UNIVERSIDADE FEDERAL RURAL DO SEMI-ÁRIDO
PRÓ-REITORIA DE GRADUAÇÃO
DEPARTAMENTO DE CIÊNCIAS EXATAS E NATURAIS
CURSO INTERDISCIPLINAR EM TECNOLOGIA DA INFORMAÇÃO
TRABALHO DE CONCLUSÃO DE CURSO (2024)

Linguagens e Ferramentas utilizadas na Especificação Formal de Software: Uma Revisão de Escopo

Languages and Tools used in Formal Software Specification: A Review of Scope

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

Resumo
Este artigo apresenta uma Revisão de Escopo (RE) que visa identificar quais são as principais linguagens e ferramentas utilizadas na especificação formal de software. A RE foi realizada em quatro fontes de busca (*ACM Digital Library*, *IEEE Xplore*, *Scopus* e *Web of Science*), utilizando uma *string* de busca e critérios de inclusão e exclusão. Após a realização das etapas definidas no protocolo, 736 trabalhos foram retornados na primeira etapa, 194 na segunda etapa, 74 na terceira e 17 foram selecionados para a etapa IV, onde foram analisados e tiveram suas informações extraídas, realizando a verificação de todos os critérios pré-definidos. Mediante os resultados apresentados, foi possível identificar as principais linguagens e ferramentas utilizadas na especificação formal de software. Os resultados revelaram que a utilização de linguagens formais, ferramentas de verificação e simulação são eficazes na garantia da implementação correta de requisitos de segurança e na detecção de erros em especificações. Além disso, são discutidos desafios enfrentados pelas abordagens tradicionais de especificação formal, como a necessidade de habilidades especializadas e a complexidade dos modelos resultantes. Este artigo ressalta a importância de tornar as ferramentas de especificação formal mais acessíveis e simplificadas, a fim de ampliar a adoção e eficácia dessas abordagens.
Palavras-chave: Especificação de software; Especificação formal; Verificação formal.

Abstract
This article presents a Scoping Review (SR) aimed at identifying the main languages and tools used in formal software specification. The SR was conducted in four search sources (ACM Digital Library, IEEE Xplore, Scopus, and Web of Science), using a search string and inclusion and exclusion criteria. After completing the protocol-defined stages, 736 works were retrieved in the first stage, 194 in the second stage, 74 in the third, and 17 were selected for stage IV, where they were analyzed and their information extracted, performing verification of all predefined criteria. Through the presented results, it was possible to identify the main languages and tools used in formal software specification. The results revealed that the use of formal languages, verification tools, and simulation are effective in ensuring the correct implementation of security requirements and in error detection in specifications. Additionally, challenges faced by traditional formal specification approaches are discussed, such as the need for specialized skills and the complexity of resulting models. This article highlights the importance of making formal specification tools more accessible and simplified to broaden the adoption and effectiveness of these approaches.
Keywords: *formal specification; formal verification; Software specification.*

Disponibilidade: DOI: http://dx.doi.org/10.18265/2447-9187a2024id8404

1 Introdução

A Engenharia de Software é uma área da Ciência da Computação que lida com a concepção,

desenvolvimento, e manutenção de softwares para computadores, com uso de métodos e técnicas de gestão de projetos (SOMMERVILLE, 2011). Um dos principais benefícios da engenharia de software,

é a viabilidade de desenvolver sistemas cada vez mais robustos, que dão suporte às necessidades das organizações e dos usuários (PUCPR, 2020).

Todavia, ainda há alguns desafios e limitações que englobam essa área. Um dos empecilhos mais predominantes é a demanda de se manter atualizado em relação às novas tecnologias e direcionamentos do mercado, assim como garantir a correta implementação de um sistema de software que ofereça maior confiabilidade no processo de desenvolvimento de softwares mais robustos (EDUCAÇÃO, 2022).

Uma das abordagens utilizadas para auxiliar no processo de desenvolvimento de software confiável são os métodos formais (SOMMERVILLE, 2011). Os métodos formais são um conjunto de técnicas que utilizam o formalismo matemático para auxiliar nas etapas de especificação, desenvolvimento e verificação de sistemas (PRESSMAN, 2003). Essas técnicas podem ajudar a identificar problemas e falhas precocemente no processo de desenvolvimento de software, reduzindo custos e tempo de desenvolvimento (FLORES, 2016). São frequentemente usados em sistemas críticos, como sistemas de aviação, sistemas médicos e sistemas de defesa. Eles também são úteis para sistemas que precisam ser altamente confiáveis, como sistemas financeiros e de segurança (SCAICO *et al*., 2007).

Para Pressman (2003) os métodos formais são importantes no desenvolvimento de software porque permitem a criação de software mais confiável, correto, de melhor qualidade, mais seguro e com redução de custos. No entanto, a aplicação de métodos formais pode exigir habilidades especializadas em matemática e lógica (SOMMERVILLE, 2011). Assim, umas das formas de auxiliar o processo de aplicação dos métodos formais é utilizar linguagens e ferramentas específicas para a sua aplicação, de modo a garantir a corretude e consistência necessárias.

Considerando o contexto apresentado, o propósito deste trabalho é realizar uma Revisão de Escopo (RE) a fim de investigar na literatura as soluções que utilizam linguagens e ferramentas destinadas à especificação formal de software. 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, tais como as linguagens Z, Alloy, Event-B, TLA+, SysML, Redes de Petri Coloridas, as ferramentas SAL, Coq, SPL e outras, destacando seu potencial para melhorar a confiabilidade e segurança de sistemas críticos, como os de segurança e Veículos aéreos não tripulados (VANTs). Os resultados revelaram que a utilização de linguagens formais, ferramentas de verificação e simulação são eficazes na garantia da implementação correta de requisitos de segurança e na detecção de erros em especificações.

2 Método da pesquisa

O presente estudo utiliza uma Revisão de Escopo (RE) como metodologia, a qual tem como objetivo identificar os principais conceitos de uma determinada área de conhecimento por meio do estudo de trabalhos já publicados (ARKSEY; O’MALLEY, 2005). Para fazer esta RE, foi realizada uma adaptação nos procedimentos metodológicos apresentados em Arksey e O’malley (2005), com retificações propostas por Levac, Colquhoun e O’brien (2010) e Peter *et al*., 2017. Dessa forma, espera se que a presente RE possa trazer importantes contribuições para o avanço do conhecimento na área de aplicação de Métodos Formais, por meio do levantamento das principais linguagens e ferramentas utilizadas na especificação de software.

Para atender aos objetivos desta pesquisa, esta RE seguirá as 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 os critérios predefinidos;
4. Avaliação e extração dos resultados e apresentação dos resultados.

2.1 Questões de 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 este questionamento, 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 as limitações identificadas?

2.2 Escopo

O escopo desta pesquisa está voltado para estudos que apresentem a utilização de linguagens e ferramentas na especificação formal de software. Assim, será possível identificar quais são as linguagens e ferramentas utilizadas para especificação de software, como são utilizadas e quais as limitações apresentadas.

2.2.1 Estratégia de busca

Esta RE busca encontrar estudos publicados entre 01 de janeiro de 2018 e 18 de março de 2023, e escritos nos idiomas inglês e português. A busca foi realizada em quatro fontes de busca relevantes na área, a citar: *ACM Digital Library*[1], *IEEE Xplore*[2], *Scopus*[3]e *Web of Science*[4]. As fontes selecionadas foram escolhidas levando em consideração a disponibilidade de acesso ao conteúdo dos artigos, bem como a quantidade significativa de publicações relacionadas à área de tecnologia.

Na sequência, foram identificadas as palavras-chave que alcançassem artigos referentes à temática desta pesquisa, a citar: Especificação formal *(Formal specification*), Verificação formal (*Formal verification*), Modelagem formal (F*ormal modeling*), Análise formal (*Formal analysis*), Prova formal (*Formal proof*), Linguagens formais (*Formal languages*), Linguagens de especificação (*Specification languages*) Ferramentas de especificação (*Specification tool*), Ferramentas de verificação (*Verification tools*).

Com isso, baseado nos termos selecionados foi possível definir a seguinte *string* ((*"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 tool" OR "Ferramentas de verificação" OR "Verification tools"*)).

2.2.2 Critério para seleção

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

• 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 do arquivo completo de forma gratuita.

3 Resultados e discussões

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

3.1 Seleção de trabalhos

-[1] https://dl.acm.org/
[2] https://ieeexplore.ieee.org/
[3] https://www.scopus.com/
[4] https://login.webofknowledge.com/

O processo de seleção e escolha dos trabalhos foi dividido em quatro etapas. Uma descrição detalhada das etapas pode ser encontrada on-line[5]. A 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.

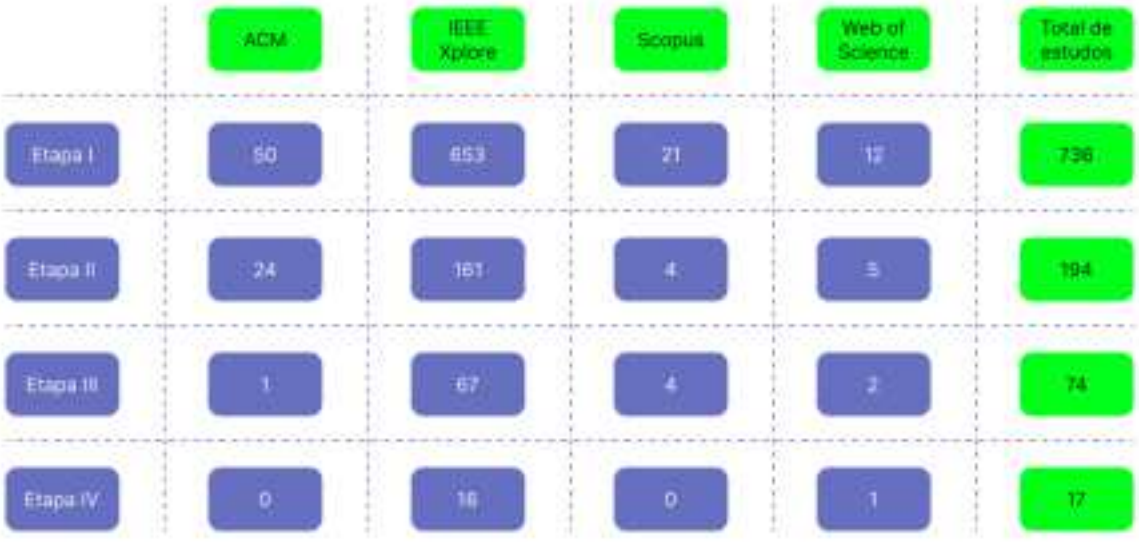
Na Etapa II, foi realizada uma análise detalhada dos títulos, palavras-chave e resumos dos trabalhos selecionados na Etapa I, visando verificar o atendimento aos critérios de inclusão CI1 e os critérios de exclusão CE1, CE3 e CE4. Os trabalhos que atenderam aos critérios de inclusão foram adicionados 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 quais eram as características principais dos trabalhos, os 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), quais eram os recursos utilizados (linguagens, ferramentas, bibliotecas e outros), quais os procedimentos adotados para validar as soluções e quais as limitações identificadas. Por fim, **17** trabalhos atenderam a todos os critérios estabelecidos e foram selecionados para responder às Questões Especificas (QEs).

A Figura 1 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 1 – 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. Destes, 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

[5] https://shorturl.at/noMX8

abordam diretamente o tema da pesquisa. A seguir, são apresentadas as respostas obtidas para as QEs com base nesses 17 trabalhos. No Quadro 1 encontram-se detalhadas as respostas identificadas a partir das Questões Específicas.

Quadro 1 – Sumarização das respostas a partir das QEs

Autores	QE1	QE2	QE3
(ROULAND *et al.*, 2019)	Linguagens: Z, 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; VINGERHOEDS, 2020)	Linguagens: SysML Ferramentas: TTool, Eclipse com plugin Papyrus, ROS	Linguagem SysML e ferramenta TTool para modelagem e simulação do sistema de controle de um VANT. Eclipse como ambiente de desenvolvimento. *Plugin Papyrus* e biblioteca ROS para comunicação	Estudo de caso baseado em uma situação educacional específica pode não ser generalizável
(BRIDA *et al.*, 2021)	Linguagens: Alloy Ferramentas: Java, 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 a Alloy e ao *Bounded Exhaustive Search*	Falta de testes da ferramenta proposta em um contexto real
(ZHANG; SALCIC; MALIK, 2019)	Linguagens: 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; ABRIANI, 2019)	Linguagens: Z Ferramentas: SAL, Z2SAL	Linguagem Z para especificação formal. Linguagem 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)	Linguagens: B, 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	Estudo de caso identificou uma lacuna no modelo HLL produzido

		para produzir modelos HLL com entrada B	
(LUKÁCS; BARTHA, 2022)	Linguagens: linguagem natural, 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)	Linguagens: Solidity, 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
(CÁRDENAS *et al*., 2022)	Linguagens: 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; LIU, 2019)	Linguagens: Redes de Petri Coloridas, 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)	Linguagens: BPMN, BPMN2, 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)	Linguagens: 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; ZAFAR, 2019)	Linguagens: UML, 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; KATAYAMA, 2020)	Linguagens: VDM++, 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; SCHOENHAAR, 2019)	Linguagens: DSLs Ferramentas: MPS, SysML/AADL	MPS para definir e estender as DSLs. SysML/AADL para compilar modelos em modelos de nível inferior	Desafios das abordagens tradicionais de especificação formal, como a necessidade de habilidades especializadas

Fonte: dados da pesquisa

3.2 QE1: Quais são as principais linguagens e ferramentas utilizadas?

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

Em Apvrille Saqui-Sannes e Vingerhoeds, (2020) é apresentada 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. Os autores utilizaram a linguagem SysML e a ferramenta TTool para a modelagem e simulação do sistema. Também foi utilizado o ambiente de desenvolvimento eclipse, o *plugin Papyrus* e a biblioteca de comunicação *Robot Operating System* (ROS).

No trabalho de Brida *et al*., (2021) é apresentada a descrição de 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 para demonstrar a eficácia da técnica proposta na identificação de reparos de especificações Alloy. A ferramenta foi desenvolvida utilizando a ferramenta Alloy e o *framework Bounded Exhaustive Search*, a linguagem Java e as bibliotecas relacionadas à linguagem Alloy também foram utilizadas.

Em Zhang, Salcic e Malik, (2019) o foco está no uso de duas linguagens e ferramentas principais: SystemJ e Redes de Petri Coloridas (do inglês *Coloured Petri Net* - CPN). SystemJ é uma linguagem de programação síncrona usada para modelagem e programação de sistemas *Globally Asynchronous Locally Synchronous* (GALS). As Redes de Petri Coloridas, por outro lado, são uma linguagem formal de modelagem usada para análise e verificação de sistemas concorrentes. Os autores também usam uma folha de estilo XSLT (*Extensible Stylesheet Language Transformation*) para realizar a tradução de arquivos AST (*Abstract Syntax Tree*) SystemJ para arquivos na linguagem CPN ML. Por fim, eles utilizam CPN Tools, uma ferramenta de software para modelagem, simulação e análise de Redes de Petri Coloridas, para analisar e verificar os modelos traduzidos.

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 SAL. 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 usada é o assistente de prova Coq, que é usado 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 também são discutidos, mas não é especificada qual linguagem de programação é usada para o desenvolvimento real de aplicações Android.

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

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, muitas técnicas e ferramentas relacionadas à Engenharia de Sistemas Baseada em Modelos (do inglês *Model-based Systems Engineering* - MBSE) são aplicadas na área ferroviária para apoiar a modelagem, como *Unified Modeling Language* (UML) ou *Systems Modeling Language* (SysML). No entanto, o trabalho não especifica nenhuma linguagem ou ferramenta específica como a principal utilizada na metodologia proposta.

Em Zhu *et al*., (2021) a principal linguagem usada é Solidity, que é uma linguagem de alto nível orientada a contratos e projetada para atingir 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 Cárdenas *et al*., (2022) as principais linguagens e ferramentas usadas são a 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 uma ferramenta *plugin* para USE que integra um framework para validar modelos de software UML.

Em Liu e Liu, (2019) os autores mencionam que as Redes de Petri Coloridas (CPN) e ferramentas CPN são usadas para modelar contratos inteligentes e atacantes, e ASK-CTL e

Ferramentas de Espaço de Estado são usadas para verificação de modelos. Além disso, o artigo 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, que é uma ferramenta para desenvolvimento e verificação de modelos de Event-B.

Em Zhao e*t al*., (2019) é usado 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, que é uma técnica de verificação de modelo. O comportamento do sistema é capturado através da caixa de ferramentas TLA+, que é validada usando o recurso TLC disponível na caixa de ferramentas TLA+.

Em Westphal, (2020) os autores sugerem selecionar algumas linguagens de descrição de software representativas que são discutidas em livros didáticos e complementar a discussão com uma cobertura totalmente formal, particularmente 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 linguagem principal usada é VDM++, que é uma linguagem de extensão orientada a objetos baseada em VDM-SL. Os autores também usaram a linguagem de programação Python e a biblioteca NLTK para processamento de linguagem natural, e o Stanford Parser para análise de dependências. Além disso, utilizaram VDMTools e VDMJ como ferramentas de suporte para VDM (Método de Desenvolvimento Viena).

Em Ratiu, Gario e Schoenhaar, (2019) o principal ambiente de trabalho usado é o *Meta Programming System* (MPS), que suporta todos os aspectos da definição de linguagens de domínio específico (do inglês *Domain Specific Languages* - 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 usadas para compilar modelos em modelos de nível inferior para consumo por ferramentas de verificação.

3.3 QE2: Como essas linguagens e ferramentas são utilizadas na especificação formal de software? Com relação à utilização das linguagens e ferramentas, em Rouland *et al*., (2019) as linguagens Z e Alloy foram utilizadas para definir precisamente os requisitos de segurança. Os *model checkers* foram utilizados para verificar a consistência e a corretude dos requisitos especificados. E a biblioteca de especificação de segurança *Security Property Library* (SPL) foi utilizada para auxiliar 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 Eclipse

foi o ambiente de desenvolvimento, o *plugin Papyrus* e a biblioteca ROS foram utilizadas 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 realizar a busca por 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 a Alloy e ao *Bounded Exhaustive Search*.

No trabalho de Zhang, Salcic e Malik, (2019) a junção do SystemJ com CPNs foi usada para modelar e analisar sistemas GALS. SystemJ foi usado para modelar o comportamento do sistema, enquanto as Redes de Petri Coloridas são usadas para verificar formalmente o modelo. A tradução de modelos SystemJ em Redes de Petri Coloridas permite o uso de uma ampla gama de ferramentas de análise e verificação para Redes de Petri Coloridas, incluindo análise de espaço de estados e verificação de modelos. Essas técnicas podem ser usadas para verificar várias propriedades do

sistema, como ser livre de *deadlock*, vivacidade e segurança. Ao utilizar métodos formais para especificar e verificar o comportamento dos sistemas GALS, é possível detectar e corrigir erros precocemente no processo de projeto, levando a sistemas mais confiáveis e seguros.

Em Siregar e Abriani, (2019) Z é uma linguagem de especificação formal que utiliza matemática para declarar sistemas e esquemas para construir e modularizar a especificação. Produz especificações mais formais e livres de ambiguidades, permitindo a realização de análises mecânicas. A linguagem SAL é uma linguagem formal usada para especificar e modelar sistemas, e o verificador de modelo SAL é usado para verificar a exatidão do modelo do sistema. Ao utilizar essas linguagens e ferramentas formais, os autores conseguiram verificar o sistema especialista e garantir sua correção.

Em Khan *et al*., (2019) o assistente de prova Coq é usado para desenvolver um modelo formal de segurança Android baseado em linguagem, que permite o raciocínio formal sobre o sistema de permissão do Android usando uma ferramenta de prova de teoremas baseada em computador. O modelo formal é construído usando a lógica por trás do provador de teoremas, e a solidez da técnica de segurança baseada na linguagem é 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. Essa abordagem permite verificação e raciocínio baseados em computador, o que é importante para a especificação formal de software.

No trabalho de Halchin *et al*., (2019) a linguagem formal B é usada para especificação formal de software, enquanto a linguagem de modelagem HLL é usada como base para verificação de propriedades de segurança, a fim de preencher a lacuna entre a especificação de software com o desenvolvimento formal em B, e as técnicas de verificação no sistema. O kit de ferramentas Isabelle/HOL e sua biblioteca de táticas são usados para descrever a semântica formal de ambas as línguas e para formalizar a relação de tradução de ambas as línguas. Os modelos Isabelle/HOL desenvolvidos são então comprovados como garantindo a correção do processo de tradução.

Em Lukács e Bartha, (2022) a metodologia proposta para sistemas de intertravamento ferroviário utiliza um ambiente de especificação que se baseia 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 neste ambiente de especificação. 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 trabalho de Zhu *et al*., (2021) o Solidity é usado para escrever contratos inteligentes - autoexecutáveis, com os termos do acordo entre cliente e fornecedor sendo escritos diretamente em linhas de código. O Event-B é usado para criar modelos abstratos dos contratos Solidity, que podem ser refinados para criar modelos mais detalhados que podem ser verificados quanto à segurança. ANTLR é usado para implementar um tradutor que pode gerar 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, podem então ser aplicadas ao modelo Evento-B para garantir que o contrato atenda aos requisitos de segurança e proteção.

Em Cárdenas *et al*., (2022) a UML e a OCL são usadas para especificar o sistema de software e seus requisitos estruturais formalmente. O Ambiente de Especificação baseado em UML (USE) é usado 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 seu comportamento.

No trabalho de Liu e Liu, (2019) os autores mencionam que o método de verificação sugerido é baseado em Redes de Petri Coloridas (CPN) para verificar contratos inteligentes no sistema *blockchain*. Os modelos CPN são executados por simulação passo a passo para validar sua correção funcional e, finalmente, a tecnologia de verificação de modelo baseada em lógica de temporização de ramificação ASK-CTL nas ferramentas CPN é utilizada para detectar vulnerabilidades latentes em contratos inteligentes. Isto sugere que CPN e ASK-CTL são usados para especificar formalmente e verificar o comportamento de contratos inteligentes de forma rigorosa e sistemática.

Em Younes *et al*., (2019) os autores propõem 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 Event-B. A especificação formal resultante pode ser verificada utilizando a plataforma Rodin. Ao utilizar um método formal como o Event-B, os autores pretendem 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 e*t al*., (2019) é utilizado o Storm, CSP e FDR para formalizar o fluxo de trabalho do Storm e verificar se ele satisfaz propriedades de consistência sequencial e sem *deadlock*. O CSP é utilizado para modelar os comportamentos de comunicação no fluxo de trabalho do Storm, e o FDR é utilizado para verificar se o modelo estabelecido 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+ é utilizada para validar e verificar propriedades do sistema usando técnicas formais. A especificação é analisada por meio do TLC, que é 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.

Em Shigyo e Katayama, (2020) o VDM++ é usado para descrever as especificações, que podem ser verificadas por teoremas matemáticos e verificações mecânicas. VDMTools e VDMJ facilitam a verificação das especificações. No trabalho de Ratiu, Gario e Schoenhaar, (2019) o MPS é usado 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 usadas para compilar modelos em modelos de nível inferior para consumo por ferramentas de verificação. Ao usar 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.

3.4 QE3: Quais as limitações identificadas?

No que diz respeito às limitações identificadas, em Rouland *et al*., (2019) as limitações incluem a necessidade de conhecimento prévio em engenharia de requisitos e métodos formais. Além disso, a validação foi limitada por se tratar de um único estudo de caso. Ainda com relação às limitações, Apvrille

Saqui-Sannes e Vingerhoeds, (2020) apresentam um estudo de caso baseado em uma situação educacional específica, o que pode não apresentar bons resultados para outras situações educacionais. No trabalho de Brida *et al*., (2021) poderia ser realizado um estudo de caso para testar a ferramenta proposta em um contexto real.

Em Zhang, Salcic e Malik, (2019) são mencionadas 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, que pode ocorrer 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 em redes de Petri coloridas pode não ser totalmente automatizada e pode exigir intervenção manual. Finalmente, os autores observam que a abordagem de tradução apresentada no artigo cobre apenas o subconjunto síncrono do SystemJ e pode não ser aplicável a toda a linguagem.

No trabalho de Siregar e Abriani, (2019) não são mencionadas explicitamente quaisquer limitações que foram identificadas durante o desenvolvimento e verificação do sistema especialista baseado em regras usando Z e SAL. No entanto, vale a pena notar que os métodos formais, em geral,

podem ser demorados e exigir um elevado nível de conhecimentos para serem utilizados de forma eficaz. Além disso, os métodos formais podem não ser adequados para todos os tipos de sistemas de software e a sua utilização pode nem sempre ser rentável.

O trabalho de Khan *et al*., (2019) não menciona explicitamente quaisquer limitações da abordagem usada para formalizar a segurança do Android baseada em linguagem. No entanto, menciona que a linguagem formal para representar aplicações Android e o sistema de tipos que impõe as melhores práticas não podem ser lidos 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 tipo devem ser definidos em um provador de teoremas.

Em Halchin *et al*., (2019) não são mencionadas explicitamente quaisquer limitações da abordagem proposta. No entanto, discute um estudo de caso relacionado a um sistema de software ferroviário e como a abordagem foi usada para identificar 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 esta violação de propriedade.

No trabalho de Lukács e Bartha, (2022) não é fornecida uma lista abrangente das limitações identificadas na metodologia proposta para sistemas de intertravamento ferroviário. No entanto, menciona que a especificação e o planejamento simultâneos são propensos a erros e que é raro que uma implementação apropriada de um sistema seja alcançada em apenas uma especificação e uma etapa de design. Além disso, o documento observa que as práticas tradicionais de desenvolvimento muitas vezes englobam usuários que têm uma participação limitada no processo e que comunicam formalmente seus requisitos em alguma forma textual, que não estão dispostos a modificar formalmente durante o desenvolvimento devido ao seu 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.

Em Zhu *et al*., (2021) menciona que escrever contratos inteligentes seguros e protegidos pode ser extremamente difícil devido a diversas lógicas de negócios, bem como a 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 determinado 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.

O trabalho Cárdenas *et al*., (2022) não menciona explicitamente nenhuma limitação identificada. No entanto, afirma que ainda não existem melhores práticas padronizadas para proteger os sistemas da Internet das Coisas (IoT), o que destaca a necessidade de melhorias nesta área. Em Liu e Liu, (2019) os autores informam que existem muitas vulnerabilidades na maioria dos contratos inteligentes devido à complexidade do ambiente distribuído e às limitações da linguagem de programação.

No trabalho de Younes *et al*., (2019) são discutidas diversas limitações da linguagem BPMN2, incluindo a ausência de uma semântica formal precisa das diversas notações utilizadas, o que muitas vezes leva a ambiguidades, e a falta de um sistema de prova que valide uma especificação BPMN2.

Essas limitações podem dificultar 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.

Em Zhao e*t al*., (2019) não são mencionadas explicitamente quaisquer limitações da abordagem usada para modelar e verificar Storm. No entanto, os autores mencionam que pesquisas anteriores sobre Storm se concentraram na análise ou na otimização do Storm, mas não houve nenhum trabalho formal de modelagem do Storm. O artigo visa preencher esta 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 algumas propriedades significativas do Storm.

Em Latif, Rehman e Zafar, (2019) os autores não mencionam explicitamente quaisquer limitações da abordagem usada. Em Westphal, (2020) os autores mencionam algumas limitações da abordagem proposta. Uma das principais é que é difícil medir se os objetivos de aprendizagem foram alcançados, pois seria necessário ver 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 bastante estáveis ao longo dos testes e tomam 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.

Em Shigyo e Katayama, (2020) os autores identificaram diversas limitações de 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.

No trabalho de Ratiu, Gario e Schoenhaar, (2019) não são mencionadas explicitamente quaisquer limitações da abordagem FASTEN. No entanto, os autores 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 Considerações finais e Trabalhos futuros

Neste trabalho, foi realizada uma RE com o objetivo de identificar quais são as principais linguagens e ferramentas utilizadas na especificação formal de software. Para tal, definiu-se um protocolo para condução da revisão. Em conclusão, as abordagens formais para modelagem, verificação e correção de sistemas de segurança e 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 ser eficaz na garantia da correta implementação de requisitos de segurança e na identificação de erros em especificações, contribuindo para a mitigação de vulnerabilidades e a prevenção de falhas em sistemas complexos.

No entanto, 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, estaremos contribuindo para a construção de sistemas mais seguros, confiáveis e resilientes, essenciais para o avanço da tecnologia e o bem-estar da sociedade. Como trabalhos futuros, pretende-se realizar uma pesquisa mais abrangente, incorporando outras fontes de busca e ampliando o período de estudo, além de incluir trabalhos em outros idiomas. Essa abordagem permitirá uma visão mais completa das práticas em especificação formal de software, possibilitando para 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. de; 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**, IEEE, v. 1, n. 2, p. 117–129, 2020.

ARKSEY, H.; O'MALLEY, L. Scoping studies: towards a methodological framework. **International journal of social research methodology**, Taylor & Francis, v. 8, n. 1, p. 19–32, 2005.

BRIDA, S. G. et al. Artifact of bounded exhaustive search of alloy specification repairs. In: **Proceedings** of the IEEE/ACM 43rd International Conference on Software Engineering: Companion Proceedings (ICSE-Companion). [S.l.], 2021, p. 209–210.

CARDENAS, H. et al. Formal uml-based modeling and analysis for securing location-based iot applications. In: **Proceedings** of the IEEE 19th International Conference on Mobile Ad Hoc and Smart Systems (MASS). [S.l.], 2022, p. 722–723.

EDUCAÇÃO, X. **O que é engenharia de software? Tudo sobre a carreira.** 2022. Disponível em: https://blog.xpeducacao.com.br/o-que-e-engenharia-de-software/. Acesso em: 16 mar. 2024.

FLORES, F. E. L. **Verificação formal na indústria.** Dissertação (Mestrado em Engenharia Elétrica) - Universidade Federal de Minas Gerais, 2016.

HALCHIN, A. et al. Certified embedding of b models in an integrated verification framework. In: **Proceedings** of the IEEE International Symposium on Theoretical Aspects of Software Engineering (TASE). [S.l.], 2019, p. 168–175.

KHAN, W. et al. **Formal analysis of language-based android security using theorem proving approach**. IEEE Access, IEEE, v. 7, p. 16550–16560, 2019.

LATIF, S.; REHMAN, A.; ZAFAR, N. A. Nfa based formal modeling of smart parking system using tla+. In: **Proceedings** of the IEEE International Conference on Information Science and Communication Technology (ICISCT). [S.l.], 2019, p. 1–6.

LEVAC, D.; COLQUHOUN, H.; O'BRIEN, K. K. Scoping studies: advancing the methodology. Implementation science, Springer, v. 5, p. 1–9, 2010.

LIU, Z.; LIU, J. Formal verification of blockchain smart contract based on colored petri net models. In: **Proceedings** of the IEEE 43rd Annual Computer Software and Applications Conference (COMPSAC). [S.l.], 2019, v. 2, p. 555–560.

LUKÁCS, G.; BARTHA, T. Conception of a formal model-based methodology to support railway engineers in the specification and verification of interlocking systems. In: **Proceedings** of the IEEE 16th International Symposium on Applied Computational Intelligence and Informatics (SACI). [S.l.], 2022, p. 000283–000288.

PETERS, M. D. et al. Scoping reviews. In: **Joanna Briggs Institute reviewer's manual**. The Joanna Briggs Institute Australia, 2017.

PRESSMAN, R. S. **Engenharia de software**. Monthly Weather Review, AMGH, v. 131, 2003, p. 34.
PUCPR, E. Áreas da Engenharia de Software: Saiba tudo sobre a carreira. 2020. Disponível em: https://ead.pucpr.br/blog/areas-engenharia-de-software. Acesso em: 16 mar. 2023.

RATIU, D.; GARIO, M.; SCHOENHAAR, H. Fasten: An open extensible framework to experiment with formal specification approaches. In: **Proceedings** of the IEEE/ACM 7th International Conference on Formal Methods in Software Engineering (FormaliSE). [S.l.], 2019, p. 41–50.

ROULAND, Q. et al. A formal methods approach to security requirements specification and verification. In: **Proceedings** of the IEEE 24th International Conference on Engineering of Complex Computer Systems (ICECCS). [S.l.], 2019, p. 236–241.

SCAICO, A. et al. **Aplicação de métodos formais no projeto de interfaces para sistemas industriais críticos**. Dissertação (Mestrado em Engenharia Elétrica) - Universidade Federal de Campina Grande, 2007.

SHIGYO, Y.; KATAYAMA, T. Proposal of an approach to generate vdm++ specifications from natural language specification by machine learning. In: **Proceedings** of the IEEE 9th Global Conference on Consumer Electronics (GCCE). [S.l.], 2020, p. 292–296.

SIREGAR, M. U.; ABRIANI, S. Verification of a rule-based expert system by using sal model checker. In: **Proceedings** of the IEEE 3rd International Conference on Informatics and Computational Sciences (ICICoS). [S.l.], 2019, p. 1–6.

SOMMERVILLE, I. **Engenharia de software**. 9. ed. [S.l.: s.n.], 2011. 63 p.

WESTPHAL, B. On complementing an undergraduate software engineering course with formal methods. In: **Proceedings** of the IEEE 32nd Conference on Software Engineering Education and Training (CSEE&T). [S.l.], 2020, p. 1–10.

YOUNES, A. B. et al. From bpmn2 to event b: A specification and verification approach of workflow applications. In: **Proceedings** of the IEEE 43rd Annual Computer Software and Applications Conference (COMPSAC). [S.l.], 2019, v. 2, p. 561–566.

ZHAO, H. et al. Modeling and verifying storm using csp. In: **Proceedings** of the IEEE 19th International Symposium on High Assurance Systems Engineering (HASE). [S.l.], 2019, p. 192–199.

ZHANG, W.; SALCIC, Z.; MALIK, A. Towards formal modeling and analysis of systemj gals systems using coloured petri nets. In: **Proceedings** of the IEEE 17th International Conference on Industrial Informatics (INDIN). [S.l.], 2019, p. 152–159.

ZHU, J. et al. Formal simulation and verification of solidity contracts in event-b. In: **Proceedings** of the IEEE 45th Annual Computers, Software, and Applications Conference (COMPSAC). [S.l.], 2021, p. 1309–1314.