UNIVERSIDADE FEDERAL DE SANTA MARIA
CENTRO DE TECNOLOGIA
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

Otávio da Cruz Mello

REFINAMENTO DE ESPECIFICAÇÕES DE REQUISITO PELA DETECÇÃO DE INCONSISTÊNCIAS, AMBIGUIDADES E INCOMPLETUDES ATRAVÉS DO PROCESSAMENTO DE LINGUAGEM NATURAL

Santa Maria, RS
2024

Otávio da Cruz Mello

REFINAMENTO DE ESPECIFICAÇÕES DE REQUISITO PELA DETECÇÃO DE INCONSISTÊNCIAS, AMBIGUIDADES E INCOMPLETUDES ATRAVÉS DO PROCESSAMENTO DE LINGUAGEM NATURAL

Dissertação de Mestrado apresentada ao Programa de Pós-Graduação em Ciência da Computação, Área de Concentração em Engenharia de Software, da Universidade Federal de Santa Maria (UFSM, RS), como requisito parcial para obtenção do grau de **Mestre em Ciência da Computação**. Defesa realizada por videoconferência.

Orientadora: Prof.ª Lisandra Manzoni Fontoura

Santa Maria, RS
2024

Otávio da Cruz Mello

REFINAMENTO DE ESPECIFICAÇÕES DE REQUISITO PELA DETECÇÃO DE INCONSISTÊNCIAS, AMBIGUIDADES E INCOMPLETUDES ATRAVÉS DO PROCESSAMENTO DE LINGUAGEM NATURAL

Dissertação de Mestrado apresentada ao Programa de Pós-Graduação em Ciência da Computação, Área de Concentração em Engenharia de Software, da Universidade Federal de Santa Maria (UFSM, RS), como requisito parcial para obtenção do grau de **Mestre em Ciência da Computação**.

Aprovado em 17 de abril de 2024:

_____

Lisandra Manzoni Fontoura, Dr. (UFSM)
(Presidenta/Orientadora)

_____

Joaquim Vinicius de Carvalho Assunção, Dr. (UFSM)

_____

Lucineia Heloisa Thom, Dr. (UFRGS)

Santa Maria, RS
2024

AGRADECIMENTOS

Este trabalho foi realizado graças ao apoio constante recebido durante seu desenvolvimento. Gostaria de expressar minha gratidão a todas as pessoas que desempenharam um papel fundamental ao longo dessa jornada:

À minha orientadora, Prof. Dra. Lisandra Manzoni Fontoura, que sempre se mostrou disponível para me guiar ao longo deste trabalho. Sua orientação foi fundamental. Agradeço também toda a dedicação, paciência e confiança que foram depositadas em mim, não apenas durante esse período, mas ao longo de toda a minha trajetória acadêmica. Suas contribuições foram verdadeiramente inestimáveis e cruciais para o meu desenvolvimento.

À meus pais, Marilene Pereira da Cruz e Marcos Porto Mello, e à minha irmã Rafaela da Cruz Mello, por toda paciência e carinho que foi dado durante o período que estava desenvolvendo este trabalho, por me incentivarem a superar os obstáculos e sempre me apoiarem. Sou profundamente grato pelo amor incondicional e pelo apoio inestimável que me proporcionaram.

À Amaury Teixeira Cassola, que foi um grande parceiro para mim ao longo desta jornada, sempre demonstrando seu apoio incondicional e sendo um pilar fundamental para que eu conseguisse alcançar meus objetivos. Obrigado por estar sempre presente quando eu precisasse desabafar e por me motivar quando eu sentia que não conseguiria. É difícil expressar o quanto sua presença é importante para mim e como sou feliz por tê-lo ao meu lado.

Agradeço de fundo do coração a todos os amigos que fazem, ou fizeram, parte da minha vida, por sempre me fazerem sorrir e me darem motivações para continuar seguindo em frente.

Por fim, agradeço a todas as pessoas que direta ou indiretamente tornaram esse trabalho realidade.

RESUMO

REFINAMENTO DE ESPECIFICAÇÕES DE REQUISITO PELA DETECÇÃO DE INCONSISTÊNCIAS, AMBIGUIDADES E INCOMPLETUDES ATRAVÉS DO PROCESSAMENTO DE LINGUAGEM NATURAL

AUTOR: Otávio da Cruz Mello
Orientadora: Lisandra Manzoni Fontoura

Contexto: A definição inadequada de requisitos é recorrente em projetos de software, resultando em especificações inconsistentes, ambíguas e incompletas. Esses problemas contribuem para a incerteza no desenvolvimento, levando a custos excessivos e atrasos no cronograma. Diante desse desafio, pesquisadores buscam abordagens capazes de antecipar e mitigar esses problemas ao longo do ciclo de vida do projeto, de forma a definir histórias claras e que atendem as necessidades do usuário. **Objetivo:** O principal objetivo desta pesquisa é propor uma técnica baseada em Processamento de Linguagem Natural (PLN) para auxiliar analistas na detecção problemáticos. O PLN é ideal para lidar com requisitos, uma vez que se concentra na linguagem natural, forma comum de expressão dos requisitos. A técnica proposta visa reduzir os riscos associados ao retrabalho, oferecendo suporte aos analistas na correção de defeitos de requisitos antes que se tornem problemas para o projeto. **Metodologia:** Inicialmente foi realizada uma revisão sistemática da literatura para definição dos principais problemas da área de Engenharia de Requisitos. Com base nesses problemas, foram selecionados desafios para uma análise mais detalhada, sendo eles ambiguidade, inconsistência e incompletude de requisitos. Dado que as histórias de usuário são comumente expressas em linguagem natural, foi desenvolvida uma solução baseada em PLN, acompanhada por uma ferramenta de apoio para aplicação prática da técnica. Por fim, com base na aplicação da ferramenta usando dados reais de empresas de *software*, avaliou-se os resultados obtidos quanto a métricas de desempenho. **Resultados**: Por meio dos resultados da aplicação usando histórias de usuários de empresas de *software*, notou-se que o sistema tem precisão satisfatória de detecção de requisitos com problemas, com cerca de 89% de acerto médio. Dessa forma o sistema baseado em PLN proposto é um potencial aliado nas etapas de especificação e validação de requisitos.

Palavras-chave: Requisitos de *software*. Processamento de Linguagem Natural. Inteligência Artificial. Engenharia de Requisitos. Ambiguidade. Inconsistência. Incompletude.

ABSTRACT

REFINEMENT OF REQUIREMENT SPECIFICATIONS THROUGH DETECTION OF INCONSISTENCIES, AMBIGUITIES, AND INCOMPLETENESS USING NATURAL LANGUAGE PROCESSING

AUTHOR: Otávio da Cruz Mello
ADVISOR: Lisandra Manzoni Fontoura

Context: The poor definition of requirements is a recurring issue in software projects, resulting in inconsistent, ambiguous, and incomplete specifications. These problems contribute to uncertainty in development, leading to excessive costs and schedule delays. Faced with this challenge, researchers seek approaches capable of anticipating and mitigating these issues throughout the project lifecycle, in order to define clear stories that meet user needs. **Goal:** The main goal of this research is proposing a technique based on Natural Language Processing to help analysts detect problematic requirements. NLP is ideal for handling requirements, as it focuses on natural language, the common form of expressing requirements. The proposed technique aims to reduce the risks associated with rework, by providing support to analysts in correcting requirement defects before they become problems for the project. **Methodology:** Initially, we conducted a systematic literature review to define the main problems in the Requirements Engineering area. Based on these problems, we selected challenges for a more detailed analysis, namely ambiguity, inconsistency, and incompleteness of requirements. Given that user stories are commonly expressed in natural language, we developed a solution based on Natural Language Processing, accompanied by a supporting tool for practical application of the technique. Finally, based on the application of the tool using real data from software companies, we evaluated the results obtained regarding performance metrics. **Results**: Through the results of the application using user stories from software companies, we noted that the system has satisfactory accuracy in detecting requirements with problems, with an average accuracy rate of about 89%. Thus, the proposed NLP-based system is a potential ally in the stages of specification and validation of requirements.

Keywords: Software requirements. Natural Language Processing. Artificial Intelligence. Requirements Engineering. Ambiguity. Inconsistency. Incompleteness.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE SIGLAS

BERT	*Bidirectional Encoder Representations from Transformers* (Representações de codificador bidirecional de transformadores)
ER	Engenharia de Requisitos
ERS	Documento de Especificação de Requisitos
GLC	Gramática Live de Contexto
IA	Inteligência Artificial
NLTK	*Natural Language Toolkit* (Conjunto de ferramentas de Linguagem Natural)
UML	Linguagem de Modelagem Unificada
PLN	Processamento de Linguagem Natural
POS	*Part-of-speech* (Parte do discurso)
RSL	Revisão Sistemática da Literatura
RUP	*Rational Unified Process* (Processo Unificado da Rational)
SWEBOK	*Software Engineering Body of Knowledge* (Corpo de Conhecimento da Engenharia de *Software*)

SUMÁRIO

1 INTRODUÇÃO

Historicamente, o desenvolvimento de *software* se apresenta como uma atividade de complexo gerenciamento. Ao final da década de 1960, cerca de 10 anos após a criação do termo *software*, os pesquisadores passaram a notar os primeiros problemas surgidos no desenvolvimento e entrega de código devido a limitação de *hardware* na época, e também a má definição e gestão de processos. Tais problemas estavam fortemente associados a questões como extrapolamento de recursos financeiros e tempo, além de falhas em atingir os requisitos esperados com um nível de qualidade ideal (FITZGERALD, 2012).

A crise do *software* se estendeu por várias décadas e havia um esforço conjunto da comunidade científica em propor uma tecnologia que fosse a "bala de prata" e, portanto, solucionasse todos os problemas enfrentados no desenvolvimento (BROOKS JR, 1987). De fato, viram-se enormes melhorias em relação a gestão e qualidade de *software* com o passar dos anos, mas ainda é comum nos dias de hoje projetos de desenvolvimento de *software* que enfrentam desafios ao longo do seu ciclo de vida, ou que acabam, até mesmo, sendo descontinuados (PRESSMAN; MAXIM, 2019) (YAN et al., 2020).

Segundo uma pesquisa realizada pelo Standish Group em seu relatório CHAOS de 2019, cerca de 31% dos projetos tendem a ser descontinuados e 52% extrapolam recursos financeiros, têm entregas após os prazos estipulados e/ou não satisfazem as necessidades que foram prometidas ao cliente (Standish Group International, 2019). Esse ainda é considerado um número muito alto, principalmente após anos de evolução de tecnologia e surgimento de novos processos e práticas para desenvolvimento.

Problemas podem surgir em diferentes etapas do projeto. Podem existir erros durante a implementação devido a fatores relacionados a características da equipe, como inexperiência, ou alocação de recursos ineficiente. Pode haver, também, erros associados a falta, ou definição inadequeda, de processos de teste, o que potencialmente resulta na falha na identificação de defeitos do sistema. Entretanto, o erro de maior custo e com maior potencial de gerar impactos que irão se propagar por todas as outras fases do ciclo de vida do projeto é aquele que ocorre logo em seu início, isto é, nas etapas de levantamento e especificação dos requisitos do *software*, compreendidas pela Engenharia de Requisitos (ER) (HULL; JACKSON; DICK, 2017) (MCCONNELL, 2004) (RUIZ; HASSELMAN, 2020).

A Engenharia de Requisitos consiste de atividades relacionadas a elicitação, análise, especificação, validação e gestão dos requisitos gerados das necessidades dos clientes. Tais requisitos são inicialmente descritos em linguagem natural pelas partes interessadas e posteriormente especificados em linguagem formal por profissionais com maior envolvimento em atividades da área (SOMMERVILLE, 2019).

Considerando que os requisitos de *software* são normalmente escritos em linguagem natural, existe uma grande tendência de que eles se tornem confusos ou difíceis de

se compreender. Problemas como falta de clareza, ambiguidades e inconsistências, por exemplo, são usuais na Engenharia de Requisitos, e se tornam grandes desafios para os profissionais da área por não serem facilmente identificados até fases posteriores do projeto, quando se torna extremamente custoso solucioná-los (JACKSON, 1995) (SHAH; JINWALA, 2015) (ZAHRIN et al., 2022).

Dessa forma, os pesquisadores vêm propondo formas inovadoras de solucionar tais problemas clássicos da Engenharia de Requisitos. É possível notar que, nos últimos anos, a comunidade científica vem buscando soluções em outras áreas da ciência, por exemplo, a Inteligência Artificial (IA), para conseguirem melhores resultados na Engenharia de Requisitos (GEROGIANNIS, 2017) (LIU; REDDIVARI; REDDIVARI, 2022).

Segundo pesquisadores, a relação de IA com áreas da Engenharia de *Software* tende a gerar resultados proveitosos (HARMAN, 2012) (KOKOL, 2024). A Engenharia de *Software* é uma área fortemente focada no conhecimento, porém tem como grande obstáculo a incerteza. A partir das técnicas e algoritmos fornecidos pela IA, tal incerteza pode ser reduzida.

O Processamento de Linguagem Natural (PLN) é um subcampo de linguística e Inteligência Artificial focado no uso de computadores para processar ou entender a linguagem humana. Técnicas de PLN são usadas em diferentes sistemas, envolvendo atividades tais como: reconhecimento de fala, compreensão da linguagem falada, sistemas de diálogo, análise lexical, análise sintática, tradução automática, gráfico de conhecimento, recuperação de informações, resposta a perguntas, análise de sentimento, computação social, geração de linguagem natural e resumo de linguagem natural (JURAFSKY; MARTIN, 2000) (DENG; LIU, 2018).

Segundo o que foi exposto, o presente trabalho relata inicialmente uma Revisão Sistemática da Literatura (RSL) realizada sobre o cenário atual da Engenharia de Requisitos, entendendo quais as principais dificuldades enfrentadas atualmente e algumas abordagens recentes e inovadoras propostas por pesquisadores do campo. Em seguida, com base nas informações que foram obtidas e apresentadas, este estudo define e descreve um método baseado em PLN capaz de identificar, dentro de uma lista de requisitos, possíveis ambiguidades e inconsistências, bem como requisitos que possam estar com problemas relacionados a sua completude, e assim alertar ao usuário correções que devem ser realizadas, sendo essa técnica suportada por uma ferramenta de apoio. A técnica concentra-se, principalmente, em metodologias ágeis, dessa forma, trabalha-se com histórias de usuário.

Diante disso, a presente pesquisa busca responder a seguinte questão de pesquisa: "É viável utilizar uma abordagem fundamentada em Processamento de Linguagem Natural para detectar de forma eficaz requisitos inconsistentes, ambíguos e incompletos, visando aprimorar a especificação de requisitos?".

A dissertação está organizada da seguinte forma: No capítulo 2, exploram-se con-

ceitos importantes para a compreensão da pesquisa. No capítulo 3, discutem-se alguns trabalhos relacionados. No capítulo 4, é apresentada a metodologia e detalhados os passos para a condução da pesquisa. No capítulo 5, é descrita a etapa de Revisão Sistemática da Literatura, na qual são identificados os principais problemas da área de Engenharia de Requisitos. No capítulo 6, apresenta-se a proposta de técnica baseada em Processamento de Linguagem Natural com base nos desafios encontrados pela condução da RSL. No capítulo 7, são demonstrados e analisados os resultados obtidos pela utilização da ferramenta de apoio. Por fim, no capítulo 8 debatem-se as conclusões atingidas a partir dos resultados, os desafios encontrados durante a pesquisa e possíveis futuros trabalhos.

2 FUNDAMENTAÇÃO TEÓRICA

Este capítulo tem como objetivo apresentar e contextualizar alguns temas base que são tratados ao longo do trabalho, de forma a situar o leitor e tornar o seu entendimento mais claro. Ao longo desta dissertação, outros conceitos adicionais são explorados e devidamente explicados.

O referencial teórico se divide em quatro partes. Inicialmente se define o que é um requisito de *software* e a diferença entre os tipos de requisitos existentes para então entrar em maiores detalhes sobre o processo de Engenharia de Requisitos em projetos de desenvolvimento de *software*. Em seguida, apresentam-se fatores importantes na definição de requisitos para a garantia de qualidade, bem como os benefícios de uma especificação de requisitos adequada e o conceito de propagação de erros em projetos. Por fim, introduzem-se conceitos de PLN, técnica abordada ao longo deste trabalho.

2.1 REQUISITOS DE *SOFTWARE*

O requisito de um *software* é um termo com várias definições na literatura. De acordo com o Instituto de Engenheiros Eletricistas e Eletrônicos (IEEE, 1990), um requisito pode ter os seguintes significados :

1. Uma condição ou capacidade necessária por um usuário para resolver um problema ou atingir um objetivo.
2. Uma condição ou capacidade que deve ser atendida ou possuída por um sistema ou componente do sistema para satisfazer um contrato, padrão, especificação ou outro documento formalmente imposto.
3. Uma representação em documento de uma condição ou capacidade como no primeiro ou segundo item.

O guia para o Corpo de Conhecimento da Engenharia de *Software*, também conhecido como guia SWEBOK, por outro lado define requisito de *software* como “uma propriedade que deve ser exibida em algo com a finalidade de resolver algum problema do mundo real“ (BOURQUE; FAIRLEY, 2014). Outras definições surgem de outras publicações, ou até mesmo são estabelecidas pelas próprias companhias que desenvolvem *software*.

Apesar de “requisito“ ser um termo subjetivo e sem uma definição consistente na indústria, pode se assumir que esse certamente tem uma característica essencial que é comum em todas definições: ele descreve algo que é necessário para alguém. No caso de

desenvolvimento de *software*, usualmente, um requisito é tratado como uma funcionalidade que um sistema deve possuir para que as necessidades do seu usuário sejam satisfeitas (SOMMERVILLE, 2019).

Um requisito pode ser classificado em duas principais categorias de acordo com seu conteúdo: requisitos funcionais e requisitos não-funcionais (LAPLANTE, 2017).

2.1.1 Requisitos funcionais

Os requisitos funcionais descrevem as funcionalidades que são requeridas pelo sistema para satisfazer as necessidades do usuário. Requisitos desse tipo podem possuir um maior ou menor nível de detalhamento, de acordo com a sua complexidade e os *stakeholders* que se deseja engajar (LAMSWEERDE, 2009).

Para definir e elucidar suas características, nessa seção tem-se exemplos de requisitos funcionais em uma aplicação de *planning poker*, técnica *gamificada* para estimativa de esforço. Pode se notar que tais requisitos descrevem especificamente algumas das funcionalidades que o sistema deve possuir para que seja possível iniciar o jogo.

- *"Como moderador, eu quero criar um novo jogo inserindo um nome e uma descrição opcional, para que eu possa começar a convidar estimadores."*
- *"Como moderador, quero convidar os estimadores dando a eles um URL onde eles possam acessar o jogo, para que possamos iniciar o jogo."*

Requisitos funcionais, como os mostrados acima, geralmente são escritos em linguagem natural, de forma a melhorar a comunicação e a compreensão das partes envolvidas. Também é possível representá-los de outras formas, como graficamente por meio de diagramas e modelos (LAPLANTE, 2017).

2.1.2 Requisitos não-funcionais

Ao passo que requisitos funcionais descrevem funcionalidades necessárias no sistema, requisitos não-funcionais especificam restrições do sistema e como ele deve se comportar em relação a alguns atributos. Requisitos não-funcionais podem ser influenciados por inúmeros atributos, entre eles os clássicos finalizados com o sufixo “ibilidade“, tais como confiabilidade, usabilidade, disponibilidade, manutenibilidade, portabilidade, escalabilidade, entre outros.

Na Figura 1 é ilustrada uma taxonomia de requisitos não-funcionais para sistemas, proposta por (ADAMS, 2015). O pesquisador estudou a literatura e definiu os 27 principais tipos de requisitos não-funcionais entre mais de duzentos.

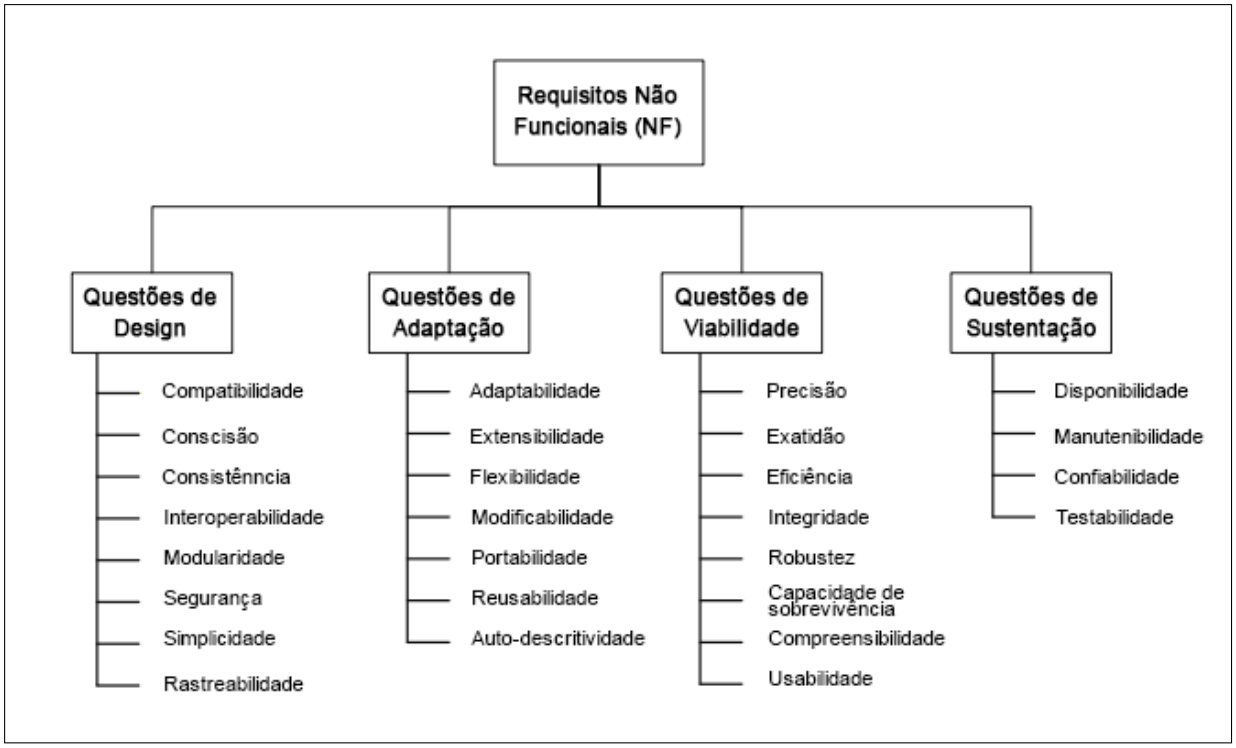

Figura 1 – Taxonomia de requisitos não funcionais para sistemas, proposta por Adams (2015).

Fonte: Adaptado de (ADAMS, 2015).

A especificação e implementação de requisitos não-funcionais é uma tarefa complexa pois, muitas vezes, eles não estão associados a um único componente e sim ao sistema como um todo, e além disso normalmente surgem como necessidades genéricas do cliente. Falhar em atingir um requisito não funcional pode significar que o sistema é inutilizável, dessa forma é correto considerá-los tão críticos quanto requisitos funcionais (SOMMERVILLE, 2019).

2.2 COMO DESCREVER UM REQUISITO?

Um requisito pode ser descrito de inúmeras formas. É possível utilizar palavras, imagens, vídeos, diagramas, entre outros meios para conseguir expressar as necessidades do cliente ou usuário para as partes interessadas. Entre as principais formas de descrição de um requisito, atualmente, estão os casos de uso e as histórias de usuário. Apesar de ambos terem a mesma função de especificar uma necessidade, cada um tem suas características específicas mais adequadas a tipos de projetos e *stakeholders* com que se está tratando.

2.2.1 Casos de uso

Casos de uso são uma das ferramentas mais clássicas e consolidadas para representação de requisitos, sendo introduzidos no ano de 1986 por Ivar Jacobson e tendo grande relevância na Engenharia de Requisitos até os dias de hoje.

A principal característica de um caso de uso é a quantidade de informações presentes em sua especificação. Tem-se, em um documento de casos de uso, um conjunto completo de interações entre sistemas e seus usuários (ou atores), e todos os possíveis cenários ocasionados por tais interações. É comum complementarmos o documento com diagramas em linguagem de modelagem unificada (UML) para melhor representação (Object Management Group, 2011). Um exemplo de um diagrama de casos de uso genérico para um sistema médico está ilustrado na Figura 2.

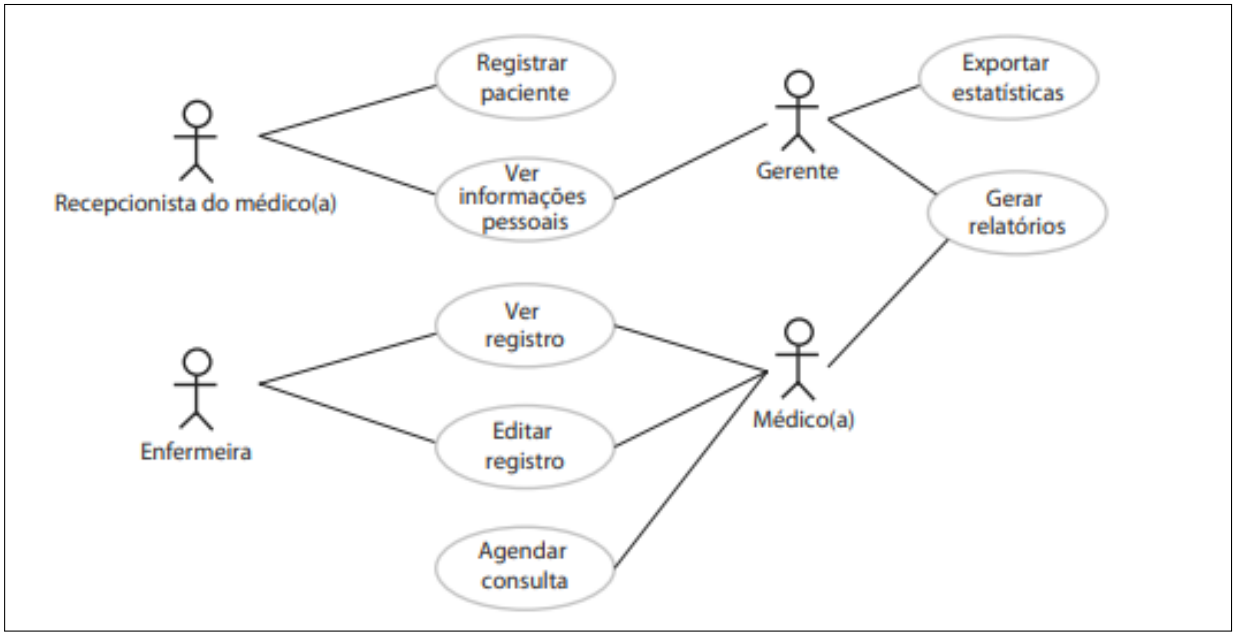

Figura 2 – Exemplo de um diagrama de casos de uso.

Fonte: Adaptado de (SOMMERVILLE, 2019).

Justamente por ser um método de representação mais antigo e ter uma documentação mais extensa e com mais informações, temos casos de uso normalmente associados a processos tradicionais, como o *Rational Unified Process* (RUP) (COHN, 2004). Por outro lado, não é totalmente incomum a utilização de casos de uso em métodos ágeis. Por mais que não seja ideal para representação de requisitos por existirem outros meios mais adequados, eles podem ser um bom complemento para requisitos ágeis (GALLARDO-VALENCIA; OLIVERA; SIM, 2007).

2.2.2 Histórias de usuário

Se, por um lado, as especificações dos casos de uso são descrições longas e detalhadas de requisitos, histórias de usuário são especificações sucintas e informais acerca de necessidades do usuário. Histórias de usuário também têm duração de vida mais curta, sobrevivendo somente em iterações nas quais são planejadas, ao contrário de casos de uso, que são permanentes ao longo do ciclo de vida do desenvolvimento do produto.

Por possuírem tais características, histórias de usuário são normalmente adotadas em projetos com características ágeis. Basicamente são descrições de funcionalidades de valor para um cliente ou usuário, e são geralmente escritas em linguagem natural e entendível em cartões que contem três informações essenciais: o usuário da funcionalidade (Quem?), a funcionalidade desejada (O quê?) e por qual razão tal funcionalidade é necessária (Por quê?), tal como ilustrado na Figura 3.

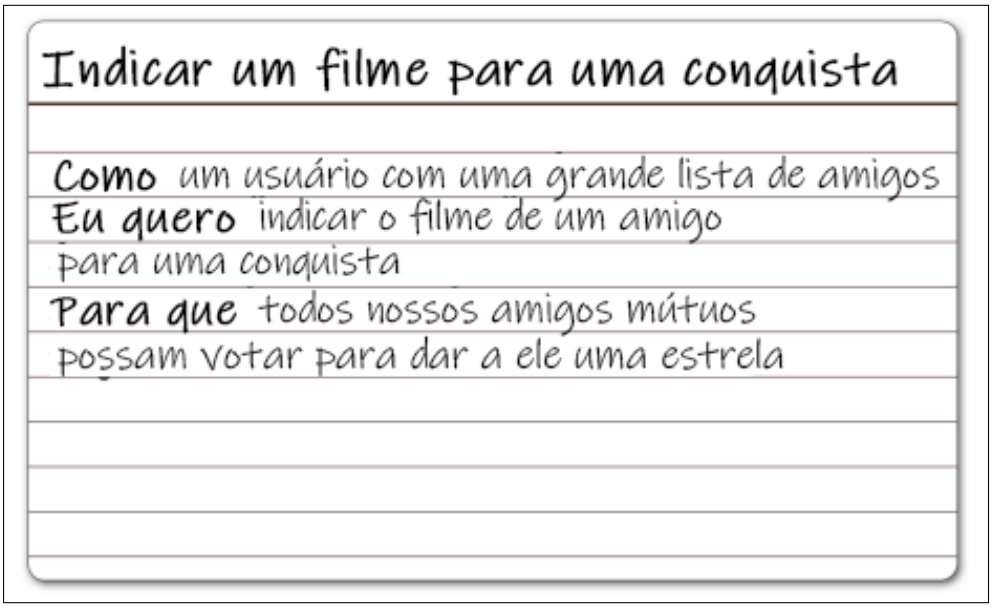

Figura 3 – Exemplo de uma história de usuário escrita em um cartão.

Fonte: Adaptado de (STELLMAN; GREENE, 2014).

Existem vantagens e desvantagens no uso de histórias de usuário. A principal vantagem é a facilidade de entendimento das histórias pelos *stakeholders*, evitando-se o excesso de detalhamento e informações desnecessárias, dando foco somente na funcionalidade de valor. Além disso, esse aspecto das histórias também torna mais fácil a sua priorização. Por outro lado, também pelo motivo de não existir detalhamento, existe o risco de histórias de usuário não contemplarem tudo que é requerido pelo cliente (PATTON; ECONOMY, 2014) (COHN, 2004).

2.3 O PROCESSO DE ENGENHARIA DE REQUISITOS

Podemos dividir o processo de Engenharia de Requisitos em duas etapas principais: a etapa de desenvolvimento de requisitos, que compreende atividades iniciais de elicitação, análise, especificação e validação dos requisitos, e a etapa de gerenciamento de requisitos, em que se dá foco a questões como rastreabilidade e volatilidade dos requisitos definidos na etapa anterior (WIEGERS; BEATTY, 2013). O principal foco da presente pesquisa está na etapa de desenvolvimento, então não discute-se profundamente sobre gerenciamento de requisitos. A Figura 4 ilustra o processo de desenvolvimento de requisitos.

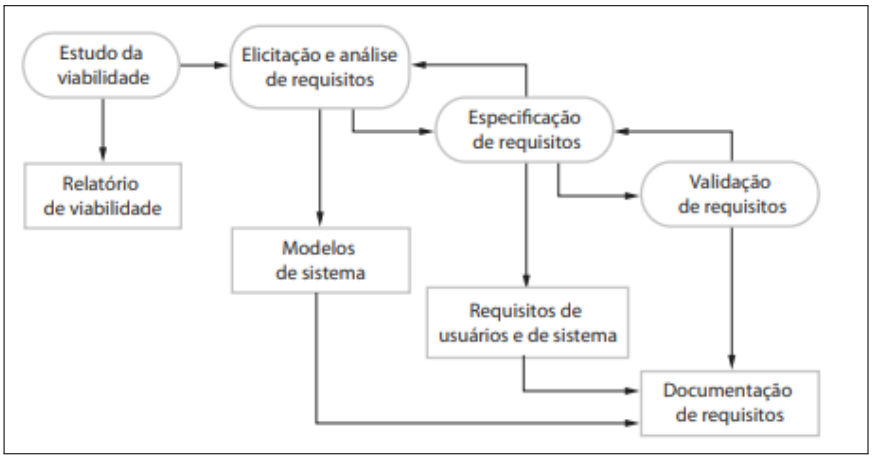

Figura 4 – Processo de Engenharia de Requisitos.

.

Fonte: Adaptado de (SOMMERVILLE, 2019).

Uma das etapas mais cruciais durante um projeto é o levantamento inicial dos requisitos funcionais e não funcionais de valor para as partes interessadas. O levantamento pode ocorrer de diferentes maneiras, desde entrevistas individuais com *stakeholders* até aplicação de questionários e *workshops*. A técnica utilizada para elicitação depende das características do projeto e, muitas vezes, é comum que até mesmo mais de uma técnica seja empregada.

O que se espera ao final da atividade de elicitação de requisitos é que exista um entendimento dos problemas e das necessidades dos *stakeholders*. Deve se tomar nota de todos os assuntos de importância discutidos durante a sessão de elicitação e organizá-los, demarcando claramente o que está dentro e fora do escopo do projeto. Também é importante ter em mente que é impossível documentar todos os requisitos necessários ao sistema logo ao início do projeto, mas deve se evitar ao máximo deixar de especificar funcionalidades críticas, pois essas podem gerar riscos futuros ao projeto (WIEGERS; BEATTY, 2013).

A atividade de análise de requisitos, que segue, está associada ao refinamento dos requisitos levantados na atividade de elicitação. Quando o levantamento inicial é feito com

o cliente, podem existir requisitos conflitantes, confusos, ou até mesmo com interpretação errônea por parte do analista. Dessa forma, é importante corrigir tais erros antes que os requisitos comecem a ser implementados, visto que esses podem produzir danos cada vez maiores na medida que o projeto avança em suas fases.

Antes de finalizar o documento de especificação dos requisitos com os *stakeholders*, é importante que algumas perguntas acerca dos requisitos sejam esclarecidas (LAPLANTE, 2017).

- Os requisitos estão completos?
- Os requisitos estão claros?
- Os requisitos podem ser implementados?

Após a atividade de análise de requisitos, é necessário documentar todas as informações obtidas até o momento em um documento formal de especificação de requisitos (ERS), que seja compreensível pelo público alvo. É importante a adoção de *templates* e padrões de documentação para melhor organização. No documento ERS, são incluídos requisitos classificados em requisitos do usuário e requisitos do sistema, que são respectivamente descrições abstratas e detalhadas das funcionalidades do sistema (SOMMERVILLE, 2019).

Por fim, a validação é a atividade final da etapa de definição de requisitos. Basicamente, é a garantia de que os requisitos especificados são representações corretas das necessidades do usuário. Para alcançar essa garantia, é preciso tanto verificar os requisitos, que consiste em identificar se os requisitos foram escritos de maneira correta, quanto validar os requisitos, ou seja, identificar se os requisitos corretos foram especificados.

Essa atividade analisa os requisitos especificados em inúmeros quesitos, sendo alguns: sua completude, sua consistência, se estão corretamente descritos e não geram dúvidas, se são necessários, entre outros (WIEGERS; BEATTY, 2013). Dentro das atividades da Engenharia de Requisitos, essa é a de maior importância para a presente pesquisa, pois envolve a correção dos requisitos especificados.

2.4 QUALIDADE DE REQUISITOS

Como mencionado anteriormente, um requisito com falhas de especificação tende a se tornar um entrave com o avanço do projeto. Isso acontece porque os requisitos definem como o sistema deve ser durante as fases pré-construção do projeto, em que ainda não existem muitos artefatos entregáveis como códigos, diagramas, modelos, entre outros. Dessa forma, tudo que é feito após a etapa de desenvolvimento de requisitos terá grande

influência do que foi definido como funcionalidade requerida pelo sistema. Se tal funcionalidade estiver mal especificada, com informações vagas ou incompletas por exemplo, existe a possibilidade de que os requisitos propaguem erros para todas as fases do ciclo de vida do projeto.

Analisando a história de usuário abaixo:

- *"Como usuário, eu quero compartilhar arquivos com outras pessoas, para que todos possam ter acesso aos meus arquivos."*

O requisito acima contém alguns problemas. Ele apresenta palavras vagas e que podem gerar múltiplas interpretações de acordo com o leitor. Por exemplo, o desenvolvedor pode considerar que arquivos de texto devem ser compartilhados no sistema por meio de redes sociais, e então desenvolve a funcionalidade da maneira que interpretou. Por outro lado, o cliente desejava que o sistema também pudesse compartilhar arquivos de foto e vídeo, não por redes sociais, mas por *e-mail* e por mensagem de texto. Dessa forma, o sistema não atende a necessidade do cliente e deve ser retrabalhado para satisfazer o requisito que foi mal especificado pelo analista, gerando aumento de custos financeiros e atrasos ao projeto.

Casos como esse se tornam mais complexos a medida que o projeto avança em suas fases finais, pois alterar um sistema quase completo é mais custoso do que alterar um sistema que está no início de seu desenvolvimento. A história de usuário utilizada como exemplo poderia ter inúmeros outros requisitos associadas a ela já implementados, e esse erro de especificação poderia representar um retrabalho em tais requisitos também.

De modo a evitar erros futuros, é ideal especificar requisitos de qualidade nas etapas iniciais de especificação. Para tal finalidade, além da existência de um processo de Engenharia de Requisitos bem definido, é preciso também existir técnicas e ferramentas que auxiliem o analista durante as atividades de levantamento, análise, especificação e validação.

2.5 PROCESSAMENTO DE LINGUAGEM NATURAL

O Processamento de Linguagem Natural (PLN) é um subcampo de IA e linguística voltado para o entendimento e manipulação da linguagem humana por meio de computadores. Este campo está efetivamente presente em várias aplicações cotidianas, incluindo, mas não se limitando a: tradução de texto, análise de sentimentos, e geração e compreensão de texto.

O PLN vem sendo exponencialmente explorado em diversas áreas da ciência devido ao seu potencial para transformar a maneira como interagimos com dados textuais e informações linguísticas (KHURANA et al., 2022). O processo de análise de linguagem

é composto de cinco fases principais (INDURKHYA; DAMERAU, 2010), como ilustrado na Figura 5.

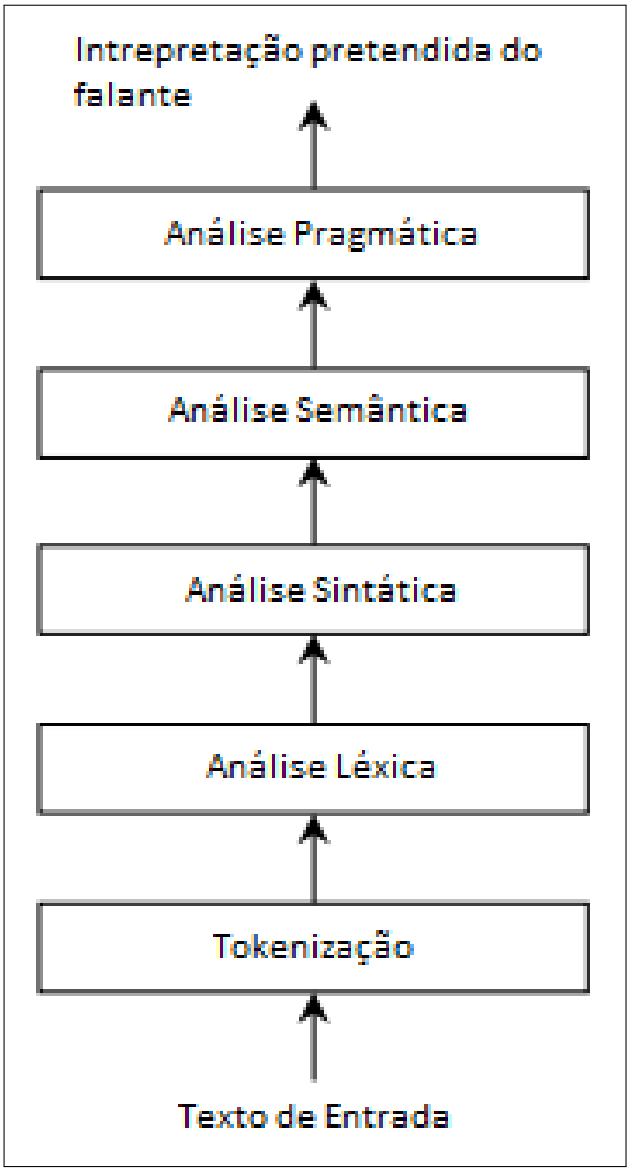

Figura 5 – Processo de análise de linguagem, proposto por Indurkhya e Damerau.

Fonte: Adaptado de (INDURKHYA; DAMERAU, 2010).

O primeiro estágio do PLN envolve a tokenização e segmentação do texto inicial, bem como a Análise Léxica dos *tokens* gerados. Basicamente, o principal objetivo dessas etapas é separar a entrada em palavras e fragmentos de texto, identificando e categorizando essas unidades de acordo com o seu significado dentro do seu contexto linguístico. Uma forma de fazer essa categorização, por exemplo, é atribuindo classes gramaticais aos *tokens* individuais. Os trechos extraídos nessa etapa continuarão sendo processados nas fases seguintes.

O processamento segue com a Análise Sintática do texto. Essa etapa envolve a análise de palavras em uma frase, de forma a entender o seu relacionamento, sua ideia e

a lógica por trás de sua organização. Isso é feito pelo uso de métodos, por exemplo, árvores sintáticas (*parse trees*), que gera todas as derivações possíveis de uma determinada expressão, e são usadas para aplicações que envolvem análise de discurso, estrutura e gramática (KOBAYASHI et al., 2020).

Em sequência, a Análise Semântica busca o significado exato da frase, de forma a conseguir interpretar palavras individuais em contextos específicos, extraindo assim conhecimento de valor. Essa etapa é especialmente útil em áreas que envolvem análise de sentimento e conteúdo, busca de informações em grandes bases e, até mesmo, tradução precisa de texto.

Por fim, a Análise Pragmática tem como objetivo re-interpretar a frase de acordo com o seu verdadeiro significado, que exige conhecimento do mundo real. A partir dessas etapas de PLN, é possível a máquina entender o sentido do texto escrito ou falado em linguagem humana (INDURKHYA; DAMERAU, 2010). Um exemplo seria fornecer como entrada a frase “Você sabe que horas são?” e analisá-la e interpretá-la como um pedido de informação do horário atual.

Atualmente, há uma ampla variedade de recursos disponíveis para os desenvolvedores focados no desenvolvimento de sistemas inteligentes destinados ao PLN. O spaCy e o *Natural Language Toolkit* (NLTK) são bibliotecas de PLN em Python que se destacam por oferecer suporte a uma variedade de tarefas, incluindo tokenização, análise gramatical, reconhecimento de entidades nomeadas e treinamento de modelos personalizados, de forma simples e eficiente (HONNIBAL; MONTANI, 2017) (BIRD; KLEIN; LOPER, 2009). Além dessas, existem outras ferramentas e frameworks, como o BERT (Bidirectional Encoder Representations from Transformers), desenvolvido pela Google, que revolucionou o campo ao permitir a compreensão mais profunda do contexto bidirecional das palavras em uma frase, melhorando significativamente a precisão em diversas tarefas de PLN (DEVLIN et al., 2018).

O Rrocessamento de Linguagem Natural surge como uma tecnologia que impacta significativamente a sociedade em vários setores. Desde a automação de tarefas administrativas até a personalização de experiências de usuário em plataformas digitais, o PLN redefine como interagimos com dados textuais e linguísticos. Com avanços contínuos em inteligência artificial e aprendizado de máquina, podemos esperar que o PLN não apenas melhore a eficiência operacional e a tomada de decisões, mas também abra novas fronteiras em áreas como assistência médica, educação personalizada e análise preditiva de mercado (RASHEED; GHWANMEH; ABUALKISHIK, 2023) (ZHANG, 2024). À medida que essa tecnologia evolui, sua capacidade de compreender e interpretar nuances linguísticas transforma fundamentalmente a maneira como o conhecimento é acessado, comunicado e aplicado.

3 TRABALHOS RELACIONADOS

Este capítulo apresenta uma análise de trabalhos que utilizam técnicas de Processamento de Linguagem Natural para desenvolver soluções para desafios da área de Engenharia de Requisitos. O principal objetivo desta seção é investigar as lacunas na literatura que a presente pesquisa pretende preencher.

Zait e Zarour (2018) propuseram uma abordagem para detecção e correção de ambiguidades usando técnicas de Processamento de Linguagem Natural e Web Semântica. A abordagem proposta identifica palavras ambíguas em requisitos e sugere possíveis interpretações e significados. O estudo foi publicado em sua fase inicial e como trabalho futuro os pesquisadores têm como intuito também detectar requisitos vagos e incompletos (ZAIT; ZAROUR, 2018).

Em um estudo similar, Asadabadi et. al (2020) propuseram uma abordagem semiautomática para identificar e clarificar ambiguidades em projetos de larga escala, que são compostos de um grande número de requisitos. A abordagem é baseada em Processamento de Linguagem Natural e técnica de computação suave usando a teoria de conjunto difuso. Os pesquisadores notaram que os requisitos ambíguos são identificados com precisão, porém a lógica de conjunto difuso não foi capaz de identificar todos os tipos de ambiguidades textuais. Assim, como trabalho futuro, os pesquisadores propuseram desenvolver ferramentas e técnicas mais potentes, capazes de resolver as ambiguidades restantes (ASADABADI et al., 2020).

Mishra e Sharma (2019) buscaram identificar e detectar palavras do domínio ambíguas dentro de um texto de linguagem natural. Os pesquisadores aplicaram a técnica de Processamento de Linguagem Natural baseada em incorporação de palavras para detectar tais ambiguidades. Os pesquisadores concluíram que a técnica é eficaz em identificar ambiguidades específicas de domínio, e pode ser aplicada em documentos de diferentes tamanhos, podendo ser útil em projetos de larga escala (MISHRA; SHARMA, 2019).

Ferrari et. al (2014) apresentaram uma proposta para detecção de ambiguidades pragmáticas em requisitos de linguagem natural. Tal tipo de ambiguidade depende do contexto do requisito, que inclui o conhecimento do leitor. A proposta é fundamentada na modelagem de grafo do conhecimento de diferentes leitores, usando o algoritmo de caminho mais curto para modelar a interpretação pragmática do requisito. A comparação de diferentes interpretações é usada pra decidir se o requisito é ou não ambíguo. O método foi avaliado em ambiente real e os resultados foram positivos (FERRARI et al., 2014).

Por fim, Dalpiaz et. al (2018) também propuseram uma abordagem voltada para detecção de ambiguidades, adicionalmente focada na identificação de requisitos incompletos. Para isso, os pesquisadores combinaram técnicas de Processamento de Linguagem Natural com técnicas de visualização de informação, para interpretação do tipo de defeito.

Em uma avaliação preliminar, foi possível notar que a abordagem proposta pode ter uma revocação melhor do que uma inspeção papel e caneta, porém sem diferença significativa na precisão. Como trabalhos para o futuro, os pesquisadores têm como objetivo testar a abordagem em larga escala e aprimorar os algoritmos de detecção de ambiguidades (DALPIAZ; SCHALK; LUCASSEN, 2018).

Na Tabela 12, é apresentada uma comparação entre o presente estudo e os trabalhos mencionados. Esta pesquisa se diferencia dos trabalhos mencionados devido ao seu objetivo específico: propor uma técnica que aborda os desafios relacionados tanto à ambiguidade quanto à incompletude e inconsistência de requisitos.

Tabela 1 – Comparação entre estudos relacionados.

Estudo	**Foco**			**Tecnologia**
	Ambig.	Incons.	Incompl.	
Zait e Zarour (2018)	✓	✗	✗	PLN, Web Semântica
Asadabadi et. al (2020)	✓	✗	✗	PLN, Computação Suave
Mishra e Sharma (2019)	✓	✗	✗	PLN
Ferrari et. al (2014)	✓	✗	✗	Grafos
Dalpiaz et. al (2018)	✓	✗	✓	PLN
Abordagem Proposta	✓	✓	✓	PLN

Essa abordagem se fundamenta no emprego de técnicas avançadas de Processamento de Linguagem Natural, visando aprimorar a qualidade da especificação e a validação de requisitos de *software* de forma simplificada. Frequentemente um documento de requisitos apresenta numerosas inconsistências como destacado pela Revisão Sistemática da Literatura executada. É válido avaliar as histórias de usuário sobre diferentes perspectivas, nesse caso, buscando ambiguidades, inconsistências e incompletudes.

O uso de uma técnica abrangente para detecção de problemas possibilita a formulação de requisitos mais claros e corretos, contribuindo para uma compreensão mais profunda das necessidades do usuário. Além disso, resulta em uma maior precisão na definição dos requisitos do *software*, o que é crucial para o sucesso do projeto e para garantir que o produto final atenda plenamente às expectativas e exigências dos usuários.

3.1 IA GENERATIVA

Os sistemas de Processamento de Linguagem Natural e os sistemas de IA Generativa, como os modelos GPT-3 e GPT-4, são diferentes abordagens para manipulação texto. O PLN concentra-se na decodificação e compreensão da linguagem humana para analisar dados e fazer previsões (NATH et al., 2022). Por outro lado, os sistemas generativos de IA visam criar conteúdos baseados em padrões e contextos aprendidos, a partir de *prompts* de pesquisa (ZHAN et al., 2024).

Em tarefas específicas, como as da presente pesquisa, as vantagens dos sistemas de PLN residem na sua capacidade de analisar texto de forma eficiente e consistente dentro do seu contexto. Em contraste, a IA generativa depende de um *prompt* fornecido para execução de uma determinada tarefa e pode produzir diferentes respostas e interpretações para uma mesma entrada (DUC et al., 2023) (MORALES et al., 2023). Tal variabilidade compromete a precisão e confiabilidade da análise dos requisitos, visto que histórias de uma mesma especificação podem ser tratadas diferentemente pela IA generativa.

Adicionalmente, pesquisadores estudam o fato de respostas produzidas por IA generativa ainda serem pouco confiáveis, até mesmo em tarefas envolvendo busca de informações ou compreensão da entrada do usuário (AMARO et al., 2023) (MEMON; WEST, 2024) (GWON et al., 2024). Em estudos comparativos, modelos como GPT-4 se mostraram menos precisos em suas respostas do que abordagens baseadas em PLN (SURAWORACHET; SEON; CUKUROVA, 2024) (HUTT et al., 2024).

A incipiência da Engenharia de *Prompts*, campo recente dentro da área de IA generativa, está parcialmente associada a muitos desses fatores, uma vez que a qualidade das respostas geradas pelos modelos depende diretamente da formulação adequada dos *prompts* fornecidos (KORZYNSKI et al., 2023). Com o avanço e a consolidação deste campo, é possível que a IA generativa eventualmente se torne uma tecnologia mais precisa e confiável.

Levando em conta essas considerações e os resultados obtidos por pesquisadores usando e aplicando ambas as abordagens, optamos por utilizar o Processamento de Linguagem Natural para a presente pesquisa, visando uma melhor precisão das respostas retornadas.

4 METODOLOGIA

A metodologia utilizada neste estudo foi planejada visando assegurar a validade da solução proposta, evidenciando sua relevância e aplicabilidade no contexto atual de Engenharia de Requisitos (ER), bem como a lacuna que visa preencher. Esta seção aborda os procedimentos e abordagens empregados para compreender o cenário atual da área de ER, desenvolver a técnica proposta, e analisar os resultados obtidos. A Figura 6 ilustra o fluxo geral de atividades que compõem a metodologia.

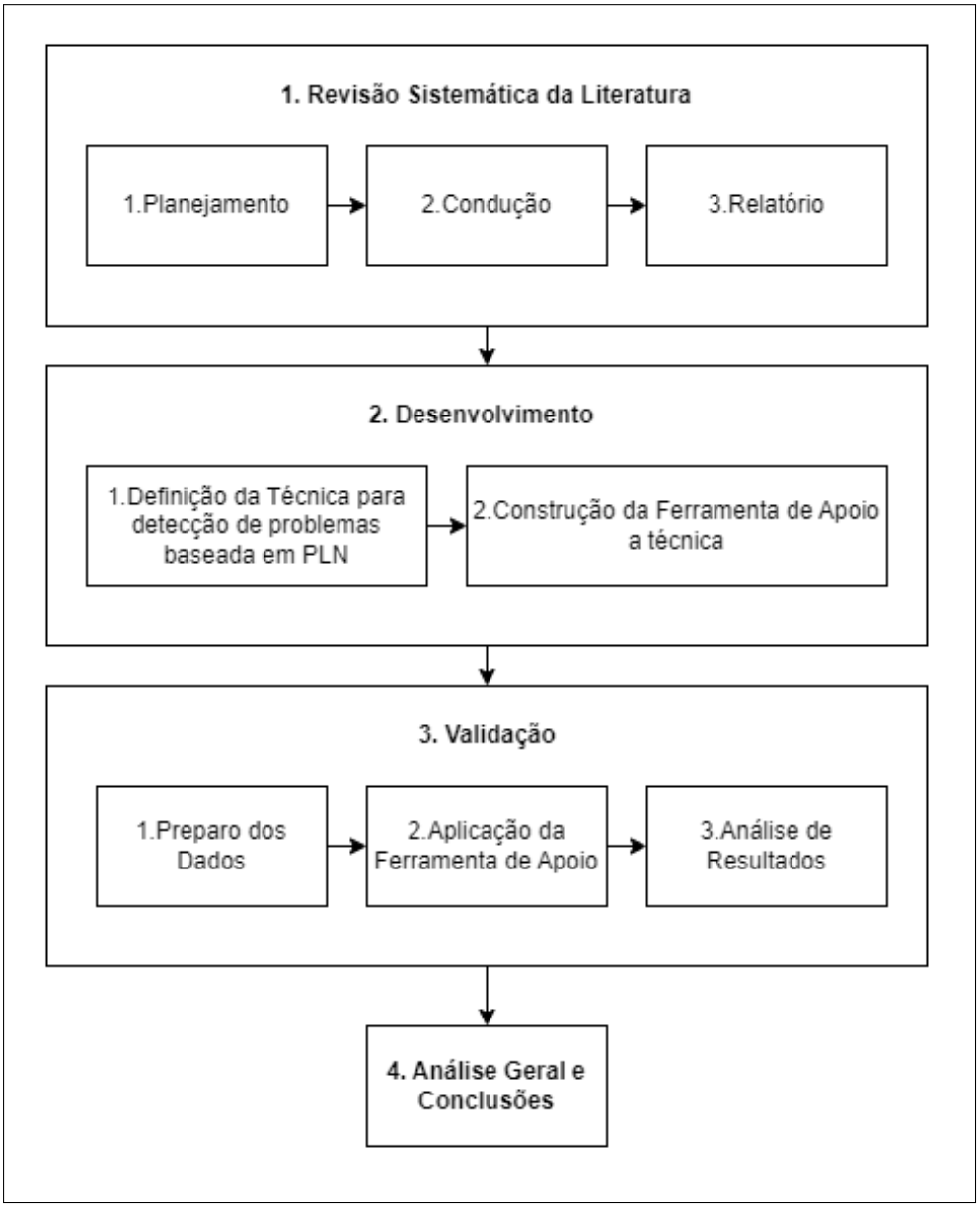

Figura 6 – Fluxo geral da metodologia.

Fonte: Autoria própria.

Inicialmente, para contextualizar os principais desafios enfrentados na área de En-

genharia de Requisitos na atualidade e as abordagens existentes para solucioná-los, foi realizada uma RSL. Essa revisão foi conduzida em três fases distintas: a fase de planejamento, que envolveu atividades como o estabelecimento de critérios de seleção e a definição da estratégia de busca; a fase de condução, que abarcou a coleta, seleção e análise dos estudos identificados; e, por fim, a fase de relatório das informações obtidas após as etapas anteriores (KITCHENHAM; CHARTERS, 2007).

Com base na RSL conduzida, foram identificados problemas principais na área que demandavam um estudo mais aprofundado. Tais problemas envolviam ambiguidade, incompletude e inconsistências de requisitos. Com base nos princípios de PLN, uma técnica foi desenvolvida para auxiliar analistas de requisitos a abordar esses problemas, definindo conceitos e atividades principais.

A técnica define três ciclos principais pelos quais a história de usuário é submetida, cada um focado em identificar a ocorrência de um dos problemas estudados: o ciclo de completude, o ciclo de não-ambiguidade e o ciclo de consistência. No ciclo de completude, a análise da história de usuário é realizada por meio de uma abordagem semântica do requisito, onde os trechos são rotulados utilizando etiquetas *PropBank*. Já nos ciclos de não-ambiguidade e consistência, a análise é conduzida através de uma abordagem sintática. Os trechos são rotulados e analisados utilizando etiquetas POS (*Part of Speech*) e são adicionalmente realizadas análises de árvores livres de contexto, no caso do ciclo de não-ambiguidade.

Em paralelo a essa técnica, foi desenvolvida uma ferramenta de apoio para possibilitar a aplicação prática e avaliação da solução proposta. Esta ferramenta foi elaborada utilizando recursos das bibliotecas spaCy e NLTK para PLN. Usando como entrada dados selecionados de empresas de *software*, adquiridos de plataformas digitais, a ferramenta foi empregada para avaliar métricas de taxas de acertos e erros. A análise interpretativa dos resultados permitiu a avaliação do desempenho da técnica com as bases de dados selecionadas.

Por fim, realizou-se uma avaliação abrangente de todo o trabalho realizado, analisando o impacto da técnica proposta e ponderando sobre possíveis melhorias futuras, visando aperfeiçoar os resultados obtidos.

5 CONTEXTUALIZANDO O CENÁRIO

Para contextualizar o cenário atual do campo de Engenharia de Requisitos, realizou-se uma pesquisa na literatura com o escopo de entender quais são os principais desafios enfrentados pelos analistas de requisitos atualmente e, assim, buscar abordagens de apoio ainda não desenvolvidas compatíveis com os resultados encontrados. Tais resultados foram obtidos a partir de uma revisão de estudos publicados por pesquisadores acerca de experiências observadas nas atividades de Engenharia de Requisitos (MELLO.; FONTOURA., 2022).

5.1 PROCEDIMENTOS PARA REVISÃO

Para a realização da RSL, foram seguidos o processo e as diretrizes propostos por Kitchenham e Charters (KITCHENHAM; CHARTERS, 2007). Tal processo consiste de três fases principais, que estão ilustradas na Figura 7. Cada uma das etapas tem procedimentos e atividades padrões que devem ser seguidas para que a pesquisa possa produzir resultados adequados e significativos.

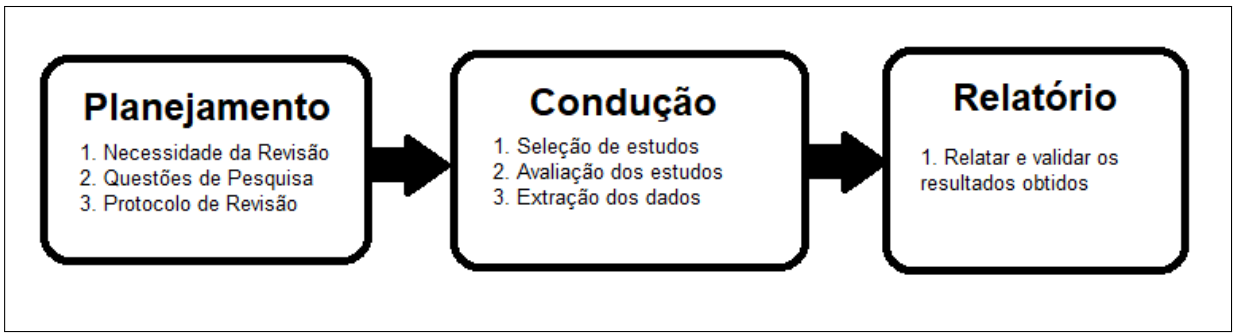

Figura 7 – Processo de Revisão Sistemática da Literatura proposto por Kitchenham e Charters.

Fonte: Autoria própria, com base nos estudos de (KITCHENHAM; CHARTERS, 2007).

5.1.1 Planejamento

A fase inicial de planejamento envolve basicamente as atividades de preparo para a realização da revisão. É no planejamento que é definido o objetivo que se deseja alcançar com a pesquisa e quais vão ser os passos para se conseguir chegar até esse objetivo. Tais passos são:

Definir a necessidade da revisão. A necessidade desta pesquisa foi originada dos problemas usualmente enfrentados por profissionais nas atividades de Engenharia de Requisitos. Na grande maioria das vezes projetos de desenvolvimento de *software* acabam sendo duramente afetados por esses problemas, tendo impactos de grande magnitude nos prazos e custos previamente estimados pelo gerente do projeto. Dessa forma, este estudo teve foco no entendimento sobre quais são as dificuldades mais frequentes nessa área e também quais soluções existem atualmente para impedir, ou ao menos reduzir, essas dificuldades.

Definir as questões de pesquisa. As questões de pesquisa auxiliam o pesquisador a dar foco ao seu estudo, respondendo perguntas específicas elaboradas por ele para alcançar o seu objetivo. Para esta revisão se definiram três questões de pesquisas a serem respondidas.

- **Q_1:** Quais os desafios mais frequentemente enfrentados na área de Engenharia de Requisitos atualmente?
- **Q_2:** Quais abordagens são propostas para impedir ou reduzir tais dificuldades?
- **Q_3:** Quais as oportunidades existem para futuras pesquisas de acordo com o cenário atual?

Para responder a questão de pesquisa Q_1, foram estudados trabalhos publicados entre 2015 e 2021 relatando problemas experienciados na área de Engenharia de Requisitos durante o desenvolvimento de projetos de *software*. A partir do mapeamento dos problemas, foram identificadas na literatura as técnicas e abordagens que existem atualmente para solucioná-los e assim responder a questão de pesquisa Q_2. Por fim, a pergunta Q_3 é respondida com base nas duas questões de pesquisa anteriores. Foram analisados os pontos que estão em aberto ou ainda podem ser explorados a fim de melhorar as atividades da área e reduzir riscos.

5.1.2 Condução

A partir da definição do protocolo de revisão, que estabelece quais métodos e procedimentos vão ser tomados na etapa de seleção e análise dos estudos, é possível começar a execução das atividades que foram planejadas na etapa de planejamento. A condução da revisão sistemática da literatura é composta dos seguintes passos:

Definição dos termos de busca. Os estudos foram recuperados das bases científicas IEEE Xplore, ACM Digital Library e ScienceDirect. Para a filtragem das publicações mais relevantes para esta revisão, foram definidos termos de busca que contemplam as

questões de pesquisa definidas. O resultado foi a seguinte string de busca:

("requirements engineering"OR "requirements management"OR "requirements elicitation"OR "requirements analysis"OR "requirements specification"OR "requirements validation"OR "software requirements") AND ("challenges"OR "issues"OR"obstacles"OR "difficulties"OR "burden"OR "problems"OR "complications"OR "hardship")

Dessa forma, a string limita a pesquisa para publicações que abordem em seu texto problemas específicos relacionados as principais atividades da Engenharia de Requisitos, bem como a área como um todo.

Definição dos Critérios de Inclusão e Exclusão. Os critérios de inclusão e exclusão filtram as publicações que serão relevantes durante o processo da revisão. Esta pesquisa usou os seguintes critérios:

Critérios de Inclusão:

- Estudos publicados entre 2015 e 2021.
- O texto deve estar disponível por completo.
- Os estudos são publicações revisadas por pares.
- Os estudos são relevantes para os termos de pesquisa que foram definidos.
- Os estudos descrevem os desafios relacionados a Engenharia de Requisitos experienciados durante o desenvolvimento de um projeto de *software*.

Critérios de Exclusão:

- Estudos que não são focados ou não discutem tópicos da Engenharia de Requisitos.
- Estudos que não descrevem nem desafios, nem soluções experienciadas na área de Engenharia de Requisitos.
- Publicações do tipo *Keynotes*, *White Papers*, ou trabalhos que contém somente um *abstract*.
- Publicações que não estejam na língua portuguesa ou na língua inglesa.
- Estudos que não satisfazem quaisquer outros critérios de inclusão que foram definidos.
- Estudos duplicados.

Seleção dos estudos. A primeira seleção se deu pela leitura do título e do *abstract* das publicações, a fim de entender se essas estavam alinhadas com o tema proposto. A seleção inicial gerou 152 resultados.

A partir desses estudos, foi realizada uma análise mais detalhada em duas etapas, em que se realizou a leitura dos artigos e se aplicou os critérios de inclusão e exclusão definidos. 36 das 152 publicações selecionadas inicialmente foram consideradas adequadas para serem utilizadas na revisão.

As quantidades de publicações retornadas a partir da execução da string de busca nas três bases definidas e após a filtragem em cada etapa uma das etapas de seleção podem ser visualizadas na Tabela 1.

Tabela 2 – Estudos selecionados.

Base	1ª Seleção	2ª Seleção	Seleção Final
IEEE Xplore	32	14	10
ACM Digital Library	49	24	11
ScienceDirect	71	35	15
Total	152	73	36

5.2 RESULTADOS

Essa seção tem como foco expor os resultados iniciais obtidos a partir da RSL performada. Para responder as questões de pesquisa que foram elaboradas, identificaram-se nos estudos selecionados os principais desafios enfrentados nas atividades da Engenharia de Requisitos, e para cada problema foi feita sua classificação em categorias relacionadas. Ao final, calculou-se a frequência de cada um dos desafios com base nas ocorrências em estudos.

O mapeamento dos desafios mais frequentes está disponível na Tabela 2. Ela auxilia a responder a primeira questão de pesquisa $\mathbf{Q_1}$. É possível visualizar que a falta de comunicação entre as partes interessadas é o problema mais citado em Engenharia de Requisitos entre as publicações selecionadas, ocorrendo em cerca de 56% dos casos.

Também com percentuais consideravelmente altos estão requisitos ambíguos, incompletos, inconsistentes ou incorretos com respectivamente 33%, 31%, 22% e 19% de ocorrência. Outro problema frequente, a manutenção da documentação aparece em seguida, sendo citada em 28% dos estudos como desafio.

Além dos já citados, outras adversidades tendem a gerar complicações ao projeto, como pouco conhecimento sobre o domínio da aplicação (22%), priorização dos requisitos (22%) e mudança de requisitos (19%). Com menor frequência estão dificuldades técnicas gerais, que envolvem geralmente falta de técnicas e ferramentas para gerenciar requisitos,

com 14% de ocorrência.

Tabela 3 – Os principais desafios da Engenharia de Requisitos segundo a análise das publicações selecionadas.

	Desafio	**Frequência (Total:36)**	**Percentual (%)**
C_1	Falta de comunicação e/ou envolvimento das partes interessadas	20	56
C_2	Requisitos ambíguos	12	33
C_3	Requisitos incompletos	11	31
C_4	Documentação	10	28
C_5	Rastreabilidade dos requisitos	9	25
C_6	Requisitos inconsistentes	8	22
C_7	Pouco conhecimento sobre o domínio	8	22
C_8	Priorização dos requisitos	8	22
C_9	Requisitos incorretos	7	19
C_{10}	Requisitos voláteis	7	19
C_{11}	Dificuldades técnicas	5	14

Para cada um dos problemas que foram identificados, buscou-se na literatura algumas das técnicas mais atuais e efetivas que existem para tentar solucioná-los, e assim responder a questão de pesquisa $\mathbf{Q_2}$.

C_1: Falta de comunicação e/ou envolvimento das partes interessadas.

Apesar da comunicação e o envolvimento dos *stakeholders* dependerem da disponibilidade dos mesmos, existem algumas metodologias clássicas para melhorar a comunicação da equipe como os métodos MUST, *Joint Application Design* (JAD), *User-Led Requirements Construction* (ULRC) e *Soft Systems Methodology* (SSM), que são focadas em definições de práticas ideais para a elicitação de requisitos (COUGHLAN; MACREDIE, 2002).

Atualmente muitas publicações propõem formas inovadoras de fazer com que essa comunicação aconteça de forma eficaz em projetos de diferentes características. Em projetos de desenvolvimento de *software* global, por exemplo, a falta de comunicação entre as partes envolvidas é um problema comum, pois a equipe e o cliente estão distribuídos em vários lugares ao redor do planeta. Dessa forma, Nadeem e Lee (2019) propuseram um *framework* baseado em Raciocínio Baseado em Casos (RBC), para armazenamento de conhecimento, e técnicas de elicitação para aprimorar o diálogo entre as partes interessadas e assim definir as melhores práticas para a coleta de requisitos no contexto de desenvolvimento de *software* global (NADEEM; LEE, 2019).

Em um estudo similar, Shahzad et. al (2021) debatem as mudanças necessárias para as tarefas de elicitação de requisitos no contexto da pandemia global de COVID-19 e propõem o uso de tecnologias baseadas em *blockchain* para aprimorar diversos fatores relacionados a Engenharia de Requisitos, incluindo a comunicação entre *stakeholders*,

conflitos em tomadas de decisões, negociações, entre outros (SHAHZAD et al., 2021).

C_2: Requisitos ambíguos.

A detecção de requisitos ambíguos é atualmente uma área muito explorada por pesquisadores, que utilizam principalmente algoritmos inteligentes para realizar a correção de documentos já especificados. Entre os campos específicos da IA, a Aprendizagem de Máquina é frequente em trabalhos relacionados a redução de ambiguidades de requisitos (SHREDA; HANANI, 2024).

Osman e Zaharin (2018) propuseram uma abordagem híbrida entre Mineração de Texto e Aprendizado de Máquina para a detecção e classificação de ambiguidades em um conjunto de dados extraídos de documentos de especificação malaios. A partir da extração das informações na etapa de mineração, o sistema aprende progressivamente a detectar requisitos ambíguos (OSMAN; ZAHARIN, 2018).

Também focados no Aprendizado de Máquina, Sharma et. al (2016) propuseram uma abordagem para detecção de ambiguidades de anáforas pronomiais nocivas em documentos de especificação de requisitos. Os pesquisadores usaram algoritmos de classificação para classificar as ambiguidades nocivas e não-nocivas. O classificador escolhido conseguiu detectar corretamente 95% dos requisitos ambíguos nocivos (SHARMA; SHARMA; BISWAS, 2016).

C_3: Requisitos incompletos.

Assim como requisitos ambíguos, requisitos incompletos é outro desafio comum na área de Engenharia de Requisitos e pode ser ocasionado pela definição precária dos requisitos na etapa de especificação. Muitas vezes se torna difícil identificar a incompletude dos requisitos antes do desenvolvimento da funcionalidade, principalmente quando os documentos não são revisados periodicamente. Dessa forma, é interessante o uso de ferramentas que auxiliem o profissional nessa tarefa.

DeVries e Cheng (2017) propuseram a abordagem Ares-EC, voltada para a detecção de requisitos incompletos usando análise simbólica e computação evolutiva para realizar a análise de modelos de requisitos hierárquicos. A abordagem foi aplicada em um sistema real, e os pesquisadores conseguiram notar que o Ares-EC foi capaz de detectar automaticamente os requisitos incompletos e gerar contra-exemplos de completude (DEVRIES; CHENG, 2017).

Kalinowski et. al (2016) conduziram um questionário em 88 organizações brasileiras e austríacas de tamanho pequeno, médio e grande para definir as causas mais comuns para requisitos incompletos ou escondidos em um projeto. Focando nessas causas, os pesquisadores discutiram práticas e ações de mitigação com base nas respostas obtidas dos respondentes (KALINOWSKI et al., 2016).

C_4: Documentação.

Ter uma documentação que não está acessível a todas as partes envolvidas, que não é gerenciada corretamente ou que simplesmente não exista são fatores de grande risco para um projeto. Para se ter a qualidade no desenvolvimento do produto de *software*, é preciso ter boas práticas definidas para documentar e gerenciar os documentos.

Salvador e Santos (2016) apresentaram em seu estudo a aplicação DoMaR, que é focada na prevenção de problemas relacionados ao gerenciamento de requisitos e a documentação a partir do mapeamento desses problemas por questionários realizados com especialistas. A partir do feedback dos especialistas, a aplicação mostrou ter potencial para solucionar os desafios citados nos questionários respondidos (SALVADOR; SANTOS, 2016).

Behutiye et. al (2017) entrevistaram profissionais de quatro empresas diferentes e apresentaram suas descobertas em relação as práticas de documentação adotadas pelas organizações. A partir de sua análise, os pesquisadores propuseram diretrizes para a documentação de requisitos no desenvolvimento de *software* ágil, em que geralmente não se dá prioridade a documentos (BEHUTIYE et al., 2017).

C_5: Rastreabilidade dos requisitos.

Rastreabilidade é a capacidade de acompanhar um requisito desde sua origem e especificação até seu desenvolvimento e implementação. Manter a rastreabilidade ao longo do ciclo de desenvolvimento do *software* é uma tarefa complexa, mas que auxilia a manutenção e controle dos requisitos do projeto.

De forma a propor uma solução para o problema da rastreabilidade, Haidrar et. al (2018) desenvolveram uma linguagem para especificação de requisitos nomeada ReqDL. A linguagem possui operadores específicos que revelam os links explícitos e implícitos entre requisitos e artefatos. A partir do uso das expressões ReqDL, a tarefa de rastreabilidade se torna menos complexa e mais compreensível (HAIDRAR et al., 2018).

Garcia e Paiva (2016) apresentaram uma ferramenta para auxiliar o usuário a manter a informação de rastreabilidade dos requisitos atualizada, sem erros. Os pesquisadores demonstraram, a partir da ferramenta desenvolvida, que criar uma ligação entre os requisitos e os artefatos de implementação auxiliou na manutenção mais do que matrizes de rastreabilidade, que são documentos comuns na área de Engenharia de Requisitos para demonstrar a relação entre artefatos (GARCIA; PAIVA, 2016).

C_6: Requisitos inconsistentes.

A consistência entre requisitos é mais um aspecto crucial para manter a qualidade do desenvolvimento do produto de *software*. A inconsistência passa despercebida na maioria das vezes, e pode ocasionar prejuízos ao projeto como retrabalho e esforço desperdiçado, gerando aumento no tempo e nos recursos necessários para um projeto ser

finalizado.

Para tratar de inconsistências de requisitos no contexto de desenvolvimento de *software* global, Gull et. al (2021) introduziram o *framework* BOMO, baseado na tecnologia *blockchain* e na metodologia de Engenharia de *Software* Dirigida por Modelo (MDSE) (KENT, 2002). Essa união torna possível gerenciar requisitos inconsistentes de forma efetiva e simples. Os resultados do estudo de caso aplicando o *framework* se mostraram positivos, auxiliando a indústria de *software* global a lidar com a inconsistência (GULL et al., 2021).

Mezghani et. al (2018) usaram o algoritmo de Aprendizado de Máquina não supervisionado, k-means, para a identificação de redundância e inconsistência em requisitos. O algoritmo k-means agrupa os dados em torno de centroides, de acordo com uma quantidade de grupos definida, ou também chamado somente de k. O algoritmo foi testado usando uma base industrial real contendo dados inconsistentes, e usando como valor de k um número definido por um profissional da área de Engenharia de Requisitos que auxiliou na pesquisa. Os experimentos produziram resultados iniciais positivos na detecção de requisitos inconsistentes e de requisitos redundantes (MEZGHANI; KANG; SEDES, 2018).

C_7: Pouco conhecimento sobre o domínio.

Um especialista de RE que não tem o conhecimento sobre o domínio da aplicação tende a falhar em executar as atividades de especificação e gerenciamento de requisitos corretamente. Entender o domínio de uma aplicação a ser desenvolvida é uma tarefa que demanda tempo e dedicação do profissional envolvido nas atividades de Engenharia de Requisitos e, por essa razão, aplicar meios de obter esse conhecimento de maneira simples e rápida no projeto pode trazer resultados benéficos.

Li et. al (2020) propuseram uma forma automatizada de extração do conhecimento de domínio a partir de documentos de requisitos para auxiliar os profissionais. A ferramenta representa os documentos em linguagem natural como um vetor usando o algoritmo Doc2Vec e a partir disso aplica algoritmos de agrupamento para criar a *cluster feature tree* inicial. Os resultados extraídos contendo palavras e frases mais importantes são retornados ao usuário para análise (LI et al., 2020).

C_8: Priorização dos requisitos.

A priorização dos requisitos é uma tarefa que tem um grande papel durante a fase de implementação. Priorizar funcionalidades é um processo complexo e leva em conta aspectos como a capacidade de desenvolvimento da equipe no momento e a urgência do cliente pelo requisito.

Atualmente existem várias maneiras de realizar a priorização dos requisitos, algumas conhecidas por produzirem melhores resultados. Entretanto, não existe um consenso sobre qual é a melhor técnica para priorizar requisitos, tudo depende das características

do projeto, da equipe e do produto. Publicações frequentemente surgem propondo novos meios de priorizar focando na usabilidade, escalabilidade e na garantia da qualidade.

Nessa perspectiva, Mkpojiogu e Hashim (2017) propuseram uma abordagem para priorização de requisitos baseada em qualidade, usando o modelo Kano. Os requisitos são priorizados de acordo com atributos que levam em consideração o ponto de vista dos *stakeholders* sobre qualidade. O resultado mostrou que o modelo produziu resultados consistentes e que pode servir como uma boa opção para a priorização de requisitos (MKPOJIOGU; HASHIM, 2017).

Yaseen et. al (2020) sugeriram uma abordagem baseada em Árvore de Extensão para a priorização dos requisitos sob a perspectiva do desenvolvedor. A abordagem foi testada usando um conjunto de requisitos do *software* ERP ODOO, que foram designados a quatro desenvolvedores de forma a existirem dependências. Os pesquisadores então realizaram as estimativas de tempo para os requisitos priorizados e não priorizados para avaliar o impacto no tempo de estimativa total do projeto. Percebeu-se uma diferença significativa entre os dois tempos, que demonstrou a importância da priorização dos requisitos (YASEEN; MUSTAPHA; IBRAHIM, 2020).

C_9: Requisitos incorretos.

Requisitos incorretos são, geralmente, definições erradas de funcionalidades requisitadas pelo cliente ou que estão em conflito com documentos especificados previamente acerca do produto. Pode acontecer por um erro do especialista no momento da elaboração do documento de especificação de requisitos ou pelo mal entendimento acerca de uma funcionalidade. Um requisito incorreto pode gerar um produto que não satisfaça as necessidades do cliente, e, dessa forma, exigir retrabalho.

De forma a melhorar a qualidade da etapa de análise dos requisitos, Nguyen et. al (2014) desenvolveram a ferramenta GUITAR, com o objetivo de detectar artefatos incorretos, incompletos e inconsistentes. A ferramenta GUITAR é baseada em ontologias do domínio de conhecimento e semântica para a análise semântica dos requisitos. A metodologia é composta de diferentes atividades que contém uma ação e um objeto, e as atividades do domínio são relacionadas. A ontologia auxilia na identificação de incompatibilidades entre artefatos e artefatos faltantes a partir dessas relações (NGUYEN; GRUNDY; ALMOSY, 2014).

C_{10}: Requisitos voláteis.

Mudanças são inevitáveis dentro de qualquer projeto. Dificilmente é possível definir tudo que se espera de um *software* no início do desenvolvimento, de modo que é necessário que o cliente realize frequentes solicitações de mudança à equipe (SOMMERVILLE; KOTONYA, 1998). Essas solicitações podem envolver não apenas a alteração de um requisito já existente no projeto, como também abarcar a incorporação de novos requisitos

que não estavam planejados anteriormente.

Assim, para que ocorram mínimos impactos negativos possíveis ao final do ciclo de vida, é necessário que um projeto de *software* tenha um Processo de Gerenciamento de Mudanças definido, adequado às suas práticas.

Ali et. al (2018) propuseram um *framework* para gerenciamento de mudanças de requisitos fundamentado na técnica de Raciocínio Baseado em Casos, no contexto de desenvolvimento de *software* global. Com a aplicação do *framework* baseado em RBC em nuvem, os autores notaram que a comunicação e coordenação da equipe global durante o gerenciamento de mudanças, que antes era desafiadora, se tornou mais eficaz. Todos os serviços demandados estavam sempre disponíveis em uma única plataforma sem restrições de tempo e espaço, diferentemente de quando o time utilizava ferramentas em diferentes sites, com diferentes *logins* e membros, e que não levavam em consideração as diferenças culturais (ALI; IQBAL; HAFEEZ, 2018).

Naz et. al (2013) definiram e descreveram um modelo que integra o gerenciamento de mudanças de requisitos com a técnica de Raciocínio Baseado em Casos. Para avaliação, os pesquisadores apresentaram seu modelo para especialistas da área e perguntaram sobre a experiência obtida assim que o modelo foi implementado, comparando assim as diferenças de performance antes e depois do seu uso. Os resultados mostram que o *framework* auxiliou na redução de impactos de custo e tempo de uma mudança, bem como na resolução de conflitos de requisitos e o no aumento de satisfação do cliente (NAZ et al., 2013).

Em relação ao desafio $\mathbf{C_{11}}$, que se refere a dificuldades técnicas, não existem soluções específicas que podem ser usadas. Tal desafio se refere a problemas gerais relacionados à organização, como a falta de ferramentas e processos claramente definidos para o gerenciamento dos requisitos.

5.3 TRÍADE DE PROBLEMAS DE REQUISITOS

Analisando os desafios mais frequentes da Engenharia de Requisitos, identificados através da condução da RSL, é evidente que em certos casos, como o desafio C_1, há uma exigência de mudança organizacional, adotando novas práticas e processos nas atividades de levantamento. Em outros casos, entretanto, os desafios podem estar relacionados a questões técnicas ou até mesmo de incerteza ou má definição dos requisitos do projeto.

Certos desafios podem ser abordados de maneira conjunta devido às suas semelhanças e à sua inclusão em uma categoria comum. Requisitos ambíguos, incompletos e inconsistentes representam alguns dos problemas mais comuns da área, ocorrendo frequentemente nos casos estudados. Tais problemas estão intrinsecamente ligados a espe-

cificação de requisitos, surgindo a partir da incerteza ou da falta de percepção por parte do analista durante a documentação. Eles são tratados durante a atividade de validação de requisitos, garantindo que os requisitos documentados atendam aos critérios de qualidade estabelecidos e sejam compreendidos de maneira clara e consistente por todas as partes envolvidas no projeto.

Esses três casos são o foco dessa pesquisa, buscando compreender as causas subjacentes desses desafios e identificar técnicas eficazes para corrigi-los de forma integrada durante o processo de validação. Para alcançar esse objetivo, foi realizado um estudo detalhado de cada desafio.

5.3.1 Ambiguidade

De acordo com a RSL performada, a ambiguidade em requisitos do projeto é um dos problemas mais comuns na Engenharia de Requisitos. Os principais sintomas dessa ambiguidade são a possibilidade de interpretar uma história de usuário de várias maneiras e a possibilidade de ter diferentes entendimentos sobre a mesma.

Dependendo da forma que o requisito foi especificado, uma funcionalidade pode não ter o resultado esperado pelo cliente devido a interpretação errônea da equipe. Isso pode causar esforço adicional para a equipe e consequentemente ocasionar atrasos e extrapolamento do orçamento (WIEGERS; BEATTY, 2013).

Existem quatro tipos principais de ambiguidades linguísticas possíveis (KAMSTIES et al., 2003):

- **Ambiguidade Léxica:** Ocorre quando uma palavra tem vários significados. Esse tipo de ambiguidade pode ser classificado em ambiguidade homônima, ocorrendo quando uma palavra tem a mesma escrita e representação fonética mas significados diferentes, e em ambiguidade polissêmica, quando uma palavra tem vários significados, mas uma única etimologia.

  Ex: ”*Ficamos sem **rede** ontem a tarde.*”.

  Nesse caso, rede tem múltiplos sentidos.

- **Ambiguidade Sintática:** Ocorre quando uma sequência de palavras tem mais de uma estrutura gramatical, cada uma com um significado diferente. Pode ser analítica, de anexo, de coordenação e elíptica.

  Ex: ”*O sistema deve mostrar o rastreamento dos dados **nos próximos dois dias**.*”

  A frase ganha duplo sentido, pois ”nos próximos dois dias” pode estar relacionado tanto ao verbo ”mostrar” quanto a ”rastreamento dos dados”.

- **Ambiguidade Semântica:** Ocorre quando existe mais de uma forma de ler a frase em seu contexto, não contendo ambiguidades léxicas ou sintáticas.

  Ex: "*Aeronaves que não são amigas **e** têm missão desconhecida **ou** potencial para entrar em espaço aéreo restrito dentro de 5 minutos deve levantar um alerta.*"

  A frase é semanticamente ambígua pois não se sabe entre "e" ou "ou" quem tem a precedência.

- **Ambiguidade Pragmática:** Ocorre quando uma frase tem vários significados no contexto em que é pronunciada. Nesse caso, não são fornecidas informações suficientes para esclarecer o sentido real da frase naquele contexto. Pode ser do tipo referencial ou dêitica.

  Ex: "*O sistema deve aceitar **assinaturas digitais.***."

  Para interpretar a ambiguidade, é preciso do conhecimento do contexto de quem escreveu. No caso acima, "assinaturas digitais" pode ser interpretado de formas diferentes por diferentes pessoas, de acordo com o conhecimento sobre o termo.

5.3.2 Incompletude

Requisitos Incompletos são outro problema frequente em Engenharia de Requisitos. Esse tipo de problema ocorre quando existem informações em falta na especificação de um requisito. Geralmente é ocasionado pela incerteza dos *stakeholders*, principalmente ao início de um projeto quando não há algumas bases e conceitos definidos (WIEGERS; BEATTY, 2013).

Um requisito expresso na forma de história de usuário pode ser considerado incompleto caso falte uma das informações cruciais de sua estrutura.

Ex: "*Gostaria de me registrar no sistema, para poder acessá-lo.*" .

Uma história de usuário é composta por papel, descrição e finalidade. No exemplo acima, a história não especifica o papel do usuário o que resulta em informações incompletas.

5.3.3 Inconsistência

Requisitos Inconsistentes são aqueles que se contradizem, seja o requisito inteiro ou somente uma parte. Esse é um tipo muito comum de problema na Engenharia de

Requisitos, pois muitas vezes os requisitos evoluem e, dessa forma, tendem a sofrer mudanças imprevisíveis. Também ocorre frequentemente em casos que existem múltiplos *stakeholders* com diferentes requisitos e prioridades (SOMMERVILLE; KOTONYA, 1998).

Uma inconsistência pode ser classificada em diferentes categorias de acordo com sua origem (LAMSWEERDE, 2009):

- **Conflito de Terminologia:** O mesmo conceito tem diferentes nomes em diferentes histórias.

  Ex 1: "*O sistema deve permitir acesso aos **participantes do programa**.*" .

  Ex 2: "*O sistema deve permitir que os **integrantes do programa** se cadastrem.*"

  "Participantes do programa" e "integrantes do programa" representam o mesmo conceito, porém não existe consistência no nome entre as histórias.

- **Conflito de Designação:** O mesmo nome designa diferentes conceitos em diferentes histórias.

  Ex 1: "*O **usuário** [final] deve visualizar telas.*" .

  Ex 2: "*O **usuário** [intermediário] deve ter acesso irrestrito.*"

  Há um conflito de designação pois o mesmo nome "usuário" está sendo usado em diferentes histórias para representar dois conceitos diferentes.

- **Conflito de Estrutura:** O mesmo conceito tem diferentes estruturas em histórias diferentes.

  Ex 1: "*Fazer backups em **pontos no tempo**.*" .

  Ex 2: "*Fazer backups em **intervalos no tempo**.*"

  "Pontos no tempo" e "intervalos no tempo" se referem ao mesmo conceito, estruturados diferentemente.

- **Conflito Forte:** As histórias não se satisfazem juntas. Um exemplo é a inconsistência lógica.

  Ex 1: "*Desejo apagar os dados de um usuário.*" .

  Ex 2: "*Os dados de um usuário nunca devem ser apagados.*"

  Nessa caso, há contradição lógica entre ambas as histórias, impedindo que ambas aconteçam ao mesmo tempo.

- **Conflito Fraco ou Divergência:** As histórias não se satisfazem juntas sob alguma condição, que quando ocorre causa um conflito forte entre elas. É uma das inconsistências mais comuns em Engenharia de Requisitos.

  Ex 1: "*Desejo fazer backup dos dados a cada **vinte e quatro horas**.*" .

Ex 2: ”*Desejo fazer backup dos dados a cada **doze horas**.*”

Há uma contradição fraca pois existem duas histórias com condições diferentes para realizar uma ação.

5.4 CENÁRIO ATUAL

Analisando o cenário atual, como definido na questão de pesquisa $\mathbf{Q_3}$, percebe-se que os pesquisadores estão explorando formas inovadoras para solucionar problemas clássicos da Engenharia de Requisitos. A partir desse trabalho, foi possível notar que a comunidade científica está buscando soluções em outras áreas, como a IA, que se mostrou presente em grande parte das publicações estudadas.

Conforme observado por Harman (2012), há uma convergência marcante entre os princípios da IA e campos da Engenharia de *Software*, e a integração dessas áreas pode potencialmente oferecer vantagens significativas, pois ambas áreas tem grande influência do conhecimento humano. Na Engenharia de *Software*, aplicações de IA podem simplificar trabalhos complexos, e atuar como aliada de analistas na mitigação de incertezas e na aplicação eficiente do conhecimento.

A partir de uma análise realizada da quantidade de estudos contendo as palavras-chave “Inteligência Artificial” e “Engenharia de *Software*”, publicados entre 2010 e 2021 nas bases IEEE Xplore, ACM Digital Library e ScienceDirect, percebe-se também uma tendência de crescimento no cruzamento dessas duas áreas pelos pesquisadores. Como pode ser visualizado na Figura 8, em 2010 cerca de 900 publicações abordavam ambos os tópicos, enquanto que no ano de 2021 esse número ultrapassou a marca de 2500, valor quase três vezes maior.

Com base nessas informações é possível notar que a IA é um ramo da ciência que tem muito a complementar a Engenharia de *Software* como um todo. Por ser uma área muito vasta e em constante ascensão, ainda existem muitas oportunidades abertas para a aplicação de técnicas inteligentes também na Engenharia de Requisitos.

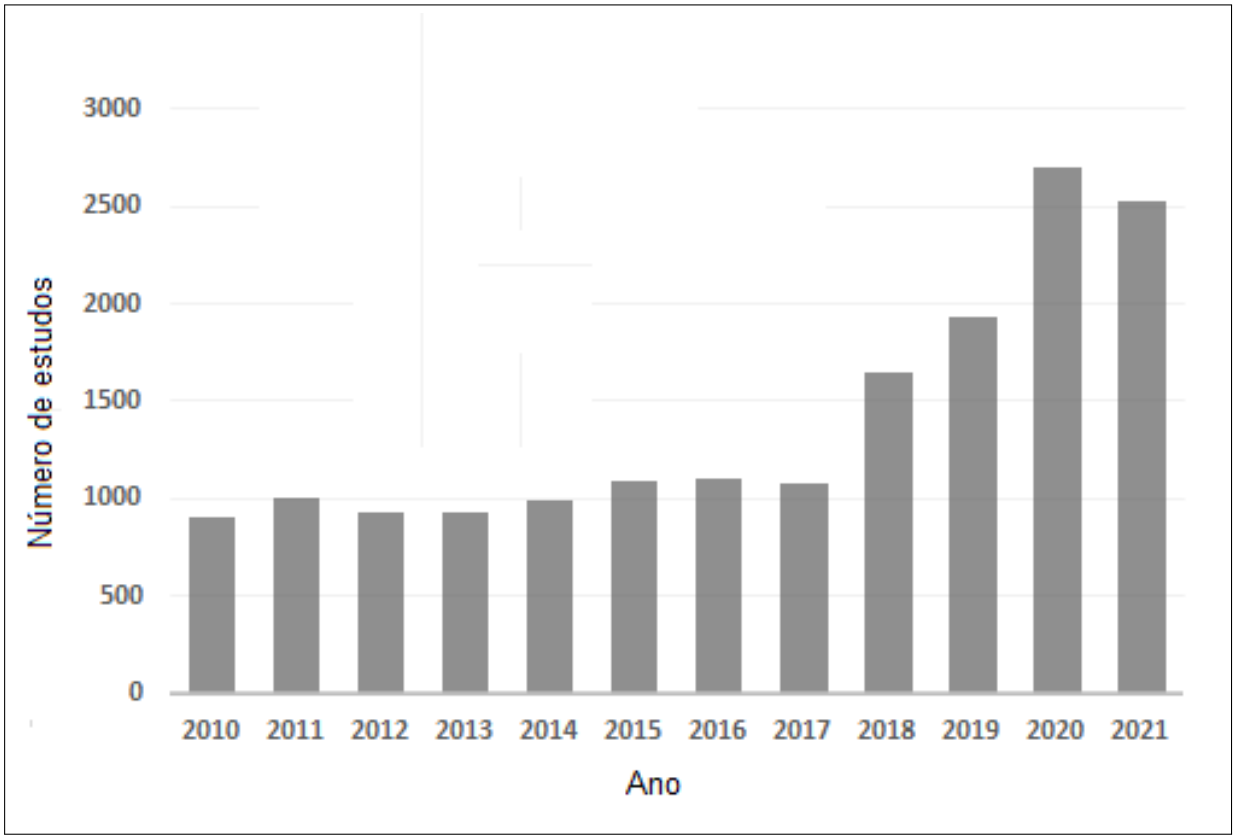

Figura 8 – Quantidade de publicações abordando Engenharia de *Software* e Inteligência Artificial usando as palavras chaves, por ano.

Fonte: Autoria própria.

6 REFINANDO REQUISITOS COM PROCESSAMENTO DE LINGUAGEM NATURAL

Este capítulo tem como foco expor e discutir a técnica desenvolvida. O objetivo da técnica é auxiliar analistas na detecção de requisitos ambíguos, incompletos e inconsistentes, visando reduzir riscos associados a potenciais retrabalhos associados a requisitos mal-especificados. Para isso é descrito o fluxo geral, bem como discutidos os subprocessos relacionados ao PLN. Essa técnica fundamenta-se, principalmente, nos princípios das metodologias ágeis e surge como uma aliada no processo de validação de histórias de usuário. A modelagem dos processos e atividades intrínsecos ao método, elaborada a partir da Linguagem de Modelagem Unificada (UML) (GOGOLLA, 2009), estão ilustrados na Figura 9.

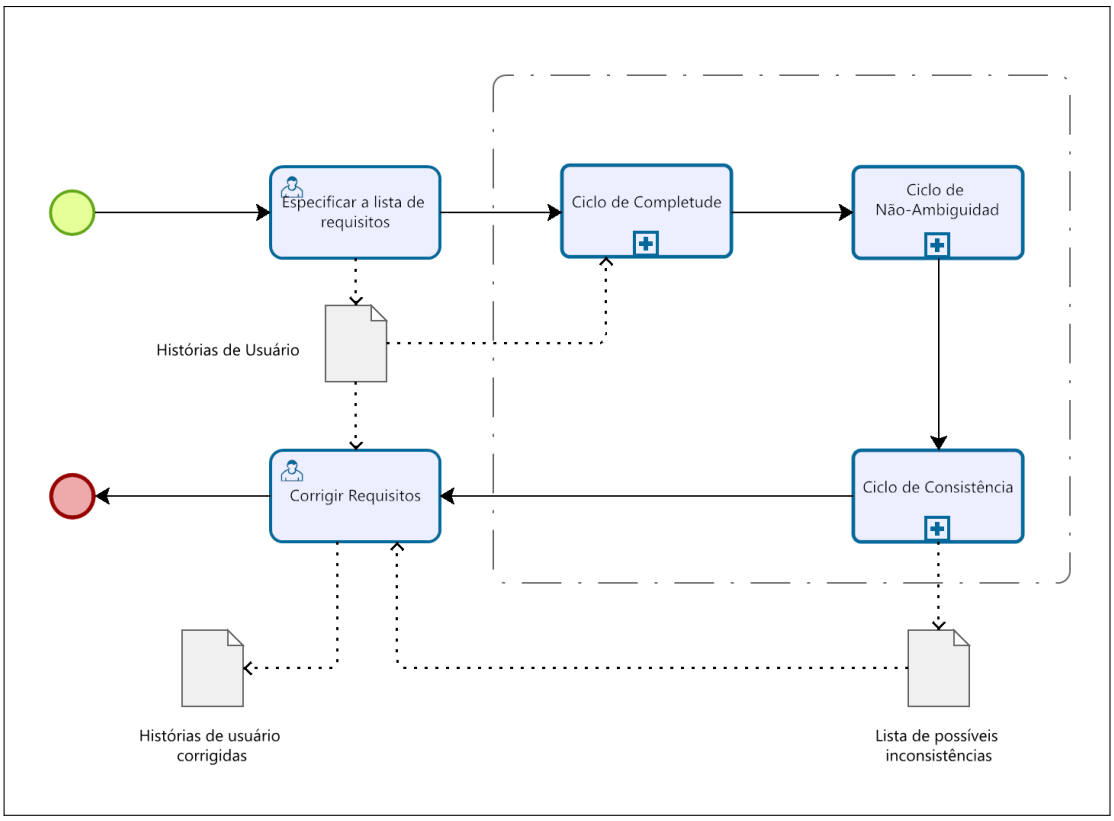

Figura 9 – Modelagem do fluxo da técnica proposta.

Fonte: Autoria própria.

O fluxo geral é composto de algumas atividades principais. Inicialmente, a atividade “Especifica lista de requisitos“, realizada pelo analista de requisitos, gera um documento de especificação contendo todos as histórias de usuário do sistema. A Seção 4.1 explica em detalhes o funcionamento dessa atividade.

Usando esse documento como entrada, a técnica desenvolvida é efetivamente executada nos subprocessos “Ciclo de completude“, “Ciclo de não-ambiguidade“ e “Ciclo de

consistência“, em que se aplicam as atividades de PLN. O sequenciamento dos ciclos foi estabelecido levando em consideração as saídas geradas e as entradas requeridas. O ciclo de completude é o ponto de partida no fluxo, já que sua saída proporciona as entradas necessárias para os ciclos subsequentes. Cada ciclo é respectivamente descrito nas Seções 4.2, 4.3 e 4.4.

Por fim, com base na lista de inconsistências gerada pelo sistema após os três ciclos de detecção de problemas. O analista de requisitos decide fazer a correção ou não da sua especificação a partir do seu conhecimento de domínio.

6.1 ESPECIFICAÇÃO DE REQUISITOS

Inicialmente um novo requisito é especificado por um analista após conversa com as partes interessadas. A história de usuário é um artefato informal em linguagem natural que contém informações sobre uma funcionalidade que deve ser implementada no sistema. Ela é escrita sob ponto de vista do usuário que irá interagir com o sistema (PMI, 2017).

Apesar de existirem inúmeras maneiras de se escrever histórias de usuário, normalmente temos a presença de três informações cruciais: o papel do usuário que irá interagir com a funcionalidade, uma breve descrição da funcionalidade que será implementada e razões para implementação e clarificações sobre tal funcionalidade (opcional). Esses aspectos do requisito são marcados pela presença dos indicadores “como”, “eu quero” e “para” nas histórias (ROBEER et al., 2016), tal como no exemplo ilustrado na Figura 10, que demonstra a análise gramatical de um requisito escrito como história de usuário.

Por mais que seja escrita em outras linguagens, a história de usuário seguirá sempre essa mesma estrutura. Essa técnica inicialmente trata apenas histórias de usuário escritas em inglês. Eventualmente, outras línguas serão contempladas.

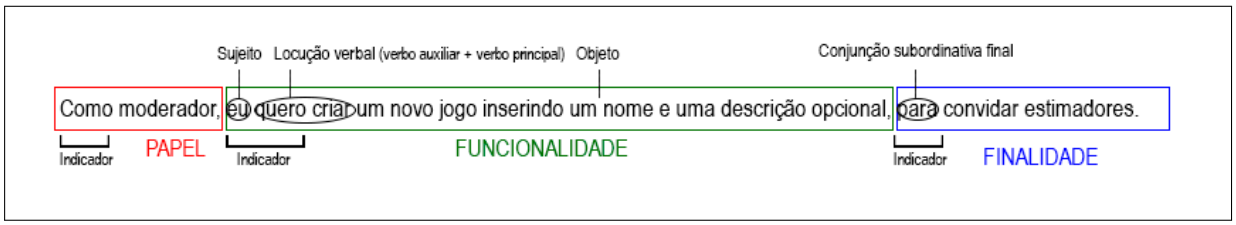

Figura 10 – Análise gramatical de uma história de usuário.

Fonte: Autoria própria.

6.2 CICLO DE COMPLETUDE

Por meio do PLN é possível que a máquina compreenda a entrada digitada pelo usuário e reporte a ele caso algum dos elementos da história não esteja presente, mesmo que os termos usados sejam diferentes. Além disso, também garante que o texto inserido pelo usuário faça sentido e tenha as informações necessárias para ser desenvolvido.

Os usuários têm a flexibilidade de escolher o formato desejado de entrada dos requisitos, desde que textual. Podem ser definidos também identificadores, que marcam uma história específica, e também estabelecer delimitadores para estruturar o conteúdo. O sistema faz a separação do arquivo de entrada em histórias a partir das configurações inseridas pelo usuário, e então inicia o fluxo da técnica proposta, apoiando o analista na garantia de completude, consistência e não-ambiguidade dos requisitos estabelecidos.

A Figura 11 detalha o primeiro subprocesso do fluxo geral, “Garantia de completude”, que descreve a verificação de um requisito em relação a característica de integridade de informações.

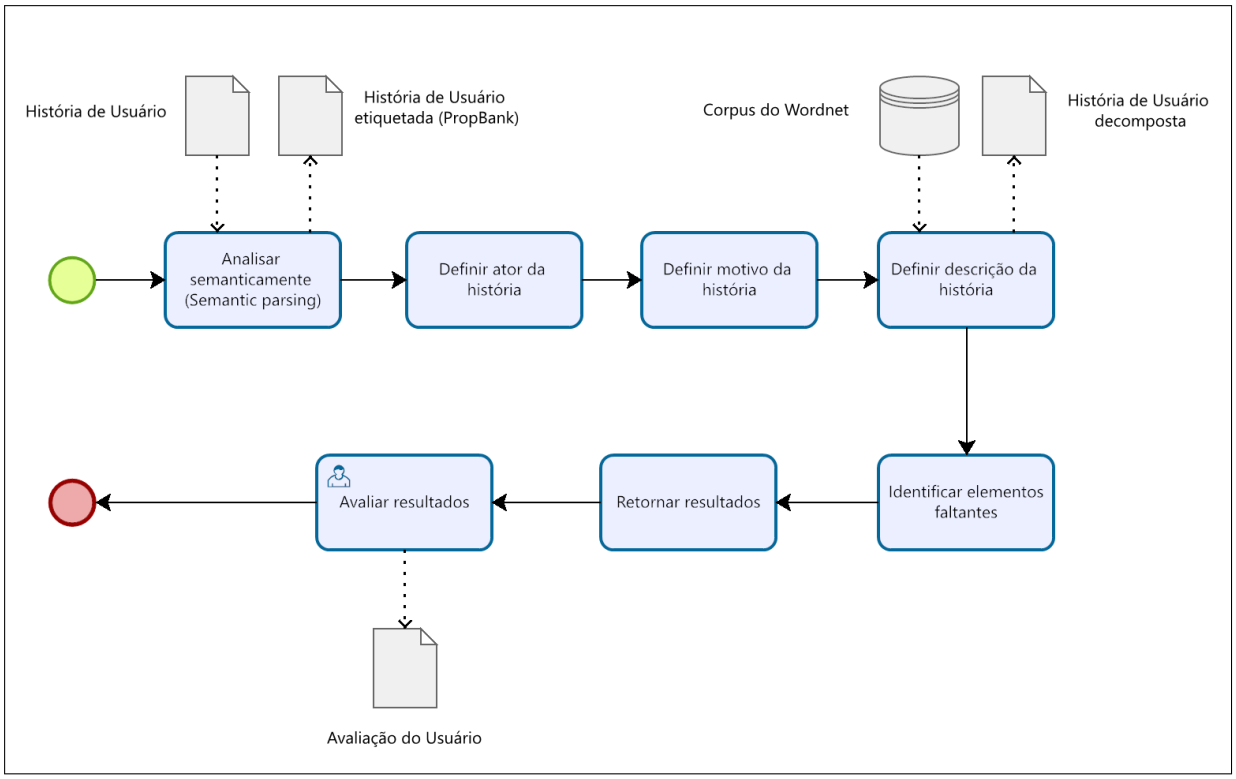

Figura 11 – Subprocesso de garantia de completude.

Fonte: Autoria própria.

A atividade de análise está presente em todos os subprocessos, apesar de existirem particularidades. A partir de um *parser*, ou analisador, a frase de entrada é decomposta em *tokens* que são etiquetados de acordo com os objetivos que se deseja atingir. No caso das atividades de verificação de completude, a história de usuário é analisada para rotu-

lagem semântica de função, usando etiquetas PropBank, um corpus que provê anotações semânticas às estruturas “predicado-argumento”. Os rótulos mais comuns descrevem argumentos de um verbo (numerados como Arg0, Arg1, Arg2, entre outros) ou modificadores de verbos (ArgM) (PALMER; GILDEA; KINGSBURY, 2005).

É possível fazer a definição do papel, da descrição da funcionalidade e do motivo, ou finalidade, por meio da filtragem de modificadores específicos, associados a cada uma dessas informações. Para a extração do papel do ator na história de usuário, primeira atividade do subprocesso, o sistema investiga a existência do demarcador “ArgM-PRD”, que denota o predicado secundário ou predicativo, tipo de expressão que atribuí características ao sujeito de uma oração.

Na sequência, a atividade ”Define motivo da história” tem como objetivo extrair a finalidade do requisito. Esse fragmento de frase é marcado pela presença do modificador “ArgM-PRP”, que indica orações de propósito ou finalidade. Tendo as informações de papel e motivo, a porção remanescente do texto é caracterizada como a descrição da funcionalidade.

Além disso, como medidas de teste e garantia para confirmar que o conteúdo extraído está efetivamente relacionado à especificação da funcionalidade, o sistema analisa a existência de palavras-chave que sinalizam demanda ou necessidade. Conforme destacado anteriormente, termos como ”quero”, ”gostaria” ou ”desejo” são frequentemente indicativos do propósito central do usuário na história. Usando o corpus do WordNet, base de dados léxica em inglês que agrupa substantivos, verbos, advérbios e adjetivos em sinônimos cognitivos (*synsets*) (FELLBAUM, 1998), verifica-se a ocorrência desses termos ou seus sinônimos no trecho em análise.

Se alguma das três partes essenciais para a história do usuário não for identificada, o sistema emite um aviso ao usuário, indicando quais informações estão ausentes na especificação. A especificação então é validada pelo analista de requisitos em conjunto com os alertas retornados pelo sistema, podendo ou não ser corrigida, considerando a validade dos alertas para os casos específicos. Essa técnica visa garantir que a especificação da história do usuário seja completa e precisa, fornecendo ao analista a oportunidade de realizar ajustes conforme necessário.

A história decomposta em trechos com ator, descrição e funcionalidade é armazenada internamente para ser utilizada nas etapas seguintes. Para melhor entendimento, a Tabela 3 mostra a história de usuário ”Como um administrador de sistema, eu quero configurar parâmetros, para facilitar a visualização de relatórios.” decomposta.

Tabela 4 – Decomposição de uma história de usuário.

Função	Trecho
Ator	**Como** um administrador de sistema
Descrição	**eu quero** configurar parâmetros
Finalidade	**para** facilitar a visualização de relatórios

Fonte: Autoria própria.

6.3 CICLO DE NÃO-AMBIGUIDADE

Após verificar a completude do requisito, é conduzida uma análise para identificar eventuais ambiguidades em seu texto. A Figura 12 fornece uma visão detalhada das atividades relacionadas ao fluxo do subprocesso que visa garantir a ausência de ambiguidades. Este procedimento tem como objetivo assegurar que o requisito seja claro e sucinto, eliminando duplicações ou sobreposições que possam resultar em interpretações ambíguas.

O subprocesso tem início com a análise sintática da história de usuário, diferentemente do fluxo de garantia de completude, em que é conduzida a análise semântica. Nesse contexto, o *parser* faz a rotulagem dos *tokens* de acordo com etiquetas de partes do discurso (POS). A rotulagem POS, também denominada ”rotulagem gramatical”, atribui a cada palavra uma etiqueta indicativa de sua função gramatical dentro da frase (PYYSALO, 2013).

Para assegurar a não-ambiguidade, o fluxo possui duas principais responsabilidades: verificar frases contendo múltiplos sentidos e analisar a ocorrência de palavras vagas. As atividades subsequentes têm o objetivo de analisar as interpretações da história, a partir da gramática livre de contexto (GLC).

A gramática livre de contexto é um modelo formal usado para descrever a estrutura sintática de uma linguagem. Ela consiste em regras que especificam como sequências de símbolos podem ser combinadas para formar frases gramaticais. Em uma GLC, as regras são definidas de maneira hierárquica, permitindo a representação de estruturas aninhadas (JURAFSKY; MARTIN, 2000). Um exemplo de uma árvore GLC, com as regras expandidas de uma frase, está ilustrado na Figura 13.

Para criar a GLC para as histórias de usuário de entrada, o sistema faz uso das etiquetas POS, convertendo-as para o formato utilizado pela GLC e incorporando algumas regras pré-definidas. A combinação desses componentes resulta na formulação de um conjunto de regras apropriadas para a construção da árvore de análise da GLC (*CFG parsing*). No caso de uma frase apresentar múltiplos significados, várias árvores serão geradas. Portanto, ao avaliar a quantidade de árvores construídas, torna-se possível determinar se a história de usuário pode ser considerada ambígua.

Com a conclusão das atividades de identificação de interpretações múltiplas, o fluxo segue com a investigação das histórias em busca de termos ambíguos. As principais

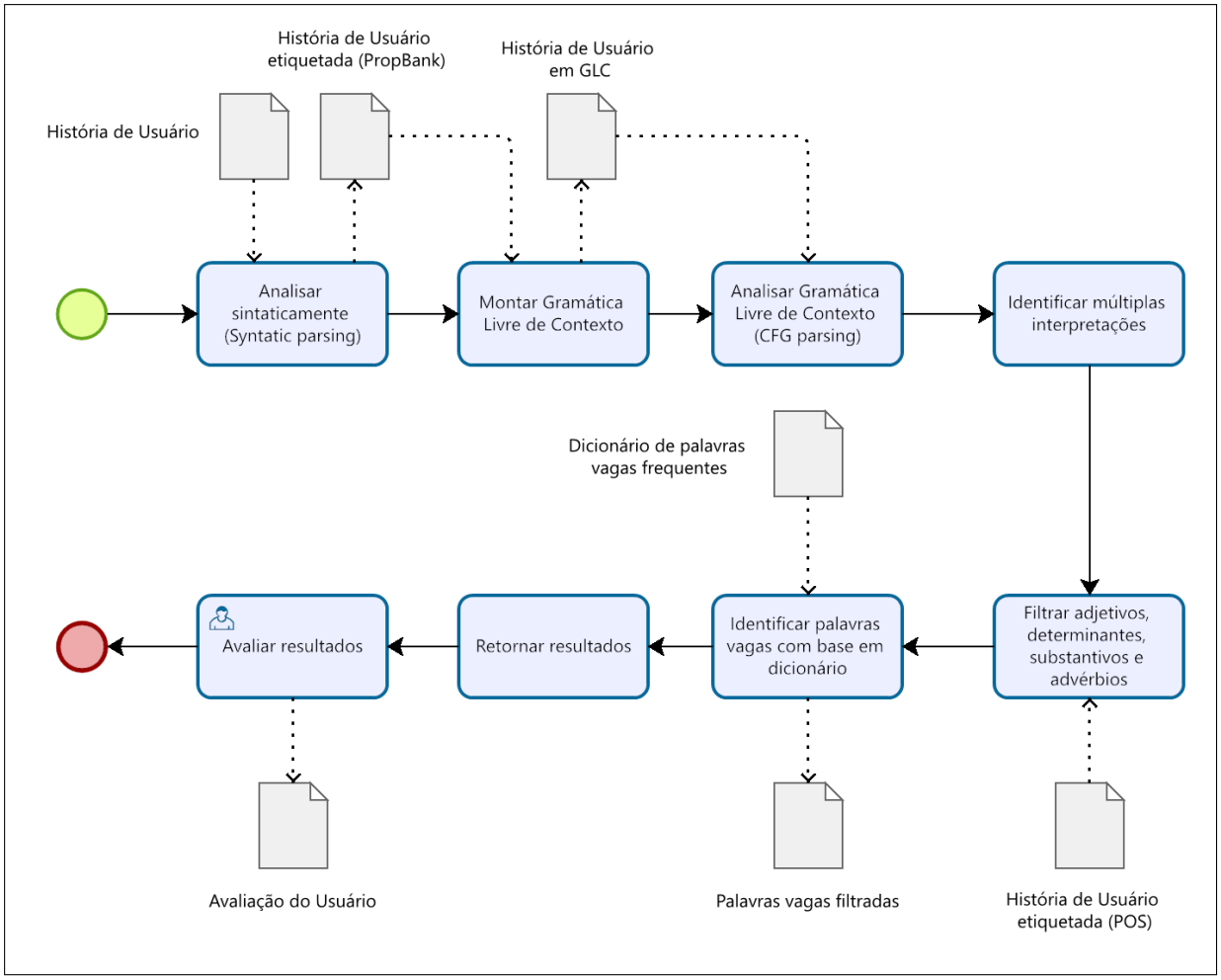

Figura 12 – Subprocesso de garantia de não-ambiguidade.

Fonte: Autoria própria.

palavras da história são filtradas de acordo com sua classe gramatical, dando destaque a substantivos, advérbios, determinantes e adjetivos, por serem componentes essenciais do discurso ou tenderem a ser vagos.

Cada palavra filtrada é buscada dentro de um dicionário de termos vagos, elaborado a partir das palavras vagas mais frequentes encontradas na literatura (AMBRAKAIT, 2016) (BOAKYE, 2007) (GLEICH; CREIGHTON; KOF, 2010). As palavras do dicionário contêm uma classificação, baseada nos estudos de Bhatia et. al (2016) e Meibauer (2018), que determina o tipo incerteza associado a elas (BHATIA et al., 2016) (MEIBAUER, 2018). A Tabela 4 demonstra os tipos de classificação e alguns termos associados.

Se ocorrer uma correspondência, o usuário é notificado sobre a detecção de uma palavra vaga e recebe informações sobre como tal palavra impacta a frase, dependendo de sua classificação. O analista pode, então, avaliar se o resultado está adequado para o contexto em questão e proceder com as alterações na especificação, se julgar que forem necessárias.

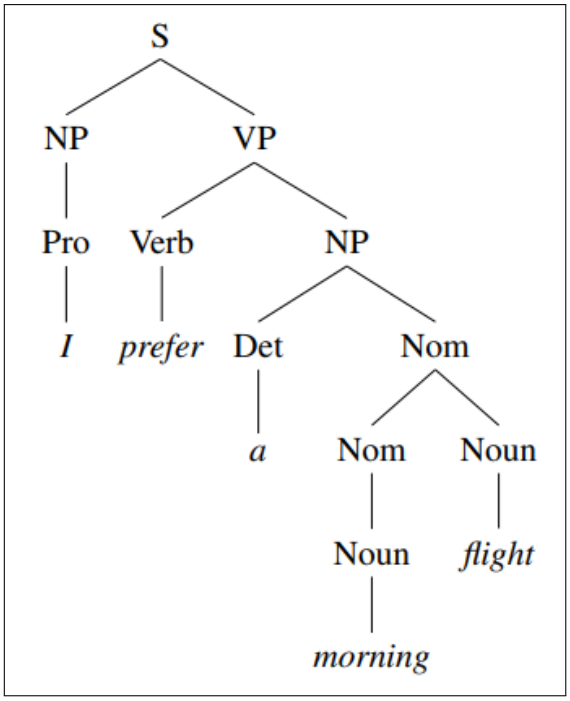

Figura 13 – Árvore de gramática livre de contexto da frase *"I prefer a morning flight"*.

Fonte: (JURAFSKY; MARTIN, 2000).

Tabela 5 – Classificação de termos vagos.

Tipo de Incerteza	Termos
Condicionalidade	depending, if practical, if applicable, once in a while...
Generalidade	usually, frequently, among others, etc, more or less...
Modalidade	maybe, can, could, possibly, probably...
Quantidade	some, many, bigger, lower, big, small...
Combinatória	appropriately, quickly, easily, enough...

Fonte: (BHATIA et al., 2016),(MEIBAUER, 2018)

6.4 CICLO DE CONSISTÊNCIA

Para assegurar a consistência do novo requisito, o sistema recupera as histórias armazenadas na base a fim de compará-las em busca de inconsistências. As histórias decompostas, originadas no ciclo de garantia de completude, são utilizadas como entrada para filtrar apenas os requisitos que envolvem o mesmo ator. Isso elimina a necessidade de processamento adicional em casos onde não há correspondência, resultando em melhorias de desempenho.

Cada história de usuário filtrada tem seus trechos de descrição e motivo comparados separadamente com os mesmos trechos das próximas histórias filtradas, até o fim da lista. A Figura 14 esquematiza as atividades do fluxo de garantia de consistência dos requisitos, apresentando uma visão geral de como esse processo ocorre. Esse ciclo é repetido de forma iterativa até que todas as histórias de usuário da lista tenham sido devidamente

analisadas.

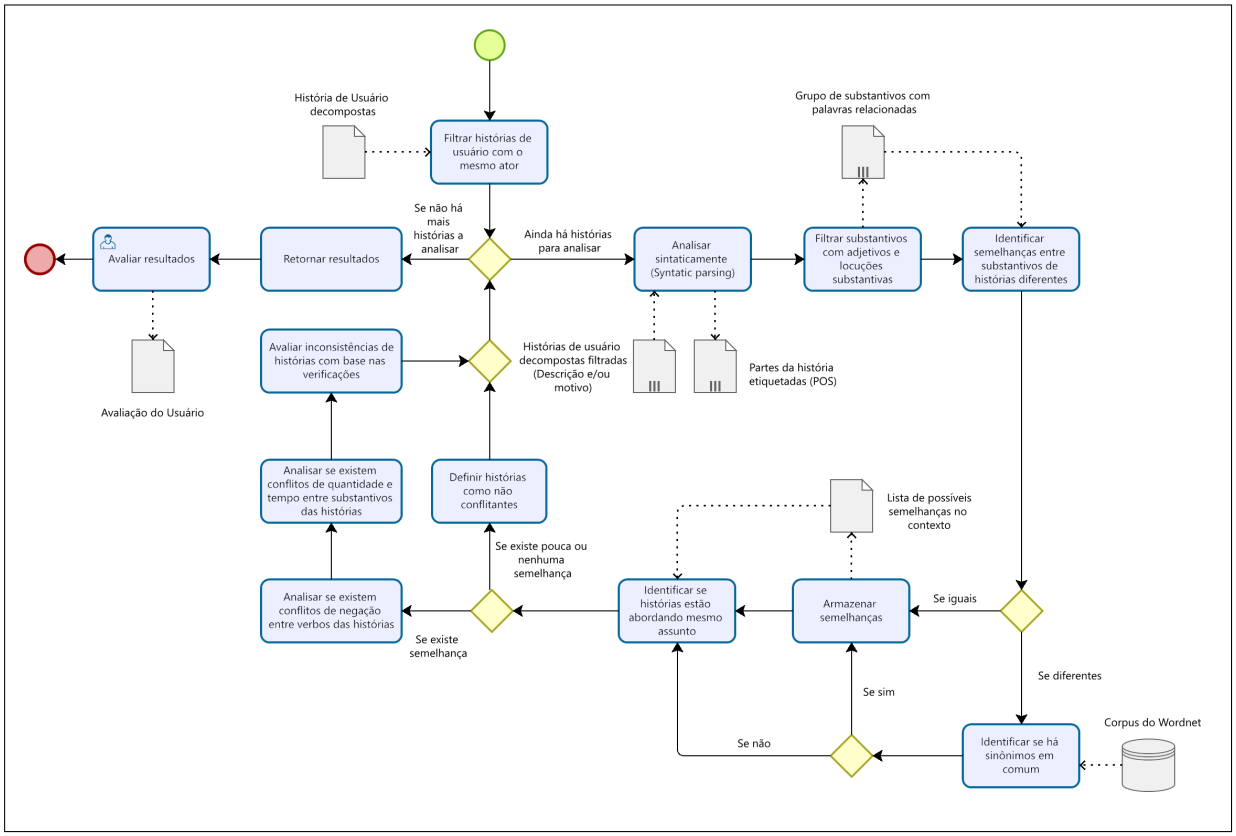

Figura 14 – Subprocesso de garantia de consistência.

Fonte: Autoria própria.

Novamente, utiliza-se técnicas de Processamento de Linguagem Natural para realizar a análise sintática das histórias de usuário decompostas e rotulá-las com etiquetas POS. Para cada trecho do par de histórias envolvidas na comparação, são filtradas as palavras com rótulo gramatical de substantivo, e todos os termos substantivos e adjetivos que estão diretamente ligados a elas. Por meio disso é possível obter todos os grupos de termos centrais das frases analisadas.

Os termos do par de histórias tem sua similaridade calculada pelo sistema. Quando existe um alto percentual de similaridade, a semelhança é armazenada, para posteriormente ser calculada a similaridade total entre as frases inteiras. Se, porventura, o cálculo resultar em um valor baixo ou mesmo nulo, se realiza o procedimento de comparação dos sinônimos dos termos, buscando uma análise de similaridade mais precisa. Se existir alguma correspondência entre os seus sinônimos, o sistema faz também o armazenamento dos termos.

Quando todos os termos centrais das histórias tiverem sido analisados quanto a semelhança, é realizado um cálculo geral entre ambas, levando em conta o total de termos, e a quantidade de vezes que se identificou que os termos possuíam semelhança nas atividades anteriores. Se o valor calculado for superior a 75%, determina-se que as fra-

ses estão abordando o mesmo assunto, caso contrário, as frases são definidas como não conflitantes, pois abordam assuntos diferentes.

Para determinar se existe conflito entre duas histórias que são semelhantes, o sistema analisa três características que são grandes indicadoras de contradição: negação, tempo e quantidade.

Primeiramente, são analisados os verbos contidos nas histórias de usuário, examinando sua associação com os substantivos já identificados. Se houver semelhança entre os verbos e existir a presença da etiqueta POS “NEG“ próxima ao verbo analisado, determina-se uma frase de negação. Se uma das histórias analisadas for marcada como negativa e a outra não, surge um conflito de negação, notificando o usuário por meio de um alerta.

Na segunda verificação, são considerados valores numéricos que acompanham e complementam substantivos, exemplificados por expressões como “24 horas“ ou “Duas telas“. Nesse processo são observados rótulos marcados como “NUM“, de classe gramatical Numeral. Caso exista discrepância entre esses valores, as histórias são identificadas como inconsistentes.

As atividades se repetem até que todas as histórias do documento de entrada tenham sido analisadas. Ao término do ciclo, os resultados gerais são fornecidos, exibindo as inconsistências encontradas. Assim como nos outros subprocessos, cabe ao usuário verificar se é apropriado ou não efetuar as alterações com base nas informações apresentadas pelo sistema.

É importante ressaltar que, se não for do interesse do usuário que as histórias de usuário sejam analisadas em todos os ciclos, é possível concentrar-se apenas nos problemas específicos passando para a próxima etapa diretamente, assim evitando processamento extra.

6.5 FERRAMENTA DE APOIO

Para o desenvolvimento do presente trabalho, as bibliotecas spaCy e NLTK foram amplamente utilizadas para a análise sintática das frases, rotulando-as com etiquetas POS (no caso dos ciclos de não-ambiguidade e consistência). A Figura 15 ilustra um exemplo prático de uma história de usuário etiquetada com rótulos POS, gerados pelo algoritmo de Processamento de Linguagem Natural do spaCy.

Cada palavra na história é rotulada conforme a sua classe gramatical. A depender do objetivo desejado, é possível filtrar apenas as palavras essenciais para a análise. No ciclo de garantia de consistência, por exemplo, se filtram as palavras que têm maior importância no texto, como substantivos, com o propósito de verificar se duas frases estão inseridas no mesmo contexto. Por outro lado, no ciclo de garantia de não-ambiguidade são

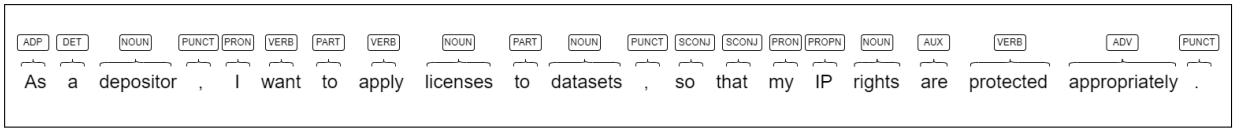

Figura 15 – História de usuário decomposta com rótulos POS.

Fonte: Autoria própria.

selecionadas as palavras que potencialmente introduzem maior incerteza ao texto, para posteriormente filtrá-las novamente por meio do dicionário de palavras vagas. Exemplos de aplicação dos rótulos POS podem ser observado no algoritmo apresentado na Figura 16, que abrange uma etapa do ciclo de garantia de não-ambiguidade, especificamente na detecção de termos frequentemente ambíguos, e nos trechos de algoritmo apresentados na Figura 17, contendo algumas verificações realizadas no ciclo de consistência em relação a negação e contradição numérica.

```
def detect_vagueness(self,sentence,nlp,vague_words):

    pos_tags = nltk.pos_tag(nltk.word_tokenize(sentence.lower()), tagset='universal')
    possible_vagues = []

    for word, pos in pos_tags:
        if pos == 'ADJ' or 'DET' or 'ADV' or 'NOUN':
            possible_vagues.append(word)

    vague_terms = []

    for term in vague_words:
        if term[0] in possible_vagues:
            vague_terms.append(term)

    if vague_terms:
            for vague_term in vague_terms:
                print("The user story contains the word \"" + vague_term[0] + "\" which can indicate the presence of " + vague_term[1] + ".")
    else:
            print("The user story does not contain vague words.")
```

Figura 16 – Algoritmo parcial de detecção de palavras vagas.

Fonte: Autoria própria.

Nesse algoritmo, são filtrados os termos com rótulo “ADJ“, “DET“, “ADV“ e “NOUN“, considerados os mais críticos nessa verificação. A seguir, verifica-se a presença dessas palavras no dicionário de termos vagos, elaborado a partir de estudos encontrados na literatura sobre o assunto. Se uma correspondência é encontrada, há a possibilidade de a frase apresentar problemas relacionados à ambiguidade.

O spaCy e o NLTK foram também essenciais para garantir o acesso ao WordNet, uma vez que as duas proporcionam integração simples e direta à base de dados. Esse suporte tornou a tarefa de identificar a presença de relações entre palavras mais intuitiva, simplificando o desenvolvimento.

Para a garantia de completude, foi utilizada a biblioteca *transformer-srl* baseada na implementação do *Allen Institute for Artificial Intelligence's Natural Language Processing* (AllenNLP), *framework* de código aberto para PLN, do estudo de Shi e Lin (2019) para rotulagem de função semântica (GARDNER et al., 2022) (SHI; LIN, 2019). Em conjunto

```
14              ...
15              for verb in verb_list:
16                  for child in verb.children:
17                      if child.dep_ == 'neg':
18                          negation = True

...

42                          if(token2.i == compat[1].i):
43                              if(token2.i < len(doc2)-1):
44                                  if doc2[token2.i+1].pos_ == 'NUM':
45                                      tokenNum2 = token2.i+1
46                              if(token2.i-1 >= 0):
47                                  if doc2[token2.i-1].pos_ == 'NUM' and tokenNum2 < 0:
48                                      tokenNum2 = token2.i-1
49                              if(tokenNum2 >= 0):
50                                  if(doc[tokenNum1].text != doc2[tokenNum2].text):
51                                      contradiction = True
52                                      break

...
```

Figura 17 – Algoritmo parcial de detecção de inconsistências.

Fonte: Autoria própria.

com a biblioteca, foi utilizado um modelo BERT pré-treinado também disponibilizado pelos criadores. Modelos BERT ou "*Bidirectional Encoder Representations from Transformers*" constituem uma categoria de modelos de linguagem pré-treinados desenvolvidos pelo Google. Eles são baseados na arquitetura *Transformer* e são projetados para entender o contexto das palavras em uma sentença, levando em consideração tanto as palavras à esquerda quanto as da direita (DEVLIN et al., 2018).

Por meio da biblioteca, que proporciona rotulagem semântica usando notações PropBank ou VerbAtlas, e do modelo pré-treinado, foi possível extrair as informações desejadas da história de usuário a partir do processamento e análise das informações resultantes. A Figura 18 demonstra uma história de usuário etiquetada com notações PropBank, após execução do algoritmo.

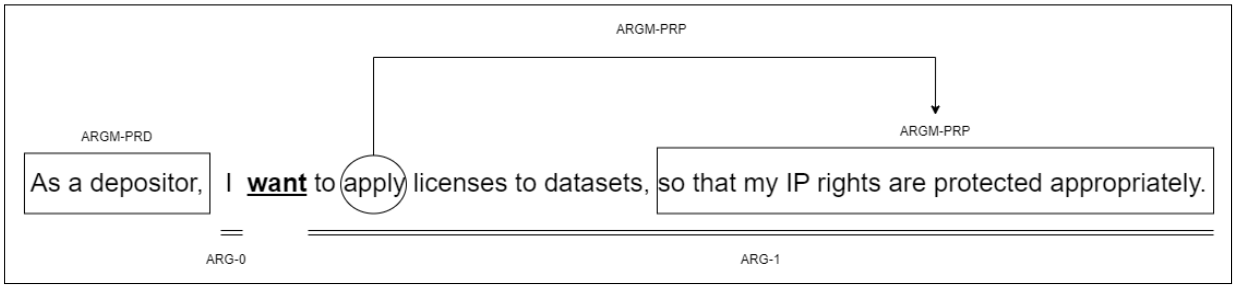

Figura 18 – História de usuário decomposta com rótulos PropBank.

Fonte: Autoria própria.

Analisando a frase a partir de “*want*“, que o algoritmo considera um verbo gatilho na história de usuário, percebemos a presença de algumas informações importantes. Temos os argumentos do verbo “*I*“, que representa quem faz ou requere a ação e nesse caso é o sujeito da oração, e “*to apply licenses to datasets, so that my IP rights are protected appropriately*“, o predicado, e também o predicativo “*As a depositor*“ que dá uma característica ao sujeito.

Mudando o foco e rotulação da frase a partir do verbo “*apply*“, conseguimos extrair a informação do motivo da ação ser realizada. A oração “*so that my IP rights are protected appropriately*“ se liga diretamente a esse verbo, justificando aplicação de licenças para proteção dos direitos de IP. A análise combinada de todos esses elementos garante a verificação de completude de um requisito, e na falta de um ou mais elementos, há possibilidade de erro na especificação. O algoritmo para checagem de papel e motivo de uma história pode ser visualizado na Figura 19.

```
def check_role(self,result,parts_sentence):
    role = ''
    previous = False
    for verb in enumerate(result["verbs"]):
        for arg_index, arg in enumerate(verb["tags"]):
            print(verb,arg)
            if(previous == True and (arg == 'I-ARGM-PRD') ):
                previous = True
            else:
                previous = False
            if(((arg == 'B-ARGM-PRD') and result["words"][arg_index].lower() in "as") or ((arg == 'I-ARGM-PRD') and previous)):
                role += result["words"][arg_index] + " "
                previous = True

    if(role != ''):
        parts_sentence.append({"role":role.capitalize().replace(",","")})
        return parts_sentence
    else:
        parts_sentence.append({"role":''})
        print("Incomplete user story. Role missing.\n")
        return parts_sentence


def check_purpose(self,result,parts_sentence):
    purpose = ''
    for verb in enumerate(result["verbs"]):
        for arg_index, arg in enumerate(verb["tags"]):
            if(arg == 'B-ARGM-PRP' and self.trigger_index == -1):
                self.trigger_index = arg_index
                purpose += result["words"][arg_index] + " "
            if(arg == 'I-ARGM-PRP'):
                purpose += result["words"][arg_index] + " "
    if(purpose != ''):
        parts_sentence.append({"purpose":purpose.capitalize().replace(",","")})
        return parts_sentence
    else:
        parts_sentence.append({"purpose":''})
        print("Incomplete user story. Purpose missing.\n")
        return parts_sentence
```

Figura 19 – Algoritmo parcial de busca por papel e motivo.

Fonte: Autoria própria.

Para indicar todo o trecho rotulado, o PropBank utiliza o prefixo “B“, para a palavra que dá inicio, e “I“, para as demais. No caso do exemplo acima, “*As*“ é rotulada como “B-ARGM-PRD“ enquanto as palavras restantes “*a*“ e “*depositor*“ são rotuladas como “I-

ARGM-PRD“. O mesmo se aplica a notação “PRP“, na checagem de motivo. O algoritmo então decompõe esses trechos em partes da história, para uso posterior.

O sistema desenvolvido atua em conjunto com o seu usuário como um assistente, alertando sobre possíveis problemas e dando soluções para que o analista consiga tornar sua especificação completa e precisa, evitando erros posteriores. Entretanto, nem sempre os alertas gerados vão ser válidos, podendo variar caso a caso. Dessa forma, cabe ao analista, e ao seu entendimento e experiência sobre o domínio, avaliar os problemas encontrados e utilizar a ferramenta de acordo com suas necessidades. O código completo pode ser acessado através do repositório na plataforma GitHub (MELLO, 2024).

7 RESULTADOS

Neste capítulo, são descritos os resultados da pesquisa, centrando no uso e análise da ferramenta de PLN desenvolvida. Exploram-se os métodos de avaliação adotados, os dados utilizados e as descobertas relevantes provenientes desse processo.

7.1 PREPARO DOS DADOS

Para avaliação da técnica proposta, foi selecionado um *dataset* publicado no Mendeley Data, repositório que permite que pesquisadores armazenem e disponibilizem seus dados de pesquisa publicamente. O conjunto de dados selecionado faz parte de um grupo de vinte e dois *datasets*, em arquivo textual, contendo requisitos expressados em formato de histórias de usuário. As histórias foram adquiridas *on-line* e compiladas pelo pesquisador, sendo originadas de empresas de *software* que as disponibilizam com permissão para divulgação (DALPIAZ, 2018). Entre os grupos, foram usados os *datasets* “g02-federalspending“ e “g12-camperplus“, contendo respectivamente 98 e 55 histórias de usuário, como entradas para a ferramenta validada.

A partir dos dados extraídos do repositório, foram filtrados os requisitos e originados três documentos para verificação. Os dois primeiros foram elaborados com histórias tendendo a serem problemáticas, e histórias presumivelmente corretas, selecionadas por meio de uma análise rápida e superficial. O terceiro documento foi constituído a partir de uma seleção aleatória de histórias. A segmentação foi realizada com o principal propósito de avaliar o desempenho da ferramenta em cenários diversos, e clarificar os resultados. A divisão final das bases, com o tamanho de cada após filtragem, está detalhada na Tabela 5.

Tabela 6 – Tamanho das bases segmentadas.

Conjunto de dados	**Tamanho**
Base tendendo a erros	40 histórias
Base tendendo a acertos	51 histórias
Base mista	100 histórias

7.2 RESULTADOS OBTIDOS

Ao longo da execução, o sistema retorna os problemas encontrados por ciclo, e o usuário pode avaliá-los como pertinentes ou não. Dado que as histórias de usuários

são altamente específicas, variando de acordo com o contexto do negócio em que estão inseridas, há situações em que o resultado da ferramenta pode não ser ideal. Ainda assim, o algoritmo mostra-se capaz de produzir resultados positivos, sinalizando corretamente falhas em uma parte considerável das situações.

Para teste e avaliação, o *software* foi executado em três instâncias distintas, uma para cada conjunto de dados. Utilizando a base com histórias tendendo a erros como ponto de partida, avaliou-se os resultados gerados pelo sistema para verificar sua capacidade de identificar problemas nas histórias. Os resultados dessa análise, incluindo a quantidade de histórias com falhas, estão detalhados na Tabela 6.

Tabela 7 – Resultados do diagnóstico de problemas da ferramenta (Base tendendo a erros)

Ciclo	Histórias com problemas	Sem Problemas
Completude	22	18
Não-ambiguidade	17	23
Consistência	2	38

Foi observado que mais da metade das histórias de usuário na primeira base apresentaram falhas relacionadas à não-completude. Dentro da tríade de problemas em requisitos, essa categoria foi a mais recorrente, seguida pela ambiguidade, que ocorreu em dezessete das quarenta histórias. Quanto às inconsistências, foi notada uma baixa incidência esperada, dado que os critérios para inconsistência são mais difíceis de serem atingidos pelas histórias em uma base de dados reduzida.

A seguir, apresentam-se três exemplos ilustrativos das falhas identificadas pelo sistema:

Ex. 1: "*As a user, I want to know about other versions of the object or its metadata that might be of use to me.*"

- **Completude:** Não há a presença do trecho de finalidade.
- **Ambiguidade:** Presença do termo vago "*might*", que indica vagueza de modalidade.

Ex. 2: "*As a Cornell faculty member, I want to have some control over how items within my collection sort in collection contents lists, so that I can control how my work is presented.*"

- **Ambiguidade:** Presença do termo vago "*some*", que indica vagueza de quantidade.

Ex. 3: "*As a depositor, I want to deposit arbitrarily large files, so that I am not limited in what files I can and cannot deposit.*"

- **Inconsistência:** O trecho é inconsistente consigo mesmo, pois depositar arquivos grandes não tem o mesmo significado de não ser limitado a arquivos que pode depositar.
- **Ambiguidade:** Presença do termo vago "*large*", que indica vagueza de quantidade.

Ainda, durante a análise do resultado da ferramenta, é avaliado se o seu diagnóstico é correto. Se forem detectados problemas e eles estão de fato presentes na história, considera-se isso um "Verdadeiro Positivo" (VP). Por outro lado, se os problemas não estão presentes, então considera-se esse caso um "Falso Positivo" (FP).

Adicionalmente, avaliou-se também os casos em que não foram detectados problemas, mesmo estando presentes, sendo esses tratados como "Falsos Negativos" (FN). Um exemplo dessa classificação para a base com dados tendendo a erros está apresentada na Tabela 7.

Tabela 8 – Verdadeiros Positivos, Falsos Positivos e Falsos Negativos (Base tendendo a erros)

Ciclo	VP	FP	FN
Completude	22	0	0
Não-ambiguidade	16	1	3
Consistência	2	0	0

Verificou-se que, no caso do ciclo de completude e consistência em bases tendendo a erros, o sistema foi bem sucedido em detectar falhas quando presentes, e conseguiu avaliá-las corretamente. No contexto do ciclo de não-ambiguidade, a taxa mediana de casos corretamente identificados foi 89%. Em um dos casos houve uma detecção equivocada de ambiguidades e em três não foram detectadas ambiguidades que estavam presentes nas histórias. As métricas de precisão, *recall* e *F1-score* podem ser visualizadas na Tabela 8.

Tabela 9 – Métricas resultantes (Base tendendo a erros)

Ciclo	Precisão	Recall	F1-Score
Completude	100%	100%	100%
Não-ambiguidade	94%	84%	89%
Consistência	100%	100%	100%

Na segunda iteração, executando o algoritmo com entrada da base de dados contendo histórias presumivelmente sem erros, foram registrados os resultados apresentados na Tabela 9.

Ao contrário da primeira execução, o sistema detectou uma quantidade reduzida de problemas. Ambiguidade e não-completude persistiram como falhas mais comuns entre as histórias, sendo que apenas uma apresentou problemas de consistência, relacionados a uma contradição consigo mesma. Os detalhes específicos dessas informações podem ser consultados na Tabela 10.

Tabela 10 – Resultados do diagnóstico de problemas da ferramenta (Base tendendo a acertos)

Ciclo	Problemas Identificados	Sem Problemas
Completude	2	49
Não-ambiguidade	3	48
Consistência	1	50

Tabela 11 – Verdadeiros Positivos (VP), Falsos Positivos (FP) e Falsos Negativos (FN) (Base tendendo a acertos)

Ciclo	VP	FP	FN
Completude	2	0	1
Não-ambiguidade	3	0	2
Consistência	1	0	0

Durante os três ciclos, os problemas identificados pelo sistema estavam efetivamente presentes nas histórias, sem apresentar equívocos. As taxas de acerto foram de 80%, 75% e 100% para a detecção de não-completude, ambiguidade e inconsistência, respectivamente. Por outro lado, no ciclo de completude, uma história apresentou problemas não identificados pelo sistema, enquanto no ciclo de não-ambiguidade, duas histórias não tiveram suas falhas detectadas. As métricas coletadas nessa base são apresentadas na Tabela 11.

Tabela 12 – Métricas resultantes (Base tendendo a acertos)

Ciclo	Precisão	Recall	F1-Score
Completude	100%	67%	80%
Não-ambiguidade	100%	60%	75%
Consistência	100%	100%	100%

Por fim, executou-se o código com uma base de histórias não-selecionadas, representando uma base de histórias real. Essa entrada, por ser consideravelmente maior que as anteriores, resultou em uma quantidade superior de falhas detectadas, como pode ser visualizado na Tabela 12.

Da mesma forma que nas demais bases, foi observada uma maior ocorrência de falhas associadas ao ciclo de completude e não-ambiguidade, totalizando 43% e 23% entre os cem casos totais, respectivamente. Não foram identificados problemas de consistência entre as histórias de usuário desse conjunto de dados. Aprofundando a análise, é possível verificar as informações detalhadas dos resultados do diagnóstico da ferramenta na Tabela 13.

Entre as 43 histórias diagnosticadas com problemas de completude pela ferramenta, cerca de quarenta efetivamente possuíam tais problemas, a partir de uma análise do conteúdo da história e do resultado retornado pela ferramenta, enquanto os três casos restantes foram equivocadamente diagnosticados. Não foram detectados falsos negativos nessa base.

Tabela 13 – Resultados do diagnóstico de problemas da ferramenta (Base mista)

Ciclo	Problemas Identificados	Sem Problemas
Completude	43	57
Não-ambiguidade	23	77
Consistência	0	100

Tabela 14 – Verdadeiros Positivos (VP), Falsos Positivos (FP) e Falsos Negativos (FN) (Base mista)

Ciclo	VP	FP	FN
Completude	40	3	0
Não-ambiguidade	19	4	3
Consistência	0	0	0

Em relação ao ciclo de não-ambiguidade, três histórias problemáticas não foram identificadas pelo sistema. Entre as 23 histórias detectadas como falhas, dezenove possuíam ambiguidade, gerada pela presença de termos vagos em todos os casos, e quatro foram erroneamente diagnosticadas. As métricas resultantes da execução dessa base estão listadas na Tabela 14.

Tabela 15 – Métricas resultantes (Base Mista)

Ciclo	Precisão	Recall	F1-Score
Completude	93%	100%	96%
Não-ambiguidade	83%	86%	84%
Consistência	-	-	-

7.3 ANÁLISE DOS RESULTADOS

Por meio dos dados e informações que foram obtidos através da execução do código com diferentes bases de entrada, foi feita uma análise geral da técnica e da ferramenta, identificando seus aspectos positivos, bem como pontos de melhorias em cada etapa do fluxo. Na Figura 20, é possível observar um gráfico que retrata a taxa média de acerto do sistema em relação a cada base que foi submetida a teste. Para o cálculo da taxa geral foram considerados as métricas de *F1-Score* nas três bases nos ciclos de completude, não-ambiguidade e consistência, se realizando então a média aritmética entre os valores calculados.

A ferramenta apresentou um resultado positivo nos três casos, tendo eficácia de, no mínimo, 85% na detecção de problemas relacionados a tríade de problemas de requisitos. No melhor caso, usando a base tendendo a erros como entrada nos testes, o sistema apresentou taxa de acerto de cerca de 96%.

De forma geral, ao analisar as estatísticas obtidas nos testes dos três conjuntos de

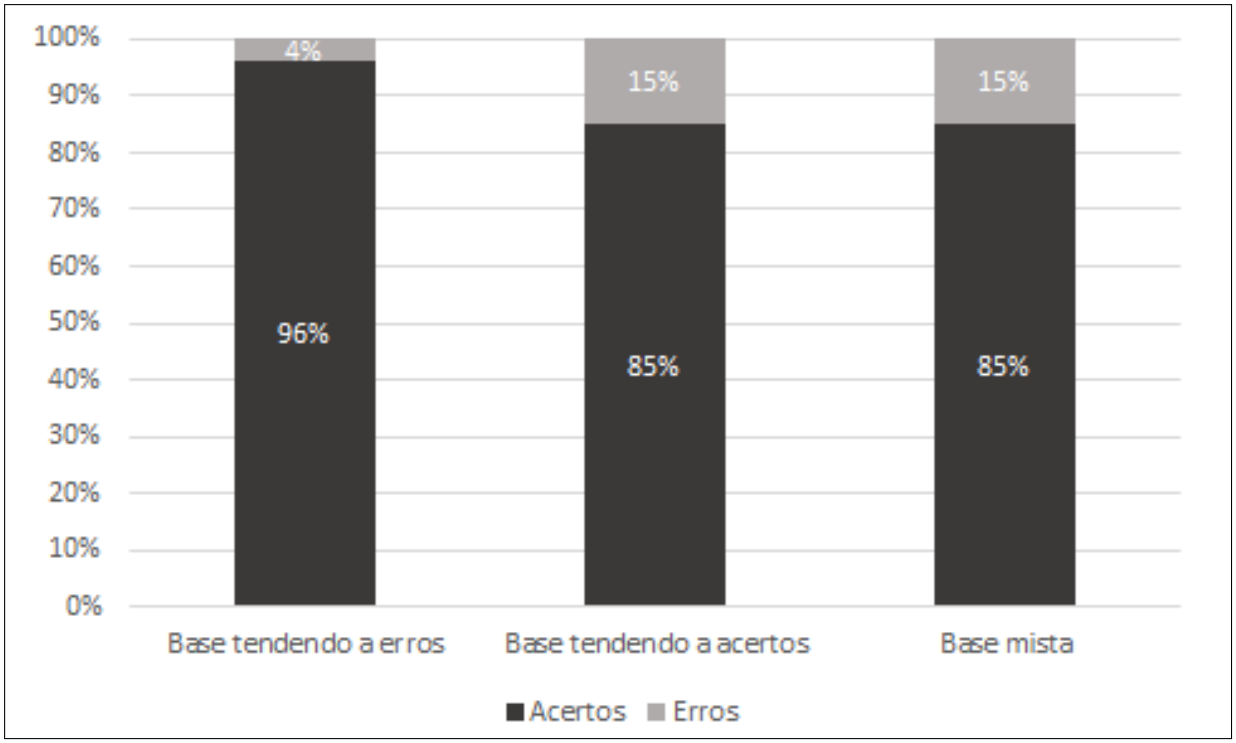

Figura 20 – Taxa média de acertos do sistema com cada base.

Fonte: Autoria própria.

dados e calcular suas proporções, constatou-se que o sistema tem uma eficácia geral de 89% na detecção dos problemas estudados. Considera-se essa uma taxa positiva, que indica que o sistema tem um desempenho satisfatório na identificação de histórias incompletas, ambíguas e inconsistentes, fornecendo aos usuários uma base sólida para corrigir e aprimorar suas especificações. No entanto, há aspectos que podem ser melhorados para alcançar resultados ainda mais precisos.

A Figura 21 detalha o desempenho geral de cada ciclo estudado, a partir das estatísticas das três bases.

Ao examinar os percentuais, observa-se que, em primeiro lugar, o ciclo de consistência não apresentou erros de diagnóstico em nenhum dos três cenários. Apesar de existirem ocorrências de inconsistências na primeira e na segunda base, o número foi consideravelmente baixo. Isso ocorre por termos uma amostragem de dados relativamente pequena, na qual os requisitos abrangem diferentes partes dos sistemas especificados, que ocasionalmente se relacionam. Mesmo em casos em que houve convergência de contextos, as histórias raramente atingiram os critérios que caracterizam uma contradição, novamente pelas particularidades do conjunto de dados.

Outro aspecto relevante visualizado a partir dos gráficos é a porcentagem de acertos no ciclo de ambiguidade, que se mostrou a menor entre os três ciclos analisados. Entre as ambiguidades, o uso de palavras vagas mostrou-se um problema mais comum do que trechos com múltiplas interpretações, representando quase a totalidade das ocorrências

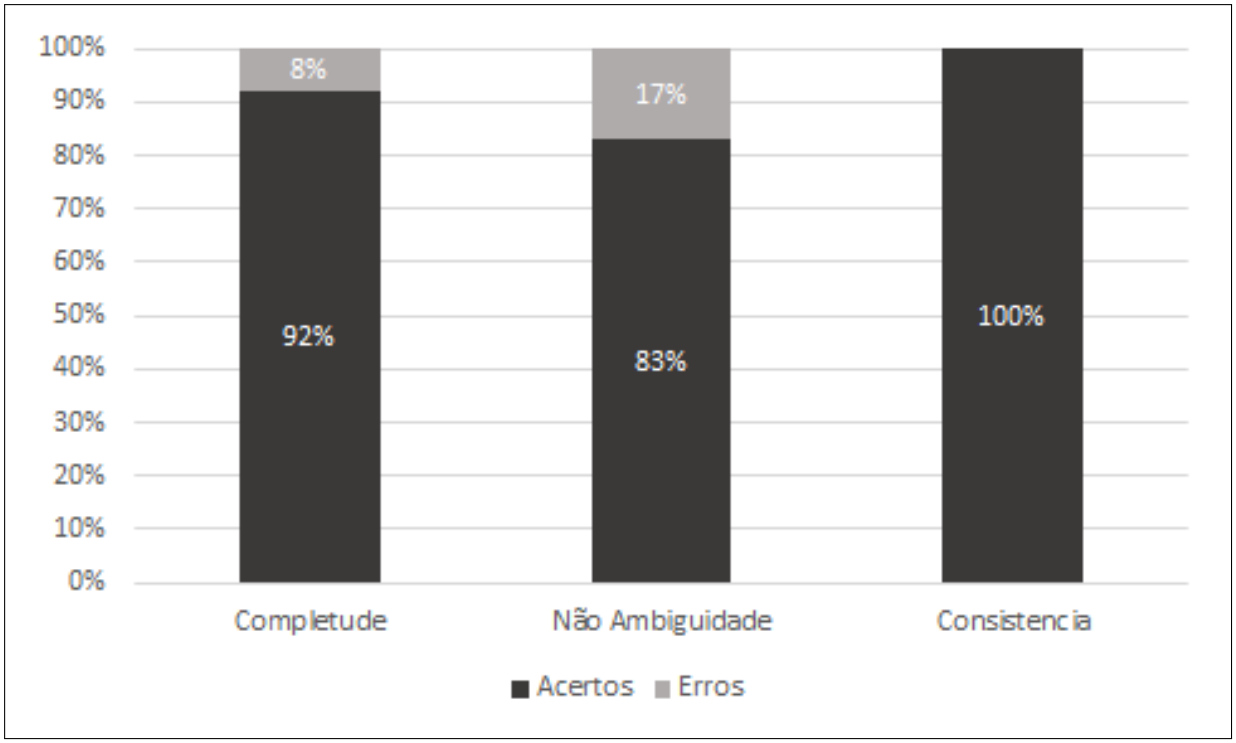

Figura 21 – Taxa média de acertos do sistema em cada ciclo.

Fonte: Autoria própria.

detectadas.

Dois fatores principais foram responsáveis pela menor porcentagem de acertos neste ciclo. Primeiramente, algumas palavras vagas não foram detectadas devido ao estágio inicial e incompleto do dicionário de palavras vagas frequentes, que está constantemente expandindo à medida que novos termos são incluídos. Em segundo lugar, em alguns dos casos nos quais houve correspondência entre palavras da história e do dicionário, observou-se uma imprevisibilidade quanto à presença ou ausência de um problema devido a contextos específicos nos quais alguns termos são empregados, o que contribuiu para a taxa de erros.

Em relação ao ciclo de completude, os erros se deram principalmente pela falha do modelo selecionado em classificar corretamente os rótulos PropBank em algumas das histórias, fazendo o algoritmo equivocadamente não detectar partes ausentes de uma história.

Em suma, foi possível avaliar que a ferramenta desenvolvida tem potencial de auxiliar o usuário em sua especificação de requisitos, retornando com taxa de acertos satisfatória possíveis falhas relacionadas a tríade de problemas mais comuns em histórias de usuário. Entretanto, existe a necessidade de adaptação do algoritmo para casos mais específicos, bem como expandir os dados que são usados na avaliação, como a base de termos ambíguos que, apesar de detectar uma quantidade considerável de palavras vagas, ainda pode ser ampliada.

8 CONSIDERAÇÕES FINAIS

O presente trabalho teve como principal objetivo propor uma técnica e ferramenta para auxiliar analistas de *software* na especificação de requisitos de seus sistemas, atuando como guias para documentá-los de maneira clara, precisa e livre de erros. Uma documentação de requisitos falha tende a ser tornar um problema durante a fase de desenvolvimento do projeto, frequentemente exigindo mais esforço e retrabalho por parte da equipe. Além disso, no contexto de métodos ágeis, no qual o foco está na entrega de *software* funcional e não na documentação extensiva, o uso de uma ferramenta para auxílio na especificação torna o processo não apenas mais preciso, mas também mais simples e ágil.

Para contextualização do cenário de Engenharia de Requisitos nos dias de hoje, foi conduzida uma RSL, com o intuito de identificar os desafios enfrentados na área. Onze problemas principais foram mapeados e, para cada, analisaram-se abordagens e técnicas propostas por pesquisadores para solucioná-los. Observando o cenário atual, percebeu-se que métodos baseados em IA vêm sendo amplamente estudados não somente na área de Engenharia de Requisitos, como em toda Engenharia de *Software*.

A partir das oportunidades de estudo ainda em aberto no tema, esta pesquisa foi desenvolvida. Dando foco em três dos principais problemas da área de Engenharia de Requisitos - ambiguidade, não completude e inconsistência - avaliou-se como a IA poderia contribuir para a qualidade da especificação, auxiliando o analista de *software* a resolver ou minimizar tais problemas. Entre as técnicas de IA, o PLN apresentou-se um campo ideal para estudo, permitindo a extração de informações de um requisito a partir das relações entre as palavras, seus significados e funções dentro de uma frase.

A técnica proposta foi segmentada em três ciclos, abordando os problemas da tríade. Em cada um dos ciclos a história é fragmentada em *tokens*, que são processados de acordo com os objetivos da etapa. Ao final, o usuário pode avaliar se os resultados apresentados são válidos e fazer as devidas alterações na sua especificação com base no *feedback* recebido pela ferramenta de apoio desenvolvida.

Como forma de avaliação, selecionamos conjuntos de dados com histórias de usuário disponíveis em um repositório público para serem utilizados como entrada. As histórias são originadas de bases reais de empresas de *software* que disponibilizam seus dados. Para entender o comportamento da ferramenta em diferentes contextos, adicionalmente os conjuntos foram segmentados em três documentos, com base em um filtro de histórias. Tais documentos incluíam histórias propensas a erros, histórias propensas a acertos e histórias selecionadas aleatoriamente.

Ao executar o sistema utilizando as três bases, foi possível notar que os resultados obtidos foram positivos e a detecção de problemas foi precisa na maioria dos casos, com

taxa de acertos de 85% com a base tendendo a acertos e a base mista, máxima de 96% com a base tendendo a erros, e precisão média de 89%. Entre os ciclos, as detecções de histórias incompletas e inconsistentes alcançaram as maiores taxas de acerto, ambas superiores a 90%, enquanto o ciclo de ambiguidade apresentou uma taxa de acerto de aproximadamente 83%.

Conclui-se a partir dessa pesquisa que, embora exista a necessidade de aprimoramentos no sistema, a ferramenta proposta baseada em PLN mostra-se promissora e benéfica para a área de Engenharia de Requisitos. As avaliações realizadas durante o estudo evidenciam taxas de acerto satisfatórias na detecção de histórias incompletas, inconsistentes e ambíguas, indicando o grande potencial do sistema como aliado aos analistas de *software* nas etapas de especificação e validação dos requisitos.

8.1 DIFICULDADES ENCONTRADAS

Durante o desenvolvimento da pesquisa, enfrentou-se uma série de obstáculos e contratempos que representaram desafios significativos.

O sistema se tornou complexo e demandou uma pesquisa extensa sobre temas de Engenharia de Requisitos, IA e outros campos externos à área de Ciência da Computação, como gramática. Adicionalmente, as etapas de desenvolvimento demandaram uma minuciosa busca por técnicas condizentes a proposta. Devido a essas circunstâncias, algumas das atividades planejadas inicialmente, como a aplicação do sistema com supervisão de analistas, foram adiadas e serão tratadas em futuras pesquisas.

Outros desafios estavam ligados diretamente ao formato das histórias e às capacidades das técnicas de PLN. Entender o contexto de uma história é uma tarefa desafiadora e pouco simples. Embora o algoritmo proposto demonstre competência nesse aspecto, ele não alcança um nível de precisão excelente. Esse aspecto torna o sistema mais propensos a erros nas detecções de problemas, principalmente nos ciclos de ambiguidade e inconsistência.

A criação do dicionário de termos vagos, utilizado no ciclo de ambiguidade, também foi um processo que demandou esforço. Determinar o tamanho ideal do dicionário se mostrou desafiador. Mesmo com uma considerável quantidade de termos vagos extraídos da literatura, ao final do estudo ficou claro que é necessário expandir o dicionário, pois sua cobertura atual é insuficiente.

Buscou-se enfrentar tais desafios de maneira estratégica, procurando encontrar soluções alternativas para superá-los.

8.2 AMEAÇAS À VALIDADE

Nesta seção, abordamos as ameaças à validade que possam potencialmente influenciar os resultados e conclusões deste estudo, destacando a importância de reconhecê-las e enfrentá-las para assegurar a confiabilidade e precisão das descobertas apresentadas.

A validade desta pesquisa é ameaçada principalmente pela maneira como os testes foram conduzidos. Embora os dados utilizados tenham sido compilados por fontes acadêmicas, baseados em informações internas disponibilizadas por empresas de *software*, o sistema requer a confirmação dos resultados da sua análise pelo usuário com quem está interagindo. Isso estabelece uma colaboração mútua entre o sistema e o analista para a aplicação eficaz desta técnica.

Entretanto, durante os testes realizados na etapa de verificação, não foi possível contar com a assistência de um especialista na avaliação, conforme inicialmente planejado. Como resultado, os resultados da ferramenta foram avaliados exclusivamente pelos pesquisadores, o que pode comprometer a imparcialidade e a precisão da avaliação realizada.

8.3 TRABALHOS FUTUROS

Para futuras pesquisas, planeja-se implementar melhorias adicionais na ferramenta proposta, com o intuito de aprimorar sua precisão na detecção de problemas. Os algoritmos de ambiguidade e consistência necessitam de alterações adicionais para alcançar uma eficácia ainda maior. Também é necessário expandir o dicionário de termos vagos atualmente existente, o que requer pesquisa adicional na literatura sobre termos que comumente indicam requisitos vagos. Por fim, considera-se complementar o estudo com uma interface visual que assegure ao usuário uma compreensão mais profunda do que está sendo analisado na história.

Apesar de não ter sido possível realizar isso neste trabalho, há planos para futuramente aplicar a ferramenta em um projeto e analisar seu desempenho ao longo de um período prolongado, sob a supervisão de um analista de requisitos. Dessa forma, ao aplicar a ferramenta em um ambiente prático com o *feedback* de profissionais envolvidos, ter-se-á oportunidade de avaliar os resultados de forma mais abrangente. Isso possibilitará identificar com exatidão não apenas os pontos mais fortes, mas também os aspectos que podem ainda necessitar melhorias que não foram visualizadas.

REFERÊNCIAS

ADAMS, K. M. Introduction to non-functional requirements. In: ____. **Nonfunctional Requirements in Systems Analysis and Design**. Cham: Springer International Publishing, 2015. p. 45–72. ISBN 978-3-319-18344-2. Disponível em: <https://doi.org/10.1007/978-3-319-18344-2_3>.

ALI, S.; IQBAL, N.; HAFEEZ, Y. Towards requirement change management for global software development using case base reasoning. **Mehran University Research Journal of Engineering and Technology**, Mehran University of Engineering and Technology, Jamshoro, Pakistan, v. 37, 2018.

AMARO, I. et al. Ai unreliable answers: A case study onăchatgpt. In: DEGEN, H.; NTOA, S. (Ed.). **Artificial Intelligence in HCI**. Cham: Springer Nature Switzerland, 2023. p. 23–40. ISBN 978-3-031-35894-4.

AMBRAKAIT, R. A contrastive analysis of vague language in spoken english of lithuanian learners of english and native speakers. Jan 2016.

ASADABADI, M. R. et al. Ambiguous requirements: A semi-automated approach to identify and clarify ambiguity in large-scale projects. **Comput. Ind. Eng.**, v. 149, p. 106828, 2020.

BEHUTIYE, W. et al. Non-functional requirements documentation in agile software development: Challenges and solution proposal. In: . [S.l.: s.n.], 2017. p. 515–522. ISBN 978-3-319-69925-7.

BHATIA, J. et al. A theory of vagueness and privacy risk perception. In: **2016 IEEE 24th International Requirements Engineering Conference (RE)**. [S.l.: s.n.], 2016. p. 26–35.

BIRD, S.; KLEIN, E.; LOPER, E. **Natural Language Processing with Python: Analyzing Text with the Natural Language Toolkit**. Beijing: O'Reilly, 2009. ISBN 978-0-596-51649-9. Disponível em: <http://www.nltk.org/book>.

BOAKYE, N. Gender aspects of vague language use: Formal and informal contexts. 2007.

BOURQUE, P.; FAIRLEY, R. E. (Ed.). **SWEBOK: Guide to the Software Engineering Body of Knowledge**. Version 3.0. Los Alamitos, CA: IEEE Computer Society, 2014. ISBN 978-0-7695-5166-1. Disponível em: <http://www.swebok.org/>.

BROOKS JR, F. No silver bullet essence and accidents of software engineering. **IEEE Computer**, v. 20, p. 10–19, 04 1987.

COHN, M. **User Stories Applied: For Agile Software Development**. USA: Addison Wesley Longman Publishing Co., Inc., 2004. ISBN 0321205685.

COUGHLAN, J.; MACREDIE, R. Effective communication in requirements elicitation: A comparison of methodologies. **Requirements Engineering**, v. 7, p. 47–60, 06 2002.

DALPIAZ, F. **Requirements Data Sets (User Stories)**. [S.l.]: Mendeley, 2018.

DALPIAZ, F.; SCHALK, I.; LUCASSEN, G. Pinpointing ambiguity and incompleteness in requirements engineering via information visualization and nlp. In: ____. [S.l.: s.n.], 2018. p. 119–135. ISBN 978-3-319-77242-4.

DENG, L.; LIU, Y. **Deep Learning in Natural Language Processing**. 1st. ed. [S.l.]: Springer Publishing Company, Incorporated, 2018. ISBN 9789811052088.

DEVLIN, J. et al. **BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding**. 2018. Cite arxiv:1810.04805Comment: 13 pages. Disponível em: <http://arxiv.org/abs/1810.04805>.

DEVRIES, B.; CHENG, B. Automatic detection of incomplete requirements using symbolic analysis and evolutionary computation. In: **SSBSE**. [S.l.: s.n.], 2017.

DUC, A. N. et al. **Generative AI in Undergraduate Information Technology Education – Insights from nine courses**. 2023. Disponível em: <https://arxiv.org/abs/2311.10199>.

FELLBAUM, C. **WordNet: An Electronic Lexical Database**. Bradford Books, 1998. Disponível em: <https://mitpress.mit.edu/9780262561167/>.

FERRARI, A. et al. Pragmatic ambiguity detection in natural language requirements. In: **2014 IEEE 1st International Workshop on Artificial Intelligence for Requirements Engineering (AIRE)**. [S.l.: s.n.], 2014. p. 1–8.

FITZGERALD, B. Software crisis 2.0. **IEEE Computer - COMPUTER**, v. 45, p. 89–91, 04 2012.

GALLARDO-VALENCIA, R.; OLIVERA, V.; SIM, S. Are use cases beneficial for developers using agile requirements? In: . [S.l.: s.n.], 2007. p. 11 – 22. ISBN 978-0-7695-3378-0.

GARCIA, J. E.; PAIVA, A. C. A requirements-to-implementation mapping tool for requirements traceability. **J. Softw.**, v. 11, p. 193–200, 2016.

GARDNER, M. et al. **AllenNLP: A Deep Semantic Natural Language Processing Platform**. 2022. Disponível em: <https://github.com/allenai/allennlp>.

GEROGIANNIS, V. Software requirements engineering: State of the art and research trends. international journal of management and applied science (ijmas), vol. 3, issue 9, 2017, pp. 66-71. p. 2394–7926, 09 2017.

GLEICH, B.; CREIGHTON, O.; KOF, L. Ambiguity detection: Towards a tool explaining ambiguity sources. In: . [S.l.: s.n.], 2010. v. 6182, p. 218–232. ISBN 978-3-642-14191-1.

GOGOLLA, M. Unified modeling language. In: ____. **Encyclopedia of Database Systems**. Boston, MA: Springer US, 2009. p. 3232–3239. ISBN 978-0-387-39940-9. Disponível em: <https://doi.org/10.1007/978-0-387-39940-9_440>.

GULL, N. et al. A block-chain oriented model driven framework for handling inconsistent requirements in global software development. In: **2021 10th International Conference on Software and Computer Applications**. New York, NY, USA: Association for Computing Machinery, 2021. (ICSCA 2021), p. 105111. ISBN 9781450388825. Disponível em: <https://doi.org/10.1145/3457784.3457799>.

GWON, Y. N. et al. The use of generative ai for scientific literature searches for systematic reviews: Chatgpt and microsoft bing ai performance evaluation. **JMIR Med Inform**, v. 12, p. e51187, May 2024. ISSN 2291-9694. Disponível em: <https://medinform.jmir.org/2024-/1/e51187>.

HAIDRAR, S. et al. A domain-specific language to manage requirements traceability. **J. Softw.**, v. 13, p. 460–480, 2018.

HARMAN, M. The role of artificial intelligence in software engineering. In: **RAISE '12: Proceedings of the First International Workshop on Realizing AI Synergies in Software Engineering**. Zurich, Switzerland: IEEE Press, 2012. p. 1–6.

HONNIBAL, M.; MONTANI, I. spaCy 2: Natural language understanding with Bloom embeddings, convolutional neural networks and incremental parsing. To appear. 2017.

HULL, E.; JACKSON, K.; DICK, J. **Requirements Engineering**. [S.l.]: Springer Cham, 2017. (Practitioner series). ISBN 9783319610733.

HUTT, S. et al. Feedback on feedback: Comparing classic natural language processing and generative ai to evaluate peer feedback. In: **Proceedings of the 14th Learning Analytics and Knowledge Conference**. New York, NY, USA: Association for Computing Machinery, 2024. (LAK '24), p. 5565. ISBN 9798400716188. Disponível em: <https://doi.org/10.1145-/3636555.3636850>.

IEEE. **IEEE Standard Glossary of Software Engineering Terminology**. 1990. 1-84 p.

INDURKHYA, N.; DAMERAU, F. J. **Handbook of Natural Language Processing**. 2nd. ed. [S.l.]: Chapman amp; Hall/CRC, 2010. ISBN 1420085921.

JACKSON, M. **Software Requirements Specifications: A Lexicon of Practice, Principles and Prejudices**. USA: ACM Press/Addison-Wesley Publishing Co., 1995. ISBN 0201877120.

JURAFSKY, D.; MARTIN, J. H. **Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition**. 1st. ed. USA: Prentice Hall PTR, 2000. ISBN 0130950696.

KALINOWSKI, M. et al. Preventing incomplete/hidden requirements: Reflections on survey data from austria and brazil. In: . [S.l.: s.n.], 2016. ISBN 978-3-319-27032-6.

KAMSTIES, E. et al. From contract drafting to software specification: Linguistic sources of ambiguity. parallelism, 12 2003.

KENT, S. Model driven engineering. In: BUTLER, M.; PETRE, L.; SERE, K. (Ed.). **Integrated Formal Methods**. Berlin, Heidelberg: Springer Berlin Heidelberg, 2002. p. 286–298. ISBN 978-3-540-47884-3.

KHURANA, D. et al. Natural language processing: state of the art, current trends and challenges. **Multimedia Tools and Applications**, Springer Science and Business Media LLC, v. 82, n. 3, p. 37133744, jul. 2022. ISSN 1573-7721. Disponível em: <http://dx.doi.org/10-.1007/s11042-022-13428-4>.

KITCHENHAM, B.; CHARTERS, S. Guidelines for performing systematic literature reviews in software engineering. v. 2, 01 2007.

KOBAYASHI, N. et al. Top-down rst parsing utilizing granularity levels in documents. **Proceedings of the AAAI Conference on Artificial Intelligence**, v. 34, p. 8099–8106, 2020.

KOKOL, P. The use of ai in software engineering: synthetic knowledge synthesis of recent research literature. 2024.

KORZYNSKI, P. et al. Artificial intelligence prompt engineering as a new digital competence: Analysis of generative ai technologies such as chatgpt. **Entrepreneurial Business and Economics Review**, v. 11, p. 25–37, 09 2023.

LAMSWEERDE, A. van. **Requirements Engineering: From System Goals to UML Models to Software Specifications**. 1st. ed. [S.l.]: Wiley Publishing, 2009. ISBN 0470012706.

LAPLANTE, P. A. **Requirements engineering for software and systems**. 3rd. ed. [S.l.]: Auerbach Publications, 2017.

LI, Y. et al. Automated extraction of domain knowledge in practice: The case of feature extraction from requirements at danfoss. In: **Proceedings of the 24th ACM Conference on Systems and Software Product Line: Volume A - Volume A**. New York, NY, USA: Association for Computing Machinery, 2020. (SPLC '20). ISBN 9781450375696. Disponível em: <https://doi.org/10.1145/3382025.3414968>.

LIU, K.; REDDIVARI, S.; REDDIVARI, K. Artificial intelligence in software requirements engineering: State-of-the-art. In: **2022 IEEE 23rd International Conference on Information Reuse and Integration for Data Science (IRI)**. [S.l.: s.n.], 2022. p. 106–111.

MCCONNELL, S. **Code Complete: A Practical Handbook of Software Construction**. 2. ed. Redmond, WA: Microsoft Press, 2004. (Best Practices for Developers). ISBN 978-0-7356-1967-8. Disponível em: <https://www.safaribooksonline.com/library-/view/code-complete-second/0735619670/>.

MEIBAUER, J. **The Oxford Handbook of Lying**. Oxford University Press, 2018. ISBN 9780198736578. Disponível em: <https://doi.org/10.1093/oxfordhb/9780198736578.001-.0001>.

MELLO, O. **Ambiguity, Inconsistency and Incompleteness Detector for User Stories**. 2024. Disponível em: <https://github.com/odcmello/Requirements-AI>.

MELLO., O.; FONTOURA., L. Challenges in requirements engineering and its solutions: A systematic review. In: INSTICC. **Proceedings of the 24th International Conference on Enterprise Information Systems - Volume 2: ICEIS,**. [S.l.]: SciTePress, 2022. p. 70–77. ISBN 978-989-758-569-2. ISSN 2184-4992.

MEMON, S. A.; WEST, J. D. **Search Engines Post-ChatGPT: How Generative Artificial Intelligence Could Make Search Less Reliable**. 2024. Disponível em: <https://arxiv.org-/abs/2402.11707>.

MEZGHANI, M.; KANG, J.; SEDES, F. Industrial requirements classification for redundancy and inconsistency detection in semios. In: . [S.l.: s.n.], 2018. p. 297–303.

MISHRA, S.; SHARMA, A. On the use of word embeddings for identifying domain specific ambiguities in requirements. In: **2019 IEEE 27th International Requirements Engineering Conference Workshops (REW)**. [S.l.: s.n.], 2019. p. 234–240.

MKPOJIOGU, E. O. C.; HASHIM, N. Quality based prioritization: An approach for prioritizing software requirements. **Journal of Telecommunication, Electronic and Computer Engineering**, v. 9, p. 17–21, 2017.

MORALES, S. et al. Generative ai in model-driven software engineering education: Friend or foe? In: **2023 ACM/IEEE International Conference on Model Driven Engineering Languages and Systems Companion (MODELS-C)**. [S.l.: s.n.], 2023. p. 110–113.

NADEEM, M.; LEE, S. U.-J. Requirement elicitation framework for global software development. **Indian journal of science and technology**, v. 12, p. 1–6, 2019.

NATH, S. et al. New meaning for nlp: the trials and tribulations of natural language processing with gpt-3 in ophthalmology. **British Journal of Ophthalmology**, v. 106, p. 889–892, 2022.

NAZ, H. et al. Effective usage of ai technique for requirement change management practices. In: . [S.l.: s.n.], 2013. p. 121–125.

NGUYEN, H.; GRUNDY, J.; ALMOSY, M. Guitar: An ontology-based automated requirements analysis tool. In: . [S.l.: s.n.], 2014.

Object Management Group. **Unified Modeling Language (UMLő)**. 2011. Disponível em: <http://www.omg.org/spec/UML/index.htm>.

OSMAN, M. H.; ZAHARIN, M. F. Ambiguous software requirement specification detection: An automated approach. In: **Proceedings of the 5th International Workshop on Requirements Engineering and Testing**. New York, NY, USA: Association for Computing Machinery, 2018. (RET '18), p. 3340. ISBN 9781450357494. Disponível em: <https://doi.org/10.1145/3195538.3195545>.

PALMER, M.; GILDEA, D.; KINGSBURY, P. The Proposition Bank: An annotated corpus of semantic roles. **Computational Linguistics**, v. 31, n. 1, p. 71–106, 2005. Disponível em: <https://aclanthology.org/J05-1004>.

PATTON, J.; ECONOMY, P. **User Story Mapping: Discover the Whole Story, Build the Right Product**. 1st. ed. [S.l.]: O'Reilly Media, Inc., 2014. ISBN 1491904909.

PMI. **Project Management Institute: The PMI Guide to BUSINESS ANALYSIS**. 2017.

PRESSMAN, R. S.; MAXIM, B. R. **Software Engineering: A Practitioner's Approach**. 9th. ed. [S.l.]: AMGH, 2019.

PYYSALO, S. Part-of-speech tagging. In: ____. **Encyclopedia of Systems Biology**. New York, NY: Springer New York, 2013. p. 1650–1650. ISBN 978-1-4419-9863-7. Disponível em: <https://doi.org/10.1007/978-1-4419-9863-7_162>.

RASHEED, Z.; GHWANMEH, S.; ABUALKISHIK, A. Z. Harnessing artificial intelligence for personalized learning: a systematic review. **Data and Metadata**, v. 2, p. 146, 2023.

ROBEER, M. et al. Automated extraction of conceptual models from user stories via nlp. In: **2016 IEEE 24th International Requirements Engineering Conference (RE)**. [S.l.: s.n.], 2016. p. 196–205.

RUIZ, M.; HASSELMAN, B. Can we design software as we talk? **Enterprise, Business-Process and Information Systems Modeling**, p. 327–334, 2020.

SALVADOR, M. E. R. F. L.; SANTOS, L. B. R. dos. Domar: An approach to prevent problems related to requirements documentation and management. In: . [S.l.: s.n.], 2016.

SHAH, U.; JINWALA, D. Resolving ambiguities in natural language software requirements: A comprehensive survey. **ACM SIGSOFT Software Engineering Notes**, v. 40, p. 1–7, 09 2015.

SHAHZAD, B. et al. Reliable requirements engineering practices for covid-19 using blockchain. **Sustainability**, v. 13, n. 12, 2021. ISSN 2071-1050. Disponível em: <https://www-.mdpi.com/2071-1050/13/12/6748>.

SHARMA, R.; SHARMA, N.; BISWAS, K. K. Machine learning for detecting pronominal anaphora ambiguity in nl requirements. **2016 4th Intl Conf on Applied Computing and Information Technology/3rd Intl Conf on Computational Science/Intelligence and Applied Informatics/1st Intl Conf on Big Data, Cloud Computing, Data Science & Engineering (ACIT-CSII-BCD)**, p. 177–182, 2016.

SHI, P.; LIN, J. Simple bert models for relation extraction and semantic role labeling. **ArXiv**, abs/1904.05255, 2019.

SHREDA, Q. A.; HANANI, A. A. Identifying non-functional requirements from unconstrained documents using natural language processing and machine learning approaches. **IEEE Access**, p. 1–1, 2024.

SOMMERVILLE, I. **Software Engineering**. 10th. ed. São Paulo, SP, Brasil: Pearson Prentice Hall, 2019.

SOMMERVILLE, I.; KOTONYA, G. **Requirements Engineering - Processes and Techniques**. 1st. ed. Hoboken, NJ, United States: Wiley, 1998.

Standish Group International. **The Standish Group Report CHAOS**. 2019.

STELLMAN, A.; GREENE, J. **Learning Agile: Understanding Scrum, XP, Lean, and Kanban**. O'Reilly, 2014. ISBN 9781449331924. Disponível em: <https://books.google-.com.br/books?id=T7BhngEACAAJ>.

SURAWORACHET, W.; SEON, J.; CUKUROVA, M. Predicting challenge moments from students' discourse: A comparison of gpt-4 to two traditional natural language processing approaches. In: **Proceedings of the 14th Learning Analytics and Knowledge Conference**. New York, NY, USA: Association for Computing Machinery, 2024. (LAK '24), p. 473485. ISBN 9798400716188. Disponível em: <https://doi.org/10.1145/3636555.3636905>.

WIEGERS, K. E.; BEATTY, J. **Software Requirements 3**. USA: Microsoft Press, 2013. ISBN 0735679665.

YAN, B. et al. A case study for software quality evaluation based on sct model with bp neural network. **IEEE Access**, v. 8, p. 56403–56414, 2020.

YASEEN, M.; MUSTAPHA, A.; IBRAHIM, N. Prioritization of software functional requirements: Spanning tree based approach. **International Journal of Advanced Computer Science and Applications**, v. 10, 01 2020.

ZAHRIN, M. F. et al. Issues in requirements specification in malaysias public sector: an evidence from a semi-structured survey and a static analysis. **International Journal of Advanced Computer Science and Applications**, v. 13, 2022.

ZAIT, F.; ZAROUR, N. Addressing lexical and semantic ambiguity in natural language requirements. In: **2018 Fifth International Symposium on Innovation in Information and Communication Technology (ISIICT)**. [S.l.: s.n.], 2018. p. 1–7.

ZHAN, T. et al. Optimization techniques for sentiment analysis based on llm (gpt-3). **Applied and Computational Engineering**, v. 67, p. 41–47, 2024.

ZHANG, Y. Design of an intelligent qamp;a system for online education platform based on natural language processing technology. **Journal of Electrical Systems**, 2024.

REFERÊNCIAS DA REVISÃO

Mahmood Alsaadi, Alexei Lisitsa, and Malik Qasaimeh. Minimizing the ambiguities in medical devices regulations based on software requirement engineering techniques. pages 1–5, 12 2019.

Tawfeeq Alsanoosy, Maria Spichkova, and James Harland. Exploratory analysis of cultural influences on requirements engineering activities based on stakeholders profile. *Procedia Computer Science*, 176:3379–3388, 01 2020.

Wasim Alsaqaf, Maya Daneva, and Roel Wieringa. Agile quality requirements engineering challenges: First results from a case study. 12 2017.

Wasim Alsaqaf, Maya Daneva, and Roel Wieringa. Quality requirements challenges in the context of large-scale distributed agile: An empirical study. *Information and Software Technology*, 110, 02 2019.

Yury V. Balashov, Victor K. Batovrin, and Yury I. Lobanovsky. Requirements engineering practice in russian aviation industry. In *2019 Actual Problems of Systems and Software Engineering (APSSE)*, pages 42–49, 2019.

Jaroslav Bendík. Consistency checking in requirements analysis. pages 408–411, 07 2017.

Elizabeth Bjarnason, Michael Unterkalmsteiner, Markus Borg, and Emelie Engström. A multi-case study of agile requirements engineering and the use of test cases as requirements. *Information and Software Technology*, 77, 03 2016.

Feng Chen, Norah Power, J. Collins, and Fuyuki Ishikawa. Contemporary requirements challenges and issues: an empirical study in 11 organizations. pages 1592–1599, 04 2019.

Vanessa Faria de Souza, Alexandre L'Erario, José A. Fabri, and Elias C. Genvigir. Improvement of software process small business using spem: A case study on requirements engineering. In *2016 11th Iberian Conference on Information Systems and Technologies (CISTI)*, pages 1–6, 2016.

Senay Demirel and Resul Das. Software requirement analysis: Research challenges and technical approaches. pages 1–6, 03 2018.

Sita devulapalli, O R S Rao, and Akhil Khare. Requirements prioritization: Parameters of relevance – an empirical study across 3 datasets. pages 1–5, 03 2016.

Xiao Fei, Chen Bin, and Zhao Siming. A methodology of requirements validation for aviation system development. pages 4484–4489, 08 2020.

Davide Fucci, Andreas Falkner, Gottfried Schenner, Fabrizio Brasca, Tomi Männistö, Alexander Felfernig, Walid Maalej, Cristina Palomares, Xavier Franch, Dolors Costal, Mikko Raatikainen, Martin Stettinger, Zijad Kurtanovi, Tero Kojo, and Lars Koenig. Needs and challenges for a platform to support large-scale requirements engineering: a multiple-case study. pages 1–10, 10 2018.

Manju Geogy and Andhe Dharani. A scrutiny of the software requirement engineering process. *Procedia Technology*, 25:405–410, 12 2016.

Allenoush Hayrapetian and Rajeev Raje. Empirically analyzing and evaluating security features in software requirements. pages 1–11, 02 2018.

Anne Hess, Philipp Diebold, and Norbert Seyff. Understanding information needs of agile teams to improve requirements communication (special issue edited by nan niu and daniel mendez). *Journal of Industrial Information Integration*, 14, 05 2018.

Mahmood Hosseini, Alimohammad Shahri, Keith Phalp, Jacqui Taylor, Fabiano Dalpiaz, and Raian Ali. Configuring crowdsourcing for requirements elicitation. volume 2015, 05 2015.

Prasara Jakkaew and Tew Hongthong. Requirements elicitation to develop mobile application for elderly. In *2017 International Conference on Digital Arts, Media and Technology (ICDAMT)*, pages 464–467, 2017.

Aleksander Jarzbowicz and Natalia Sitko. Agile requirements prioritization in practice: Results of an industrial survey. *Procedia Computer Science*, 176:3446–3455, 2020. Knowledge-Based and Intelligent Information Engineering Systems: Proceedings of the 24th International Conference KES2020.

Tracy-Lee Kotze and Hanlie Smuts. Model for knowledge capturing during system requirements elicitation in a high reliability organization: a case study. 09 2018.

Pekka Mäkiaho, Timo Poranen, and Zheying Zhang. Requirements management in students' software development projects. In *Proceedings of the 18th International Conference on Computer Systems and Technologies*, CompSysTech'17, page 203210, New York, NY, USA, 2017. Association for Computing Machinery.

Pedro Marques, Murilo Silva, Camila Gusmão, Diego Castro, and Marcelo Schots. Requirements engineering out of the classroom: Anticipating challenges experienced in practice. In *2020 IEEE 32nd Conference on Software Engineering Education and Training (CSEE T)*, pages 1–9, 2020.

Juliana Medeiros, Alexandre Vasconcelos, Carla Silva, and Miguel Goulão. Quality of software requirements specification in agile projects: A cross-case analysis of six companies. *Journal of Systems and Software*, 142, 04 2018.

Juliana Medeiros, Alexandre Vasconcelos, Carla Silva, and Miguel Goulão. Requirements specification for developers in agile projects: Evaluation by two industrial case studies. *Information and Software Technology*, 117:106194, 09 2019.

Jorge Melegati, Alfredo Goldman, Fabio Kon, and Xiaofeng Wang. A model of requirements engineering in software startups. *Information and Software Technology*, 109, 02 2019.

Daniel Méndez Fernández and Stefan Wagner. Naming the pain in requirements engineering: Design of a global family of surveys and first results from germany. pages 183–194, 04 2013.

Dorgival Netto, Carla Silva, and João Araújo. Identifying how the brazilian software industry specifies legal requirements - a qualitative study. 09 2019.

Sara Nilsson, Erik Sundin, and Mattias Lindahl. Integrated product service offerings challenges in setting requirements. *Journal of Cleaner Production*, 201, 08 2018.

Martha Palmer, Daniel Gildea, and Paul Kingsbury. The Proposition Bank: An annotated corpus of semantic roles. *Computational Linguistics*, 31(1):71–106, 2005.

Michael C. Panis. An analysis of requirements-related problems that occurred in an organization using a mature requirements engineering process. In *2020 IEEE 28th International Requirements Engineering Conference (RE)*, pages 291–299, 2020.

Andrés Paz and Ghizlane El-Boussaidi. Building a software requirements specification and design for an avionics system: An experience report. 04 2018.

Wichai Puarungroj, Narong Boonsirisumpun, Suchada Phromkhot, and Nattiya Puarungroj. Dealing with change in software development: A challenge for requirements engineering. pages 1–5, 12 2018.

Syed Raza. Managing ethical requirements elicitation of complex socio-technical systems with critical systems thinking: A case of course-timetabling project. *Technology in Society*, 66:101626, 08 2021.

Olga Sankowski, Jan Küchenhof, Florian Dambietz, Marc Zuefle, Anne Wallisch, Dieter Krause, and Kristin Paetzold. Challenges in early phase of product family development processes. *Procedia CIRP*, 100:840–845, 01 2021.

Alberto Silva. Linguistic patterns and linguistic styles for requirements specification (i): An application case with the rigorous rsl/business-level language. pages 1–27, 07 2017.

Kostas Stylidis, Monica Rossi, Casper Wickman, and Rikard Söderberg. The communication strategies and customer's requirements definition at the early design stages: An empirical study on italian luxury automotive brands. volume 50, 08 2016.

Saurabh Tiwari and Atul Gupta. A systematic literature review of use case specifications research. *Inf. Softw. Technol.*, 67(C):128158, nov 2015.

Krzysztof Wnuk. Involving relevant stakeholders into the decision process about software components. In *2017 IEEE International Conference on Software Architecture Workshops (ICSAW)*, pages 129–132, 2017.