Linguagens e ferramentas utilizadas na especificação formal de software: uma revisão de escopo

Caio Alberto Nunes Marques[1]*, Maria Adriana Ferreira da Silva[2] , José Lucas Santana de Andrade[3], Alysson Filgueira Milanez[4]

[1] caio.marques883@gmail.com, [2] maria.silva78326@alunos.ufersa.edu.br, [3] jose.andrade82065@alunos.ufersa.edu.br, [4] alysson.milanez@ufersa.edu.br. Universidade Federal Rural do Semi-Árido (UFERSA), Pau dos Ferros, Rio Grande do Norte, Brasil
* autor correspondente

Resumo
Este artigo apresenta uma revisão de escopo com o objetivo de identificar as principais linguagens e ferramentas utilizadas na especificação formal de software. A revisão foi realizada em quatro bases de dados (ACM Digital Library, IEEE Xplore, Scopus e Web of Science), utilizando uma *string* de busca específica e critérios de inclusão e exclusão bem definidos. Após as etapas definidas no protocolo e pesquisa, foram retornados 736 trabalhos na primeira fase, 194 na segunda, 74 na terceira e 17 selecionados para a última fase, onde foram analisados detalhadamente e tiveram suas informações extraídas, verificando-se todos os critérios pré-definidos. Os resultados permitiram identificar as principais linguagens e ferramentas empregadas na especificação formal de software. Revelou-se que a utilização de linguagens formais, ferramentas de verificação e simulação são eficazes para garantir a correta implementação dos requisitos de segurança e para detectar erros em especificações. Além disso, foram discutidos os desafios enfrentados pelas abordagens tradicionais de especificação formal, como a necessidade de habilidades especializadas e a complexidade dos modelos resultantes. Este artigo destaca a importância de tornar as ferramentas de especificação formal mais acessíveis e simplificadas, visando aumentar a adoção e eficácia dessas abordagens.
Palavras-chave: especificação de software; métodos formais; modelagem formal; validação de software; verificação formal.

Languages and tools used in formal software specification: a scoping review

Abstract
This article presents a scoping review to identify the main languages and tools used in formal software specifications. The review was conducted across four databases (ACM Digital Library, IEEE Xplore, Scopus, and Web of Science), employing a specific search string and well-defined inclusion and exclusion criteria. Following the stages defined in the research protocol, 736 papers were retrieved in the first phase, 194 in the second, 74 in the third, and 17 were selected for the final phase, where they were analyzed in detail and had their information extracted, ensuring all predefined criteria were met. The results identified the primary languages and tools employed in formal software specification. It was revealed that using formal languages, verification tools, and simulation effectively ensures the correct implementation of security requirements and detects specification errors. Moreover, the challenges traditional formal specification approaches faced, such as the need for specialized skills and the complexity of resulting models, were discussed. This article highlights the importance of making formal specification tools more accessible and simplified to enhance their adoption and effectiveness.
Keywords*: formal methods; formal modeling; formal verification; software specification; software validation.*

1 Introdução

A Engenharia de Software é uma área da Ciência da Computação que se dedica a concepção, desenvolvimento e manutenção de softwares para computadores, utilizando métodos e técnicas de gestão de projetos (Sommerville, 2019). Um dos principais benefícios dessa disciplina é a viabilidade

de desenvolver e projetar sistemas cada vez mais robustos, de modo que atendam às necessidades das organizações e dos usuários (Pressman; Maxim, 2021).

Entretanto, a Engenharia de Software enfrenta desafios, tais como: melhorar a confiabilidade do programa, minimizar a complexidade dos sistemas, simplificar a manutenção, garantir a segurança, equilibrar prazos, orçamentos e qualidade, além de assegurar a correta implementação dos sistemas de software (Rezende, 2006; Sommerville, 2019).

Uma das abordagens utilizadas para auxiliar no desenvolvimento de software confiável são os métodos formais (Sommerville, 2019). Esses métodos consistem em um conjunto de técnicas que utilizam o formalismo matemático para apoiar as etapas de especificação, desenvolvimento e verificação de sistemas (Ferrari; Beek, 2022; Sommerville, 2019). Essas técnicas permitem identificar problemas e falhas precocemente no processo de desenvolvimento de software, reduzindo custos e tempo de desenvolvimento (Roggenbach *et al*., 2021; Sommerville, 2019). São frequentemente aplicados em sistemas críticos, como os de aviação, médicos e de defesa, além de serem úteis em sistemas que exigem alta confiabilidade, como os financeiros e de segurança (Scaico *et al*., 2007).

Os métodos formais são essenciais no desenvolvimento de software, pois auxiliam na melhor compreensão dos requisitos e na identificação precoce de inconsistências, utilizando uma linguagem formalmente definida. Isso também contribui para a redução de custos com testes, uma vez que as verificações ocorrem tanto no programa quanto na especificação (Roggenbach *et al*., 2021; Sommerville, 2019). No entanto, a aplicação desses métodos pode exigir habilidades especializadas em matemática e lógica (Sommerville, 2019). Assim, uma das formas de facilitar a aplicação dos métodos formais é por meio do uso de linguagens e ferramentas específicas, que garantem a correção e consistência necessárias.

Em sistemas críticos, as linguagens B e Z são fundamentais, pois auxiliam na definição de requisitos formais essenciais para sistemas precisos e complexos (Lecomte, 2023; Scaico *et al*., 2007). A utilização dessas linguagens facilita a formalização e verificação de especificações, assegurando a conformidade e a correção dos sistemas. Além disso, ferramentas de modelagem são cruciais para a aplicação eficiente desses métodos em sistemas críticos (Lecomte, 2023).

A UML e suas extensões, como a OCL, têm dominado a modelagem de software desde 1997 (Hilken; Hamann, 2020). A UML é uma ferramenta prática que permite modelar diferentes fases de um sistema, desde a concepção inicial até a codificação final, sendo aplicável a qualquer sistema, especialmente em diagramas orientados a objetos. Sua capacidade de simplificar conceitos complexos, adaptar-se às mudanças tecnológicas e promover uma comunicação eficiente é fundamental (Bernhardt *et al*., 2023; Xavier, *et al*., 2019).

Essas linguagens são amplamente utilizadas para verificação formal tanto na academia quanto na indústria. Existe uma crescente demanda na indústria por profissionais com domínio avançado em UML. Em resposta a essa demanda, as universidades estão adequando seus currículos às mudanças no mercado de trabalho, resultando em mais tempo dedicado ao ensino de UML e outras disciplinas relacionadas (Bernhardt *et al*., 2023).

A literatura já apresenta estudos sobre a utilização da UML. Um exemplo é o estudo de Xavier *et al*. (2019), que fornece uma perspectiva atualizada sobre como a UML é utilizada na indústria de software, especialmente em relação aos métodos ágeis. Os autores analisam os resultados de um questionário aplicado a diversas empresas de desenvolvimento de software em Minas Gerais, com o objetivo de investigar a utilização da UML nos processos ágeis de desenvolvimento.

Considerando o contexto apresentado, o objetivo deste trabalho é realizar uma Revisão de Escopo (RE) para investigar as soluções que utilizam linguagens e ferramentas destinadas à especificação formal de software na literatura. Para isso, foi definida a seguinte questão de pesquisa: "Quais são as principais linguagens e ferramentas utilizadas na especificação formal de software disponíveis na literatura?"

Esta revisão resultou na identificação das principais linguagens e ferramentas utilizadas na especificação formal de software, como as linguagens Z, Alloy, Event-B, TLA+, SysML, Redes de Petri Coloridas e as ferramentas SAL, Coq e SPL, destacando seu potencial para melhorar a confiabilidade e segurança de sistemas críticos, como os de segurança e Veículos Aéreos Não Tripulados (VANTs).

Os resultados desta pesquisa podem ter impactos significativos tanto na academia quanto na indústria. Na academia, a identificação das principais linguagens e ferramentas de verificação pode ajudar a melhorar e ampliar os estudos nessa área. Para a indústria, os insights obtidos destacam a eficácia da utilização de linguagens formais, ferramentas de verificação e simulação na garantia da implementação correta de requisitos de segurança e na detecção de erros em especificações.

O restante deste trabalho está organizado da seguinte forma: a Seção 2 apresenta uma revisão dos trabalhos relacionados; a Seção 3 descreve a metodologia utilizada; a Seção 4 expõe os resultados e discussões obtidas a partir da aplicação do protocolo. Por fim, a Seção 5 apresenta as conclusões e as sugestões de trabalhos futuros.

2 Trabalhos relacionados

Nesta seção, são apresentados alguns trabalhos relacionados que realizaram revisões no contexto de métodos formais de software. Rizvi, Khan e Asthana (2013) conduziram uma revisão sistemática da literatura que enfatiza o papel dos métodos formais na melhoria dos requisitos de software. O estudo descreve como esses métodos podem melhorar a clareza, precisão e consistência dos requisitos. O trabalho destaca os benefícios dos métodos formais para a qualidade dos requisitos, a necessidade de sua adoção contínua e sua implementação desde as fases iniciais do ciclo de desenvolvimento.

O estudo de Atoum *et al*. (2021) apresenta os resultados de uma revisão sistemática da literatura sobre a validação de requisitos. A pesquisa analisa as técnicas de validação mais adotadas e discute como melhorar a qualidade do software final por meio dessas técnicas. São abordadas a avaliação de qualidade, as principais questões e desafios na validação de requisitos de software, fornecendo *insights* valiosos para a melhoria da qualidade do software.

Rajabli, *et al*. (2020) realizaram uma revisão que abrange estudos relevantes sobre a segurança crítica de sistemas, com foco em técnicas de teste e verificação, incluindo a análise de critérios de cobertura. Os autores destacam a necessidade de padronização na terminologia utilizada nos processos de verificação e validação, a fim de acelerar o progresso na pesquisa sobre veículos autônomos.

O trabalho de Hofer-Schmitz e Stojanović (2020) revisa a verificação formal de protocolos de Internet das Coisas (IoT), ressaltando a importância dessa abordagem na identificação de vulnerabilidades. Os autores categorizam os métodos de verificação em quatro campos principais: verificações funcionais, verificações de propriedades de segurança, esquemas aprimorados e verificações de implementação. O estudo também fornece uma visão geral das propriedades de segurança, ferramentas e verificadores de modelo utilizados, abordando também os desafios e questões em aberto no campo da IoT.

Este artigo apresenta uma revisão de escopo com o objetivo de identificar as principais linguagens e ferramentas de especificação formal, examinando sua aplicação e destacando as limitações encontradas nos estudos analisados. O Quadro 1 apresenta uma comparação entre os trabalhos relacionados e o presente estudo.

Quadro 1 – Comparativo de trabalhos

Trabalhos	Foco da revisão	Principais aspectos
Rizvi, Khan e Asthana (2013)	Papel dos métodos formais na melhoria de requisitos de software	Necessidade de adoção contínua e implementação desde as fases iniciais do ciclo de desenvolvimento
Atoum *et al.* (2021)	Validação de requisitos de software	Técnicas de validação de requisitos, avaliação de qualidade, principais questões e desafios
Rajabli *et al.* (2020)	Verificação e validação (V&V) de software para carros autônomos seguros	Requisitos e desafios na V&V, oportunidades e questões abertas, transferência de tecnologia
Hofer-Schmitz e Stojanović (2020)	Verificação formal de protocolos utilizados na Internet das Coisas (IoT)	Importância da verificação formal, classificação das aplicações, propriedades de segurança, ferramentas e verificadores de modelo, desafios e questões em aberto
Este artigo	Identificação das principais linguagens e ferramentas de especificação formal	Aplicação das linguagens e ferramentas e limitações encontradas nos estudos analisados

Fonte: dados da pesquisa

3 Método da pesquisa

O presente estudo adotou uma Revisão de Escopo (RE) como metodologia, com o objetivo de identificar os principais conceitos de uma determinada área de conhecimento por meio da análise de trabalhos já publicados (Arksey; O'Malley, 2005). Para a realização desta RE, os procedimentos metodológicos originalmente propostos por Arksey e O'Malley (2005) foram adaptados, incorporando ajustes sugeridos por Levac, Colquhoun e O'Brien (2010) e Peters *et al*. (2024).

Levac, Colquhoun e O'Brien (2010) propuseram diversas melhorias ao trabalho de Arksey e O'Malley (2005), recomendando maior clareza na definição das questões de pesquisa e na utilização de critérios para a seleção dos estudos. Além disso, os critérios de inclusão e exclusão foram aprimorados conforme as recomendações de Peters *et al.* (2024). Esses ajustes foram devidamente incorporados ao presente estudo.

Espera-se que a presente RE possa contribuir significativamente para o avanço do conhecimento na aplicação de métodos formais, identificando as principais linguagens e ferramentas utilizadas na especificação de software.

Para atender aos objetivos desta pesquisa, a RE seguiu as seguintes etapas:

1. Formulação da questão de pesquisa e objetivos;
2. Identificação dos estudos relevantes para a temática abordada;
3. Seleção de estudos de acordo com critérios predefinidos;
4. Avaliação, extração e apresentação dos resultados.

A Figura 1 ilustra o processo metodológico estabelecido para esta pesquisa.

Figura 1 – Número de trabalhos retornados nas quatro etapas

Fonte: dados da pesquisa

Inicialmente, foi definida a seguinte Questão de Pesquisa (QPE): "Quais são as principais linguagens e ferramentas utilizadas na especificação formal de software disponíveis na literatura?". Para responder a essa questão, foram elaboradas três Questões Específicas (QE):

- QE1: quais são as principais linguagens e ferramentas utilizadas?
- QE2: como essas linguagens e ferramentas são utilizadas na especificação formal de software?
- QE3: quais são as limitações identificadas?

O escopo desta pesquisa concentrou-se em estudos que abordam a utilização de linguagens e ferramentas na especificação formal de software. Assim, foi possível identificar quais linguagens e ferramentas são utilizadas para a especificação de software, como são aplicadas e quais são as suas limitações.

Esta RE buscou encontrar estudos publicados entre 1º de janeiro de 2018 e 18 de março de 2023, escritos em inglês e português. A escolha do intervalo de publicações foi baseada nos últimos cinco anos até março de 2023, data em que a pesquisa foi realizada. A busca foi conduzida em quatro bases de dados relevantes na área: ACM Digital Library, IEEE Xplore, Scopus e Web of Science. As fontes foram selecionadas considerando a disponibilidade de acesso ao conteúdo dos artigos e a quantidade significativa de publicações relacionadas à área de tecnologia.

Foram identificadas as seguintes palavras-chave para alcançar artigos pertinentes à temática da pesquisa: especificação formal (*formal specification*), verificação formal (*formal verification*),

modelagem formal (*formal modeling*), análise formal (*formal analysis*), prova formal (*formal proof*), linguagens formais (*formal languages*), linguagens de especificação (*specification languages*), ferramentas de especificação (*specification tools*), ferramentas de verificação (*verification tools*). As palavras-chave foram selecionadas com base em sua relevância para os objetivos da pesquisa.

Com base nesses termos, a seguinte *string* de busca foi definida: "especificação formal" OR "*formal specification*" OR "verificação formal" OR "*formal verification*" OR "modelagem formal" OR "*formal modeling*" OR "análise formal" OR "*formal analysis*" OR "prova formal" OR "*formal proof*" AND "linguagens formais" OR "*formal languages*" OR "linguagens de especificação" OR "*specification languages*" AND "ferramentas de especificação" OR "*specification tools*" OR "ferramentas de verificação" OR "*verification tools*".

Para selecionar os estudos de maior relevância, foram definidos Critérios de Inclusão (CI) e Critérios de Exclusão (CE):

- **Critérios de Inclusão (CI):**
  - CI1: estudos com foco na pesquisa;
  - CI2: estudos publicados entre 01 de janeiro de 2018 e 18 de março de 2023;
  - CI3: estudos publicados nos idiomas Inglês e Português;
  - CI4: estudos que apresentam linguagem ou ferramenta de especificação.

- **Critérios de Exclusão (CE):**
  - CE1: estudos que não tenham foco na pesquisa;
  - CE2: estudos que não apresentam informações suficientes para responder a nenhuma das questões de pesquisa;
  - CE3: estudos repetidos nas fontes de busca;
  - CE4: estudos que não sejam de revistas, conferências ou jornais;
  - CE5: estudos que não possibilitem download gratuito do arquivo completo.

4 Resultados e discussões

Esta seção apresenta os resultados e discussões obtidos com a aplicação do protocolo.

4.1 Seleção de trabalhos

A seleção dos trabalhos foi realizada em pares, com a seleção inicial dos artigos sendo realizada de forma independente pelos autores. Cada autor avaliou os estudos identificados, segundo os critérios de seleção estabelecidos anteriormente. Após a revisão independente, os resultados foram comparados e discutidos para resolver as inconsistências, garantindo, dessa forma, a inclusão apenas dos estudos que atendessem aos critérios estabelecidos.

O processo de seleção e escolha dos trabalhos foi dividido em quatro etapas. Uma descrição detalhada das etapas pode ser encontrada on-line[1]. Etapa I consistiu na pré-seleção dos artigos a partir da aplicação das *strings* de busca nas fontes de dados. Nessa etapa, fez-se a verificação dos critérios de inclusão CI2 e CI3, resultando em 736 trabalhos selecionados.

Na Etapa II, foi realizada uma análise detalhada dos títulos, palavras-chave e resumos dos trabalhos selecionados na Etapa I, visando verificar o atendimento aos critérios de inclusão CI1 e os critérios de exclusão CE1, CE3 e CE4. Os trabalhos que atenderam aos critérios de inclusão foram adicionados à Lista de Selecionados (LS1), enquanto os que foram excluídos foram organizados na Lista de Removidos (LR1). Ao final da Etapa II, 194 trabalhos foram selecionados para a Etapa III.

A Etapa III consistiu na leitura da introdução, metodologia e conclusão dos trabalhos contidos na LS1, levando em consideração os critérios de inclusão CI1 e CI4, bem como os critérios de exclusão CE1, CE2 e CE5. Com base nesses critérios, os estudos que não se enquadraram nas diretrizes predefinidas foram excluídos e adicionados em uma segunda lista de removidos (LR2), enquanto os trabalhos que atenderam aos critérios foram incluídos em uma segunda lista de trabalhos selecionados (LS2). Nessa etapa, buscou-se identificar as principais características dos trabalhos, os

[1] Dados da pesquisa. Disponível em: https://periodicos.ifpb.edu.br/index.php/principia/article/view/8404.

principais conceitos abordados e a metodologia utilizada. Ao final da Etapa III, 74 trabalhos foram selecionados para a próxima fase.

A Etapa IV foi conduzida com base nos trabalhos contidos na LS2. Inicialmente, foram realizadas leituras integrais de todos os trabalhos e uma análise detalhada de todos os critérios de inclusão e exclusão predefinidos. Nessa etapa, fez-se a identificação do propósito do trabalho (linguagem, ferramenta, abordagem ou outro), dos recursos utilizados (linguagens, ferramentas, bibliotecas e outros), dos procedimentos adotados para validar as soluções e das limitações identificadas. Por fim, 17 trabalhos atenderam a todos os critérios estabelecidos e foram selecionados para responder às Questões Específicas (QEs).

A Figura 2 apresenta o quantitativo de trabalhos retornados em cada fonte de pesquisa, para cada etapa do processo de avaliação. Ao final do processo de seleção dos trabalhos, partiu-se para a sua análise.

Figura 2 – Número de trabalhos retornados nas quatro etapas

Fonte: dados da pesquisa

Dos 74 trabalhos que passaram para a Etapa IV, todos foram analisados com base em todos os critérios de inclusão e exclusão. Desses, apenas 17, das fontes de busca IEEE e Web of Science, foram selecionados para responder às Questões Específicas (QEs). Os artigos foram selecionados de acordo com uma aplicação rigorosa dos critérios de inclusão e exclusão, visando selecionar apenas aqueles que abordam diretamente o tema da pesquisa. A seguir, são apresentadas as respostas obtidas para as QEs com base nesses 17 trabalhos. No Quadro 2, encontram-se detalhadas as respostas identificadas a partir das Questões Específicas.

Quadro 2 – Sumarização das respostas a partir das QEs2

Autores	**QE1**	**QE2**	**QE3**
Rouland *et al.*, 2019	Linguagem de especificação formal: Z Linguagem de modelagem: Alloy Ferramentas: *Model checkers*, SPL	Linguagens Z e Alloy para definir requisitos de segurança. *Model checkers* para verificar a consistência e corretude dos requisitos. SLP para auxiliar na especificação dos requisitos de segurança	Necessidade de conhecimento prévio em engenharia de requisitos e métodos formais. Limitação na validação devido a ser um único estudo de caso
Apvrille, Saqui-Sannes e	Linguagem de modelagem: SysML	Linguagem SysML e ferramenta TTool para	Estudo de caso baseado em uma situação educacional

Vingerhoeds, 2020	Ferramentas: TTool, Eclipse com plugin Papyrus, ROS	modelagem e simulação do sistema de controle de um VANT. Eclipse como ambiente de desenvolvimento. *Plugin Papyrus* e biblioteca ROS para comunicação	específica pode não ser generalizável
Brida *et al*., 2021	Linguagem de modelagem: Alloy Linguagem de programação: Java Ferramentas: bibliotecas do Alloy, *framework* Bounded Exhaustive Search	Linguagem Alloy para desenvolver a ferramenta. *Framework* Bounded Exhaustive Search para busca por reparações em especificações defeituosas de modelos Alloy, utilizando a linguagem Java e bibliotecas relacionadas	Falta de testes da ferramenta proposta em um contexto real
Zhang, Salcic e Malik, 2019	Linguagem de modelagem: SystemJ, Redes de Petri Coloridas Ferramentas: CPN Tools, XSLT	SystemJ e CPNs para modelar e analisar sistemas GALS. XSLT para traduzir o modelo SystemJ em Redes de Petri Coloridas	Problema da explosão do espaço de estados na modelagem. Necessidade de intervenção manual na tradução de modelos SystemJ em Redes de Petri Coloridas
Siregar e Abriani, 2019	Linguagem de especificação formal: Z Ferramentas: SAL, Z2SAL	A linguagem Z para especificação formal. A ferramenta SAL e verificador de modelo SAL para verificar a exatidão do modelo do sistema	Métodos formais podem ser demorados e exigir alto nível de conhecimento. Nem todos os sistemas de software podem ser adequados para métodos formais
Khan *et al*., 2019	Linguagens: não especificada Ferramentas: Assistente de prova Coq, verificadores de tipo	Assistente de prova Coq para desenvolver um modelo formal de segurança Android. Verificadores de tipo para verificar a solidez do modelo formal	A formalização da segurança do Android através de linguagem não é compatível com ferramentas de verificação automática.
Halchin *et al*., 2019	Linguagem de especificação formal: B Linguagem de programação: HLL Ferramentas: Isabelle/HOL, B2HLL	Linguagem B para especificação formal. Linguagem HLL para verificação de propriedades de segurança. Kit de ferramentas Isabelle/HOL para descrever a semântica formal. B2HLL para produzir modelos HLL com entrada B	Estudo de caso identificou uma lacuna no modelo HLL produzido
Lukács e Bartha, 2022	Linguagem: linguagem natural Linguagem de modelagem: UML, SysML Ferramentas: Ferramentas MBSE	Linguagem natural estruturada, diagramas UML, método tabular e máquinas de estado UML para especificação de requisitos, interfaces, configuração e comportamento	Especificação e planejamento simultâneos propensos a erros. Participação limitada dos usuários no processo de desenvolvimento

Zhu *et al*., 2021	Linguagem de programação: Solidity Linguagem de especificação formal: Event-B Ferramentas: ANTLR	Solidity para escrever contratos inteligentes. Event-B para criar modelos abstratos dos contratos Solidity. ANTLR para implementar um tradutor do modelo abstrato Event-B a partir do contrato Solidity	Dificuldade em lidar com sistemas complicados e explosão de estado na verificação do modelo
Cardenas *et al*., 2022	Linguagem de modelagem: UML, OCL Ferramentas: USE	UML e OCL para especificação formal. USE como ambiente de especificação para analisar e validar os modelos UML	Ausência de melhores práticas padronizadas para proteger sistemas IoT
Liu e Liu, 2019	Linguagem de modelagem: Redes de Petri Coloridas Linguagem de especificação formal: ASK CTL Ferramentas: Ferramentas CPN	Redes de Petri Coloridas para verificar contratos inteligentes. ASK CTL para verificar vulnerabilidades latentes em contratos inteligentes	Muitas vulnerabilidades em contratos inteligentes devido à complexidade do ambiente distribuído
Younes *et al*., 2019	Linguagem de modelagem: BPMN, BPMN2 Linguagem de especificação formal: Event-B Ferramentas: Rodin	BPMN2 para especificação. Event-B para transformar especificações BPMN em especificações formais. Rodin para verificação	Limitações da linguagem BPMN2, como falta de semântica formal precisa e sistema de prova
Zhao *et al*., 2019	Linguagem de especificação formal: CSP Ferramentas: FDR, Storm	CSP para modelar os comportamentos. FDR para verificar o modelo. Storm para processamento.	Dificuldade em lidar com a complexidade do fluxo de trabalho do Storm
Latif, Rehman e Zafar, 2019	Linguagem de modelagem: UML, Linguagem de especificação formal: TLA+ Ferramentas: Recurso TLC	TLA+ para validar e verificar propriedades do sistema, TCL para verificação do modelo.	Não especificado
Westphal, 2020	Linguagens: Linguagem de descrição de software representativa Ferramentas: Não especificada	Métodos formais para análise formal de requisitos, modelos de software ou programas. Engenharia de Requisitos e Design usando formalismos adequados	Dificuldade em medir se os objetivos de aprendizagem foram alcançados. Requer tempo e esforço significativos dos alunos e instrutores
Shigyo e Katayama, 2020	Linguagem de Especificação formal: VDM++, Linguagem de programação: Python Ferramentas: VDMTools, VDMJ, NLTK, Stanford Parser	VDM++ para descrever especificações. VDMTools e VDMJ para verificação das especificações	Limitações na geração de tipos de dados e tratamento de especificações complexas de linguagem natural
Ratiu, Gario e Schoenhaar, 2019	Linguagem de especificação formal: DSLs	MPS para definir e estender as DSLs. SysML/AADL para compilar modelos em	Desafios das abordagens tradicionais de especificação formal, como a necessidade

	Ferramentas: MPS, SysML/AADL	modelos de nível inferior	de habilidades especializadas

Fonte: dados da pesquisa

4.2 QE1: principais linguagens e ferramentas utilizadas

No trabalho de Rouland *et al*. (2019) foi apresentada uma abordagem que utiliza especificações formais para modelar os requisitos de segurança e verificações automáticas para assegurar que o sistema implemente esses requisitos de maneira correta e segura. Para validar a abordagem, foi realizado um estudo de caso visando avaliar a aplicabilidade e eficácia da abordagem em sistemas reais. Foram utilizadas as linguagens Z e Alloy, bem como ferramentas de apoio à verificação formal, como *model checkers* e bibliotecas voltadas para a especificação de segurança, como a Security Property Library (SPL).

Apvrille, Saqui-Sannes e Vingerhoeds (2020) apresentaram uma metodologia para o uso do SysML (System Modeling Language) e da TTool no design de Veículos Aéreos Não Tripulados (VANTs). Para validar a metodologia, foi realizado um estudo de caso educacional. Foram utilizadas a linguagem SysML e a ferramenta TTool para a modelagem e simulação do sistema. Adicionalmente, foram utilizados o ambiente de desenvolvimento Eclipse, o *plugin* Papyrus e a biblioteca de comunicação *Robot Operating System* (ROS).

No trabalho de Brida *et al*. (2021) foi descrita uma ferramenta de busca para a identificação e correção de erros em especificações Alloy. A ferramenta foi validada por meio de testes e experimentos computacionais, demonstrando a eficácia da técnica proposta na identificação de reparos de especificações Alloy. A ferramenta foi desenvolvida utilizando a linguagem Alloy, o *framework* Bounded Exhaustive Search, a linguagem Java e bibliotecas relacionadas.

Em Zhang, Salcic e Malik (2019) o foco está no uso das linguagens e ferramentas SystemJ e Redes de Petri Coloridas (CPN). SystemJ é uma linguagem de programação síncrona usada para modelagem e programação de sistemas GALS (*Globally Asynchronous Locally Synchronous*). As CPNs são uma linguagem formal de modelagem utilizada para análise e verificação de sistemas concorrentes. Os autores também utilizam uma folha de estilo XSLT para realizar a tradução de arquivos SystemJ para CPN ML e utilizam a ferramenta CPN Tools para modelagem, simulação e análise de CPNs.

Em Siregar e Abriani (2019) as principais linguagens e ferramentas utilizadas são a linguagem formal Z para modelar o sistema especialista baseado em regras e o verificador de modelo SAL (*Symbolic Analysis Laboratory*) para verificar os teoremas adicionados à especificação. Z2SAL é usado para traduzir a especificação Z em uma especificação SAL, que pode então ser verificada pelos verificadores de modelo SAL.

No trabalho de Khan *et al*. (2019) a principal ferramenta utilizada é o assistente de prova Coq, empregado para desenvolver um modelo matemático de segurança para Android. A linguagem formal para representar aplicações Android e o sistema de tipos que impõe as melhores práticas são discutidos, embora não se especifique a linguagem de programação usada para o desenvolvimento real de aplicações Android.

Em Halchin *et al*. (2019) a abordagem proposta utiliza a linguagem formal B e a linguagem de modelagem HLL. O kit de ferramentas Isabelle/HOL e sua biblioteca de táticas são empregados para formalizar a relação de tradução de ambas as linguagens e para garantir a correção do processo de tradução. Além disso, a ferramenta B2HLL é usada para produzir modelos HLL a partir dos modelos de entrada B.

No trabalho de Lukács e Bartha (2022) a metodologia proposta para sistemas de intertravamento ferroviário envolve o uso de métodos formais, como lógica matemática e matemática discreta, para especificar e verificar a funcionalidade do sistema. Além disso, são aplicadas várias técnicas e ferramentas relacionadas à Engenharia de Sistemas Baseada em Modelos (do inglês *Model-based Systems Engineering* – MBSE), como *Unified Modeling Language* (UML) e *Systems Modeling Language* (SysML). Contudo, não se especifica uma linguagem ou ferramenta principal utilizada na metodologia proposta.

Em Zhu *et al*. (2021) a principal linguagem utilizada é Solidity, uma linguagem de alto nível orientada a contratos, projetada para a Máquina Virtual Ethereum. O método proposto envolve a tradução de contratos Solidity para modelos Event-B usando um tradutor implementado com ANTLR. O modelo abstrato Event-B pode então ser refinado manualmente de acordo com propriedades específicas.

No trabalho de Cardenas *et al*. (2022) as principais linguagens e ferramentas utilizadas são UML, a Linguagem de Restrição de Objetos (do inglês *Object Constraint Language* – OCL) e o Ambiente de Especificação baseado em UML (USE). Os autores desenvolveram um *plugin* para USE que integra um *framework* para validar modelos de software UML.

Em Liu e Liu (2019) são mencionadas as Redes de Petri Coloridas (CPN) e ferramentas CPN para modelar contratos inteligentes e atacantes, e ASK-CTL e Ferramentas de Espaço de Estado para verificação de modelos. O artigo também cita vários outros estudos que utilizaram diferentes linguagens e ferramentas para analisar sistemas *blockchain* e contratos inteligentes, como o *framework* BIP, Solidity e Vyper.

Em Younes *et al*. (2019) as principais linguagens e ferramentas utilizadas são BPMN2, Event-B e meta-modelagem. O artigo propõe uma abordagem orientada a modelos que transforma uma especificação BPMN em uma especificação formal usando a notação Event-B. A abordagem é baseada em meta-modelagem, que envolve a definição de um conjunto de regras para mapear elementos BPMN para construções do Event-B. O processo de transformação é implementado através da plataforma Rodin, uma ferramenta para desenvolvimento e verificação de modelos Event-B.

Em Zhao *et al*. (2019) são utilizados principalmente a estrutura de computação de processamento de fluxo distribuído Storm, a linguagem formal CSP (*Communicating Sequential Processes*) e a ferramenta de verificação e refinamento de software FDR (*Failures-Divergences Refinement*) para modelar e verificar os comportamentos de comunicação no fluxo de trabalho do Storm.

No trabalho de Latif, Rehman e Zafar (2019) as principais linguagens e ferramentas utilizadas são UML, modelo baseado em autômatos e a linguagem formal TLA+. A linguagem TLA+ é utilizada para definir propriedades do sistema e verificá-lo através do TLC, uma técnica de verificação de modelo. O comportamento do sistema é capturado através da caixa de ferramentas TLA+ e validado usando o recurso TLC disponível na mesma.

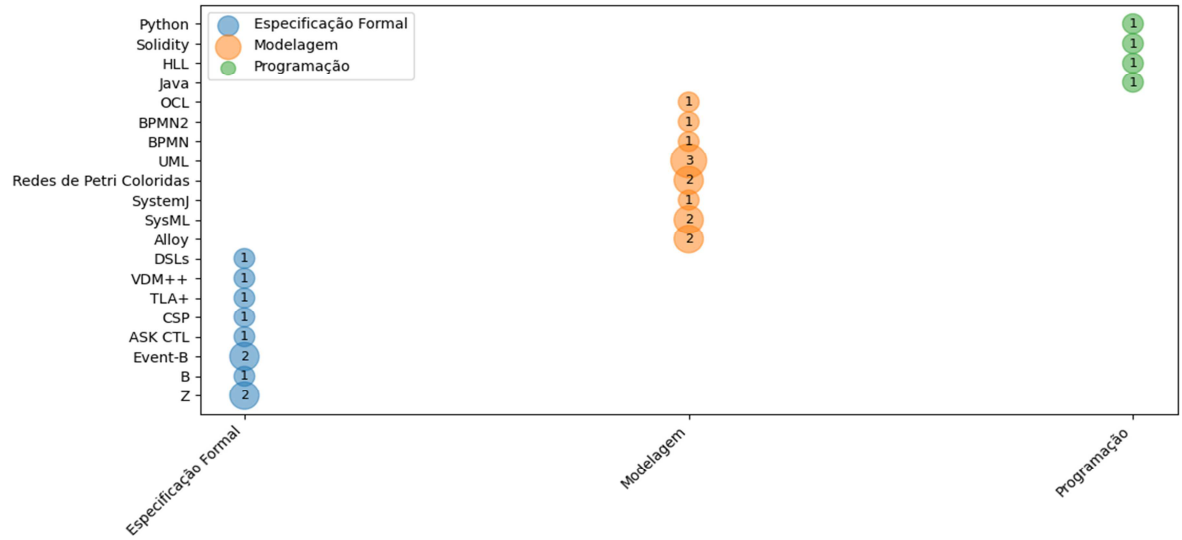
Westphal (2020) sugeriu selecionar algumas linguagens de descrição de software representativas, discutidas em livros didáticos, e complementar a discussão com uma cobertura formal de algumas semânticas formais dessas linguagens. Também propõem o uso de técnicas e ferramentas para análise de artefatos para propriedades.

No trabalho de Shigyo e Katayama (2020) a principal linguagem utilizada é VDM++, uma linguagem de extensão orientada a objetos baseada em VDM-SL. Foram utilizadas também a linguagem de programação Python e a biblioteca NLTK para processamento de linguagem natural, além do Stanford Parser para análise de dependências. VDMTools e VDMJ foram usados como ferramentas de suporte para VDM.

Em Ratiu, Gario e Schoenhaar (2019) o principal ambiente de trabalho utilizado é o Meta Programming System (MPS), que suporta todos os aspectos da definição de linguagens de domínio específico (DSLs), como sintaxe abstrata, editores avançados, restrições sensíveis ao contexto, geradores de código e analisadores. O artigo também menciona SysML/AADL como ferramentas de modelagem que podem ser utilizadas para compilar modelos em modelos de nível inferior para consumo por ferramentas de verificação.

A Figura 3 ilustra a distribuição de linguagens em três categorias distintas: especificação formal, modelagem e programação. Cada bolha representa uma linguagem, cuja posição horizontal é determinada pela frequência de uso e o tamanho, pela relevância nos trabalhos analisados. As linguagens de especificação formal Z, Event-B, UML, Redes de Petri Coloridas, SysML e Alloy foram as mais frequentemente mencionadas nos trabalhos analisados.

Figura 3 – Principais linguagens utilizadas por categoria



Fonte: dados da pesquisa

4.3 QE2: Utilização de linguagens e ferramentas na especificação formal de software

Em Rouland *et al*. (2019) as linguagens Z e Alloy foram empregadas para definir precisamente os requisitos de segurança. Os *model checkers* foram utilizados para verificar a consistência e a correção dos requisitos especificados. A biblioteca de especificação de segurança, Security Property Library (SPL), auxiliou na especificação dos requisitos de segurança.

No trabalho de Apvrille, Saqui-Sannes e Vingerhoeds (2020) a linguagem SysML e a ferramenta TTool foram utilizadas para a modelagem e simulação do sistema de controle de um VANT. O ambiente de desenvolvimento foi o Eclipse, utilizando o *plugin* Papyrus e a biblioteca ROS para a comunicação. Em Brida *et al*. (2021) a ferramenta foi desenvolvida utilizando a linguagem Alloy e o *framework* Bounded Exhaustive Search foi utilizado para buscar reparações em especificações defeituosas de modelos Alloy. O processo de busca foi executado em um *cluster* de computadores utilizando a linguagem Java e as bibliotecas relacionadas à Alloy e ao Bounded Exhaustive Search.

No estudo de Zhang, Salcic e Malik (2019) a combinação de SystemJ com Redes de Petri Coloridas (CPNs) foi usada para modelar e analisar sistemas GALS. SystemJ foi utilizado para modelar o comportamento do sistema, enquanto as CPNs foram empregadas para verificar formalmente o modelo. A tradução de modelos SystemJ para Redes de Petri Coloridas permite o uso de uma ampla gama de ferramentas de análise e verificação, incluindo análise de espaço de estados e verificação de modelos. Estas técnicas podem verificar várias propriedades do sistema, como ausência de *deadlock*, vivacidade e segurança. O uso de métodos formais na especificação e verificação de sistemas GALS possibilita a detecção e correção de erros precocemente no processo de projeto, resultando em sistemas mais confiáveis e seguros.

Em Siregar e Abriani (2019) a linguagem Z foi empregada para declarar sistemas e construir especificações de forma matemática, produzindo especificações formais e livres de ambiguidades, permitindo a realização de análises mecânicas. A linguagem SAL foi utilizada para especificar e modelar sistemas, com o verificador de modelo SAL sendo usado para garantir a exatidão do modelo do sistema. Utilizando essas linguagens e ferramentas formais, os autores conseguiram verificar o sistema especialista e assegurar sua correção.

No trabalho de Khan *et al*. (2019) o assistente de prova Coq foi utilizado para desenvolver um modelo formal de segurança Android baseado em linguagem, permitindo o raciocínio formal sobre o sistema de permissão do Android usando uma ferramenta de prova de teoremas baseada em computador. O modelo formal foi construído utilizando a lógica do provador de teoremas, e a solidez da técnica de segurança baseada em linguagem foi verificada mecanicamente. O modelo formal e a prova de solidez são legíveis por máquina e sua exatidão pode ser verificada usando prova Coq e verificadores de tipo, permitindo verificação e raciocínio baseados em computador, essenciais para a especificação formal de software.

Em Halchin *et al*. (2019) a linguagem formal B foi usada para especificação formal de software, enquanto a linguagem de modelagem HLL foi empregada na verificação de propriedades de segurança. O kit de ferramentas Isabelle/HOL e sua biblioteca de táticas descreveram a semântica formal e formalizaram a relação de tradução entre as duas linguagens. Os modelos desenvolvidos no Isabelle/HOL garantiram a correção do processo de tradução.

Em Halchin *et al*. (2019) a linguagem formal B foi utilizada para especificação formal de software, enquanto a linguagem de modelagem HLL foi empregada como base para verificação de propriedades de segurança. O kit de ferramentas Isabelle/HOL e sua biblioteca de táticas foram utilizados para descrever a semântica formal de ambas as linguagens e formalizar a relação de tradução entre elas. Os modelos Isabelle/HOL desenvolvidos garantem a correção do processo de tradução.

No trabalho de Lukács e Bartha (2022) a metodologia proposta para sistemas de intertravamento ferroviário baseia-se em quatro pilares relacionados a um componente: requisitos, interfaces, configuração e comportamento. A metodologia emprega linguagem natural estruturada para especificação de requisitos, diagramas de componentes UML para definições de interface, um método tabular específico para especificação de parâmetros e máquinas de estado UML para descrição dos aspectos comportamentais. Essas técnicas e ferramentas são usadas para apoiar a especificação, o planejamento e a verificação do sistema, modelando sistemas complexos e identificando erros nas fases iniciais do ciclo de vida. Os modelos executáveis podem ser testados no início de sua produção e, utilizando ferramentas analíticas adequadas, a verificação do modelo pode ser aplicada para permitir o controle dos modelos.

No estudo de Zhu *et al*. (2021) o Solidity foi utilizado para escrever contratos inteligentes, com os termos do acordo entre cliente e fornecedor sendo codificados diretamente. O Event-B foi empregado para criar modelos abstratos dos contratos Solidity, que podem ser refinados para criar modelos mais detalhados e verificados quanto à segurança. ANTLR foi utilizado para implementar um tradutor que gera automaticamente o modelo abstrato Event-B a partir do contrato Solidity. Técnicas formais de verificação, como prova de teoremas, verificação de modelo e execução simbólica, foram aplicadas ao modelo Event-B para garantir que o contrato atende aos requisitos de segurança e proteção.

Em Cardenas *et al*. (2022) a UML e a OCL foram utilizadas para especificar formalmente o sistema de software e seus requisitos estruturais. O Ambiente de Especificação baseado em UML (USE) foi utilizado para analisar e validar os modelos UML. A ferramenta *plugin* desenvolvida pelos autores integra um *framework* para validar modelos UML e facilita a análise do comportamento do sistema.

No trabalho de Liu e Liu (2019) foi sugerido um método de verificação baseado em Redes de Petri Coloridas (CPN) para verificar contratos inteligentes no sistema *blockchain*. Os modelos CPN foram executados por simulação passo a passo para validar sua correção funcional. A tecnologia de verificação de modelo baseada em lógica de temporização de ramificação ASK-CTL nas ferramentas CPN foi utilizada para detectar vulnerabilidades latentes em contratos inteligentes. CPN e ASK-CTL foram utilizados para especificar formalmente e verificar o comportamento de contratos inteligentes de forma rigorosa e sistemática.

Em Younes *et al*. (2019) foi proposta uma abordagem orientada a modelos que transforma uma especificação BPMN em uma especificação formal utilizando a notação Event-B. A abordagem é baseada em meta-modelagem, que envolve a definição de um conjunto de regras para mapear elementos BPMN para construções Event-B. A especificação formal resultante pode ser verificada utilizando a plataforma Rodin. Ao utilizar um método formal como o Event-B, os autores visam superar as limitações do BPMN2, que carece de uma semântica formal precisa e de um sistema de prova para validação de especificações. O uso de Event-B pode permitir a detecção de erros, design aprimorado, maior confiança na precisão, identificação precoce de defeitos e melhor compreensão dos sistemas de software.

No trabalho de Zhao *et al*. (2019) foram utilizados Storm, CSP e FDR para formalizar o fluxo de trabalho do Storm e verificar se ele satisfaz propriedades de consistência sequencial e ausência de deadlock. O CSP foi utilizado para modelar os comportamentos de comunicação no fluxo de trabalho do Storm, e o FDR foi utilizado para verificar se o modelo satisfaz as propriedades desejadas. Esta

abordagem é um exemplo de especificação formal de software, que envolve o uso de modelos matemáticos e métodos formais para especificar e verificar sistemas de software.

Em Latif, Rehman e Zafar (2019) a linguagem TLA+ foi utilizada para validar e verificar propriedades do sistema usando técnicas formais. A especificação foi analisada por meio do TLC, uma ferramenta de verificação de modelo que verifica a exatidão da especificação.

No trabalho de Westphal, (2020) os autores sugerem que métodos formais podem ser utilizados para a análise formal de requisitos, modelos de software ou programas. O conteúdo da área de Engenharia de Requisitos geralmente apresenta uma visão de diferentes linguagens de especificação, incluindo linguagens formais de descrição de software. Ao introduzir métodos formais de forma abrangente, é possível criar uma progressão gradual de dificuldade, começando com formalismos que possuem sintaxe abstrata mais simples e avançando gradualmente para formalismos mais complexos, como Statecharts e verificação de programas. Na área temática de Design, formalismos como diagramas de classes e objetos UML, OCL e variantes dos Statecharts de Harel podem ser usados para modelagem estrutural e comportamental de software. A seleção de sub linguagens adequadas permite a apresentação desses formalismos com uma semântica formal adequada.

No trabalho de Westphal (2020) os autores sugerem que métodos formais podem ser utilizados para a análise formal de requisitos, modelos de software ou programas. A área de Engenharia de Requisitos geralmente apresenta uma visão de diferentes linguagens de especificação, incluindo linguagens formais de descrição de software. Ao introduzir métodos formais de forma abrangente, é possível criar uma progressão gradual de dificuldade, começando com formalismos que possuem sintaxe abstrata mais simples e avançando gradualmente para formalismos mais complexos, como Statecharts e verificação de programas. Na área de projetos, formalismos como diagramas de classes e objetos UML, OCL e variantes dos Statecharts de Harel podem ser utilizados para modelagem estrutural e comportamental de software. A seleção de sub-linguagens adequadas permite a apresentação desses formalismos com uma semântica formal apropriada.

Em Shigyo e Katayama (2020) o VDM++ foi utilizado para descrever especificações, que podem ser verificadas por teoremas matemáticos e verificações mecânicas. VDMTools e VDMJ facilitaram a verificação das especificações. No trabalho de Ratiu, Gario e Schoenhaar (2019), o MPS foi utilizado para definir e estender Linguagens de Domínio Específico (DSLs) para especificação formal de software. Essas DSLs fornecem construções abstratas de domínio específico de forma incremental, permitindo a modelagem de uma ampla categoria de sistemas. SysML/AADL podem ser usados para compilar modelos em modelos de nível inferior para consumo por ferramentas de verificação. Utilizando essas linguagens e ferramentas, os engenheiros podem trabalhar com suas linguagens de modelagem e fluxos de trabalho usuais e, em seguida, compilar esses modelos em modelos de nível inferior que podem ser consumidos por ferramentas de verificação.

4.4 QE3: Limitações identificadas

Em relação às limitações identificadas, Rouland *et al*. (2019) destacam a necessidade de conhecimento prévio em engenharia de requisitos e métodos formais. Além disso, a validação foi limitada a um único estudo de caso. Apvrille, Saqui-Sannes e Vingerhoeds (2020) apresentam um estudo de caso baseado em uma situação educacional específica, o que pode não produzir resultados satisfatórios para outras situações educacionais. Brida *et al*. (2021) sugerem a realização de um estudo de caso para testar a ferramenta proposta em um contexto real.

Zhang, Salcic e Malik (2019) mencionam algumas limitações e desafios no uso de redes de Petri coloridas para modelagem formal e análise de sistemas SystemJ GALS. Um dos desafios é o problema da explosão do espaço de estados na modelagem de sistemas complexos. Os autores sugerem o uso de técnicas de abstração para mitigar esse problema. Além disso, a tradução de modelos SystemJ GALS para redes de Petri coloridas pode não ser totalmente automatizada e pode exigir intervenção manual. A abordagem de tradução apresentada cobre apenas o subconjunto síncrono do SystemJ e pode não ser aplicável a toda a linguagem.

Siregar e Abriani (2019) não mencionam explicitamente limitações durante o desenvolvimento e verificação do sistema especialista baseado em regras usando Z e SAL. Contudo, métodos formais, em geral, podem ser demorados e exigir um elevado nível de conhecimento para serem utilizados de

forma eficaz. Além disso, esses métodos podem não ser adequados para todos os tipos de sistemas de software e sua utilização pode nem sempre ser rentável.

Khan *et al*. (2019) não mencionam explicitamente limitações da abordagem usada para formalizar a segurança do Android baseada em linguagem. No entanto, a linguagem formal para representar aplicações Android e o sistema de tipos que impõem as melhores práticas não são legíveis por ferramentas de verificação assistidas por computador e, portanto, não são adequados para raciocínio mecânico e verificação automática. Para permitir a verificação e o raciocínio baseados em computador, a linguagem e o verificador de tipos devem ser definidos em um provador de teoremas.

Halchin *et al*. (2019) não mencionam explicitamente limitações da abordagem proposta. Contudo, um estudo de caso relacionado a um sistema de software ferroviário identificou um requisito no nível do sistema que não foi atendido pelo modelo HLL produzido. O mecanismo de prova revelou um contraexemplo, e o cenário correspondente foi analisado para compreender o risco relacionado a essa violação de propriedade.

Lukács e Bartha (2022) não fornecem uma lista abrangente das limitações identificadas na metodologia proposta para sistemas de intertravamento ferroviário. Entretanto, mencionam que a especificação e o planejamento simultâneos são propensos a erros e que é raro alcançar uma implementação adequada do sistema em apenas uma etapa de design. O documento observa que práticas tradicionais de desenvolvimento muitas vezes envolvem usuários que têm uma participação limitada no processo e que comunicam formalmente seus requisitos em alguma forma textual, não dispostos a modificá-los formalmente durante o desenvolvimento devido ao processo de aprovação e custos. A metodologia proposta visa abordar essas limitações, envolvendo mais os usuários no processo de desenvolvimento e fornecendo uma abordagem estruturada e rigorosa para especificação e verificação.

Zhu *et al*. (2021) mencionam que escrever contratos inteligentes seguros e protegidos pode ser extremamente difícil devido a diversas lógicas de negócios, vulnerabilidades e limitações da plataforma. O método proposto para traduzir contratos Solidity em modelos Event-B pode ajudar a resolver algumas dessas vulnerabilidades, mas não é uma solução completa. O artigo também observa que a verificação do modelo pode enfrentar uma explosão de estado quando um sistema se torna complicado. Além disso, o método proposto requer conhecimentos em métodos formais e pode não ser acessível a todos os desenvolvedores de contratos inteligentes.

Cardenas *et al*. (2022) não mencionam explicitamente limitações identificadas. Contudo, afirmam que ainda não existem melhores práticas padronizadas para proteger sistemas da Internet das Coisas (IoT), destacando a necessidade de melhorias nesta área. Liu e Liu (2019) informam que há muitas vulnerabilidades na maioria dos contratos inteligentes devido à complexidade do ambiente distribuído e às limitações da linguagem de programação.

Younes *et al*. (2019) discutem diversas limitações da linguagem BPMN2, incluindo a ausência de uma semântica formal precisa das notações utilizadas, levando a ambiguidades, e a falta de um sistema de prova que valide uma especificação BPMN2. Essas limitações dificultam a garantia de que os fluxos de trabalho BPMN2 estejam livres de inconsistências e erros. O artigo propõe uma abordagem baseada em modelo que utiliza o Event-B para superar essas limitações e fornecer uma especificação formal que pode ser verificada usando a plataforma Rodin.

Zhao *et al*. (2019) não mencionam explicitamente limitações da abordagem usada para modelar e verificar Storm. Contudo, os autores mencionam que pesquisas anteriores sobre Storm focaram na análise ou otimização, mas não houve trabalho formal de modelagem. O artigo visa preencher essa lacuna aplicando CSP para modelar formalmente os comportamentos de comunicação entre processos simultâneos no fluxo de trabalho do Storm e usando FDR para estabelecer um modelo CSPM e verificar propriedades significativas do Storm.

Latif, Rehman e Zafar (2019) não mencionam explicitamente limitações da abordagem usada. Westphal (2020) menciona algumas limitações da abordagem proposta. Uma das principais é a dificuldade de medir se os objetivos de aprendizagem foram alcançados, pois seria necessário observar os alunos em situações reais. Como aproximação, o exame do curso cobre a maioria das linguagens de descrição de software formalmente introduzidas e, com a mudança nas tarefas do exame, os autores observam resultados de exame estáveis ao longo dos testes, tomando isso como uma indicação de que não perderam completamente seus objetivos de ensino. Outra limitação é que a abordagem requer uma

quantidade significativa de tempo e esforço tanto dos alunos quanto dos instrutores, pois envolve o aprendizado e a aplicação de métodos e ferramentas formais. Por fim, os autores reconhecem que a abordagem proposta pode não ser adequada para todos os cursos de graduação em engenharia de software, pois depende dos objetivos, do conteúdo e do público do curso.

Shigyo e Katayama (2020) identificaram diversas limitações em sua abordagem. Uma delas é que a abordagem proposta gera apenas variáveis e tipos reais na especificação VDM++, não podendo gerar outros tipos de dados. Outra limitação é que a abordagem considera apenas substantivos na especificação da linguagem natural e não considera outras classes gramaticais. Além disso, a abordagem pode não ser capaz de lidar com especificações complexas de linguagem natural que incluem múltiplas sentenças ou estruturas gramaticais complexas. Finalmente, a abordagem pode não ser capaz de lidar com todos os casos de ambiguidade na especificação da linguagem natural.

Ratiu, Gario e Schoenhaar (2019) não mencionam explicitamente limitações da abordagem FASTEN. Contudo, mencionam desafios que as abordagens tradicionais de especificação formal enfrentam, como a necessidade de habilidades especializadas de especificação formal, a perda de conceitos de alto nível durante o processo de codificação e os modelos resultantes sendo tipicamente detalhados e difíceis de revisar e alterar. O artigo também menciona o desafio da compreensibilidade das especificações formais no contexto da manutenção e evolução de sistemas.

4.5 Ameaças à validade

Considerando a relevância dos estudos secundários para fundamentar decisões baseadas em evidências, é de suma importância identificar as ameaças à validade dessas pesquisas. Para a pesquisa em questão, foram identificadas as seguintes ameaças: possíveis vieses na seleção dos estudos, mudanças nas tecnologias e práticas de engenharia de software que podem ter influenciado a seleção dos trabalhos, afetando a precisão das conclusões. Foram utilizadas diversas bases de dados acadêmicas e científicas, como IEEE Xplore, ACM Digital Library, Scopus e Web of Science, para garantir uma cobertura ampla e diversificada de estudos relevantes. Isso ajuda a evitar o viés de seleção que poderia surgir se apenas uma ou poucas fontes fossem utilizadas. Além disso, foram realizadas buscas em outros idiomas quando possível, garantindo que estudos relevantes de diferentes regiões e comunidades científicas fossem incluídos.

5 Considerações finais e trabalhos futuros

Neste estudo, realizou-se uma revisão sistemática com o objetivo de identificar as principais linguagens e ferramentas utilizadas na especificação formal de software. Para tal, foi definido um protocolo para a condução da revisão. Conclui-se que as abordagens formais para modelagem, verificação e correção de sistemas de segurança e Veículos Aéreos Não Tripulados (VANTs) apresentam um potencial significativo para melhorar a confiabilidade e a segurança desses sistemas críticos. A utilização de linguagens formais, ferramentas de verificação e simulação, e técnicas de busca demonstrou-se eficaz na garantia da correta implementação de requisitos de segurança e na identificação de erros em especificações, contribuindo para a diminuição de vulnerabilidades e a prevenção de falhas em sistemas complexos.

Entretanto, é importante reconhecer as limitações identificadas, tais como a exigência de conhecimentos específicos em métodos formais, o tempo e esforço necessários para aplicar essas abordagens e a dificuldade de lidar com especificações complexas de linguagem natural. Essas limitações destacam a necessidade contínua de desenvolver ferramentas mais acessíveis e simplificar os processos de aplicação de métodos formais, a fim de tornar essas abordagens mais amplamente adotadas e eficazes. Apesar das limitações, os resultados apresentados nos trabalhos analisados evidenciam o impacto positivo das abordagens formais na garantia da segurança e confiabilidade de sistemas críticos. As ferramentas e linguagens utilizadas, como Z, Alloy, SysML e Event-B, demonstraram sua utilidade e versatilidade em diferentes contextos, indicando um potencial de aplicação em uma variedade de sistemas e domínios.

Portanto, é fundamental continuar investindo em pesquisas e desenvolvimentos nessa área, buscando superar as limitações identificadas e aprimorar ainda mais as abordagens formais para modelagem, verificação e correção de sistemas de segurança e VANTs. Ao fazer isso, estará se contribuindo para a construção de sistemas mais seguros, confiáveis e resilientes, essenciais para o

avanço da tecnologia e o bem-estar da sociedade. Em pesquisas futuras, pretende-se explorar outras fontes de busca, tais como Elsevier e SpringerLink. Além disso, planeja-se ampliar o período de estudo para identificar outras linguagens e ferramentas, bem como incluir trabalhos em outros idiomas. Essa abordagem permitirá uma visão mais completa das práticas em especificação formal de software, possibilitando uma compreensão mais ampla e aprofundada do tema.

Financiamento

Esta pesquisa não recebeu financiamento externo.

Conflito de interesses

Os autores declaram não haver conflito de interesses.

Referências

APVRILLE, L.; SAQUI-SANNES, P.; VINGERHOEDS, R. An educational case study of using SysML and TTool for unmanned aerial vehicles design. **IEEE Journal on Miniaturization for Air and Space Systems**, v. 1, n. 2, p. 117-129, 2020. DOI: https://doi.org/10.1109/JMASS.2020.3013325.

ARKSEY, H.; O'MALLEY, L. Scoping studies: towards a methodological framework. **International Journal of Social Research Methodology**, v. 8, n. 1, p. 19-32, 2005. DOI: https://doi.org/10.1080/1364557032000119616.

ATOUM, I.; BAKLIZI, M. K.; ALSMADI, I.; OTOOM, A. A.; ALHERSH, T.; ABABNEH, J.; ALMALKI, J.; ALSHAHRANI, S. M. Challenges of software requirements quality assurance and validation: a systematic literature review. **IEEE Access**, v. 9, p. 137613-137634, 2021. DOI: https://doi.org/10.1109/ACCESS.2021.3117989.

BERNHARDT, G. H.; PFIZ, F. S.; HO, S. L.; RODRIGUES, J. P. L.; ZANINI, E. O. A utilização da linguagem de modelagem unificada (UML) na engenharia de software: uma avaliação da adoção e relevância. *In*: ENCONTRO CIENTÍFICO CULTURAL INTERINSTITUCIONAL, 21., Cascavel, 2023. **Anais [...].** Cascavel: FAG, 2023. Disponível em: https://www4.fag.edu.br/anais-2023/Anais-2023-48.pdf. Acesso em: 12 ago. 2024.

BRIDA, S. G.; REGIS, G.; ZHENG, G.; BAGHERI, H.; NGUYEN, T.-V.; AGUIRRE, N.; FRIAS, M. Artifact of bounded exhaustive search of alloy specification repairs. *In*: 2021 IEEE/ACM INTERNATIONAL CONFERENCE ON SOFTWARE ENGINEERING: COMPANION PROCEEDINGS (ICSE-COMPANION), 43., 2021, Madrid. **Proceedings […].** Madrid: IEEE, p. 209-210, 2021. DOI: https://doi.org/10.1109/ICSE-Companion52605.2021.00093.

CARDENAS, H.; ZIMMERMAN, R.; VIESCA, A. R.; AL LAIL, M.; PEREZ, A. J. Formal UML-based modeling and analysis for securing location-based IoT applications. *In*: IEEE INTERNATIONAL CONFERENCE ON MOBILE AD HOC AND SMART SYSTEMS (MASS), 19., 2022, Denver. **Proceedings […].** Denver: IEEE, 2022, p. 722-723. DOI: https://doi.org/10.1109/MASS56207.2022.00109.

FERRARI, A.; BEEK, M. H. T. Formal methods in railways: a systematic mapping study. **ACM Computing Surveys**, v. 55, n. 4, p. 1-37, 2023. DOI: https://doi.org/10.1145/3520480.

HALCHIN, A.; AIT-AMEUR, Y.; SINGH, N. K.; FELIACHI, A.; ORDIONI, J. Certified embedding of B models in an integrated verification framework. *In*: 2019 INTERNATIONAL SYMPOSIUM ON THEORETICAL ASPECTS OF SOFTWARE ENGINEERING (TASE), 2019, Guilin. **Proceedings […]**. Guilin: IEEE, 019. p. 168-175. DOI: https://doi.org/10.1109/TASE.2019.000-4.

HILKEN, F.; HAMANN, L. History of the USE tool 20 Years of UML/OCL modeling made in Germany. JOT – The Journal of Object Technology, v. 19, n. 3, p. 3:1-13, 2020. DOI: http://dx.doi.org/10.5381/jot.2020.19.3.a20.

KHAN, W.; KAMRAN, M.; AHMAD, A.; KHAN, F. A.; DERHAB, A. Formal analysis of language-based Android security using theorem proving approach. **IEEE Access**, v. 7, p. 16550-16560, 2019. DOI: https://doi.org/10.1109/ACCESS.2019.2895261.

HOFER-SCHMITZ, K.; STOJANOVIĆ, B. Towards formal verification of IoT protocols: a review. **Computer Networks**, v. 174, 107233, 2020. DOI: https://doi.org/10.1016/j.comnet.2020.107233.

LATIF, S.; REHMAN, A.; ZAFAR, N. A. NFA based formal modeling of smart parking system using TLA+. *In*: IEEE INTERNATIONAL CONFERENCE ON INFORMATION SCIENCE AND COMMUNICATION TECHNOLOGY (ICISCT), 2019, Karachi. **Proceedings […]**. Karachi: IEEE, 2019. DOI: https://doi.org/10.1109/CISCT.2019.8777445.

LECOMTE, T. Teaching and training in formalisation with B. *In*: DUBOIS, C.; SAN PIETRO, P. (Eds.). **Formal Methods Teaching**. FMTea 2023. Lecture Notes in Computer Science, v. 13962. Cham: Springer, 2023. DOI: https://doi.org/10.1007/978-3-031-27534-0_6.

LEVAC, D.; COLQUHOUN, H.; O'BRIEN, K. K. Scoping studies: advancing the methodology. **Implementation Science**, v. 5, p. 1-9, 2010. DOI: https://doi.org/10.1186/1748-5908-5-69.

LIU, Z.; LIU, J. Formal verification of blockchain smart contract based on colored petri net models. *In*: IEEE ANNUAL COMPUTER SOFTWARE AND APPLICATIONS CONFERENCE (COMPSAC), 43., 2019, Milwaukee. **Proceedings […]**. Milwaukee: IEEE, 2019. p. 555-560. DOI: https://doi.org/10.1109/COMPSAC.2019.10265.

LUKÁCS, G.; BARTHA, T. Conception of a formal model-based methodology to support railway engineers in the specification and verification of interlocking systems. *In*: IEEE INTERNATIONAL SYMPOSIUM ON APPLIED COMPUTATIONAL INTELLIGENCE AND INFORMATICS (SACI), 16., 2022, Timisoara. **Proceedings […]**.Timisoara: IEEE, 2022, p. 283-288. DOI: https://doi.org/10.1109/SACI55618.2022.9919532.

PETERS, M. D. J.; GODFREY, C.; MCINERNEY, P.; MUNN, Z.; TRICCO, A. C.; KHALIL, H. Scoping reviews. *In*: AROMATARIS E, LOCKWOOD, C.; PORRITT, K.; PILLA, B.; JORDAN, Z. (Eds.). **JBI Manual for Evidence Synthesis**. JBI, 2024. DOI: https://doi.org/10.46658/JBIMES-24-09.

PRESSMAN, R. S.; MAXIM, B. R. **Engenharia de software**. 9. ed. Porto Alegre: McGraw Hill, 2021.

RAJABLI, N.; FLAMMINI, F.; NARDONE, R.; VITTORINI, V. Software verification and validation of safe autonomous cars: a systematic literature review. **IEEE Access**, v. 9, p. 4797-4819, 2021. DOI: https://doi.org/10.1109/ACCESS.2020.3048047.

RATIU, D.; GARIO, M.; SCHOENHAAR, H. FASTEN: an open extensible framework to experiment with formal specification approaches. *In*: IEEE/ACM INTERNATIONAL CONFERENCE ON FORMAL METHODS IN SOFTWARE ENGINEERING (FORMALISE), 7., Montreal, 2019. **Proceedings […]**. 2019, p. 41-50. DOI: https://doi.org/10.1109/FormaliSE.2019.00013.

REZENDE, D. A. **Engenharia de software e sistemas de informação**. Rio de Janeiro: Brasport, 2006.

RIZVI, S. W. A.; KHAN, R. A.; ASTHANA, R. Improving software requirements through formal methods. **International Journal of Information and Computation Technology**, v. 3, n. 11, p. 1217-1223, 2013. Disponível em: https://www.ripublication.com/irph/ijict_spl/13_ijictv3n11spl.pdf. Acesso em: 12 ago. 2024.

ROGGENBACH, M.; CERONE, A.; SCHLINGLOFF, B.-H.; SCHNEIDER, G.; SHAIKH, S. A. **Formal methods for software Engineering**: languages, methods, applications domains. Springer, 2021. DOI: https://doi.org/10.1007/978-3-030-38800-3.

ROULAND, Q.; HAMID, B.; BODEVEIX, J.-P.; FILALI, M. A formal methods approach to security requirements specification and verification. *In*: 2019 INTERNATIONAL CONFERENCE ON ENGINEERING OF COMPLEX COMPUTER SYSTEMS (ICECCS), 24., 2019, Guangzhou. **Proceedings […]**. Guangzhou: IEEE, 2019. p. 236-241. DOI: https://doi.org/10.1109/ICECCS.2019.00033.

SCAICO, A. **Aplicação de métodos formais no projeto de interfaces para sistemas industriais críticos**. Dissertação (Mestrado em Engenharia Elétrica) – Universidade Federal de Campina Grande, Campina Grande, 2007. Disponível em: http://dspace.sti.ufcg.edu.br:8080/jspui/handle/riufcg/6270. Acesso em: 13 ago. 2024.

SHIGYO, Y.; KATAYAMA, T. Proposal of an approach to generate vdm++ specifications from natural language specification by machine learning. *In*: IEEE Global Conference on Consumer Electronics (GCCE), 9., 2020, Kobe. **Proceedings […]**. Kobe: IEEE, 2020, p. 292-296. DOI: https://doi.org/10.1109/GCCE50665.2020.9292047.

SIREGAR, M. U.; ABRIANI, S. Verification of a rule-based expert system by using SAL model checker. *In*: IEEE International Conference on Informatics and Computational Sciences (ICICoS), 3., 2019, Semarang. **Proceedings […]**. Semarang: IEEE, 2019. DOI: https://doi.org/10.1109/ICICoS48119.2019.8982426.

SOMMERVILLE, I. **Engenharia de software**. 10. ed. São Paulo: Pearson, 2019.

WESTPHAL, B. On complementing an undergraduate software engineering course with formal methods. *In*: IEEE CONFERENCE ON SOFTWARE ENGINEERING EDUCATION AND TRAINING (CSEE&T), 32., 2020, Munich. **Proceedings […]**. Munich: IEEE, 2020, p. 1-10. DOI: https://doi.org/10.1109/CSEET49119.2020.9206234.

YOUNES, A. B.; HLAOUI, Y. B. D.; AYED, L. B.; BESSIFI, M. From BPMN2 to event B: a specification and verification approach of workflow applications. *In*: IEEE ANNUAL COMPUTER SOFTWARE AND APPLICATIONS CONFERENCE (COMPSAC), 43., 2019, Milwaukee. **Proceedings […]**. Milwaukee: IEEE, 2019, p. 561-566. DOI: https://doi.org/10.1109/COMPSAC.2019.10266.

XAVIER, A.; MARTINS, F.; PIMENTEL, R.; CARVALHO, D. Aplicação da UML no contexto das metodologias ágeis. *In*: ENCONTRO NACIONAL DE COMPUTAÇÃO DOS INSTITUTOS FEDERAIS, 6., 2019, Belém. **Anais [...]**. Belém: SBC. DOI: https://doi.org/10.5753/encompif.2019.6353.

ZHAO, H.; ZHU, H.; FANG, Y.; XIAO, L. Modeling and verifying storm using CSP. *In*: IEEE INTERNATIONAL SYMPOSIUM ON HIGH ASSURANCE SYSTEMS ENGINEERING (HASE), 19., 2019, Hangzhou. **Proceedings […]**.Hangzhou: IEEE, 2019. p. 192-199. DOI: https://doi.org/10.1109/HASE.2019.00037.

ZHANG, W.; SALCIC, Z.; MALIK, A. Towards formal modeling and analysis of SystemJ GALS systems using coloured petri nets. *In*: IEEE INTERNATIONAL CONFERENCE ON INDUSTRIAL INFORMATICS (INDIN), 17., 2019, Helsinki. **Proceedings [...]**. Helsinki: IEEE, 2019, p. 152-159. DOI: https://doi.org/10.1109/INDIN41052.2019.8972025.

ZHU, J.; HU, K.; FILALI, M.; BODEVEIX, J.-P.; TALPIN, J.-P.; CAO, H. formal simulation and verification of solidity contracts in event-B. *In*: 2021 IEEE ANNUAL COMPUTERS, SOFTWARE, AND APPLICATIONS CONFERENCE (COMPSAC), 45., 2021, Madrid. **Proceedings [...]**. Madrid: IEEE, 2021. p. 1309-1314. DOI: https://doi.org/10.1109/COMPSAC51774.2021.00183.