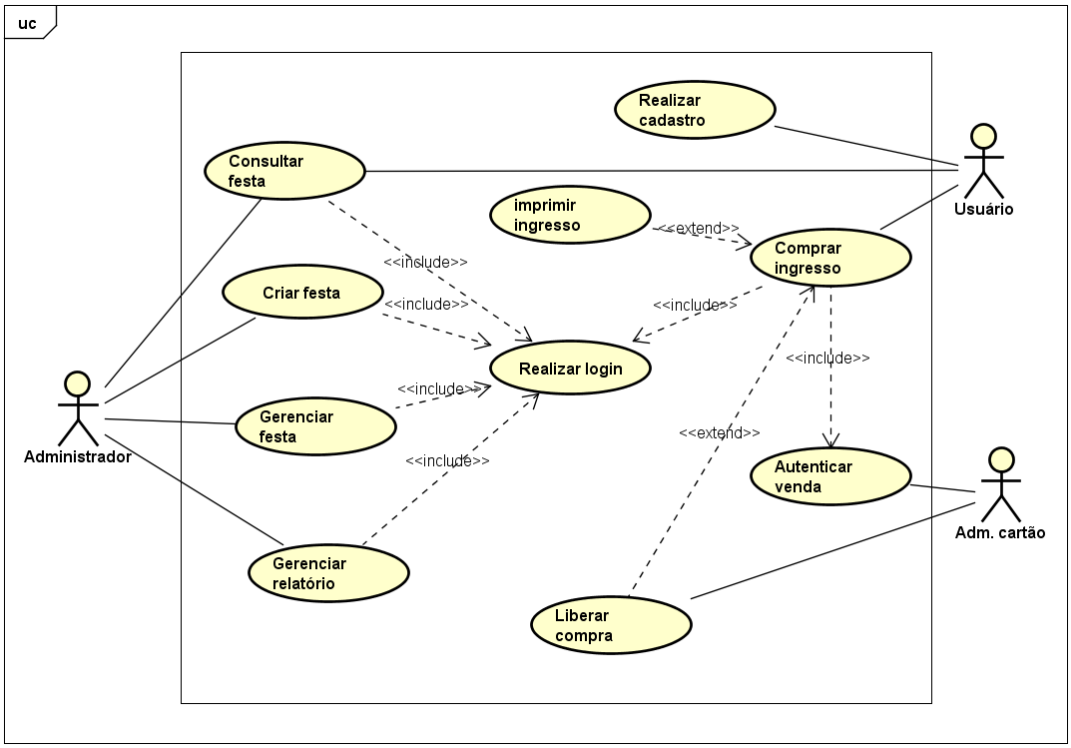

Figura 8 - Diagrama de caso de uso para um sistema de gerenciamento de festas. Fonte: Autoria própria.

as classes necessárias para o desenvolvimento desse sistema, bem como seus respectivos atributos, métodos e relacionamentos.

É importante destacar que este diagrama é um dos mais utilizados da UML, pois é útil em todo o ciclo de vida de software para uma variedade de membros da equipe. Em geral, esses diagramas permitem a validação do design e a comunicação do design entre os indivíduos e as equipes. Por exemplo, os analistas de negócios podem usar diagramas de classe ou de objeto para modelar os ativos e os recursos atuais de um negócio. Por outro lado, os desenvolvedores podem utilizar os diagramas de classes para projetar e documentar as classes a serem codificadas referente ao sistema a ser desenvolvido.

Para que um software seja desenvolvido com eficiência e integridade, podem ser utilizados diversos artefatos como os diagramas, bem como a forma como um sistema é estruturado, a qual é definida por meio da arquitetura de software.

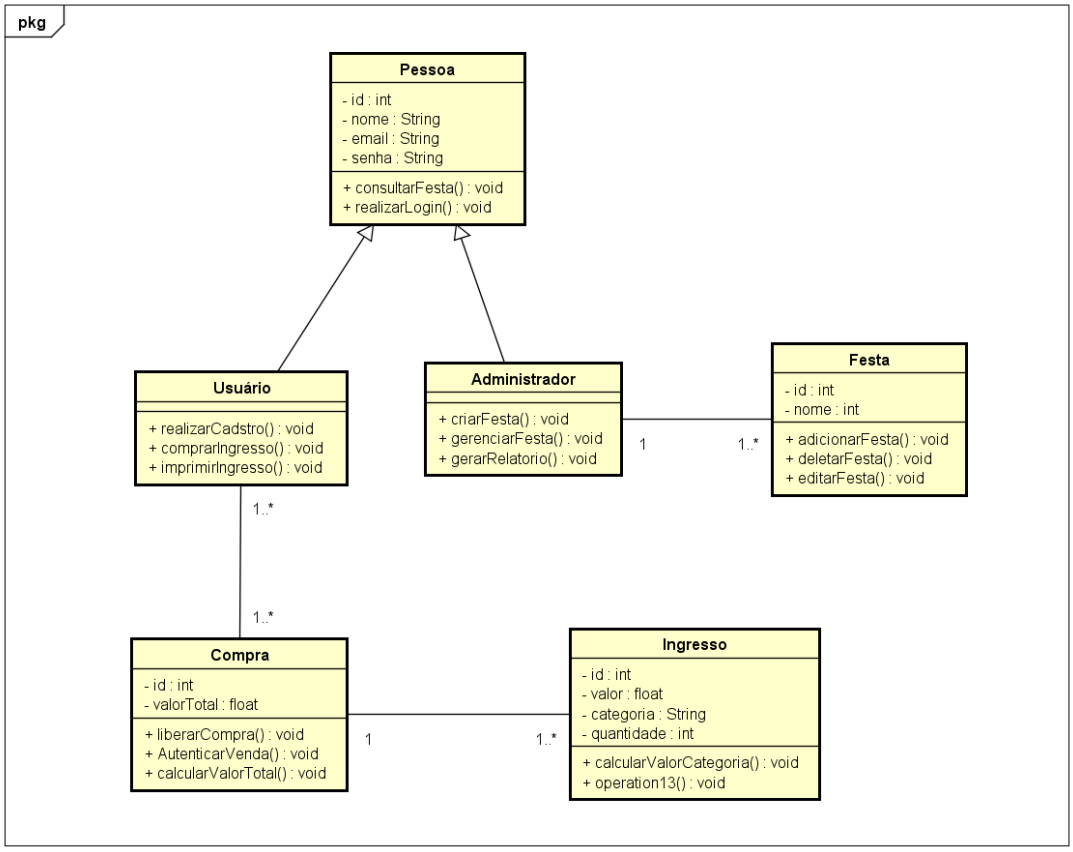

Figura 9 - Diagrama de classes para um sistema de gerenciamento de festas. Fonte: Autoria própria.

2.5 Arquitetura de Software

Segundo Bass, Clements e Kazman (2003), a Arquitetura de Software (AS) define a forma da estrutura de um sistema, bem como seus elementos e relacionamentos. Dessa forma, uma AS, quando adequadamente documentada, facilita a compreensão da estrutura de um sistema, evitando a compreensão a partir do código fonte e ajudando nas comunicações com a equipe de desenvolvimento e com cliente (SOMMERVILLE, 2016).

Além disso, a AS permite identificar as decisões na construção do software que influenciam no sucesso de software (PRESSMAN, 2016). Assim, o desenvolvimento da arquitetura de um sistema afeta fatores como reuso, manutenibilidade, extensibilidade e escalabilidade (SOMMERVILLE, 2016), os quais podem definir o sucesso ou a decadência de um projeto.

É importante destacar que, antes da criação de uma AS, devem ser realizados estudos quanto ao domínio da aplicação para que os padrões arquiteturais aplicados

não ocasionem problemas estruturais durante o desenvolvimento. Um dos problemas arquiteturais mais recorrentes em sistemas mal projetados é a erosão arquitetural (do inglês, *architecture erosion*). Neste problema, as implementações realizadas no software acabam ultrapassando os limites arquiteturais suportados pelo mesmo, fazendo com que os padrões pré-definidos não sejam mais seguidos (BASS; CLEMENTS; KAZMAN, 2003).

Neste contexto, para evitar os problemas derivados da erosão arquitetural e garantir a conformidade do sistema de acordo com a arquitetura definida, devem ser definidos padrões a serem aplicados e seguidos ao longo do ciclo de vida deste do projeto. Para isso, é necessária a utilização de um estilo arquitetural, de modo a considerar o contexto da aplicação.

Estilos arquiteturais funcionam como uma estrutura guia para detalhar os componentes e módulos do software (PRESSMAN, 2016). É importante ressaltar que existem vários tipos de estilos arquiteturais, como arquitetura em camadas, modelo de repositórios, arquitetura cliente-servidor, arquitetura orientada a objetos, arquitetura orientada a serviços, entre outros (SOMMERVILLE, 2016). Cada estilo arquitetural possui uma maneira de organizar o sistema, assim como características que auxiliam os stakeholders a determinarem quando este deve ser utilizado. Essas características, de uma maneira geral, são extraídas dos requisitos não funcionais do software (PRESSMAN, 2016).

Para garantir a qualidade arquitetural do projeto da ferramenta REMO, o estilo de arquitetura a ser utilizado é a Arquitetura Orientada a Serviços (do inglês, *Service-Oriented Architecture - SOA*).

2.5.1 Arquitetura Orientada a Serviços

Segundo Erl (2016), do ponto de vista arquitetural, um serviço pode ser definido como uma atividade com contexto bem definido. Essas atividades podem ser executadas separadamente, em locais diferentes, de forma independente das demais, garantindo a interoperabilidade entre elas.

Apesar de ser de extrema importância que todos os serviços operem em conjunto para que o sistema funcione por completo, os serviços podem continuar sendo realizados de forma independente, sem que um afete diretamente na performance do outro. No contexto da AS, um sistema pode possuir diversas funcionalidades, as quais podem ser agrupadas pelos seus serviços, conforme a definição da SOA.

SOA é um estilo arquitetural que tem como principal objetivo conceder o acopla-

mento de unidades de sistema que possuam contextos semelhantes, permitindo assim a reutilização dos serviços definidos pelos acoplamentos (HE, 2003). A representação do estilo SOA é apresentada na Figura 10.

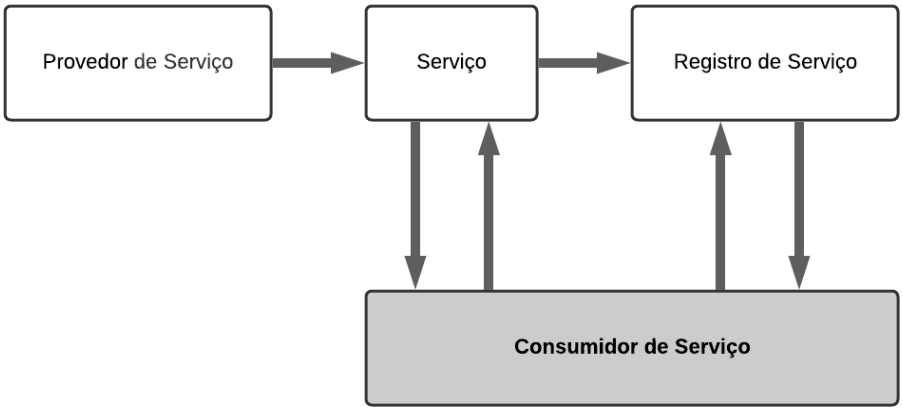

Figura 10 - Arquitetura Orientada a Serviços.
Fonte: Adaptado de DAMASCENO (2012).

O provedor do serviço é a entidade responsável pela criação, disponibilização e descrição de um serviço. Após o serviço estar disponível, a entidade de registro de serviço armazena e localiza a descrição dos serviços, além de prover as informações necessárias para a entidade que irá consumir o próprio serviço. Portanto, a utilização desse estilo arquitetural, também, pode facilitar a integração entre sistemas de alta complexidade, além de serem úteis em contextos que existam funcionalidades independentes.

2.6 Considerações Finais

Os conceitos apresentados neste Capítulo estão relacionados a este trabalho, o qual visa análise e projeto de uma ferramenta que viabilize em uma única aplicação a especificação de requisitos e a modelagem de software no processo de desenvolvimento de software ágil. Com essa ferramenta será possível reduzir o esforço e otimizar o tempo da equipe envolvida, uma vez que será permitida a união de várias atividades e a atualização de diferentes artefatos automaticamente, que comumente são tratados separadamente. No Capítulo 3 é apresentado o processo de condução do Mapeamento Sistemático, bem como os resultados alcançados, os quais corroboram a importância da ferramenta proposta neste trabalho.

3 MAPEAMENTO SISTEMÁTICO

Para a compreensão a respeito do estado da arte da área deste trabalho, foi conduzido um Mapeamento Sistemático (MS) (do inglês, *Systematic Mapping Study - SMS* visando identificar ferramentas para a especificação de software e/ou geração de diagramas no contexto tradicional e ágil. Para a condução deste MS foi utilizado o processo proposto por Kitchenham et al. (2010), o qual consiste em três fases: *(i)* Planejamento; *(ii)* Condução; e *(iii)* Análise.

Este capítulo é organizado da seguinte maneira: na Seção 3.1 é apresentado o planejamento do MS, evidenciando a Questão de Pesquisa (QP) definida; a estratégia em torno da realização da busca automática, assim como os critérios de seleção dos estudos primários; e a extração de dados a partir dos estudos obtidos. Na Seção 3.2 é apresentado o processo de condução do MS a partir da QP definida. A Seção 3.3 detalha a síntese e análise dos estudos primários. Na Seção 3.4 são apresentadas as ameaças à validade do MS realizado. E finalmente, a Seção 3.5 são descritas as considerações finais deste Capítulo.

3.1 Planejamento do Mapeamento Sistemático

Nesta fase foi definido e estabelecido o protocolo de orientação para a realização do MS. A partir dele, são definidas as informações que serão utilizadas durante a fase de Condução (ver Seção 3.2), tais como: QP; estratégia de busca; critérios de inclusão e exclusão; procedimentos para seleção dos estudos; e extração e síntese dos dados.

A partir da necessidade de investigação em relação ao tema de pesquisa, foi elaborada a seguinte QP: **Quais ferramentas têm sido propostas para especificação de software e geração de diagramas?**

Esta QP visa identificar estudos que evidenciam ferramentas para auxiliar na especificação de software, bem como na geração automática de diagramas com base nos requisitos especificados.

3.1.1 Estratégia para Busca Automática

A estratégia de busca iniciou com a construção da *string* de busca. Essa *string* de busca foi criada por meio da identificação das palavras-chaves relacionadas à QP; identificação dos sinônimos relacionados às palavras-chaves especificadas; união entre os sinônimos de uma mesma palavra-chave por meio do operador *booleano* "OR"; e união dos conjuntos de sinônimos por meio do operador *booleano* "AND". A *string* de busca gerada é apresentada na Tabela 1.

Tabela 1 - *String* de busca

(tool OR framework OR prototype) AND ("software requirement" OR "requirement engineering" O

Fonte: Autoria própria.

Após a construção da *string* de busca foram definidas as fontes de busca. Para este MS foram definidas quatro fontes de busca, as quais foram selecionadas de acordo com as diretrizes propostas por Brereton et al. (2007). As bases utilizadas são apresentadas na Tabela 2.

Tabela 2 - Fontes de busca automática

Fonte/Base de Busca	Endereço Eletrônico
IEEEXplore	www.ieeexplore.com
ACM	http://dl.acm.org/
Scopus	www.scopus.com
Science Direct	https://www.sciencedirect.com/

Fonte: Autoria própria.

Para assegurar que a *string* de busca retornasse estudos relevantes, foi elaborada uma lista de controle contendo quatro estudos, os quais estão indexados nas fontes de busca utilizadas. Esses estudos são: Ries, Capozucca e Guelfi (2018), Abdouli, Karaa e Ghezala (2016), Gulia e Choudhury (2016), e Chang et al. (2014).

3.1.2 Critérios para seleção de estudos primários

Para realizar a seleção dos estudos retornados nas fontes de busca, foram definidos alguns Critérios de Inclusão (CI) e Critérios de Exclusão (CE). Esses critérios garantem que sejam incluídos somente os estudos que satisfaçam o contexto em torno da QP. Os critérios definidos para a condução deste MS são:

- **Critérios de Inclusão (CI):**
  - **CI_1:** estudos primários que apresentam ferramentas de especificação de software no contexto tradicional e ágil;
  - **CI_2:** estudos primários que apresentam ferramentas de geração de diagramas a partir da especificação de requisitos.
- **Critérios de Exclusão (CE):**
  - **CE_1:** estudos primários com textos incompletos;
  - **CE_2:** estudos primários que não sejam *full paper* ou *short paper* (pôsteres, tutoriais, relatório técnicos, teses e dissertações);
  - **CE_3:** estudos primários que sejam uma versão anterior de um estudo mais completo sobre a mesma investigação;
  - **CE_4:** estudos primários que não estejam escritos em inglês ou português;
  - **CE_5:** estudos primários indisponíveis para download.

3.1.3 Extração de Dados

Para a realização da extração dos dados dos estudos foi utilizada a ferramenta *Google Sheets*, do *Google Drive*[1]. Para auxiliar o processo de organização e categorização dos dados foi criada uma planilha para armazenar as seguintes informações: *(i)* título do estudo; *(ii)* base onde o estudo foi retornado; *(iii)* ano de publicação; *(iv)* tipo da publicação; *(v)* país do primeiro autor; *(vi)* objetivo do estudo; *(vii)* se a ferramenta proposta era paga ou não; *(viii)* se a ferramenta contempla a manipulação de artefatos; *(ix)* se a ferramenta contemplava controle de versão; *(x)* as fases da especificação de software; e *(xi)* diagramas gerados.

É importante destacar que cada estudo primário incluído possui um contexto específico. Para uma melhor compreensão destes estudos, foram criadas quatro categorias de acordo com as funcionalidades das ferramentas abordadas. As categorias são:

- **C_1 - Manipulação de artefatos:** contempla documentação e importação ou exportação de arquivos.
- **C_2 - Implementação:** contempla controle de versão e geração de código a partir da ferramenta.

[1] https://www.google.com.br/drive/apps.html.

- **C_3 - Especificação:** contempla as atividades da especificação de software como: elicitação, especificação, validação e gerenciamento de requisitos.
- **C_4 - Modelagem:** contempla a geração de diagramas.

3.2 Condução do Mapeamento sistemático

O processo de condução deste MS é apresentado na Figura 11. O processo iniciou com a aplicação da *string* de busca na base da *IEEEXplore*, na qual retornaram um total de 133 estudos, na base *ACM* 116 estudos, na base *Scopus* 437 estudos, e na base *Science Direct* 16 estudos.

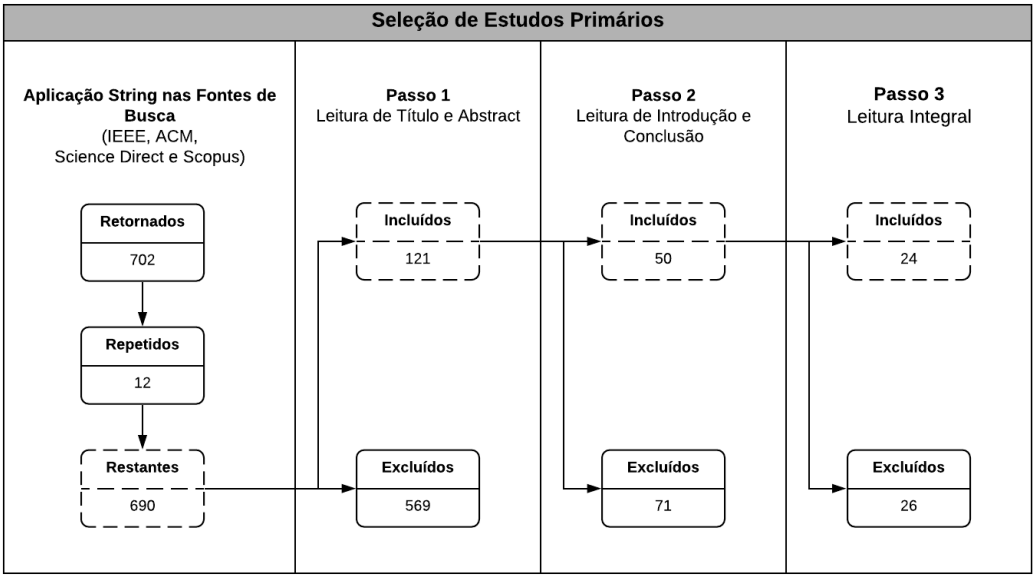

Figura 11 - Processo de seleção dos estudos primários.
Fonte: Autoria própria.

A partir da execução da *string* de busca nas bases de dados, um total de 702 estudos foram retornados. Deste total de estudos, 12 deles foram constatados como repetidos. A partir disso, foi realizada a leitura dos títulos e resumos dos 609 estudos restantes e aplicado os critérios de inclusão e exclusão, resultando na inclusão de 121 estudos. No passo 2, foi realizada a leitura da introdução e conclusão dos 121 estudos restantes, novamente junto à aplicação dos critérios de inclusão e exclusão, restando apenas 50 estudos. Por fim, no passo 3 foi realizada a leitura na íntegra dos estudos relevantes, incluindo no total 24 estudos primários.

3.3 Análise e Síntese dos estudos primários

Os 24 estudos primários foram selecionados por abordarem ferramentas que têm sido utilizadas para especificação de software e geração de diagramas no contexto tradicional e ágil. A investigação por estes tipos de ferramentas passou a ter maior evidência a partir do ano de 2002 até o ano de 2019, conforme é apresentado na Figura 12.

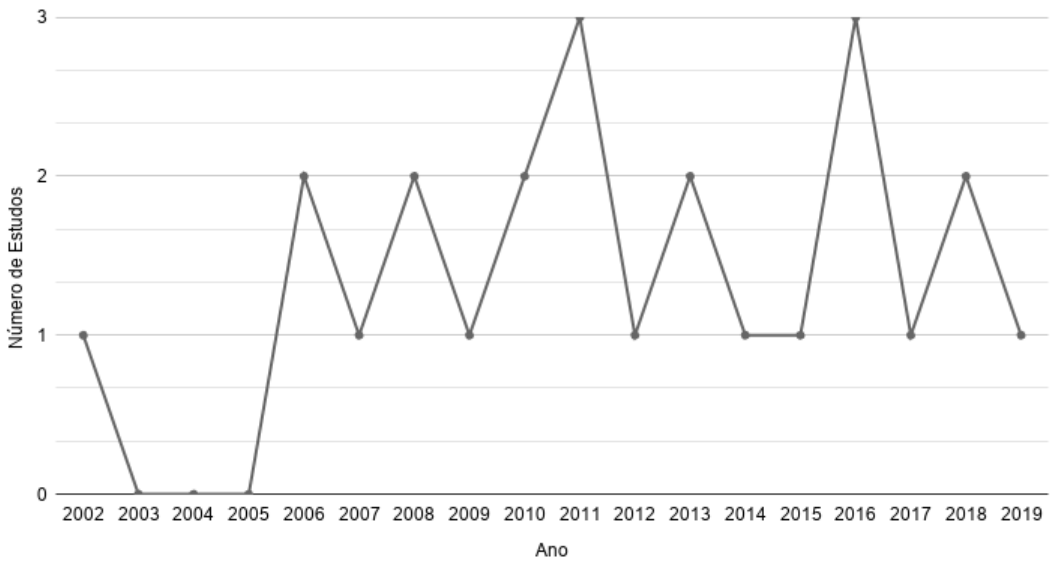

Figura 12 - Número de publicações por ano.
Fonte: Autoria própria.

É importante destacar que os estudos identificados estão distribuídos entre 17 países diferentes, levando em consideração o país de origem do primeiro autor do estudo. De origem brasileira, foram retornados apenas os estudos de Esteca et al. (2012) e Anchiêta, Sousa e Moura (2013). Dentre os demais países de origem dos estudos, estão: Jordânia, Taiwan, Índia, Tunísia, Itália, Irlanda e Luxemburgo, Turquia, Estados Unidos, Malásia, Paquistão, Irá, Alemanha, Tailândia, Nova Zelândia e Canadá, conforme é demonstrado na Figura 13.

Na liderança no ranking de estudos incluídos neste MS, ficaram empatados, Taiwan e Índia, com três estudos publicados cada um ((LU et al., 2008), (CHANG et al., 2011) e (CHANG et al., 2014)); e ((MOHANAN; SAMUEL, 2018), (GULIA; CHOUDHURY, 2016) e (KUMAR; SANYAL, 2008)), respectivamente. Os países com dois estudos retornados, foram provenientes da Jordânia, do Brasil e do Paquistão, sendo que os demais países tiveram apenas um estudo incluído ao final da condução deste MS.

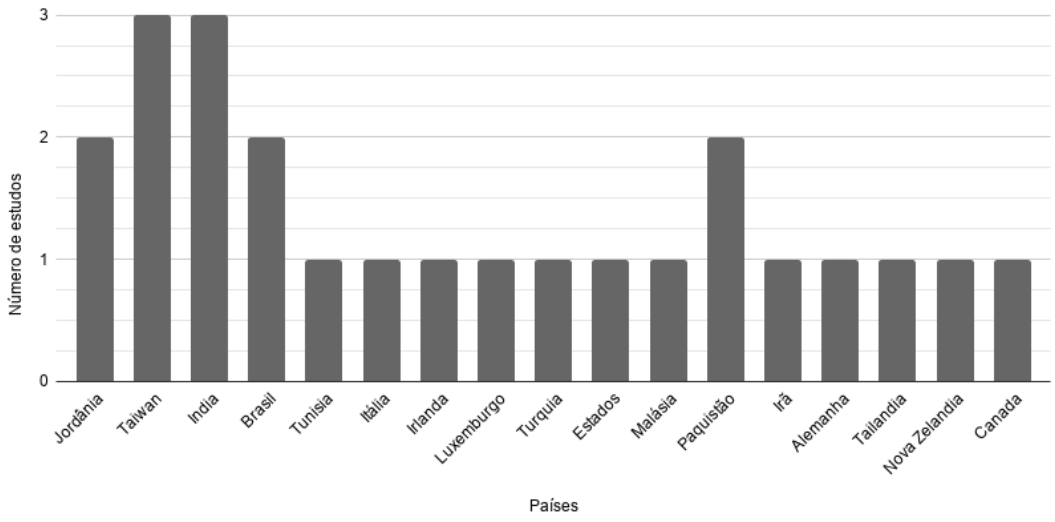

Figura 13 - Número de estudos primários distribuídos por país.
Fonte: Autoria própria.

A Tabela 3 apresenta os 24 estudos primários selecionados, relacionando-os às suas respectivas categorias (primeira coluna); a referência dos estudos (segunda coluna); a disponibilidade das ferramentas podendo ser paga, *open source* e Não Especificado (NE) (terceira coluna); à fonte de busca onde o estudo foi encontrado (quarta coluna); e o tipo de publicação (quinta coluna).

A partir da *string* aplicada nas fontes de busca, 33% (8/24) dos estudos primários selecionados foram identificados na base *IEEE*, 54% (13/24) na base *Scopus* e 13% (3/24) estudos restantes foram retornados na *ACM*. Além dos estudos publicados em conferências, retornados a partir da busca nas bases de dados, somente dois estudos foram publicados em periódicos.

Quanto a categorização dos estudos, é possível observar na Figura 14 que 42% (10/24) estudos se encaixaram somente na categoria "C_4". Por outro lado, os estudos não categorizados na "C_4" foram enquadrados em ao menos duas ou três categorias. Além disso, é importante destacar que somente o estudo de Cavada et al. (2009) abordou aspectos de Especificação (categoria C_3) junto a Modelagem (categoria C_4).

3.3.1 Ferramentas para especificação de software e geração de diagramas (QP)

A QP proposta neste MS visou orientar o processo de busca de estudos, com a intenção de identificar ferramentas que têm sido utilizadas para especificação de software

Tabela 3 - Visão geral dos estudos primários selecionados.

Categorias	Estudos Primários	Disponb.	Fontes	Tipo de Publica
C_4	Alkhader, Hudaib e Hammo (2006)	NE	IEEE	Conferência
	Kumar e Sanyal (2008)	NE	IEEE	Conferência
	Gulia e Choudhury (2016)	NE	IEEE	Conferência
	Abdouli, Karaa e Ghezala (2016)	NE	IEEE	Conferência
	Mohanan e Samuel (2018)	NE	Scopus	Periódico
	Nassar, Alhroob e Imam (2017)	NE	Scopus	Conferência
	Bozyiğit, Aktaş e Kilinç (2019)	NE	Scopus	Periódico
	Iqbal e Bajwa (2016)	NE	IEEE	Conferência
	Afreen e Bajwa (2011)	NE	Scopus	Conferência
	Anchiêta, Sousa e Moura (2013)	NE	Scopus	Conferência
C_1 e C_3	Chang et al. (2011)	NE	IEEE	Conferência
	Bojnord, Ahmad e Bojnord (2013)	NE	Scopus	Conferência
	Durán et al. (2002)	NE	Scopus	Conferência
	Phanthanithilerd e Prompoon (2015)	NE	Scopus	Conferência
	Kamalrudin, Grundy e Hosking (2010)	NE	Scopus	Conferência
	Somé (2006)	NE	Scopus	Conferência
C_1 e C_4	Chang et al. (2014)	NE	IEEE	Conferência
	Ries, Capozucca e Guelfi (2018)	*Open Source*	ACM	Conferência
	Ibrahim e Ahmad (2010)	NE	Scopus	Conferência
C_3 e C_4	Cavada et al. (2009)	NE	ACM	Conferência
C_1, C_2 e C_3	Lu et al. (2008)	NE	IEEE	Conferência
	Esteca et al. (2012)	*Open Source*	IEEE	Conferência
C_1, C_2 e C_4	Deeptimahanti e Sanyal (2011)	NE	ACM	Conferência
C_1, C_3 e C_4	Dascalu et al. (2007)	*Open Source*	Scopus	Conferência

Fonte: Autoria própria.

e a geração automática de diagramas, visando apoiar a especificação de requisitos e modelagem de software.

Quanto às ferramentas abordadas pelos estudos, foram identificadas informações referentes à sua disponibilidade, com a finalidade de identificar se as soluções eram *open source* ou não. Nesse contexto, somente 13% (3/24) dos estudos primários identificaram as ferramentas como *open source*, enquanto os 87% restante não mencionaram nenhum detalhe sobre a disponibilidade de suas ferramentas. Por se tratarem de ferramentas aplicadas em estudos científicos, a ausência dessa informação ou a não disponibilidade gratuita dessas ferramentas, pode acarretar na incompreensão do contexto ou perda de interesse na utilização das mesmas.

As ferramentas abordadas nos estudos selecionados contemplam de um a três diagramas. Dos 24 estudos primários, somente oito (33%) não abordaram a geração de

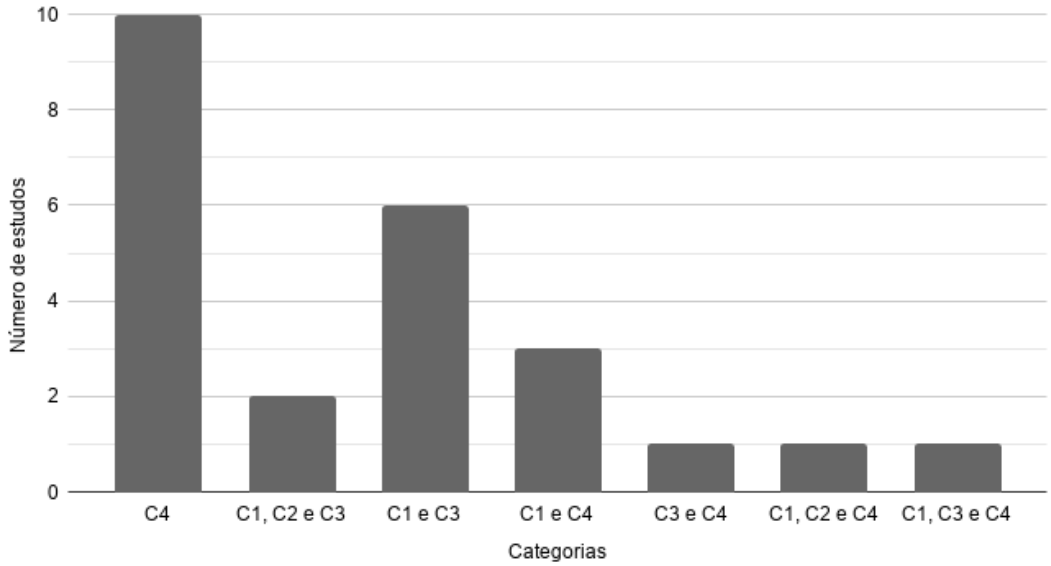

Figura 14 - Ocorrência de estudos primários por categoria.
Fonte: Autoria própria.

diagramas. É interessante destacar, também, que nenhum dos estudos incluídos no MS contemplaram, em conjunto, as quatro categorias definidas na Subseção 3.1.3. Na Tabela 4, é apresentada a relação dos estudos, as respectivas categorias nos quais foram classificados, bem como os diagramas gerados.

Dentre a variedade de diagramas identificados nos estudos que se enquadram na categoria C_4, o mais abordado pelas ferramentas foi o Diagrama de Classes, o qual foi encontrado em um total 46% (11/24) dos estudos primários. Com um total de 25% (6/24), o diagrama de Caso de Uso também foi um diagrama muito recorrente nos estudos, o que corrobora com a sua importância. Já referente ao diagrama de objetos, acredita-se que a falta de estudos seja recorrente à pouca utilização deste diagrama, por tratar de informações específicas dentro de um projeto, o que justifica o fato de que somente um estudo tratou da geração desse diagrama.

A ferramenta chamada *EuRailCheck*, apresentada por Cavada et al. (2009), se enquadra nas categorias C_3 e C_4. Essa ferramenta foi desenvolvida para atender as necessidades de formalização e validação de requisitos descritos em linguagem natural. A metodologia aplicada na ferramenta é dividida em três etapas: *(i)* fragmentação e categorização; *(ii)* formalização; e *(iii)* validação formal. Cada uma dessas etapas aborda a divisão dos requisitos em partes menores conforme suas categorias, sendo realizada a formalização dessas categorias em conceitos e diagramas em UML e a validação dos

Tabela 4 - Diagramas contemplados pelos estudos classificados na categoria C_4.

Categorias	Estudos Primários	Diagramas
C_4	Alkhader, Hudaib e Hammo (2006)	Classes
	Bozyiğit, Aktaş e Kilinç (2019)	Classes
	Afreen e Bajwa (2011)	Classes
	Mohanan e Samuel (2018)	Classes
	Iqbal e Bajwa (2016)	Atividade
	Nassar, Alhroob e Imam (2017)	Sequência
	Kumar e Sanyal (2008)	Caso de Uso, Classes
	Anchiêta, Sousa e Moura (2013)	Caso de Uso, Classes
	Gulia e Choudhury (2016)	Atividades, Sequência
	Abdouli, Karaa e Ghezala (2016)	Objetos, Atividade, Classes
C_1 e C_4	Ibrahim e Ahmad (2010)	Classes
	Chang et al. (2014)	Atividades, Caso de Uso
	Ries, Capozucca e Guelfi (2018)	Sequência, Caso de Uso, Classes
C3 e C_4	Cavada et al. (2009)	Sequência, Classes
C_1, C_2 e C_4	Deeptimahanti e Sanyal (2011)	Caso de Uso, Classes
C_1, C_3 e C_4	Dascalu et al. (2007)	Caso de Uso

Fonte: Autoria própria.

problemas e análise dos resultados, respectivamente. A criação dos diagramas em UML é realizada utilizando um editor vinculado à ferramenta por meio de *plugin*. Nele, o usuário pode montar os diagramas de classes e sequência em linguagem UML com auxílio de Linguagem Natural Controlada (do inglês, *Controlled Natural Language- CNL*).

Com a intenção de abordar o contexto de modelagem de sistemas, os estudos Alkhader, Hudaib e Hammo (2006), Kumar e Sanyal (2008), Abdouli, Karaa e Ghezala (2016), Gulia e Choudhury (2016), Mohanan e Samuel (2018), Nassar, Alhroob e Imam (2017), Bozyiğit, Aktaş e Kilinç (2019), Iqbal e Bajwa (2016), Afreen e Bajwa (2011) e Anchiêta, Sousa e Moura (2013) apresentaram suas soluções relacionadas a categoria C_4.

Alkhader, Hudaib e Hammo (2006) apresenta um *framework* para automatizar a transformação de requisitos em linguagem natural para diagramas UML. Essa automatização ocorre por meio do processamento dos requisitos, sendo gerado um modelo em Linguagem Extensível de Marcação (do inglês, *Extensible Markup Language - XML*) contendo as informações necessárias para a criação de diagramas de classes. Os documentos de especificações eram obtidos a partir de um repositório, os quais foram classificados conforme sua complexidade (simples, intermediários e complexos). Essas informações, com relação à complexidade, foram utilizadas para realização de testes comparativos de desempenho da ferramenta.

A ferramenta apresentada por Kumar e Sanyal (2008), visa auxiliar os desenvolvedores por meio da geração de diagramas em UML, a partir de requisitos especificados em linguagem natural. A geração dos diagramas ocorre diretamente pelo uso da ferramenta, sem a necessidade da utilização, a parte, de ferramentas de modelagem em UML. O processo para a geração dos diagramas em UML ocorre por meio da identificação das informações necessárias a partir das solicitações dos *stakeholders*, as quais estão inseridas nos requisitos. A partir disso, são identificados os atores, os casos de uso relacionados a cada ator e então é gerado o diagrama de caso de uso. Com esse diagrama são identificadas as classes, atributos, métodos e relacionamentos para a geração do diagrama de classes. É necessária pouca interação humana, necessitando apenas que o usuário selecione as classes a serem utilizadas, as quais foram identificadas automaticamente.

Gulia e Choudhury (2016) apresentaram a criação dos diagramas de atividades e de sequência por meio da especificação de requisitos em linguagem natural. Neste estudo é analisada a entrada inserida como texto e extraídas as frases assim como as atividades a serem utilizadas para a geração dos diagramas. Os procedimentos realizados para a geração destes dois diagramas ocorrem em diferentes instantes de tempo e utilizando diferentes técnicas. O usuário pode selecionar qual será o diagrama a ser gerado. Com isso, para cada uma das opções são disponibilizadas interfaces diferentes, nas quais o usuário insere os requisitos em um campo de texto e o diagrama é gerado com base nessa informação.

Por parte de Abdouli, Karaa e Ghezala (2016), é apresentado um *survey* de estudos relacionados ao processo de transformação de requisitos em diagramas UML. Esses estudos têm foco na análise de requisitos. Em geral, são apresentadas algumas técnicas utilizadas para a extração das informações necessárias para a geração dos diagramas, assim como as próprias ferramentas estudadas durante o *survey*. Dentre as ferramentas citadas, é importante destacar SUGAR e UMGAR, incluídas também neste MS, as quais foram propostas por Kumar e Sanyal (2008) e Deeptimahanti e Sanyal (2011), respectivamente.

O estudo de Abdouli, Karaa e Ghezala (2016) foi projetado baseado em Inspeção (do inglês, *Inspection*), Processamento de Linguagem Natural - PLN (do inglês, *Natural language processing - NPL*), heurísticas, padrões e gráficos e conceitos linguísticos/ontologias. A geração dos diagramas pode ocorrer de duas formas: *(i)* de forma direta, por meio da ferramenta; e *(ii)* geração de documentos (arquivos XML) a serem utilizados em ferramentas à parte.

Mohanan e Samuel (2018) apresentam a ferramenta SUCM, desenvolvida por meio da utilização da biblioteca *OpexNLP* do Apache. Com essa biblioteca são gerados

os diagramas de classe a partir da especificação de requisitos do sistema. A ferramenta utiliza Semântica do Vocabulário e Regras de Negócios (do inglês, *Semantics of Business Vocabulary and Business Rules - SBVR*) para a extração das informações de classes, objetos, atributos e relacionamentos de um texto contendo os requisitos, escritos em inglês. Com essas informações extraídas e processadas, o sistema realiza a geração automática do diagrama de classes.

A ferramenta SUCM foi avaliada por meio da execução de três estudos de caso, sendo eles um sistema robótico, um sistema para automação de bibliotecas e um sistema de automação de lojas de varejo. Para cada um desses cenários, foi inserido um texto contendo as informações a serem tratadas com requisitos. Em alguns dos casos, a performance da ferramenta foi afetada devido ao processamento incorreto de algumas informações do texto, ocasionando, por exemplo na criação de classes desnecessárias.

Em Nassar, Alhroob e Imam (2017) foi realizada uma Abordagem Algoritmica para Desenho de Diagramas de Sequência (do inglês, *Algorithmic Approach for Sketching Sequence Diagram - AASSD*), a qual realiza a leitura das informações de uma descrição de caso de uso, extração das informações com a utilização de PLN e realização do pré-processamento dessas informações para posteriormente utilizá-las para geração do diagrama de sequência.

O autor também destaca a importância da formulação do texto, para que a AASSD consiga o melhor resultado possível, recomendando que seja utilizada um nível de inglês simplificado com poucas descrições implícitas contidas no texto. É mencionado também o fato da ferramenta ser semi-automática, fazendo com que o usuário precise realizar algumas interações para a conclusão da formulação dos diagramas.

No trabalho de Bozyiğit, Aktaş e Kilinç (2019) é criada uma ferramenta para a geração de diagramas de classe com base em textos de especificação de requisitos, descritos em linguagem turca. Essa linguagem possui algumas dificuldades para a realização da PLN, devido ao fato da mesma possuir uma morfologia mais complexa, quando comparada com linguagens, como inglês, por exemplo.

O processo de entrada das informações para a geração do diagrama é realizada por meio da tokenização, onde é realizada a separação da sentença em grupos de palavras. Alinhado a esse processo de extração, foi utilizada uma base de dados para a validação das informações, a qual possuía vinte textos de especificação de requisitos escritos em turco, junto às suas traduções para o inglês. Para a validação da ferramenta, foi utilizada a especificação de um cenário exemplo de um restaurante, no qual as classes foram

devidamente criadas, junto a seus atributos, e relacionadas as demais classes, conforme o texto de requisitos do exemplo.

A aplicação de um método para realizar o mapeamento automático de SBVR para geração de diagramas de atividade, é apresentada por Iqbal e Bajwa (2016), por meio da ferramenta chamada SBVR2ACTIVITY. A especificação dos requisitos em SBVR é processada para posteriormente separar as informações e gerar o diagrama com base nas mesmas. A proposta é avaliada por meio de um estudo de caso aplicado em um cenário fictício de um caixa de autoatendimento, no qual os elementos contidos no texto de especificação de requisitos foram identificados e processados, gerando as estruturas do diagrama de atividades.

Utilizando as mesmas técnicas, em (AFREEN; BAJWA, 2011) é apresentada a ferramenta nomeada SBVR2UML, a qual diferentemente de (IQBAL; BAJWA, 2016), a ferramenta foca na geração de diagrama de classes. Novamente a partir do texto de especificação de requisitos, são extraídas as informações para a geração do diagrama. Nesse caso, os autores utilizaram o cenário fictício de um sistema de controle de bibliotecas, para a avaliação da ferramenta, do qual foram identificadas as classes, atributos e relacionamentos os quais foram extraídos a partir do texto.

Já em Anchiêta, Sousa e Moura (2013) é apresentada uma ferramenta, a qual foi desenvolvida em Java. A ferramenta tem o foco na geração de protótipos de telas, diagrama de classes e diagrama de caso de uso de forma automática, por meio da extração de características encontradas em texto de especificação de requisitos, com auxílio de técnicas de PLN. As informações extraídas são armazenadas em uma estrutura de dados, posteriormente utilizada para a geração dos gráficos e do protótipo. Com relação ao diagrama de casos de uso, a ferramenta consegue inclusive definir estruturas de ligação de *"extend"* e *"include"*, identificando todos os atributos corretamente.

Os estudos de Lu et al. (2008) e Esteca et al. (2012) estão enquadrados nas categorias C_1, C_2 e C_3, uma vez que buscaram permitir a manipulação de artefatos, em conjunto com opções de implementação e especificação.

No estudo de Lu et al. (2008) foi apresentado um editor de modelagem de requisitos. Esse editor suporta a integração de documentos de requisitos formais ou informais com diagramas em UML; análise e design; e código fonte. A ferramenta possui foco no gerenciamento de requisitos, permitindo a reunião de informações obtidas por meio de entrevista com cliente, bem como a especificação e manutenção dos requisitos. Além disso, a ferramenta também permite que o analista de requisitos realize a exportação das

informações em um arquivo XML, o qual pode ser utilizado em ferramentas à parte para criação de diagramas.

Esteca et al. (2012) propõem a expansão de uma ferramenta já existente chamada Ferramenta de Suporte à Elicitação de Requisitos (FSER). Essa ferramenta primeiramente foi desenvolvida para atender às necessidades da elicitação de requisitos junto a clientes, e com a expansão, foi tratado o gerenciamentos destes requisitos. Esse módulo de extensão foi criado baseado em padrões definidos pela norma IEEE 830, contribuindo também nas atividades de gerência de requisitos e desenvolvimento de requisitos, definidas pelo Modelo de Capacidade e Maturidade Integrado (do inglês, *Capability Maturity Model Integration - CMMI*). A ferramenta dispõe de um formulário de coleta de dados para os requisitos a serem especificados, bem como os casos de uso necessários para desenvolvimento. Além disso, é possível gerar um documento com a extensão *".PDF"* contendo as informações relacionadas ao levantamento de requisitos, podendo verificar também a versão de cada documento.

Os estudos de Chang et al. (2011), Bojnord, Ahmad e Bojnord (2013), Durán et al. (2002), Phanthanithilerd e Prompoon (2015), Kamalrudin, Grundy e Hosking (2010) e Somé (2006) estão enquadrados nas categorias C_1 e C_3, por buscarem permitir a manipulação de artefatos, em conjunto com as atividades relacionadas à elicitação, especificação ou validação.

Chang et al. (2011) apresenta uma ferramenta para elicitação de requisitos baseada em um *template* e modelagem baseada em Linguagem de Modelagem de Sistemas (do inglês, *Systems Modeling Language - SysML*). Com essa modelagem é possível reunir informações com os usuários, por meio de uma interface dedicada a entrevistas. Além disso, com o *template* é possível especificar os RFs e RNFs. A modelagem utilizando SysML ocorre a partir da geração de um arquivo XML contendo as informações necessárias para a geração dos diagramas. Apesar de não ter mencionado diretamente, no estudo é utilizada a ferramenta de modelagem UML *Papyrus*, baseada em *Eclipse*.

Bojnord, Ahmad e Bojnord (2013) apresenta a ferramenta WEBSTUIRE, a qual busca auxiliar desenvolvedores na elicitação e especificação de requisitos de interfaces. Além disso, a ferramenta permite a modelagem de interfaces e diagramas de sequências, a fim de deixar mais clara a usabilidade do protótipo. Os requisitos de interface são especificados por meio da adição de etapas na sequência de cada caso de uso. O usuário pode especificar o tipo da etapa, quem realiza o envio da informação, a mensagem relacionada, além de quem recebe a mensagem específica.

Em Durán et al. (2002) é apresentada REM (REquirements Manager), na qual podem ser especificados os requisitos, conflitos e defeitos utilizando linguagem natural. Os projetos criados a partir dessa ferramenta podem ser iniciados a partir de um projeto base e são armazenados em uma base de dados relacional. Na ferramenta podem ser definidos os atores relacionados aos casos de uso, assim como a especificação de cada caso de uso e relacionando-os aos respectivos atores. Além disso, o sistema disponibiliza uma matriz de rastreabilidade, por meio da qual todos os requisitos podem ser acompanhados.

O estudo de Phanthanithilerd e Prompoon (2015) têm o foco na verificação das características dos requisitos e nos diagramas de caso de uso e de classes modelados em UML. No sistema são importados os documentos contendo os requisitos especificados, assim como os diagramas de caso de uso e classes. Com isso, as características do texto de requisitos e dos diagramas são extraídas para posteriormente serem comparadas, verificando a ambiguidade, consistência e rastreabilidade entre esses três artefatos vinculados à ferramenta. Por fim, os resultados dessas verificações são apresentados ao usuário.

Kamalrudin, Grundy e Hosking (2010) apresenta uma alternativa para permitir que analistas de requisitos possam extrair casos de uso essenciais a partir da descrição dos requisitos, importados para a ferramenta. A ferramenta consegue identificar esses casos de uso a partir de uma base de dados que é pré-existente na ferramenta, fazendo com que os requisitos sejam avaliados conforme sua essencialidade. Após extraídas essas informações, é apresentada uma lista de casos de uso para o usuário, o qual pode ainda adicionar novos itens, assim como editar e excluir os pré-definidos pela ferramenta. Os autores ainda destacam a possibilidade de integração da ferramenta com outras relacionadas a ER, permitindo uma usabilidade mais completa das funcionalidades propostas.

A ferramenta nomeada UCEd foi apresentada por Somé (2006) e realiza a geração de um modelo de estado com base na descrição de casos de uso. Para cada especificação de casos de uso, é apresentado o resultado do modelo de estado na própria interface do sistema. Durante esse processo, a ferramenta também faz a detecção de possíveis inconsistências da especificação, as quais podem ser validadas manualmente pelo usuário. Quanto à inserção de informações na ferramenta, é permitido tanto realizar a importação por meio de um arquivo de intercâmbio de metadados (do inglês, *Metadata Interchange - XMI*), quanto realizar a especificação manual por meio de um módulo específico da ferramenta.

Abordando funcionalidades relacionadas à manipulação de artefatos (categoria C_1) assim como modelagem, envolvendo tanto design manual quanto geração automática de diagramas (categoria C_4), os estudos de Chang et al. (2014), Ries, Capozucca e Guelfi

(2018) e Ibrahim e Ahmad (2010) apresentam suas soluções.

Chang et al. (2014) apresenta um mecanismo para coleta de requisitos de usuários, com a ajuda de um *template* de requisitos. A partir desse *template*, ocorre a geração automática de diferentes diagramas, baseados em SysML, tais como: diagramas de caso de uso e diagramas de atividades. Um documento inicial é gerado com base no *template* de requisitos da ferramenta e posteriormente pode ser ajustado por meio da ferramenta *Papyrus*. Após validados os requisitos contidos neste documento, os diagramas de caso de uso e atividades podem ser gerados utilizando a Linguagem para Especificação de Restrições em Objetos (do inglês, *Object Constraint Language - OCL*) e a Liguagem de Transformação ATLAS (do inglês, *ATLAS Transformation Language - ATL*), respectivamente.

No estudo de Ries, Capozucca e Guelfi (2018) é apresentada a metodologia intitulada "Messir". Essa metodologia utiliza uma linguagem de domínio específico (do inglês, *Domain-Specific Languages-DSL*), a qual faz parte de uma ferramenta *open source*, chamada *Excalibur*. Na ferramenta são inseridas as informações dos requisitos utilizando a metodologia mencionada e são gerados diagramas em UML. A ferramenta também permite a geração de um documento de análise de requisitos por meio de um documento *LaTeX*.

Já em Ibrahim e Ahmad (2010) é apresentada uma ferramenta para auxiliar na análise de requisitos e extração de diagrama de classes (*RACE - Requirements Analysis and Class Diagram Extraction*), a qual possui foco na aplicação de PLN e técnicas de ontologias de domínio para obter resultados acerca do diagrama de classes. Para o desenvolvimento da ferramenta, foi utilizado *OpenNPL* juntamente a um algoritmo para simplificação de palavras, desenvolvido com a linguagem C#. Com relação ao diagrama de classes, a ferramenta também dispõe de opções para visualização, adição, deleção e renomeação de classes, assim como ajuste dos relacionamentos das classes.

O único estudo incluído nas categorias C_1, C_2 e C_4 foca somente na geração de diagramas estáticos em UML. O estudo de Deeptimahanti e Sanyal (2011) apresenta a ferramenta nomeada UMGAR, que é uma extensão da ferramenta SUGAR proposta por Kumar e Sanyal (2008). UMGAR oferece suporte semi-automático para desenvolvimento de modelos UML estáticos e dinâmicos, a partir da especificação de requisitos em linguagem natural. Para cada cenário de caso de uso selecionado é disposto um diagrama de classes. Além disso, para evitar os problemas relacionados à edição e atualização dos diagramas em UML, a ferramenta permite a geração de arquivos XMI, os quais podem ser importados em quaisquer ferramentas de modelagem que possuam essa funcionalidade. Também é possível realizar a geração de códigos na linguagem *Java* para cada um dos diagramas de

classe criados, agilizando o processo de desenvolvimento do sistema a ser implementado.

Por fim, estando exclusivamente enquadrado nas categorias C_1, C_3 e C_4, Dascalu et al. (2007) desenvolveu a ferramenta nomeada STORM (*Supporting Tool for the Organization of Requirements Modelling*), a qual possui funcionalidades com foco na especificação do software, podendo ter um ótimo desempenho quando utilizada em conjunto com *Model-Driven Development (MDD)*.

A ferramenta proposta permite que o usuário tenha acesso a funcionalidades relacionadas a diversas atividades da especificação do software. Nesta atividade é permitido separar os requisitos entre funcionais e não funcionais, definindo a prioridade de cada um junto a uma matriz de rastreabilidade. Com relação à geração automática de diagramas, o sistema disponibiliza uma interface específica para preenchimento das informações a serem utilizadas no diagrama de caso de uso. Quanto à geração de documentação, é possível gerar os arquivos no formato .PDF.

3.4 AMEAÇAS À VALIDADE

A principal ameaça à realização do MS está relacionada ao fator humano, devido aos possíveis erros que podem ocorrer durante o processo de seleção dos estudos primários, acarretando na imprecisão da extração dos dados. Neste MS foram identificadas as seguintes ameaças à validade:

- **validade de conclusão:** representa a relação estatisticamente significativa entre o tratamento e os resultados (WOHLIN et al., 2012). Para reduzir as ameaças da seleção dos estudos por apenas um pesquisador, foi realizada uma análise de 10% dos estudos primários incluídos e excluídos, pela professora orientadora, sem haver interferência na qualidade do MS. Quanto à extração dos dados, devido à falta de informações por parte dos autores de alguns estudos, foi necessário buscar informações em outras fontes, para que alguns pontos tratados fossem esclarecidos.

- **validade interna:** refere-se à relação da causa entre o tratamento e os resultados (WOHLIN et al., 2012). A ameaça à validade interna foi identificada em casos que alguns estudos não puderam ser incluídos no processo de condução do MS, devido à sua indisponibilidade de acesso ou download.

- **validade de construção:** trata da relação entre teoria e aplicação (WOHLIN et al., 2012). A ameaça à validade de construção é identificada quando estudos relevantes

para QP são excluídos. Para mitigar essa ameça, foi desenvolvido um protocolo de pesquisa seguindo as diretrizes propostas por Kitchenham et al. (2010).

- **validade externa:** refere-se à generalização de resultados num campo externo ao estudo (WOHLIN et al., 2012). Com o protocolo desenvolvido e ter considerado a inclusão de trabalhos na língua portuguesa e inglesa, apesar da *string* de busca ter sido desenvolvida somente com termos em inglês, foi possível obter um conjunto representativo de estudos em maior escala.

3.5 Considerações Finais

Durante o desenvolvimento deste Capítulo, foram apresentados os resultados obtidos a partir da condução do MS, por meio do qual foram identificadas ferramentas existentes para especificação de software e geração de diagramas.

Após a leitura e extração dos dados dos 24 estudos primários selecionados, foi possível verificar características das ferramentas apresentadas, juntamente às suas respectivas vantagens e desvantagens. A partir disso, pode-se concluir que as ferramentas analisadas não contemplaram algumas atividades relacionadas à especificação de software, como por exemplos priorização de requisitos. Portanto, é necessária a concepção de uma ferramenta que permita ao usuário especificar, gerenciar os requisitos e gerar diagramas.

4 REMO - UMA FERRAMENTA PARA ESPECIFICAÇÃO DE REQUISITOS ÁGIL E MODELAGEM DE SISTEMAS

Este Capítulo apresenta a análise e projeto de uma ferramenta para auxiliar as atividades de elicitação, especificação, validação, gerenciamento de requisitos e modelagem de sistemas. Na Seção 4.1 é apresentada a análise comparativa entre as metodologias, a nomenclatura utilizada para a criação das funcionalidades, bem como a descrição detalhada dos módulos da ferramenta REMO. Na Seção 4.2 são descritas as funcionalidades, requisitos funcionais, não funcionais da ferramenta, bem como os diagramas de caso de uso e de classe. Na Seção 4.3 é apresentada a arquitetura desenvolvida utilizando o estilo arquitetural SOA. Por fim, na Seção 4.4 são pontuadas as considerações finais deste Capítulo.

4.1 Ferramenta REMO

Atualmente, a utilização de ferramentas visando auxiliar as atividades do processo de desenvolvimento de software estão cada vez mais presentes no cotidiano das organizações. Existe uma variedade dessas ferramentas, como SUGAR (KUMAR; SANYAL, 2008), EuRailCheck (CAVADA et al., 2009) e UMGAR (DEEPTIMAHANTI; SANYAL, 2011), as quais oferecem suporte à especificação de software. No entanto, os engenheiros de requisitos, muitas vezes, necessitam utilizar mais de uma ferramenta para a realização de suas atividades, como por exemplo a modelagem de software.

Neste contexto, o presente trabalho aborda a análise e o projeto de uma ferramenta chamada ***R****equirements sp****E****cification and Software* ***MO****deling (REMO)* que viabilizará a especificação de requisitos e a modelagem de software para o contexto ágil. A ferramenta foi projetada com base nas características das metodologias ágeis *Scrum*, *XP*, *FDD* e *OpenUP*. Essa análise foi realizada com a finalidade de realizar a combinação das melhores características e boas práticas de cada uma para gerar funcionalidades genéricas que possam ser utilizadas independente da metodologia adotada por uma organização. Essas metodologias foram selecionadas devido às suas especificidades e por serem as mais utilizadas em processos de desenvolvimento de software (RIGBY; SUTHERLAND;

TAKEUCHI, 2016).

Para realizar essa análise, primeiramente foram identificadas as características e as práticas de cada metodologia relacionadas aos módulos GPR, ERA e MOS, conforme pode ser visto na Tabela 5. Nesta Tabela os campos preenchidos com um "X" indicam que a metodologia possui a determinada prática, porém não possui um nome específico; e os campos com "$-$" indicam que durante a análise não foi identificada a prática na referente metodologia.

Tabela 5 - Análise comparativa entre as metodologias

Práticas	
Documentação inicial	
Papéis	Gerente de projeto, especialista do negócio, arc
Cliente presente	
Desenvolvimento iterativo	
Lista de funcionalidades	
Priorização das tarefas da iteração	
Lista de tarefas da iteração	
Projeto da Iteração	
Acompanhamento do processo de iterações	

Fonte: Autoria própria.

Após a análise das metodologias foram definidas as nomenclaturas adotadas para cada módulo da ferramenta REMO, conforme é apresentado na Tabela 6. Na primeira coluna são descritos os módulos da ferramenta, na segunda coluna as metodologias utilizadas como base e, por fim, na terceira coluna a nomenclatura adotada para a ferramenta REMO.

A descrição de cada nomenclatura definida para a ferramenta REMO é detalhada a seguir:

- **Papéis dos *stakeholders*:** consiste em todos as pessoas envolvidas que irão atuar diretamente no projeto para sua concretização. Dentre tais papéis é importante destacar *Product Owner*, *Scrum Master*, designer, desenvolvedor, testador e cliente;
- ***Product Backlog* com EUs:** contém o escopo do produto a ser desenvolvido. Consiste em um conjunto de funcionalidades utilizando o formato de EUs, as quais serão inseridas no *Product Backlog*;
- ***Sprint*:** será utilizada para definir o processo de desenvolvimento iterativo, no qual serão realizadas as tarefas visando as entregas parciais;

Tabela 6 - Nomenclatura da ferramenta REMO com base nas metodologias ágeis.

Módulos	Metodologia	Nomenclatura adotada pela REMO
GPR	*LSD*, *Scrum*, *XP*	Criação do projeto Descrição da visão do produto Definição dos papéis dos colaboradores
	OpenUP	Gerenciamento do projeto
ERA	*Scrum* *XP*	*Product Backlog* Estória do Usuário
	FDD	Prioridade e Dependência
	Scrum	*Sprint* Tarefas *Sprint Backlog*
	OpenUP	Gerenciamento de mudanças
MOS	*FDD*	Diagrama de caso de uso Diagrama de classe

Fonte: Autoria própria.

- ***Sprint Backlog* com Tarefas:** consiste no escopo definido para a execução da *Sprint*, contendo as tarefas detalhadas para as EUs;
- **Geração de Documento:** consiste na compilação das informações dos requisitos especificados na ferramenta em um documento que auxilie os *stakeholders* durante o desenvolvimento;
- **Modelagem:** geração dos diagramas de caso de uso e classes, com base nas informações sobre os requisitos inseridas na ferramenta.

Conforme pode ser visto Tabela 6, foram definidas nomenclaturas específicas que foram utilizadas na ferramenta *REMO*. Na Figura 15 é apresentada uma visão geral da ferramenta ilustrando os três módulos definidos: (1) Gerenciamento de Projetos (GPR); (2) Especificação de Requisitos Ágil (ERA); e (3) Modelagem de Software (MOS). Nas próximas subseções serão realizadas as descrições de cada um dos módulos, sendo o GPR em 4.1.1, ERA em 4.1.2 e MOS em 4.1.3.

4.1.1 Módulo 1: Gerenciamento de Projetos (GPR)

Os sistemas computacionais existem para solucionar e/ou automatizar as necessidades identificadas pelos usuários. Essas necessidades são analisadas a partir de processos já existentes de clientes, os quais são transformados em software. Para auxiliar o desenvolvimento de software, o gerenciamento de projetos é fundamental, uma vez que

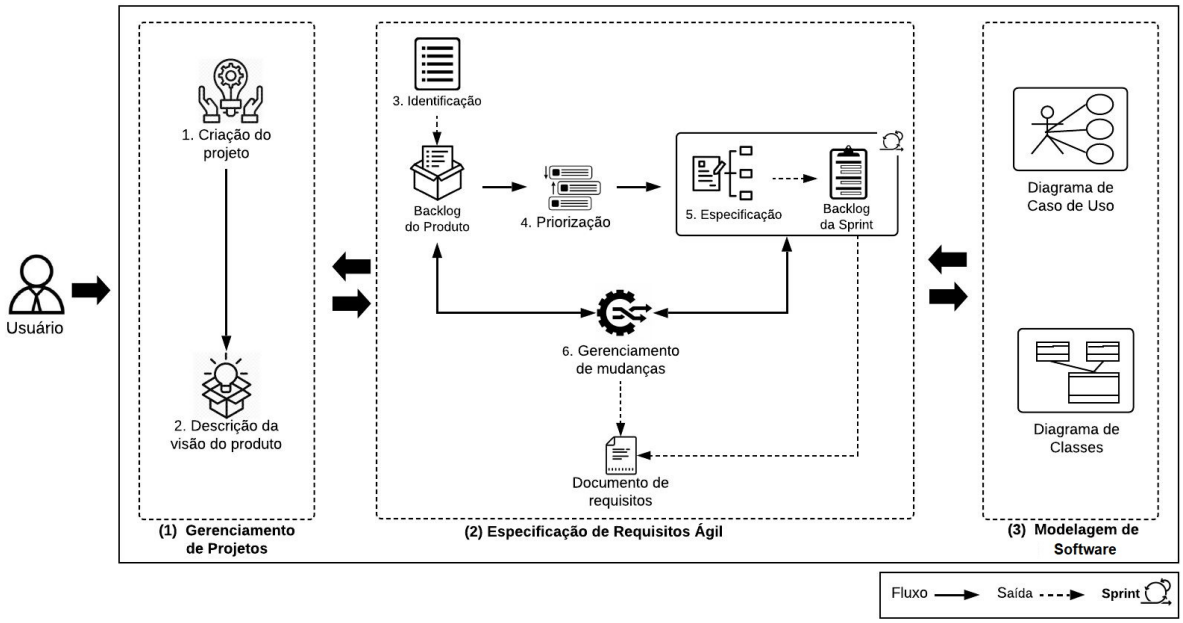

Figura 15 - Visão geral da ferramenta *REMO*.
Fonte: Autoria própria.

este permite aplicar conhecimento, habilidades, ferramentas e técnicas às atividades do projeto para atender aos seus requisitos (PMI, 2018). Portanto, o módulo GPR (módulo 1) é composto por duas fases: (1) criação do projeto; e (2) descrição da visão do produto, conforme apresentado na Figura 16.

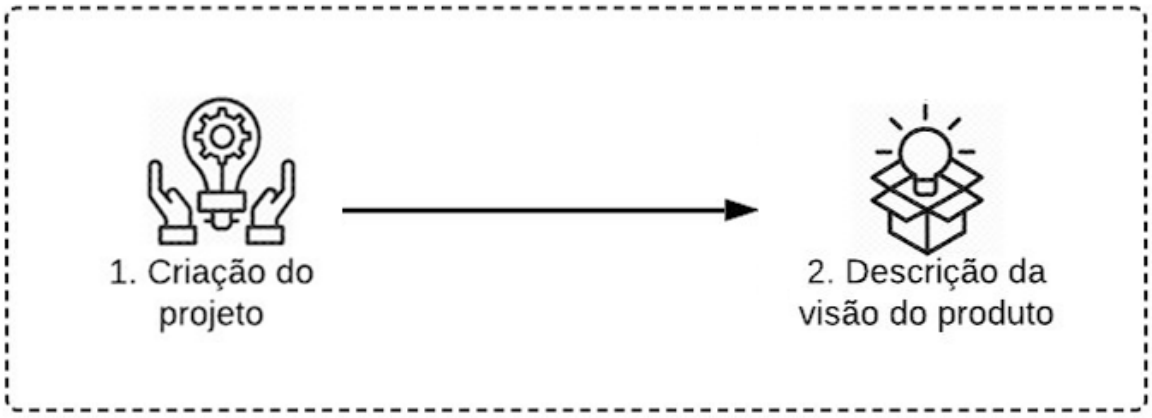

Figura 16 - Módulo GPR da ferramenta *REMO*.
Fonte: Autoria própria.

A partir do momento que um projeto é criado (fase 1), o ponto de partida consiste em identificar visão do produto (fase 2). Nesta fase, o objetivo é compreender as necessidades dos usuários e dos negócios, bem como ajudar a mapear soluções que

atendam as expectativas dos *stakeholders*. Ao final desta etapa, será obtida uma clara visão do que precisa ser feito, auxiliando na garantia de entrega do que realmente satisfaz os objetivos identificados. Caso o projeto já esteja criado, é possível editá-lo. O modelo a ser utilizado para a visão do produto será o chamado "Teste do Elevador" (do inglês, *pitch*). Um exemplo de *pitch* é apresentado na Tabela 7.

Tabela 7 - Exemplo de *pitch*.

*"**REMO** é uma **ferramenta de especificação de requisitos ágil e modelagem de software***

Fonte: Autoria própria.

O *pitch* é uma breve descrição que deve conter informações úteis e diretas para uma comunicação efetiva, apresentando todas as informações necessárias (SATOW; EISELE, 2012). Com o *pitch* será possível apresentar quem é o público-alvo do produto, qual o problema desse público ou oportunidade de mercado que será suprida. Portanto, a partir da descrição da visão do produto os requisitos podem ser identificados, priorizados, especificados e gerenciados no próximo módulo.

A Figura 17 apresenta a sequencia de passos para a utilização do módulo GPR (módulo 1), assim como a demonstração de quais serviços são utilizados em cada etapa.

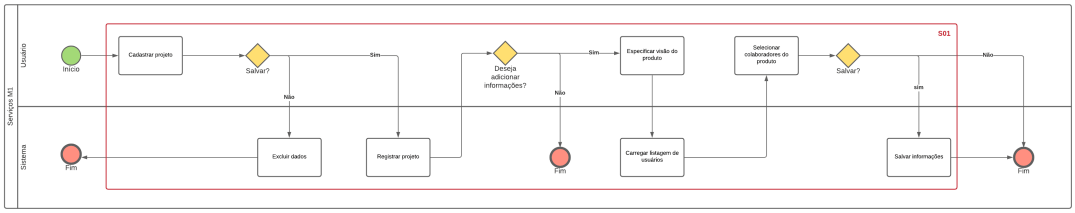

Figura 17 - Fluxograma da utilização do módulo GPR.
Fonte: Autoria própria.

4.1.2 Módulo 2: Especificação de Requisitos Ágil (ERA)

Este módulo visa orientar o processo de identificação, priorização e especificação dos requisitos, a fim de garantir a qualidade das informações coletas. O módulo ERA (módulo 2) é composto por quatro fases inter-relacionadas, conforme é apresentado na Figura 18.

A partir da definição do projeto no módulo GPR (módulo 1), a fase de identificação (fase 3) é iniciada visando coletar os requisitos de um sistema a ser desenvolvido. Esta fase

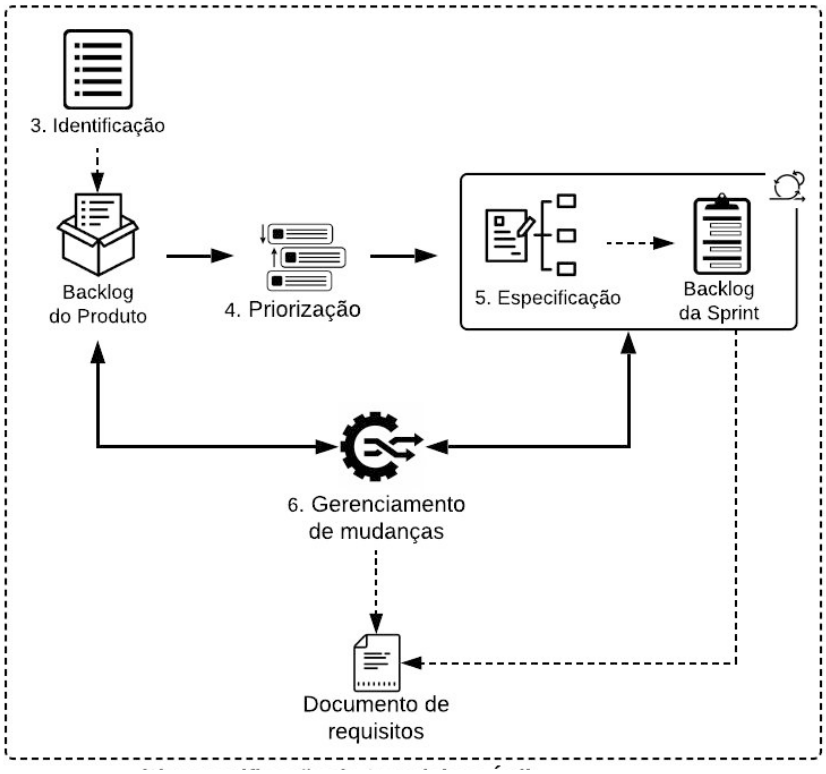

Figura 18 - Módulo ERA da ferramenta *REMO*.
Fonte: Autoria própria

consiste na elicitação e análise dos requisitos por meio de Estórias de Usuário (EUs), as quais são adicionadas no *Backlog* do Produto. As EUs criadas representam as descrições comportamentais do sistema a ser desenvolvido. As EUs seguem um formato específico que foi adaptado de Leffingwell (2011), conforme é apresentada na Tabela 8.

Tabela 8 - Exemplo de *EU*

"Como ***Usuário****, eu preciso* ***Cadastrar Usuários*** *para* ***controlar quem tem permissão para***

Fonte: Autoria própria

Além de conter as descrições comportamentais, na especificação das EUs serão definidas as estimativas e prioridades para o desenvolvimento de cada uma delas. As estimativas de cada EU são definidas considerando o tempo para o seu desenvolvimento, e o nível de prioridade é definido pelos *stakeholders*.

Com a lista de requisitos identificada, o foco da fase de priorização (fase 4) consiste na priorização dos requisitos listados no *backlog* do produto. Os requisitos serão priorizados automaticamente pela ferramenta de acordo com o grau de prioridade e a estimativa.

Aqueles requisitos com prioridades elevadas serão associados pelo usuário na *sprint* atual criada. Requisitos de sistema não priorizados permanecem aguardando priorização no *backlog* do produto.

A partir dos requisitos priorizados, a fase de especificação dos requisitos (fase 5) ocorre em cada *sprint* criada. Nessa fase, as EUs associadas a cada *sprint* serão detalhadas e poderão ser divididas em tarefas menores. Cada tarefa conterá descrição, responsável, status, estimativa, prioridade, dependências, bem como arquivos externos que podem ser anexados como por exemplo, interfaces, arquitetura. Como resultado desta fase será obtido o *backlog* da *sprint* e o documento de requisitos.

A etapa de gerenciamento de mudanças (fase 6) consiste no processo de controlar as mudanças identificadas nos requisitos conforme a evolução do projeto. É nesta etapa que são realizadas as alterações, correções e validações de requisitos já especificados, bem como a inclusão de novos no *backlog* do produto. Esta etapa tem como saída o documento de requisitos atualizado, bem como atualizar os artefatos gerados com base nessas informações.

Após as fases 5 e 6, um documento de requisitos simplificado pode ser gerado automaticamente a fim de auxiliar a compreensão do time e de todos os *stakeholders* envolvidos em cada projeto. As informações geradas neste módulo serão utilizadas como entrada para o módulo MOS (módulo 3), conforme apresentado na Figura 15, para a geração dos diagramas de caso de uso e de classe.

A Figura 19 apresenta a sequência de passos para a utilização do módulo ERA (módulo 2), assim como a demonstração de quais serviços são utilizados em cada etapa.

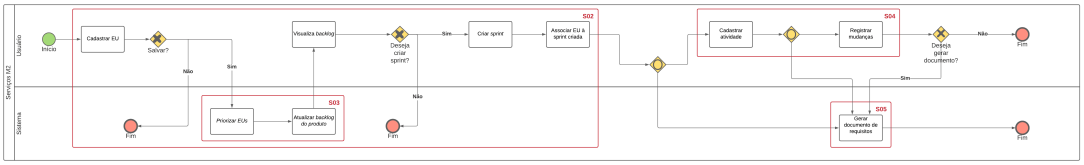

Figura 19 - Fluxograma da utilização do módulo ERA.
Fonte: Autoria própria.

4.1.3 Módulo 3: Modelagem de Software (MOS)

Os diagramas UML são fundamentais em projetos de desenvolvimento de software, devido auxiliarem na compreensão do comportamento e da estrutura do sistema a

ser desenvolvido. Neste módulo os diagramas de caso de uso e de classe são gerados automaticamente a partir dos requisitos especificados no módulo ERA (módulo 2). A Figura 20 apresenta o módulo MOS da ferramenta REMO.

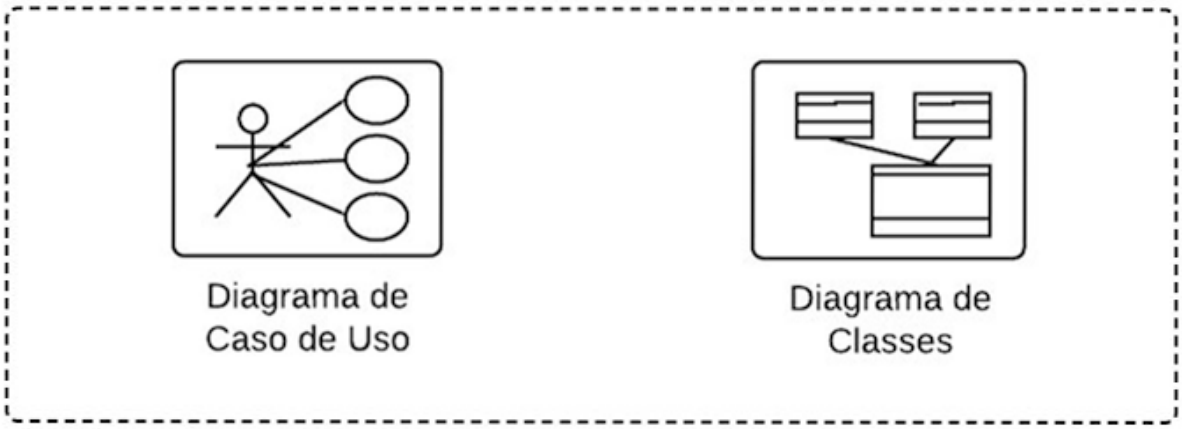

Figura 20 - Módulo MOS da ferramenta *REMO*.
Fonte: Autoria própria.

O diagrama de caso de uso auxilia na compreensão das necessidades do cliente, uma vez que permite identificar as principais funcionalidades do sistema e como os usuários interagem com estas. Por outro lado, o diagrama de classes apresenta as informações relacionadas à estrutura dos sistemas, ou seja, detalhes de implementação, como atributos, métodos e como as classes se relacionam. Esses dois diagramas foram selecionados por serem os mais abordados em ferramentas e/ou abordagens identificadas no MS conduzido, conforme pode ser visto na Seção 3.3.

Os diagramas serão gerados para cada projeto, e poderão ser atualizados a partir das modificações realizadas no módulo ERA (módulo 2). O diagrama de caso de uso será gerado por meio da extração de palavras-chaves de cada EU e o diagrama de classe com base nas informações descritas nas tarefas criadas na fase de especificação de requisitos (fase 5). As classes, atributos, métodos e relacionamentos serão identificadas automaticamente por meio de restrições de campos disponibilizadas nas tarefas.

A Figura 21 apresenta a sequencia de passos para a utilização do módulo MOS (módulo 3), assim como a demonstração de quais serviços são utilizados em cada etapa.

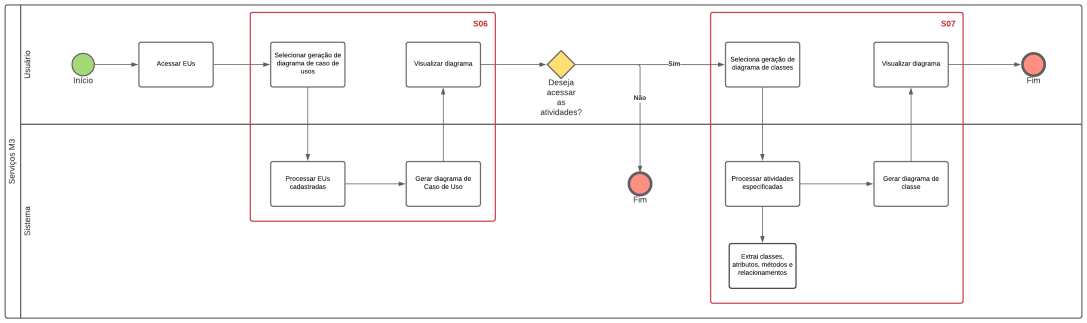

Figura 21 - Fluxograma da utilização do módulo MOS.

Fonte: Autoria própria.

4.2 Funcionalidades

Para o desenvolvimento da ferramenta foram definidas oito funcionalidades (F= f_1, f_2, f_3, ..., f_8). Na Tabela 9 são apresentadas essas funcionalidades de acordo com os módulos da ferramenta apresentados na Figura 15.

Tabela 9 - Funcionalidades da ferramenta REMO.

Módulos	**Funcionalidades**
Gerenciamento de Projetos (GPR)	f_1 - Cadastro de projeto. f_2 - Gerenciamento de projeto.
Especificação de Requisitos Ágil (ERA)	f_3 - Identificação de requisitos. f_4 - Priorização de requisitos. f_5 - Especificação de requisitos. f_6 - Gerenciamento de mudanças.
Modelagem de Software (MOS)	f_7 - Geração de diagrama de casos de uso. f_8 - Geração de diagrama de classes.

Fonte: Autoria Própria.

Para o desenvolvimento de cada funcionalidade, um ou mais RFs foram identificados para o desenvolvimento da ferramenta. Na Tabela 4.2 são listados os 25 RFs junto às suas respectivas descrições, distribuídos entre as oito funcionalidades apresentadas na Tabela 9.

c|c|p12.5cm

f_1	RF01	O usuário realiza login. O acesso da ferramenta pode ser obtido por meio do nome de usuário ou e-mail e senha.
	RF02	O usuário realiza seu cadastro. O cadastro deve ter: nome, senha, nome do usuário e e-mail.

RF03 O usuário cadastra um projeto. O projeto deve possuir ID, nome, estimativa (horas planejadas).

RF04 O usuário especifica a visão do produto. A descrição da visão do produto será realizada com base no *template* de um *pitch*.

RF05 O usuário identifica os colaboradores do projeto. Os colaboradores podem ser o time de desenvolvimento, o responsável pelo projeto e/ou o cliente.

f_2 RF06 O usuário gerencia os projetos, editando ou arquivando-os.

f_3 RF07 O usuário cadastra as EUs. Para cada requisito, o usuário pode criar uma ou várias EUs. As EUs devem possuir: ID, Nome, Status (Iniciar, Desenvolvendo, Finalizado), Estimativa (em horas), Prioridade (alta (5), média (3), baixa (1)), descrição da necessidade do cliente em texto com formato específico.

RF08 O usuário acompanha as EUs criadas, visualizando informações como: nome, status, prioridade e estimativa.

RF09 O usuário gerencia as EUs, editando ou arquivando-as.

RF10 O usuário deve estimar as EUs em termos de esforço (horas).

RF11 O sistema apresenta a listagem das EUs inseridas no *backlog* do produto, as quais não estão vinculados a nenhuma *sprint*.

f_4 RF12 O sistema ordena automaticamente os requisitos com base na prioridade e na estimativa.

RF13 O usuário vinculará os requisitos priorizados na *sprint* a ser criada.

RF14 O usuário cria *sprint*. A *sprint* deve ter: ID, nome, data de início e de término.

RF15 O usuário pode consultar as *sprints* criadas para o desenvolvimento do projeto.

RF16 O usuário pode gerenciar as *sprints* criadas, editando ou arquivando-as.

f_5 RF17 O usuário cadastra as tarefas. Para cada EU criada, o usuário pode criar uma ou várias tarefas. As atividades devem conter: ID, responsáveis, descrição, status (analisando, analisado, desenvolvendo, desenvolvido, testando, entregue), prioridade (muito baixa, baixa, média, alta, muito alta) e dependência.

RF18 O usuário cadastra recurso na tarefa. O recurso consiste no *upload* de arquivo externo que possa auxiliar na compreensão da atividade cadastrada.

RF19 O usuário gerencia as tarefas editando ou arquivando-as.

f_6 RF20 O usuário registra as modificações realizadas nas tarefas. O registro deve conter: data, descrição da alteração e responsável pela alteração.

RF21 O usuário pode visualizar histórico das alterações realizadas.

RF22 O usuário pode desfazer uma alteração realizada, ou retornar para uma versão anterior, a partir do histórico de alterações.

RF23 O sistema gera documento de requisitos com base em um *template*, contendo as informações especificadas na ferramenta.

f_7 RF24 O sistema gera o diagrama de caso de uso automaticamente, com base nas estórias de usuário identificadas.

f_8 RF25 O sistema gera o diagrama de classes automaticamente, com base nas EUs e tarefas criadas.

Fonte: Autoria própria.

A partir dos 25 RFs apresentados, foram identificados nove Requisitos Não Funcionais (RNFs), a fim de garantir a qualidade do produto. Para o levantamento destes RNFs foi utilizado como base a ISO/IEC 25010 (ISO, 2011). Esses requisitos estão relacionados a seis características de qualidade do sistema, distribuídas entre suas respectivas sub-características, conforme apresentado na Tabela 4.2.

c|c|p7.5cm|c

Compatibilidade RNF01 A arquitetura deve permitir que possam ser executadas tarefas paralelamente, mantendo a integridade das informações.
RNF02 A arquitetura deve permitir que todos os componentes do sistema possam trocar informações e reutiliza-las. RF01...RF25

Usabilidade RNF03 A interface de usuário deve ser de fácil operação e controle dos componentes do sistema. RF01...RF25
RNF04 A interface do usuário deve permitir que pessoas com diferentes características e capacidades possam acessar todas as funcionalidades do sistema. RF01...RF25

Confiabilidade RNF05 A arquitetura deve permitir que o sistema se mantenha acessível e operável sempre quando necessário, para utilização RF01...RF25

Segurança RNF06 A arquitetura deve fornecer segurança dos dados usuários RF01
RNF07 A arquitetura deve fornecer a integridade dos projetos cadastrados RF03
RNF08 A arquitetura deve permitir que sejam realizadas alterações na ferramenta, sem que sejam afetados os demais módulos. RF01...RF25

Portabilidade RNF09 A arquitetura deve permitir que o sistema possa se adaptar a diferentes ambientes de utilização. RF01...RF25

Fonte: Autoria própria.

Para facilitar a compreensão das funcionalidades do sistema e como estão relacionadas, foi desenvolvido o diagrama de caso de uso da ferramenta, conforme é apresentado na Figura 22. Entre os atores envolvidos no diagrama estão somente o usuário que irá interagir com a ferramenta e o sistema, o qual realiza algumas ações automaticamente, como geração de diagramas e de documento de requisitos, por exemplo.

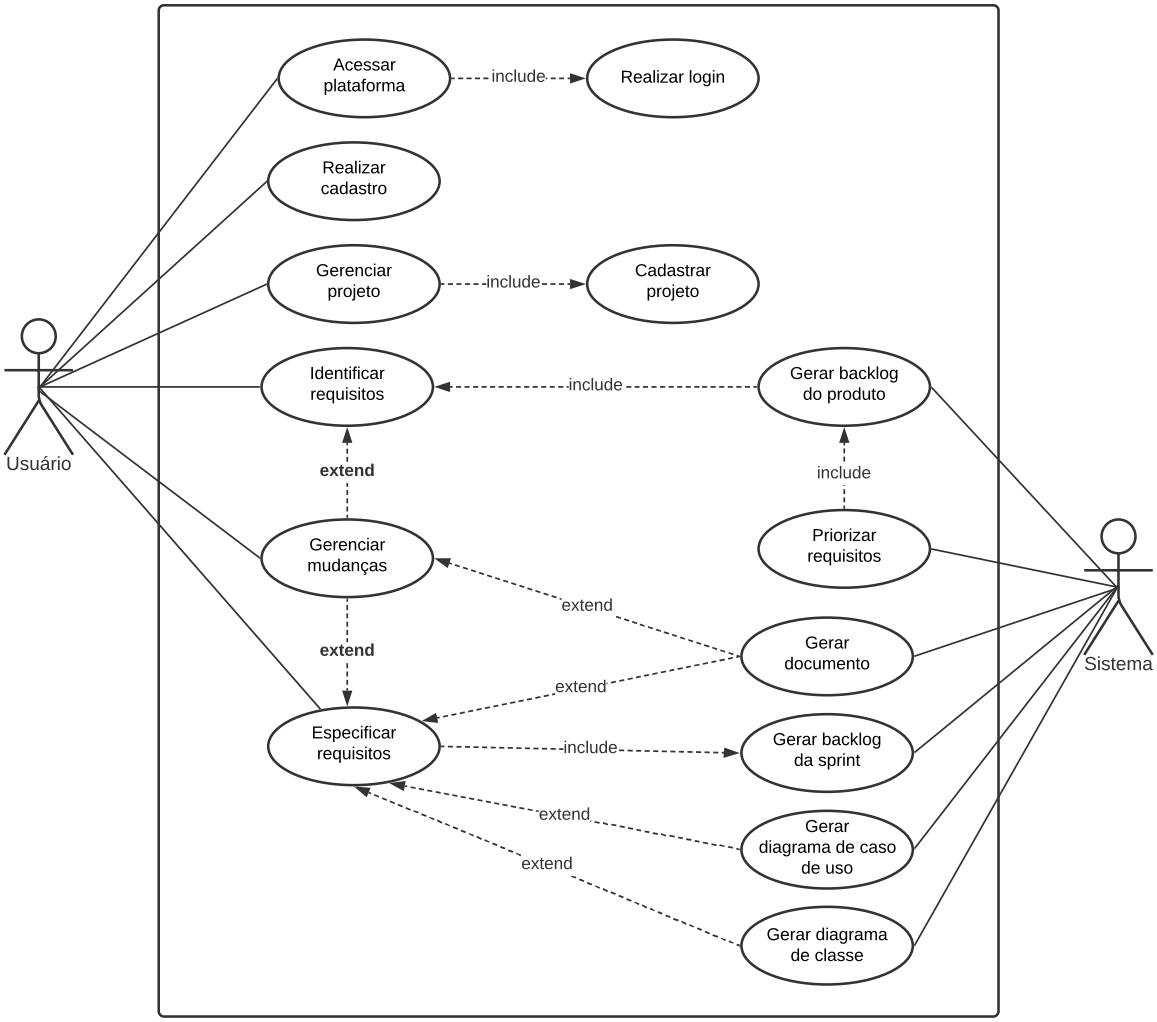

Figura 22 - Diagrama de caso de uso da ferramenta *REMO*.
Fonte: Autoria própria

Além disso, alguns casos de uso são obrigatórios em decorrência da execução de um outro caso de uso, como por exemplo a realização de login ao acessar a ferramenta

(*include)*. Por outro lado, também foram identificados casos de uso que são opcionais, como por exemplo a geração do documento de requisitos após ter especificado os requisitos (*extend*).

Após definidas as informações que abrangem as funcionalidades da ferramenta bem como suas restrições, foi necessário definir a parte estrutural da ferramenta para facilitar o seu desenvolvimento. Como a ferramenta tem foco na modularidade e a possibilidade de expansão de funcionalidades, as informações sobre os serviços disponibilizados e a arquitetura que disponibilizará a aplicação dos mesmos é de extrema importância.

4.3 Arquitetura da REMO

Outra contribuição do presente trabalho está relacionada à arquitetura da ferramenta. A criação de uma arquitetura de software é fundamental, pois auxilia no design do sistema, define como o sistema será estruturado, como seus elementos trabalharão juntos e facilita o reúso. Para facilitar o desenvolvimento da arquitetura SOA da ferramenta, foi modelado o diagrama de classes da ferramenta, conforme é apresentado na Figura 23.

Dentre as principais classes, as quais envolvem os dados obrigatórios para a utilização completa da ferramenta estão a classe "Projeto", "Estória Usuário", "Sprint", "Tarefa" e "Diagrama". As demais classes, como a "Usuário" e "Histórico Mudança", são as classes auxiliares, sendo utilizadas para delimitação de acesso à ferramenta e registro de alterações, respectivamente. Além disso, as entidades *"enumeration"* definem as opções *default* disponíveis para os respectivos atributos.

Para o projeto da ferramenta será utilizado o estilo arquitetural SOA. Este estilo foi selecionado devido proporcionar: *(i)* uma fácil adaptação das aplicações às novas tecnologias; *(ii)* alto encapsulamento das funcionalidades; *(iii)* facilidade para criar novos serviços compondo serviços já existentes; *(iv)* acessibilidade na integração de aplicações com outros sistemas; e *(v)* redução de custos nas organizações para cooperação entre elas.

É importante ressaltar que quanto à evolução da ferramenta, a SOA auxilia com relação a uma futura expansão da ferramenta para outras aplicações. Nesse caso, os serviços desenvolvidos para a *REMO* poderiam ser reutilizados pelos demais ambientes de implantação, por meio de um *Web service*, sem a necessidade de desenvolver novamente, a funcionalidade disponibilizada por esse serviço. Quanto aos benefícios da SOA relacionados à manutenção, é importante destacar que os serviços podem ser alterados, visando melhorias ou correções, sem afetar o funcionamento dos demais serviços e a utilidade da funcionalidade

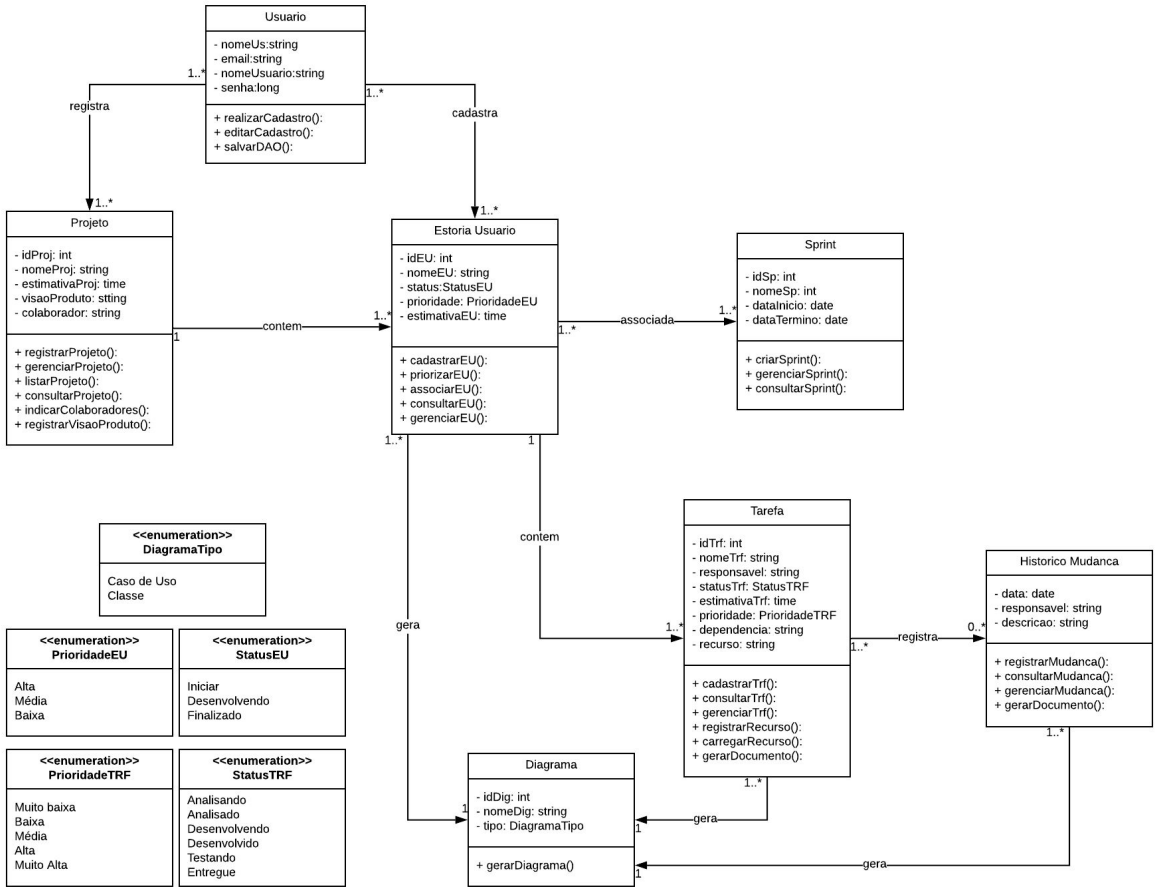

Figura 23 - Diagrama de classes da ferramenta *REMO*.
Fonte: Autoria própria;

disponibilizada por esse serviço.

A Tabela 10 apresenta os serviços levantados para cada um dos módulos da ferramenta *REMO*, especificados na Seção 4.2. Foi realizado o levantamento de sete serviços para atender cada um dos três módulos da *REMO*. O serviço S_1 é responsável pelas duas funcionalidades do módulo GPR. Já para os demais módulos (ERA e MOS) foi designado um serviço para cada funcionalidade restante (do serviço S_2 ao S_7), devido a necessidade de garantia de disponibilidade e interdependência dessas funcionalidades.

A partir dos serviços identificados, a Figura 24 ilustra a representação da arquitetura criada para a ferramenta *REMO*. Na Figura é possível observar o relacionamento da camada de serviços com as demais camadas do sistema, o que representa a interação dos serviços disponibilizados com todas as outras partes do sistema. Tanto a camada de apresentação quanto a camada de dados possuem interação com o usuário, uma vez que os dados são inseridos pelo usuário e a camada de apresentação é o meio pelo qual o usuário tem acesso

à ferramenta.

Além da relação das camadas de apresentação e dados, as capacidades também estão relacionadas aos serviços. Isso se dá pelo fato de que os serviços disponibilizados precisam garantir que as capacidades sejam seguidas para o funcionamento correto do sistema, sem prejuízos ao usuário. As cinco capacidades que devem serem cumpridas são: compatibilidade; usabilidade; segurança; confiabilidade; e portabilidade.

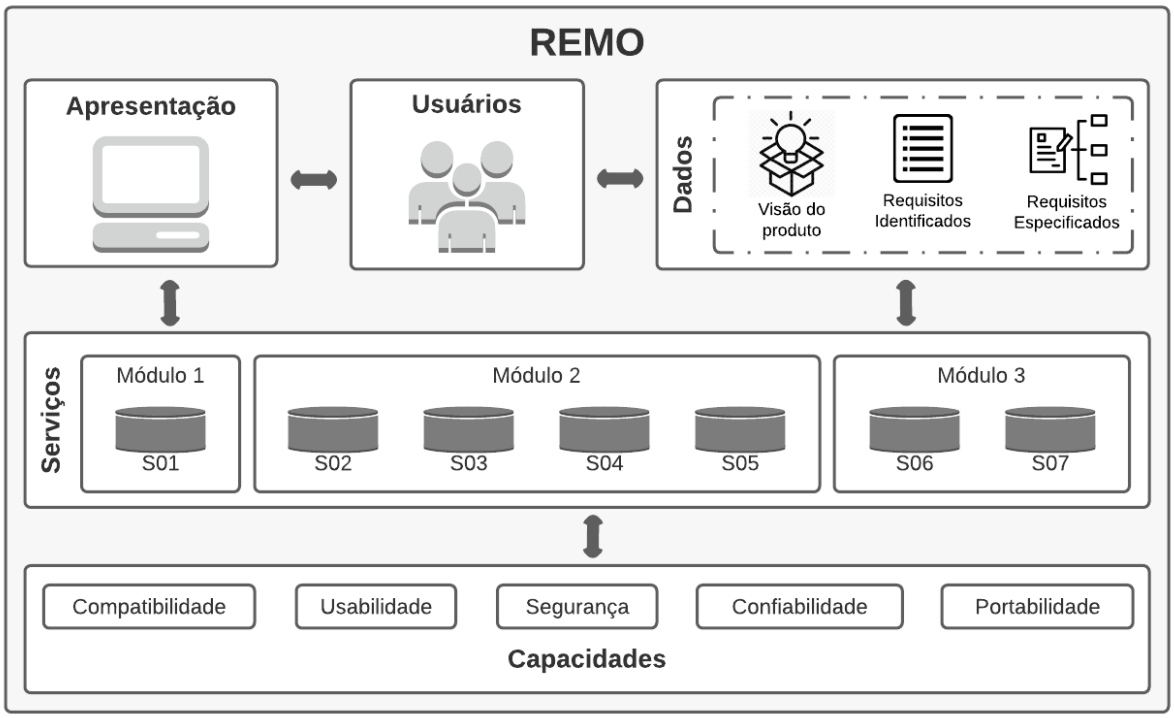

Figura 24 - Representação da arquitetura da ferramenta *REMO*.
Fonte: Autoria própria.

Portanto, a arquitetura proposta buscou fornecer suporte para que todos os módulos possam operar em conjunto, ou seja, cada um disponibilizando as suas respectivas funcionalidades sem que haja interferências operacionais entre as mesmas. Além disso, é importante destacar que esta arquitetura permite que futuramente sejam aplicadas melhorias, a fim de contemplar novas funcionalidades.

4.4 Considerações Finais

Conforme os resultados apresentados no Capítulo 3, é possível observar a ausência de uma ferramenta que contemple diferentes atividades a serem realizadas por um engenheiro de requisitos. Nesse contexto, o principal objetivo da ferramenta *REMO*

é auxiliar a execução das atividades relacionadas à elicitação, especificação, validação, gerenciamento de requisitos e modelagem de software. Portanto, um dos diferenciais da ferramenta é permitir que a partir das modificações realizadas nos requisitos, os respectivos diagramas sejam gerados automaticamente.

Para análise e projeto da ferramenta, foi realizada a especificação de requisitos funcionais e não funcionais, a geração de diagramas comportamentais e estruturais, bem como o desenvolvimento de uma arquitetura utilizando o estilo arquitetural SOA. No próximo Capítulo são apresentados a avaliação da arquitetura e os resultados alcançados.

Tabela 10 - Serviços disponibilizados pela ferramenta *REMO*.

Módulo	Funcionalidades	ID Serviço
GPR	f_1 - Cadastro de projeto	S_1
	f_2 - Gerenciamento de projeto	
ERA	f_3 - Identificação de requisitos	S_2
	f_4 - Priorização de requisitos	S_3
	f_5 - Especificação de requisitos	S_4
	f_6 - Gerenciamento de mudanças	S_5
MOS	f_7 - Geração de diagrama de casos de uso	S_6
	f_8 - Geração de diagrama de classes	S_7

Fonte: Autoria própria.

5 AVALIAÇÃO EXPERIMENTAL E RESULTADOS

Durante a condução deste trabalho, foram desenvolvidos artefatos e coletadas informações a fim de avaliar a ferramenta proposta. Nesta direção, este Capítulo apresenta um estudo de caso conduzido para para avaliar a arquitetura da ferramenta REMO apresentada no Capítulo 4. Na Seção 5.1 é apresentado o planejamento do estudo de caso, no qual foi utilizado o modelo GQM para definir os objetivos deste estudo de caso. Na Seção 5.2 são descritas como foi realizada a análise dos dados, a discussão acerca dos resultados e as ameaças à validade relacionadas ao estudo de caso. Por fim, na Seção 5.3 são apresentadas as considerações finais deste Capítulo.

5.1 Estudo de Caso

O estudo de caso teve como objetivo avaliar eficácia da arquitetura proposta para o desenvolvimento da ferramenta REMO. É importante enfatizar que o estudo foi conduzido para identificar possíveis melhorias do ponto de vista arquitetônico, ou seja, se o projeto de arquitetura impacta na capacidade da ferramenta REMO de cumprir seus objetivos de negócios e satisfazer seus requisitos. Neste estudo, foram utilizadas as diretrizes recomendadas por (WOHLIN et al., 2012) para a condução do estudo de caso.

5.1.1 Planejamento e Design do Estudo de Caso

O modelo *Goal-Question-Metric* (GQM) (BASILI; WEISS, 1984) foi utilizado para definir o objetivo do estudo de caso que pode ser resumido da seguinte forma:

"Analisar a ***REMO-AS*** *com o propósito de* ***avaliar o design da arquitetura proposta*** *a partir do ponto de vista de* ***especialistas em arquitetura*** *no contexto da* ***ferramenta REMO****."*

Para atingir o objetivo, foi realizado um estudo de caso exploratório, qualitativo e holístico de caso único que foca na ferramenta REMO e observa exclusivamente a

arquitetura REMO-AS.Neste sentido, as seguintes Questões de Pesquisa (QPs) foram investigadas:

- ***QP*$_1$:** Quão eficaz é a arquitetura REMO-AS para auxiliar o desenvolvimento da ferramenta REMO visando cumprir seus objetivos de negócio?
- ***QP*$_2$:** Quais melhorias podem ser realizadas no design da arquitetura REMO-AS?

O objetivo da primeira questão de pesquisa (QP_1) visa verificar se a arquitetura REMO-AS é eficaz para apoiar o desenvolvimento da ferramenta REMO. Para responder esta QP foram analisados a visão geral da arquitetura, visão arquitetural e o design arquitetural. Por fim, a segunda questão (QP_2) visou identificar possíveis melhorias a respeito da REMO-AS relacionadas à visão geral da arquitetura, visão e design arquitetural.

Este estudo foi realizado com 12 especialistas em arquitetura de software de seis Universidades diferentes. Os especialistas possuem experiência variando entre inferior a um ano e superior a seis anos. O grupo de especialistas foi constituído por uma mulher e 11 homens com idade entre 22 e 50 anos.

5.1.2 Preparação, coleta e análise dos dados

Para auxiliar a coleta dos dados foi utilizado como método o questionário, o qual foi selecionado devido a sua adequação técnica, praticidade de uso e considerações éticas envolvidas no processo, para tentar responder as questões que haviam sido formuladas.

O questionário foi desenvolvido baseado nas QP_1 e QP_2 e utilizando a ferramenta *Google Forms*[1]. O questionário (Apêndice **??**) é semiestruturado, contendo 23 questões, divididas entre questões abertas, de múltipla escolha e com escala de avaliação. Para responder as questões com escala de avaliação foi utilizada a escala *Likert* que consiste em cinco níveis, sendo o nível mais alto representado por "Concordo totalmente" e o nível mais baixo "Discordo totalmente" (LIKERT, 1932).

Essas questões foram agrupadas em quatro seções: *(i)* Perfil dos participantes; *(ii)* Visão geral da arquitetura; *(iii)* Visão arquitetural; e *(iv)* Design arquitetural. A primeira seção do questionário visou identificar perfil dos participantes assim como sua experiência na área de Arquitetura de Software. A segunda seção teve enfoque na avaliação arquitetural, por meio de uma visão geral da arquitetura proposta. A terceira seção buscou

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

avaliar as informações relacionadas aos diagramas desenvolvidos para apoiar a REMO-AS. Por fim, a última seção visou identificar informações sobre a capacidade arquitetural além de relacionamentos entre os dados da arquitetura. É importante destacar que ao final de cada seção foi disponibilizada uma questão aberta para permitir que os especialistas deixassem um comentário ou sugestão sobre a REMO-AS.

Para auxiliar os especialistas a responderem o questionário, foi disponibilizado um material de apoio sobre a ferramenta REMO (Apêndice **??**). Nesse material foi apresentada uma visão geral da ferramenta REMO, as funcionalidades, os requisitos funcionais e não funcionais, a arquitetura desenvolvida e os diagramas de caso de uso e classes.

A análise dos resultados, obtidos a partir da aplicação do questionário, foi realizada levando em consideração tanto fatores relacionados aos participantes especialistas quanto aos fatores relacionados diretamente ao trabalho. Esses especialistas foram selecionados de foram não probabilística utilizando a técnica por conveniência, por meio da seleção de candidatos conhecidos. Sobre eles, foi analisado o tempo de experiência dos mesmos, assim como a sua formação. No que concerne às peculiaridades deste trabalho, algumas sugestões não foram aplicadas, devido a exigência de detalhes sobre a arquitetura, os quais ainda não foram especificados.

5.2 Análise e Discussão dos Resultados

Nesta seção são detalhados os resultados alcançados a partir da condução do estudo de caso. O primeiro resultado está relacionado aos perfis dos participantes. Somente um dos 12 participantes se definiu sendo do sexo feminino, o que representou uma participação de 91,7% (11/12) do público masculino. Quanto à idade dos participantes, a faixa etária com maior número de participantes foi dos 22 aos 30 anos, com 58,3% (7/12) do total de participantes. Em segundo lugar ficou a faixa etária dos 31 aos 40 anos, com 25% (3/12). Por fim, a representatividade dos participantes com mais de 40 anos de idade, foi de 16,7% (2/12).

Quanto aos aspectos demográficos dos participantes desse estudo, foram retornadas respostas de oito municípios divididos entre seis unidades federativas do Brasil, dentre elas Minas Gerais (MG), Paraná (PR), Santa Catarina (SC), Goiás (GO), São Paulo (SP) e Bahia (BA). A Figura 25 apresenta a relação de quantidade de participantes por cidade e estado onde cada um se disse localizado.

Com base na Figura, é possível verificar que a cidade contendo mais participantes

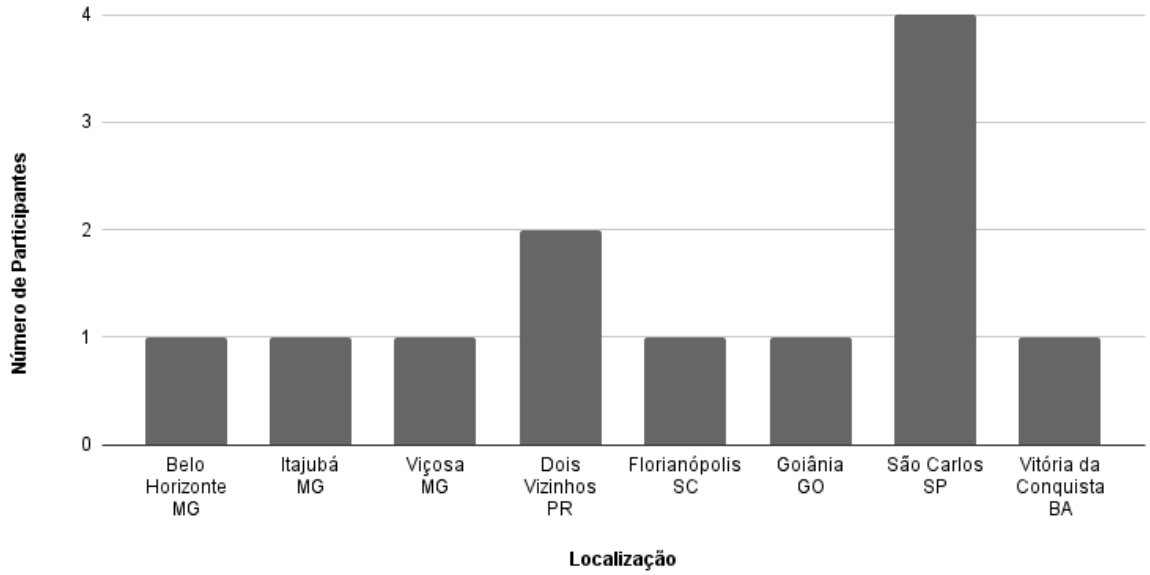

Figura 25 - Número de participantes de acordo com os seus respectivos estados e cidades.

Fonte: Autoria própria.

é São Carlos - SP, com um total de 4 participantes, e a segunda cidade é Dois Vizinhos - PR com 2 participantes. Os outros três participantes estão divididos em diferentes cidades de MG, dentre elas Belo Horizonte, Itajubá e Viçosa. Os demais participantes são de Florianópolis - SC, Goiânia - GO e Vitória da Conquista - BA.

É importante ressaltar que dentre os fatores levados em consideração para realizar as melhorias na arquitetura da ferramenta foram o grau de formação e o tempo de experiência com AS. A Figura 26 apresenta a relação dessas duas informações, mostrando a quantidade total de participantes que se enquadraram em cada uma das especificações.

Analisando a Figura 26, é possível notar que os participantes do maior agrupamento têm o grau de Mestre e possuem experiência de 2 a 4 anos na área de AS. Por outro lado, somente um participante possui o Ensino Superior completo e com 1 a 2 dois anos de experiência. Além disso, é possível notar que 75% (9/12) dos participantes possui grau de Mestre ou Doutor e possuem acima de 2 anos de experiência em AS, os quais contribuíram para a melhoria da arquitetura proposta.

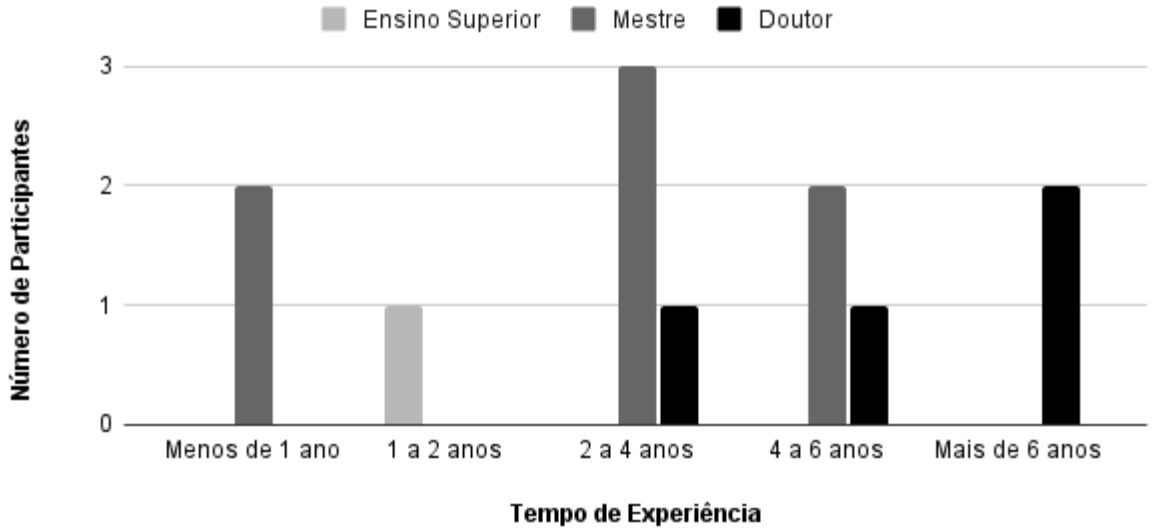

Figura 26 - Número de participantes de acordo com a sua formação e experiência com AS.

Fonte: Autoria própria.

5.2.1 QP_1: Quão eficaz é a arquitetura REMO-AS para auxiliar o desenvolvimento da ferramenta REMO visando cumprir seus objetivos de negócio?

Essa QP visa analisar as informações referentes à avaliação geral da arquitetura, considerando aspectos como objetivo da ferramenta, artefatos disponibilizados, conceitos utilizados para o desenvolvimento da arquitetura além de questões relacionadas à manutenção ou evolução da ferramenta. As respostas para essa QP estão apresentadas conforme o contexto arquitetural no qual as melhorias estão distribuídas, sendo *(i)* Visão Geral da Arquitetura, *(ii)* Visão Arquitetural e *(iii)* Design Arquitetural.

- ***(i)* Visão Geral da Arquitetura:**

Com relação ao questionamento a respeito da clareza na apresentação do objetivo da ferramenta REMO, 42% (5/12) dos participantes "Concordaram Totalmente" com tal aspecto. Apenas dois participantes se mantiveram "Neutro", com uma representatividade de 17% (2/12), sendo que nenhum participante discordou parcialmente ou totalmente com a afirmativa.

Avaliando a suficiência de artefatos para auxiliar o desenvolvimento da ferramenta, apenas 25% (3/12) dos participantes "Concordaram Totalmente" com essa afirmativa e 25% (3/12) dos "Concordaram Parcialmente". O restante dos participantes foram "Neutros" ou

"Discordaram Parcialmente", além de terem registrado observações que levaram a definição de melhorias a serem realizadas nos artefatos criados para a representação da arquitetura. Algumas das melhorias identificadas foram referentes aos diagramas de atividades, mas, como os diagramas de atividades foram utilizados somente no documento de auxílio à avaliação arquitetural, os mesmos não foram atualizados.

Sobre os conceitos apresentados relacionados a SOA, 17% (2/12) dos participantes "Concordaram Totalmente" com o que foi ilustrado no arquivo de apresentação da arquitetura. Os participantes enfatizaram sobre as nomenclaturas utilizadas acerca desse tema, sugerindo melhora no detalhamento dessas especificidades, assim como substituir o termo "agente", o qual havia sido utilizado para designar os serviços criados para a SOA. 17% (2/12) dos participantes que "Discordaram Parcialmente" mencionaram a ausência da descrição de alguns termos utilizados no documento, os quais estão definidos nesse trabalho, assim como a ausência da especificação de informações relacionadas aos RNFs definidos para a ferramenta REMO.

Quanto a possibilidade de a arquitetura proposta apresentar riscos durante o seu ciclo de vida, 42% (5/12) dos participantes "Discordaram Totalmente" ou "Discordaram Parcialmente" com essa afirmativa, apesar de terem apresentado ressalvas sobre a forma como arquitetura irá auxiliar nas mudanças futuras da ferramenta. Além disso, foram sugeridas melhorias às visões já existentes, assim como a criação de novas visões, as quais serão apresentadas a seguir.

- ***(ii)* Visão Arquitetural:**

No que concerne a primeira questão a respeito da Visão Arquitetural, 75% (9/12) dos participantes "Concordaram Parcialmente" ou "Concordaram Totalmente" que as visões arquiteturais definidas para a ferramenta REMO são suficientes. Apesar dessa grande concordância, foram sugeridas a criação de novas visões para auxiliar na fase de implantação, assim como para facilitar a compreensão de como seria realizada a comunicação entre os módulos da ferramenta.

Um ponto importante está relacionado com o auxílio dos diagramas de caso de uso ou classe no processo de implementação do sistema. 83% (10/12) dos participantes "Concordaram Parcialmente" ou "Concordaram Totalmente" sobre o diagrama de caso de uso e 67% (8/12) sobre o diagrama de classes. Uma justificativa para uma menor concordância com o diagrama de classes é o fato do mesmo ter obtido mais sugestões de melhorias. A Figura 27 apresenta a representatividade dos participantes de acordo com os

seus respectivos níveis de concordância.

Na Figura 27 é visível o grande número de participantes que concordaram totalmente com a necessidade e o auxílio dos diagramas de caso de uso e classe para implementação da ferramenta. Para o diagrama de caso de uso, nenhum participante "Discordou Totalmente" e "Discordou Parcialmente". No entanto, para o diagrama de classes esse cenário foi um pouco diferente, uma vez que 8% (1/12) dos participantes "Discordaram Totalmente" e 17% (2/12) "Discordaram Parcialmente", justificando a necessidade de melhorias indicadas para esse diagrama.

- ***(iii)* Design Arquitetural:**

No que concerne ao design arquitetural, foi avaliado junto aos participantes se a arquitetura proposta atende aos atributos de qualidade. 75% (9/12) dos participantes "Concordaram Parcialmente" ou "Concordaram Totalmente". O restante dos participantes, 17% (2/12) foram "Neutros" e 8% (1/12) "Discordou Parcialmente" da afirmação, porém não deixaram sugestões de melhorias para aplicar à arquitetura.

Um total de 58% (7/12) dos participantes concordou que a arquitetura proposta atende às necessidades relacionadas a manutenibilidade, sendo que 25% (3/12) "Concordaram Totalmente" e 33% (4/12) "Concordaram Parcialmente", conforme é apresentado na Figura 28. Apesar do elevado nível de concordância, ainda houveram várias sugestões de melhorias a serem realizadas na arquitetura, como por exemplo, a necessidade de explicitar de forma mais clara como os RNFs implicariam positivamente na arquitetura.

É importante destacar que 58% (7/12) dos participantes "Concordaram Totalmente" com os RFs elicitados e especificados para a ferramenta proposta. No entanto, quando analisada a avaliação dos atributos de qualidade da arquitetura, 42% (5/12) foram "Neutros" ao suporte dos mesmos na arquitetura, assim novamente reforçando a necessidade de melhorias arquiteturais relacionadas a esse quesito. A Figura 29 apresenta a relação das respostas dos participantes sobre os RFs e RNFs da arquitetura proposta.

Na Figura 29 pode ser analisado o baixo número de participantes que discordaram com esses requisitos, sendo que somente 8% (1/12) "Discordou Totalmente" com os RFs e que 8% (1/12) "Discordou Parcialmente" com os RNFs. Essas avaliações foram atribuídas pelo mesmo participante, porém o mesmo deixou somente observações de melhorias a respeito dos RNFs, as quais também foram indicadas por outros participantes que avaliaram esses requisitos com maior nível de concordância.

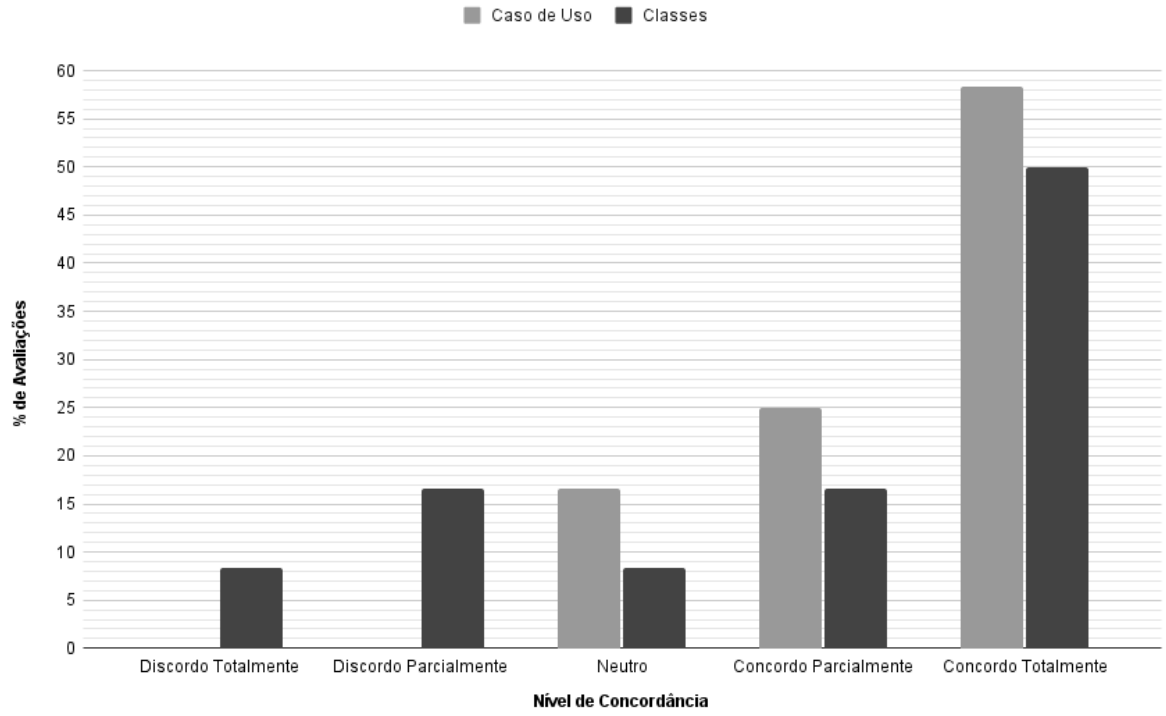

Figura 27 - Resultados referentes ao auxílio dos diagramas de caso de uso e de classes no processo de implantação da ferramenta REMO.

Fonte: Autoria própria.

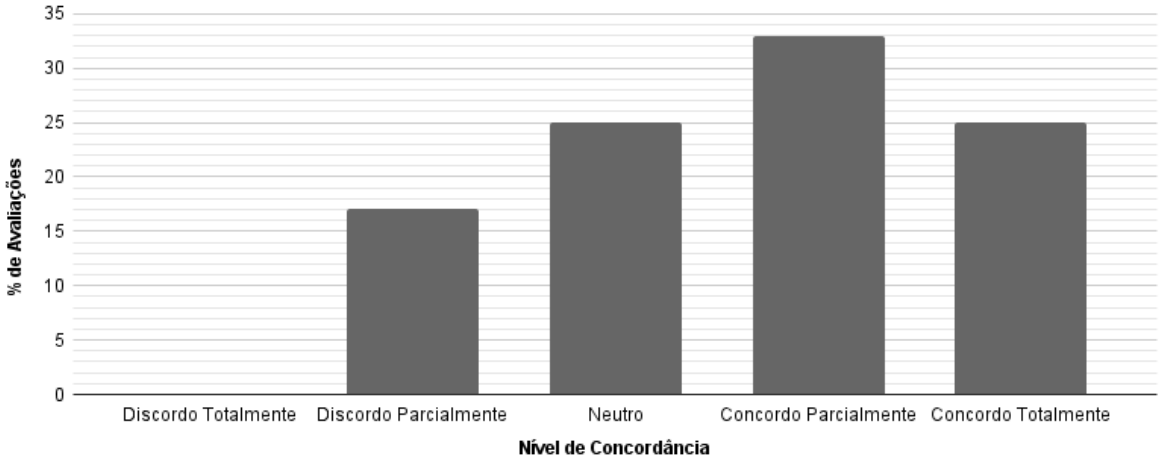

Figura 28 - Respostas dos participantes com relação aos atributos de manutenibilidade disponibilizados na arquitetura.

Fonte: Autoria própria.

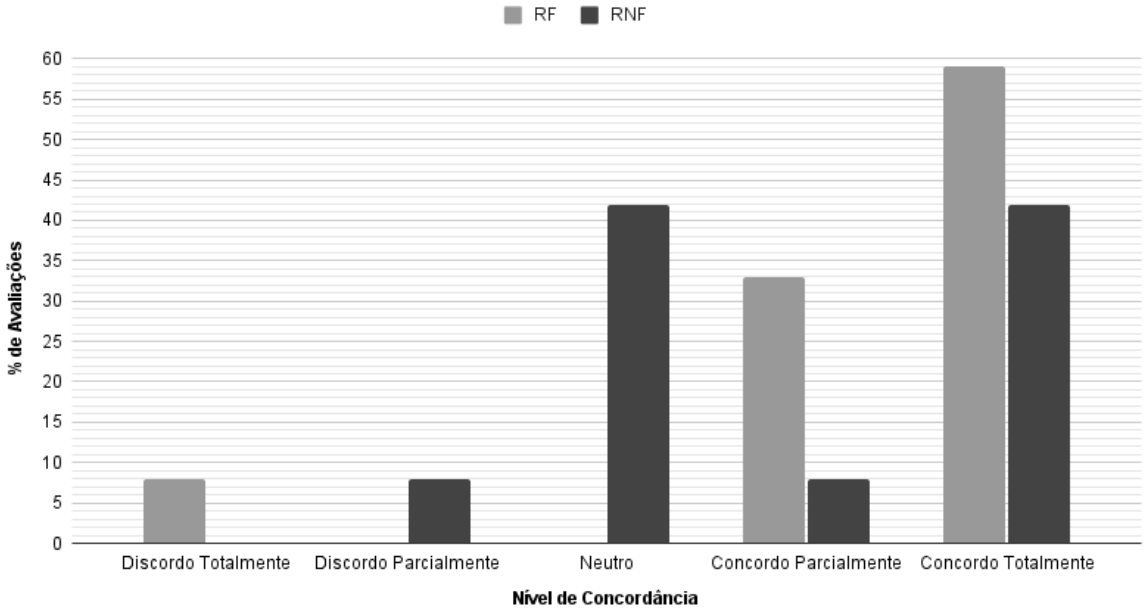

Figura 29 - Relação entre da avaliação dos RFs e RNFs da arquitetura proposta da ferramenta REMO.

Fonte: Autoria própria

Apesar do percentual significativo de concordância com esses requisitos, foi verificado de acordo com os comentários dos participantes a necessidade de melhorias nos atributos de qualidade. Essas melhorias estão relacionadas aos RNFs, o que reforçou a necessidade de revisão desses requisitos para a arquitetura proposta.

5.2.2 QP_2: - Quais melhorias podem ser realizadas no design da arquitetura REMO-AS?

Para responder a esta QP, foram levadas em consideração todas as observações deixadas pelos participantes no questionário. Algumas melhorias foram sugeridas diretamente pelos participantes, enquanto outras foram extraídas a partir da interpretação do nível de concordância atribuído em conjunto com a observação realizada. Além disso, foi verificado se a melhoria em questão seria viável para o contexto de aplicação no qual se encontra o atual trabalho. A Tabela 11 apresenta as melhorias extraídas a partir do *feedback* dos participantes.

A primeira coluna apresenta a representação dos sete participantes que apresentaram sugestões; a segunda coluna contempla o ID das 14 melhorias identificadas; a terceira coluna apresenta o contexto no qual as melhorias foram aplicadas; e por fim, na quarta coluna é realizada uma breve descrição da melhoria. Primeiramente, para iniciar a resolução

Tabela 11 - Melhorias extraídas após execução do questionário de avaliação.

Participante	ID	Contexto	Descrição da Melhoria
Participante 1	M01	Visão Projeto	Complementar arquitetura com informações sobre as restrições dos RNFs.
Participante 2	M02	Visão Geral	Especificar como a arquitetura SOA irá auxiliar com manutenções ou evoluções da ferramenta.
	M03	Visão Geral	Adicionar opção de edição de projetos nas visões relacionadas.
	M04	Visão Projeto	Adicionar representação de padrões arquiteturais da arquitetura.
Participante 3	M05	Design Arquitetural	Criar diagrama de sequência para demonstrar comunicação entre os módulos.
	M06	Visão Projeto	Adicionar "manutenibilidade" como requisito não funcional.
Participante 4	M07	Visão Geral	Criar visão de componentes definindo as interfaces, de modo a facilitar o desenvolvimento e teste.
	M08	Visão Geral	Melhorar diagrama de classes, ajustando os relacionamentos.
	M09	Visão Projeto	Mudar RNF08 para característica de disponibilidade.
Participante 5	M10	Visão Geral	Especificar tipo de usuário na Figura 15.
	M11	Design Arquitetural	Mudar "apresentação" para "interface" na Figura 24
	M12	Design Arquitetural	Adicionar "extend" de "identificar requisitos" para "especificar requisitos" na Figura 22
	M13	Design Arquitetural	Adicionar tipos de documentos no caso de uso de "gerar documento" na Figura 22
Participante 6	M14	Visão Geral	Alterar "gerenciamento de mudanças" para "gerenciamento de mudanças de requisitos" na Figura 15

Fonte: Autoria própria.

da melhoria M01, foi necessário ajustar a Tabela 4.2, apresentada na Seção 4.2, adicionando à mesma as informações relacionadas a restrição de cada RNF. As melhorias realizadas são apresentadas na Tabela 12.

Além da melhoria M01, na Tabela 12 também foram incluídas as melhorias M06 e M09, as quais também tratavam o contexto dos RNFs da ferramenta REMO. Em decorrência da realização das melhorias, esses requisitos foram reavaliados, ocasionando

na exclusão do RNF08, o qual consta na Tabela 4.2.

Referente a melhoria M02, a sugestão de melhoria do contexto de aplicação da SOA na ferramenta REMO foi inserido diretamente na Seção 4.3 do Capítulo 4 referente à utilização desse estilo arquitetural com relação às futuras evoluções e manutenções a serem realizadas.

Para as melhorias M03, M10 e M14 foram realizadas atualizações na Figura 15, apresentada na Seção 4.1, referente à "Visão geral da ferramenta REMO". Dessa forma, foram realizadas três modificações: *(i)* adição da funcionalidade de "Edição de projetos" no módulo GPR (módulo 1); *(ii)* definição do tipo de usuário como *"Stakeholder"* que irá utilizar a ferramenta; e *(iii)* alteração da nomenclatura da atividade 6 para "Gerenciamento de mudanças de requisitos" do módulo ERA (módulo 2). Essas modificações são apresentadas na Figura 30.

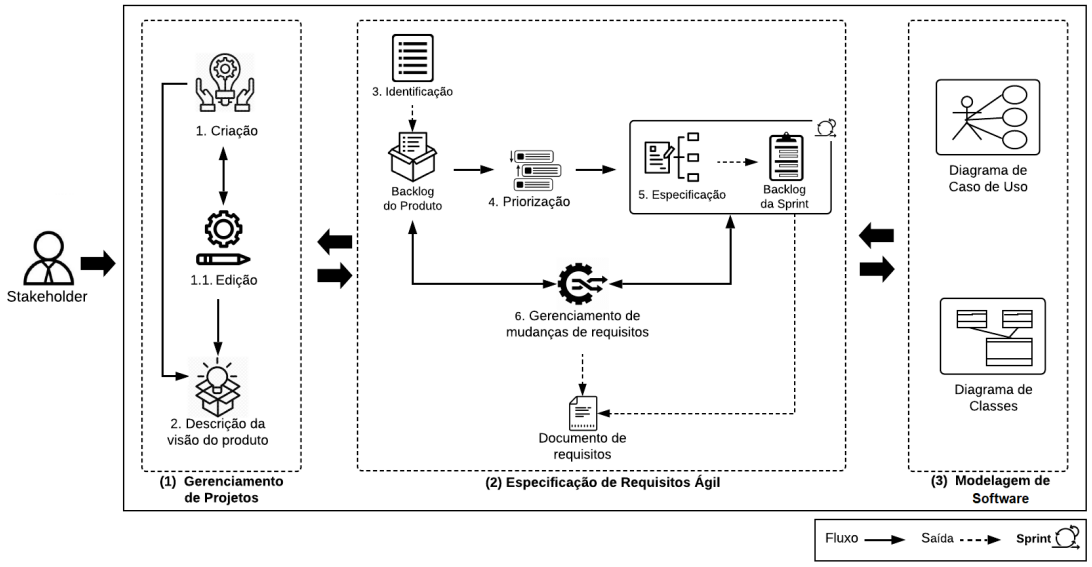

Figura 30 - Visão geral da ferramenta REMO atualizada.
Fonte: Autoria própria

Para auxiliar o desenvolvimento e teste da ferramenta e resolver as melhorias M04 e M07, foi criada uma visão de componentes da arquitetura da ferramenta. Nesta visão são definidas as interfaces requeridas e fornecidas, as quais estão agrupadas de acordo com os seus respectivos módulos. Externo a esses módulos existem dois componentes, os quais estão relacionados à interface da base de dados e à técnica PLN, a qual poderá ser substituída por algum outro método para geração dos diagramas de caso de uso e classes. O diagrama de componentes é apresentado na Figura 31.

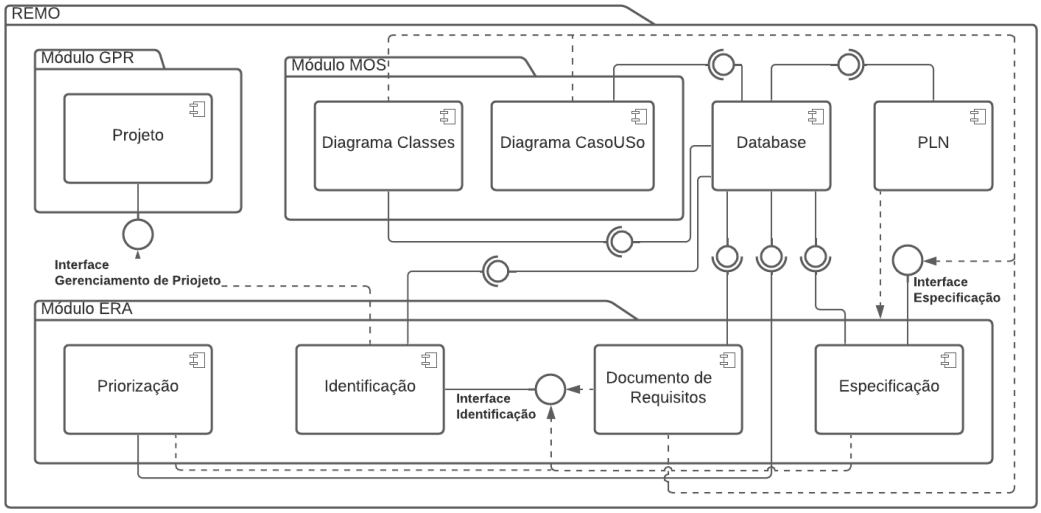

Figura 31 - Visão de componentes da ferramenta REMO.
Fonte: Autoria própria.

Com a finalidade de demonstrar a comunicação entre os três módulos da ferramenta, bem como facilitar a compreensão de como a mesma seria utilizada, foi desenvolvido um Diagrama de Sequência para agregar às demais visões arquiteturais. Esse diagrama foi desenvolvido para resolver a melhoria M05, ou seja, apresentar uma sequência de passos desde a criação de um projeto até a geração de um diagrama de caso de uso na ferramenta REMO. O diagrama é apresentado na Figura 32.

Em decorrência das melhorias M07 e M11, foram realizadas algumas atualizações na "Representação da arquitetura da ferramenta REMO", apresentada na Figura 24. Neste contexto, foi adicionada a nova capacidade de "Disponibilidade" conforme descrito pela melhoria M07 e alterada a nomenclatura utilizada no componente de "Apresentação" por "Interface", conforme é ilustrado na Figura 33.

Para que a melhoria M08 fosse aplicada, o diagrama de classes da ferramenta REMO foi atualizado ajustando a sua organização, bem como os seus relacionamentos. Como o histórico de mudanças irá registrar todas as alterações realizadas em um projeto, foi adicionada à respectiva classe o relacionamento com todas as demais classes, com exceção da classe de usuário. Além disso, foram ajustados alguns atributos e métodos, conforme é apresentado na Figura 34.

Por fim, para implantar às melhorias M12 e M13 foram realizadas algumas melhorias no diagrama de caso de uso criado inicialmente para o projeto, conforme apresentado na Figura 22. Para a M12 foi adicionado o *extend* do caso de uso de

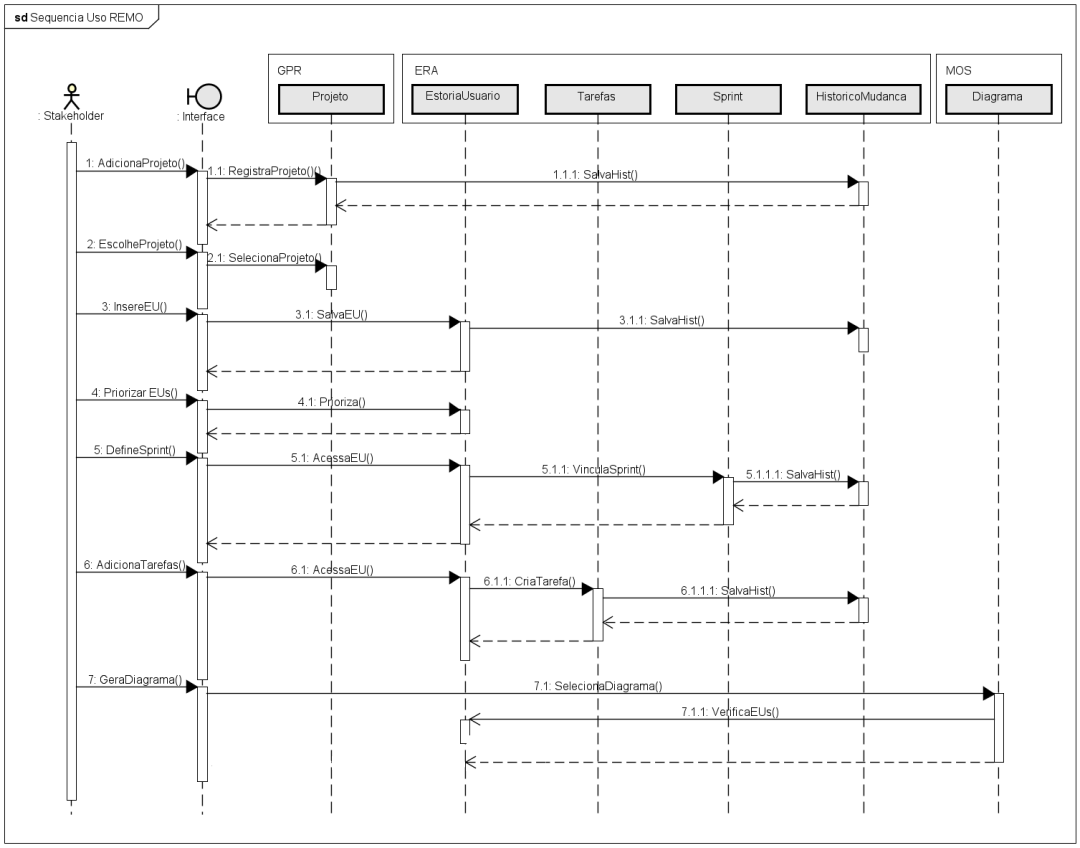

Figura 32 - Diagrama de Sequência referente a utilização da ferramenta REMO. Fonte: Autoria própria.

"Identificar Requisitos" para "Especificar Requisitos", por ser uma tarefa que pode ser subsequente. Já para a M13, foi apenas alterado o caso de uso principal para "Gerar Documento de Requisitos" (ver Figura 35).

5.2.3 Ameaças à Validade

Nesta seção são detalhadas as possíveis ameaças à validade que podem afetar os valores e as conclusões a respeito da arquitetura apresentada, seguindo as diretrizes de Wohlin et al. (2012).

As ameaças à validade externa estão relacionadas à generalização dos resultados. Somente 12 participantes avaliaram a arquitetura proposta, podendo esta representatividade ser um problema que geralmente prejudica a condução de pesquisas. Com a finalidade de minimizar esse problema, foram selecionados participantes especialistas e com experiência em AS.

As ameaças à validade por construção estão preocupadas com a relação entre

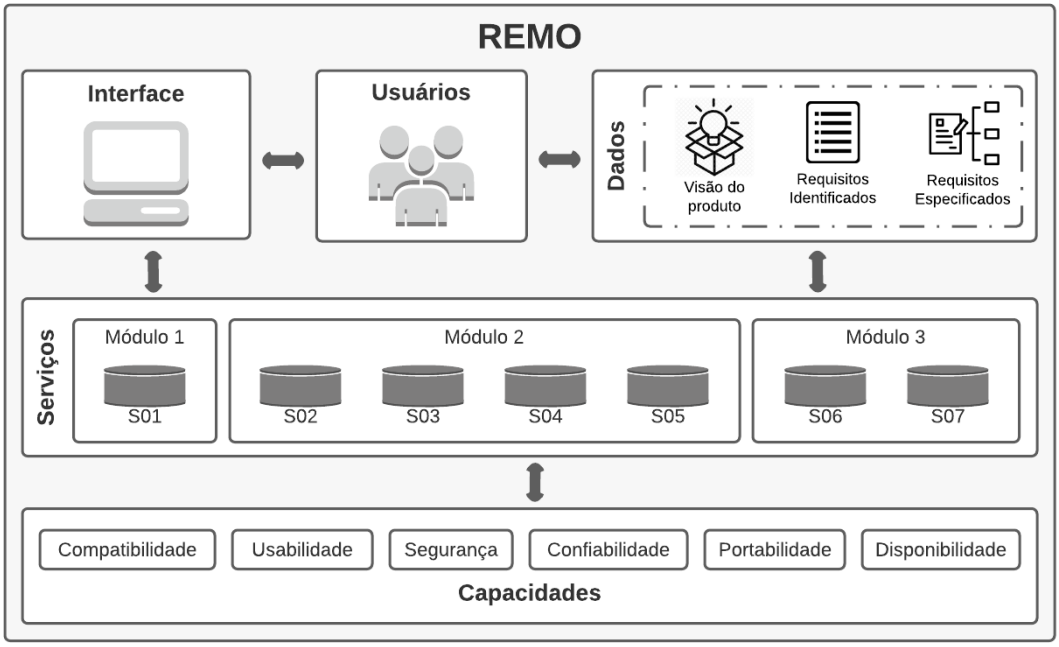

Figura 33 - Representação da arquitetura da ferramenta REMO atualizada.
Fonte: Autoria própria.

a teoria e o que é observado. No desenvolvimento e análise da arquitetura proposta, possíveis equívocos podem ter sido cometidos. Para auxiliar a compreensão do estudo de caso foi desenvolvido um documento de apoio contendo as informações necessárias para orientar os participantes na avaliação da arquitetura proposta. Para mitigar esse risco foi realizado um estudo de caso piloto com a respectiva documentação, a fim de assegurar que as informações estavam coerentes e bem apresentadas.

Por fim as ameaças à validade interna caracterizam o grau de confiabilidade entre os resultados esperados e os resultados obtidos. Para aumentar a garantia de assertividade entre esses resultados, foi realizada a validação relacionada tanto às informações de funcionalidades quanto à arquitetura da ferramenta REMO. Após a realização do estudo de caso, as melhorias identificadas foram aplicadas à arquitetura, conforme os *feedbacks* obtidos dos participantes.

5.3 Considerações Finais

No desenvolvimento desse Capítulo foram apresentados o processo de condução do Estudo de Caso, bem como os resultados obtidos referente a avaliação da arquitetura proposta para a ferramenta REMO. Dentre os resultados obtidos, estão a validação

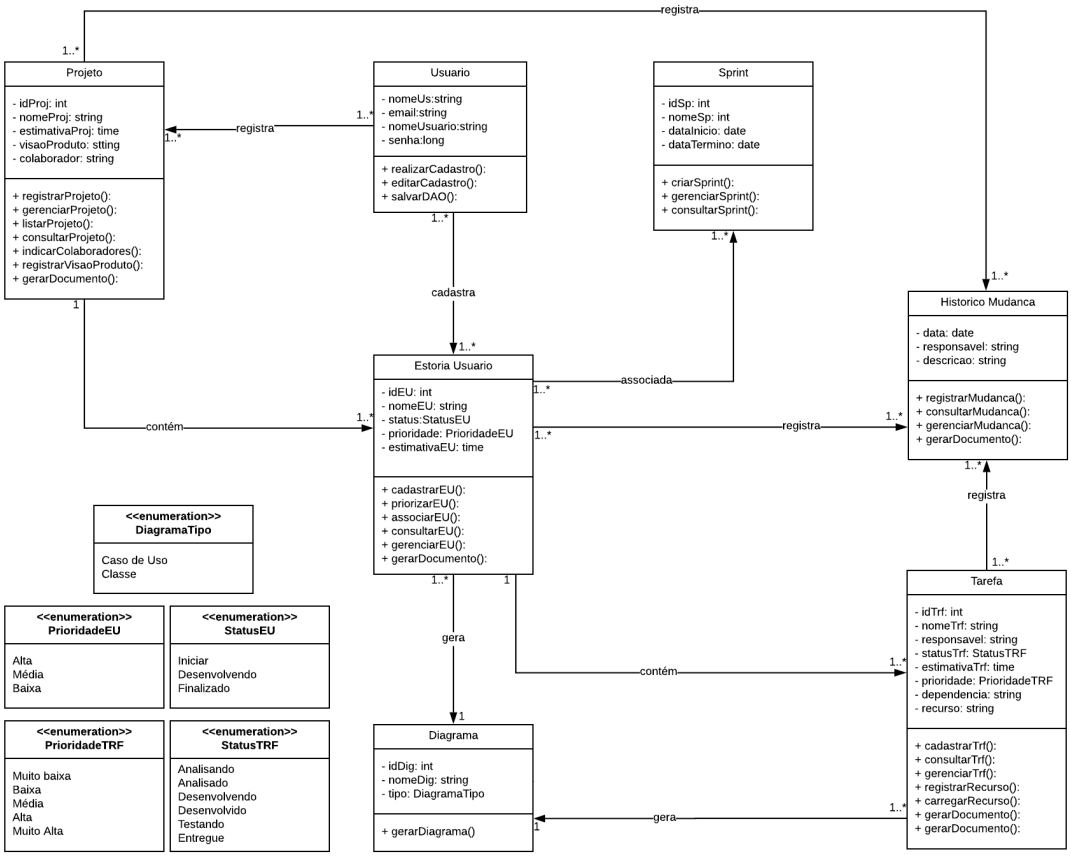

Figura 34 - Diagrama de classes da ferramenta REMO atualizado.
Fonte: Autoria própria.

arquitetural da ferramenta proposta, a qual, de forma geral foi bem avaliada, porém com algumas ressalvas. Em consequência dessas ressalvas, pode-se mencionar como resultado a primeira versão validada de uma arquitetura SOA.

É importante ressaltar que a REMO-AS desenvolvida não é, necessariamente, a última versão, sendo que a mesma ainda poderá ser complementada e evoluída com o passar do tempo conforme as futuras necessidades apresentadas. Essas futuras necessidades podem estar relacionadas tanto a expansão da ferramenta, permitindo a adição de novas funcionalidades, assim como a possibilidade de aplicação em outras plataformas.

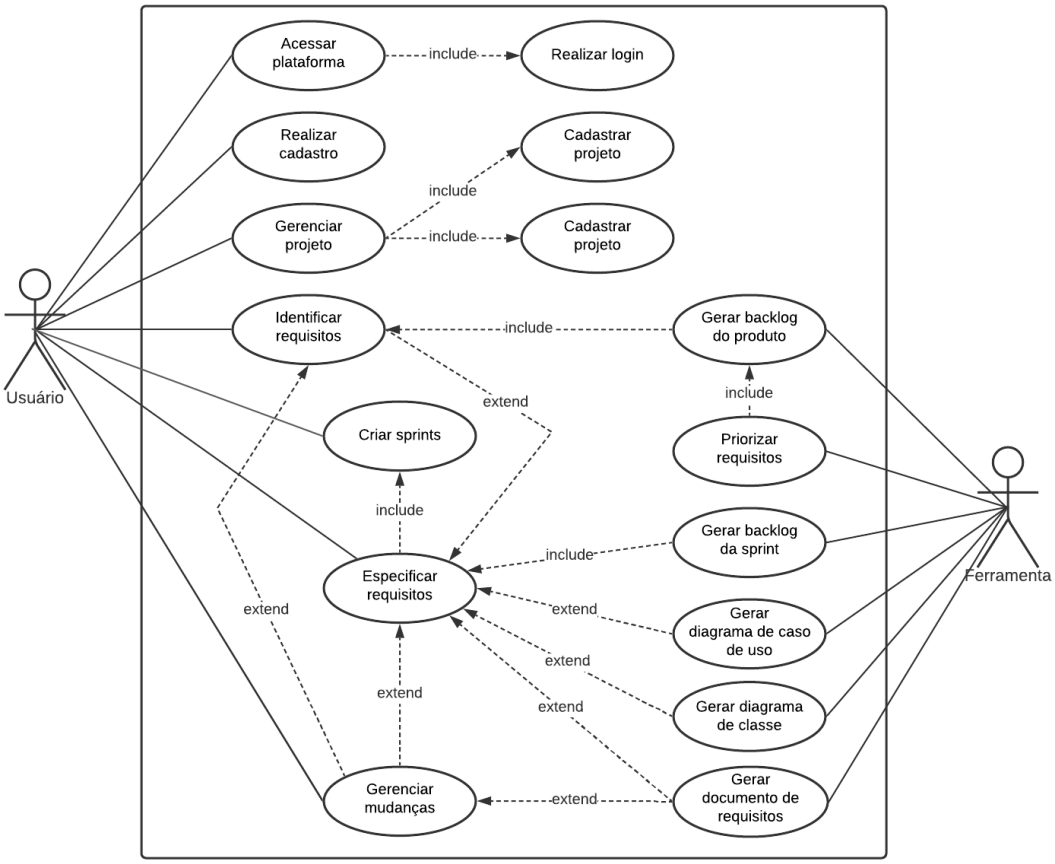

Figura 35 - Diagrama de caso de uso da ferramenta REMO

Fonte: Autoria própria

Tabela 12 - Requisitos Não Funcionais da ferramenta REMO atualizados.

Característica	ID	Descrição	Restrição
Compatibilidade	RNF01	A arquitetura deve permitir que possam ser executadas tarefas paralelamente, mantendo a integridade das informações.	As operações realizadas não devem impactar n usabilidade do outro mó
	RNF02	A arquitetura deve permitir que todos os componentes do sistema possam trocar informações e reutiliza-las.	A transferência de info módulos deve permanec durante o uso da ferram
Usabilidade	RNF03	A interface de usuário deve ser de fácil operação e controle dos componentes do sistema.	O usuário deve cons ferramenta de forma complicações.
	RNF04	A interface do usuário deve permitir que pessoas com diferentes características e capacidades possam acessar todas as funcionalidades do sistema.	Uma capacidade física d influenciar na usabilidad
Confiabilidade	RNF05	A arquitetura deve permitir que o sistema se mantenha acessível e operável sempre quando necessário, para utilização.	Quando o usuário funcionalidade da ferra deve estar disponível utilização.
Segurança	RNF06	A arquitetura deve fornecer segurança dos dados usuários.	O acesso às informaçõ deve ser restrito aos usu
	RNF07	A arquitetura deve fornecer a integridade dos projetos cadastrados	As informações inserid devem ser mantidas no
Disponibilidade	RNF08	A arquitetura deve permitir que as alterações realizadas nos serviços não impliquem nos demais serviços.	A alteração de serviços do respectivo serviço.
	RNF09	A arquitetura facilitar a criação de novos serviços.	Novos serviços poderã sem alteração da estrutu
Portabilidade	RNF10	A arquitetura deve permitir que o sistema possa se adaptar a diferentes ambientes de utilização.	A ferramenta poderá diferentes dispositivos, interface do mesmo.

Fonte: Autoria própria.

6 CONSIDERAÇÕES FINAIS

Este trabalho teve como principal objetivo a apresentação da análise e projeto da ferramenta REMO para auxiliar as atividades de elicitação, especificação, validação gerenciamento de requisitos e modelagem de sistemas. Após a definição de todos os conceitos relacionados ao âmbito deste trabalho, foi realizado um MS com objetivo de analisar o cenário atual das ferramentas CASE que também abrangem o contexto deste trabalho.

A partir da condução do MS, foi possível notar a ausência de uma ferramenta *Web* a qual permitisse a execução das atividades de elicitação, especificação, validação gerenciamento de requisitos e modelagem de sistemas no âmbito do desenvolvimento ágil de software. Este fato contribuiu para a reafirmação da relevância deste trabalho, para que fosse válida a concepção de uma ferramenta neste contexto.

Com os resultados alcançados a partir do MS, foi proposta a ferramenta REMO. As funcionalidades e as nomenclaturas da ferramenta foram selecionadas com base em um estudo de ferramentas e modelos de processos ágeis para a extração das melhores características. A partir disso foi criada uma arquitetura SOA especificamente para a REMO com a finalidade de fornecer suporte a todos os módulos da ferramenta, junto aos serviços disponibilizados pelos mesmos.

Como parte da pesquisa foi realizado um estudo de caso para avaliar se as funcionalidades e a arquitetura proposta são eficazes para o desenvolvimento da ferramenta REMO. Esse estudo de caso foi conduzido com participantes experientes na área da arquitetura de software. Os resultados alcançados a partir do estudo de caso realizado apontaram algumas melhorias as quais foram aplicadas na arquitetura da ferramenta, auxiliando para que a mesma tenha o melhor desempenho, suportando o desenvolvimento da ferramenta e futuras mudanças.

Com base nos resultados alcançados, as principais contribuições deste trabalho são:

- *(i)* visão geral sobre o estado da arte relacionado às ferramentas para especificação, gerenciamento de requisitos e modelagem de software;
- *(ii)* criação da arquitetura SOA para a ferramenta REMO, contemplando os diferentes módulos e serviços voltados para a execução das atividades de elicitação, especificação, validação gerenciamento de requisitos e modelagem de software;
- *(iii)* definição das funcionalidades para a ferramenta REMO, direcionada ao contexto ágil, capaz de gerenciar as atividades relacionadas às fases de especificação de software e modelagem de software, bem como a geração de artefatos (documentos e diagramas);
- *(iv)* produção de artigos científicos.

Durante a condução e avaliação da análise e projeto da ferramenta REMO foram identificadas algumas limitações. Dentre tais limitações, é importante ressaltar a compreensão da arquitetura proposta, por parte dos participantes do estudo de caso. Esse fator está relacionado ao documento disponibilizado em conjunto com o questionário. Algumas informações nele dispostas, segundo alguns dos participantes, poderiam estar confusas ou com falta de detalhes, podendo ter interferido na compreensão completa da AS proposta.

Ainda com relação ao estudo de caso realizado, a compreensão do *feedback* dos participantes também foi considerada uma limitação. O fato da comunicação entre o condutor do questionário e os participantes ser restrita a textos e ter sido realizada somente uma vez, a compreensão das observações repassadas pelos participantes pode ter ocorrido diferentemente do real objetivo do participante.

A forma como será gerado o diagrama de classes de forma automática também pode ser considerada uma limitação. Esse diagrama irá utilizar as informações que constituem as tarefas especificadas na ferramenta. No entanto, ainda não foi identificada como essas informações serão tratadas para a geração do diagrama. Já o diagrama de casos de uso poderá ser gerado a partir das estórias de usuário especificadas, pois já existem estudos que tratam esse tipo de geração de diagramas.

Outra limitação está relacionada à priorização automática dos requisitos. Existem alguns algoritmos que priorizam os requisitos automaticamente com base na estimativa e prioridade especificada, porém não foi determinado qual algoritmo e quais atributos serão utilizados para essa funcionalidade na ferramenta REMO.

Com base no trabalho conduzido foi possível perceber a ausência de uma ferramenta web para o contexto ágil que integre e viabilize as fases de especificação de requisitos e projeto de software. Portanto, como trabalhos futuros, os módulos da ferramenta serão desenvolvidos separadamente e posteriormente serão integrados. Além disso, a arquitetura em si poderá ser evoluída, tanto para a implementação de novas funcionalidades quanto para expansão para outras plataformas.

REFERÊNCIAS

ABDOULI, M.; KARAA, W. B. A.; GHEZALA, H. B. Survey of works that transform requirements into uml diagrams. In: IEEE. **2016 IEEE 14th International Conference on Software Engineering Research, Management and Applications (SERA)**. [S.l.], 2016. p. 117–123.

AFREEN, H.; BAJWA, I. S. Generating uml class models from sbvr software requirements specifications. In: **23rd Benelux Conference on Artificial Intelligence (BNAIC 2011)**. [S.l.: s.n.], 2011. p. 23–32.

ALKHADER, Y.; HUDAIB, A.; HAMMO, B. Experimenting with extracting software requirements using nlp approach. In: IEEE. **2006 International Conference on Information and Automation**. [S.l.], 2006. p. 349–354.

ANCHIÊTA, R. T.; SOUSA, R. F. de; MOURA, R. S. Using nlp techniques for identifying gui prototypes and uml diagrams from use cases. In: **SEKE**. [S.l.: s.n.], 2013. p. 48–53.

BASILI, V.; WEISS, D. A methodology for collecting valid software engineering data. **IEEE Transactions on Software Engineering**, v. 10, n. 6, p. 728–738, 1984.

BASS, L.; CLEMENTS, P.; KAZMAN, R. **Software architecture in practice**. [S.l.]: Addison-Wesley Professional, 2003.

BECK, K. et al. Manifesto for agile software development. 2001.

BECK K.; ANDRES, C. **Extreme Programming Explained: Embrace Change**. 2^{nd}. ed. [S.l.]: Addison-Wesley, 2004. 224 p.

BOJNORD, A. S.; AHMAD, R. B.; BOJNORD, H. S. Webstuire: web-based support tool for user interface requirements elicitation. In: IEEE. **ICCKE 2013**. [S.l.], 2013. p. 52–58.

BOOCH, G.; RUMBAUGH, J.; JACOBSON, I. **UML: guia do usuário**. 2^{nd}. ed. Rio de Janeiro: Elsevier, 2006. 474 p.

BOZYIğIT, F.; AKTAş, O.; KILINç, D. Automatic concept identification of software requirements in turkish. **Turkish Journal of Electrical Engineering & Computer Sciences**, The Scientific and Technological Research Council of Turkey, v. 27, n. 1, p. 453–470, 2019.

BRERETON, P. et al. Lessons from applying the systematic literature review process within the software engineering domain. **Journal of systems and software**, Elsevier, v. 80, n. 4, p. 571–583, 2007.

CAVADA, R. et al. Supporting requirements validation: the eurailcheck tool. In: IEEE. **2009 IEEE/ACM International Conference on Automated Software Engineering**. [S.l.], 2009. p. 665–667.

CHANG, C. et al. A sysml based requirement modeling automatic transformation approach. In: IEEE. **2014 IEEE 38th International Computer Software and Applications Conference Workshops**. [S.l.], 2014. p. 474–479.

CHANG, C.-H. et al. A sysml-based requirement supporting tool for embedded software. In: IEEE. **2011 Fifth International Conference on Secure Software Integration and Reliability Improvement-Companion**. [S.l.], 2011. p. 202–206.

COSSENTINO, M.; HILAIRE, V.; SEIDITA, V. The openup process. In: **Handbook on Agent-Oriented Design Processes**. [S.l.]: Springer, 2013. p. 491–566.

DAMASCENO, J. **Introdução à Composição de Serviços Web**. 2012.

DASCALU, S. et al. A software tool for requirements specification on using the storm environment to create srs documents. Citeseer, 2007.

DEEPTIMAHANTI, D. K.; SANYAL, R. Semi-automatic generation of uml models from natural language requirements. In: ACM. **Proceedings of the 4th India Software Engineering Conference**. [S.l.], 2011. p. 165–174.

DURÁN, A. et al. Supporting requirements verification using xslt. In: IEEE. **Proceedings IEEE Joint International Conference on Requirements Engineering**. [S.l.], 2002. p. 165–172.

EBERT, C.; ABRAHAMSSON, P.; OZA, N. Lean software development. **IEEE Software**, IEEE Computer Society, v. 29, n. 05, p. 22–25, 2012.

ERL, T. **SOA Principles of Service Design (paperback)**. [S.l.]: Prentice Hall Press, 2016.

ESTECA, A. M. N. et al. Computational support for the process of software requirement specification. In: IEEE. **2012 XXXVIII Conferencia Latinoamericana En Informatica (CLEI)**. [S.l.], 2012. p. 1–10.

FERNANDES, B. B. et al. Metodologia fdd aplicada a especificação dos requisitos no processo gre do nível g de maturidade do modelo mps.br. **XIII Encoinfo – Encontro de Computação e Informática do Tocantins**, 2011.

GOYAL, S. Major seminar on feature driven development. **Jennifer Schiller Chair of Applied Software Engineering**, 2008.

GULIA, S.; CHOUDHURY, T. An efficient automated design to generate uml diagram from natural language specifications. In: IEEE. **2016 6th International Conference-Cloud System and Big Data Engineering (Confluence)**. [S.l.], 2016. p. 641–648.

HE, H. What is service-oriented architecture. **Publicação eletrônica em: http://webservices.xml.com/pub/a/ws/2003/09/30/soa.html**, v. 30, p. 1–5, 2003.

IBRAHIM, M.; AHMAD, R. Class diagram extraction from textual requirements using natural language processing (nlp) techniques. In: IEEE. **2010 Second International Conference on Computer Research and Development**. [S.l.], 2010. p. 200–204.

IQBAL, U.; BAJWA, I. S. Generating uml activity diagram from sbvr rules. In: IEEE. **2016 Sixth International Conference on Innovative Computing Technology (INTECH)**. [S.l.], 2016. p. 216–219.

ISO, I. IEC25010: 2011 systems and software engineering–systems and software quality requirements and evaluation (square)–system and software quality models. **International Organization for Standardization**, v. 34, p. 2910, 2011.

KAMALRUDIN, M.; GRUNDY, J.; HOSKING, J. Tool support for essential use cases to better capture software requirements. In: **Proceedings of the IEEE/ACM international conference on Automated software engineering**. [S.l.: s.n.], 2010. p. 255–264.

KITCHENHAM, B. et al. Systematic literature reviews in software engineering–a tertiary study. **Information and software technology**, Elsevier, v. 52, n. 8, p. 792–805, 2010.

KUMAR, D. D.; SANYAL, R. Static uml model generator from analysis of requirements (sugar). In: IEEE. **2008 Advanced Software Engineering and Its Applications**. [S.l.], 2008. p. 77–84.

LEFFINGWELL, D. **Agile Software Requirements: Lean Requirements Practices for Teams,Programs, and the Enterprise**. [S.l.]: Addison-Wesley, 2011.

LIKERT, R. A technique for the measurement of attitudes. **Journal Archives of Psychology**, v. 22, n. 140, p. 1–55, 1932.

LIN, Y. et al. Scrum conceptualization using k-crio ontology. In: SPRINGER. **International Symposium on Data-Driven Process Discovery and Analysis**. [S.l.], 2011. p. 189–211.

LU, C. et al. A requirement tool to support model-based requirement engineering. In: IEEE. **2008 32nd Annual IEEE International Computer Software and Applications Conference**. [S.l.], 2008. p. 712–717.

MOHANAN, M.; SAMUEL, P. Natural language processing approach for uml class model generation from software requirement specifications via sbvr. **International Journal on Artificial Intelligence Tools**, World Scientific, v. 27, n. 06, p. 1850027, 2018.

NASSAR, H. A.; ALHROOB, A.; IMAM, A. T. An algorithmic approach for sketching sequence diagram (aassd). In: **Proceedings of the International Conference on Advances in Image Processing**. [S.l.: s.n.], 2017. p. 156–160.

NUSEIBEH, B.; EASTERBOOK, S. Requirements engineering: A roadmap. In: **Proceedings of the 22nd International Conference on Software Engineering (ICSE)**. Limerick, Ireland: ACM, 2000. p. 37–45. ISBN 1-58113-253-0.

PHANTHANITHILERD, N.; PROMPOON, N. Requirements characteristics verification method and tool based on rules constructed form software component relationships. In: IEEE. **2015 Second International Conference on Trustworthy Systems and Their Applications**. [S.l.], 2015. p. 61–70.

PHILIPPUS, E. Architecture spikes. **Agile Werken ImprovemenT BV**, v. 1, p. 1–3, 2009.

PMI. **Um Guia do Conhecimento em Gerenciamento de Projetos (Guia PMBOK)**. 6^{th}. ed. [S.l.]: Project Management Institute, 2018. 756 p.

POPPENDIECK, M.; POPPENDIECK, T. **Lean software development: an agile toolkit**. [S.l.]: Addison-Wesley, 2003.

PRESSMAN, R. S. **Software engineering: A practitioner's approach**. 8^{th}. ed. [S.l.]: McGraw-Hill, 2016. 976 p.

RECH, J. Handling of software quality defects in agile software development. In: **Software Applications: Concepts, Methodologies, Tools, and Applications**. [S.l.]: Igi Global, 2009. p. 242–265.

RIES, B.; CAPOZUCCA, A.; GUELFI, N. Messir: a text-first dsl-based approach for uml requirements engineering (tool demo). In: ACM. **Proceedings of the 11th ACM SIGPLAN International Conference on Software Language Engineering**. [S.l.], 2018. p. 103–107.

RIGBY, D. K.; SUTHERLAND, J.; TAKEUCHI, H. Embracing agile. **Harvard business review**, v. 94, n. 5, p. 40–50, 2016.

RIVAS, M. A.; SOUZA, E. Godoy de. Análise comparativa da utilização do modelo tradicional (waterfall) de desenvolvimento de projetos e o modelo ágil (agile) em fábricas de software. **Revista de Sistemas e Computação-RSC**, v. 4, n. 1, 2014.

SATOW, P.; EISELE, M. **Pitch development method**. [S.l.]: Google Patents, ago. 16 2012. US Patent App. 13/371,164.

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

SOMÉ, S. S. Supporting use case based requirements engineering. **Information and Software Technology**, Elsevier, v. 48, n. 1, p. 43–58, 2006.

SOMMERVILLE, I. **Software Engineering**. 10^{th}. ed. [S.l.]: Pearson Addison-Wesley, 2016. 816 p.

WOHLIN, C. et al. **Experimentation in software engineering**. [S.l.]: Springer Science & Business Media, 2012.