

Universidade de Brasília – UnB
Faculdade de Ciências e Tecnologia em Engenharias – FCTE
Engenharia de Software

DevSecOps: um estudo de caso sobre o desenvolvimento do produto Brasil Participativo

Autor: Chaydson Ferreira da Aparecida
Orientador: Msc. Hilmer Rodrigues Neri

Brasília, DF
2025



Chaydson Ferreira da Aparecida

DevSecOps: um estudo de caso sobre o desenvolvimento do produto Brasil Participativo

Monografia submetida ao curso de graduação em Engenharia de Software da Universidade de Brasília, como requisito parcial para obtenção do Título de Bacharel em Engenharia de Software.

Universidade de Brasília – UnB

Faculdade de Ciências e Tecnologia em Engenharias – FCTE

Orientador: Msc. Hilmer Rodrigues Neri

Brasília, DF

2025

Chaydson Ferreira da Aparecida

DevSecOps: um estudo de caso sobre o desenvolvimento do produto Brasil Participativo/ Chaydson Ferreira da Aparecida. – Brasília, DF, 2025-

99 p. : il. (algumas color.) ; 30 cm.

Orientador: Msc. Hilmer Rodrigues Neri

Trabalho de Conclusão de Curso – Universidade de Brasília – UnB
Faculdade de Ciências e Tecnologia em Engenharias – FCTE , 2025.

1. DevSecOps. 2. Segurança de Software. I. Msc. Hilmer Rodrigues Neri. II. Universidade de Brasília. III. Faculdade UnB Gama. IV. DevSecOps: um estudo de caso sobre o desenvolvimento do produto Brasil Participativo

CDU 02:141:005.6

XXXXXXXXXXX

Chaydson Ferreira da Aparecida

DevSecOps: um estudo de caso sobre o desenvolvimento do produto Brasil Participativo

Monografia submetida ao curso de graduação em Engenharia de Software da Universidade de Brasília, como requisito parcial para obtenção do Título de Bacharel em Engenharia de Software.

Trabalho aprovado. Brasília, DF, 17 de dezembro de 2025:

Msc. Hilmer Rodrigues Neri
Orientador

Dr. Renato Coral Sampaio
Convidado 1

Dr. Tiago Alves da Fonseca
Convidado 2

Brasília, DF
2025

Agradecimentos

Aos meus pais e à minha irmã, pelo apoio não somente na minha jornada acadêmica, mas em toda a minha vida. Por entenderem que as noites sem dormir, os finais de semana, os feriados e os momentos em que estive ausente eram necessários para que eu pudesse alcançar esse sonho.

À minha namorada e futura esposa, que nunca me deixou desistir e por várias vezes foi o meu alicerce. Por acreditar em mim mesmo quando eu não acreditava. Por ter sido paciente e compreensiva durante todo esse período. Por ter passado junto comigo pelos momentos mais difíceis e ter vivido também os momentos mais incríveis.

Ao Prof. Hilmer, que esteve comigo durante toda a elaboração do trabalho e foi para mim mais que um mestre. Por toda a paciência que teve comigo durante esse ano. Por acreditar no meu potencial e ter me estendido a mão.

Aos meus amigos, que deixaram mais leve os momentos mais difíceis. Por todas as conversas, risadas e momentos de descontração que me ajudaram a seguir em frente.

A todos que acreditaram que esse sonho era possível e fizeram parte desses momentos, a minha gratidão.

“Se eu pude enxergar adiante, é porque me apoiei nos ombros de gigantes.“
(Isaac Newton)

Resumo

O presente trabalho tem como objetivo analisar a adoção de práticas de DevSecOps no ciclo de vida de desenvolvimento do produto de software livre Brasil Participativo, investigando os impactos na segurança do projeto. Desse modo, foi realizado um estudo de caso, estruturado pela abordagem *Goal Question Metric* (GQM) e fundamentado por uma revisão da literatura. O método envolveu a integração de ferramentas de análise estática, dinâmica e de composição de software na pipeline de integração contínua para a coleta automatizada de dados quantitativos. Os resultados indicaram que a automação proporcionou a identificação eficaz de vulnerabilidades nas perspectivas de qualidade interna e externa, apesar de a pipeline ter apresentado indicadores iniciais de baixo desempenho devido à alta taxa de falhas detectadas. Conclui-se que a incorporação de práticas de DevSecOps viabilizou a detecção precoce de falhas e forneceu parâmetros quantitativos fundamentais para a tomada de decisão, reforçando a importância do monitoramento contínuo em ambientes de desenvolvimento ágil.

Palavras-chave: DevSecOps. Segurança de Software. Estudo de Caso. CI/CD. Qualidade de Software

Abstract

The present work aims to analyze the adoption of DevSecOps practices in the development lifecycle of the open-source software product Brasil Participativo, investigating the impacts on the project's security. Therefore, a case study was conducted, structured by the Goal Question Metric (GQM) approach and grounded in a literature review. The method involved integrating static, dynamic, and software composition analysis tools into the continuous integration pipeline for the automated collection of quantitative data. The results indicated that automation enabled the effective identification of vulnerabilities from the perspectives of internal and external quality, although the pipeline presented initial low performance indicators due to the high rate of detected failures. It is concluded that the incorporation of DevSecOps practices enabled the early detection of flaws and provided fundamental quantitative parameters for decision-making, reinforcing the importance of continuous monitoring in agile development environments.

Key-words: DevSecOps. Software Security. Case Study. CI/CD. Software Quality.

Lista de ilustrações

Figura 1 – Abordagem GQM 30
Figura 2 – Atividades da Primeira Etapa 32
Figura 3 – Cronograma da Primeira Etapa 33
Figura 4 – Atividades da Segunda Etapa 34
Figura 5 – Cronograma da Segunda Etapa 34
Figura 6 – Protocolo PICO 48
Figura 7 – Seleção dos Artigos 55
Figura 8 – Volume Anual de Artigos entre 2009 e 2025 62
Figura 9 – Artigos Publicados em Revistas 62
Figura 10 – Quantidade de Artigos com Validação Experimental 63
Figura 11 – Tipos de Validação Experimental dos Artigos 64
Figura 12 – Organograma das equipes do projeto Brasil Participativo 67
Figura 13 – BPMN do processo de desenvolvimento do Brasil Participativo 68
Figura 14 – SAST - Média de Vulnerabilidades por *Merge* 82
Figura 15 – DAST - Média de Vulnerabilidades por *Merge* 83
Figura 16 – SCA - Média de Vulnerabilidades por *Merge* 84
Figura 17 – Proporção de Bibliotecas com Vulnerabilidades por *Merge* 85
Figura 18 – Taxas de Aviso ou Falha por Quinzena 86

Lista de tabelas

Tabela 1 – Análise de Tendência de Vulnerabilidades - SAST (Brakeman) 87
Tabela 2 – Análise de Tendência de Alertas - DAST (ZAP) 87

Lista de Quadros

Quadro 1 – GQM Adaptado 30
Quadro 2 – PICO Adaptado 49
Quadro 3 – Protocolo de Busca 50
Quadro 4 – Artigos Selecionados 52
Quadro 5 – Vulnerabilidades Reduzidas 61
Quadro 6 – Períodos Analisados 79
Quadro 7 – Atividades Concluídas 93
Quadro 8 – Objetivos Concluídos 93

Lista de abreviaturas e siglas

BPMN	Business Process Model and Notation
CFR	Change Fail Rate
CI/CD	Integração Contínua e Entrega/Implantação Contínua
CSRF	Cross-Site Request Forgery
CVE	Common Vulnerabilities and Exposures
CVSS	Common Vulnerability Scoring System
CWE	Common Weakness Enumeration
DAST	Dynamic Application Security Testing
DevOps	Development and Operations
DevSecOps	Development, Security, and Operations
DDoS	Distributed Denial of Service
DORA	DevOps Research and Assessment
DSOMM	DevSecOps Maturity Model
GNU AGPL	GNU Affero General Public License
GQM	Goal Question Metric
IaC	Infraestrutura como Código
IAST	Interactive Application Security Testing
ISO/IEC	International Organization for Standardization/International Electrotechnical Commission
KPIs	Indicadores-Chave de Desempenho
LAPP1S	Laboratório Avançado de Produção, Pesquisa e Inovação em Software
MGI	Ministério da Gestão e da Inovação em Serviços Públicos
MITM	Man-in-the-Middle
MRs	Merge Requests

OWASP	Open Web Application Security Project
QE-1	Questão Específica 1
QE-2	Questão Específica 2
SAMM	Software Assurance Maturity Model
SAST	Static Application Security Testing
SCA	Software Composition Analysis
SGSI	Sistema de Gestão de Segurança da Informação
SNPS/SGPR	Secretaria Nacional de Participação Social da Secretaria-Geral da Presidência da República
SQL	Structured Query Language
SQuaRE	Requisitos e Avaliação de Qualidade de Sistemas e Software
SSRF	Server-Side Request Forgery
XSS	Cross-Site Scripting
XXE	XML External Entity
ZAP	Zed Attack Proxy

Lista de símbolos

$\sum$	Somatório

Sumário

Lista de Quadros 17

1 INTRODUÇÃO 27
1.1 Contexto 27
1.2 Problema 29
1.3 Questão de Pesquisa 30
1.4 Objetivos 31
1.5 Estrutura do Trabalho 31
1.6 Cronograma e Atividades 32
1.6.1 Primeira Etapa 32
1.6.2 Segunda Etapa 34

2 REFERENCIAL TEÓRICO 37
2.1 Engenharia de *Software* Experimental 37
2.1.1 Estudo de Caso 38
2.1.1.1 Definição 38
2.1.1.2 Preparação e Coleta de Dados 39
2.1.1.3 Análise de Dados Coletados 39
2.1.1.4 Relato dos Resultados 39
2.2 Modelos de Qualidade de *Software* 40
2.3 Segurança de *Software* 42
2.4 *DevOps* 43
2.5 *DevSecOps* 44

3 REVISÃO ESTRUTURADA DA LITERATURA 47
3.1 Protocolo 47
3.1.1 *String* de Busca 47
3.2 Seleção dos Artigos 50
3.3 Resultados 55
3.3.1 Quais práticas de *DevSecOps* são mais prevalentes nas organizações? . . . 56
3.3.2 Quais modelos de segurança são mais comumente aplicados em ambientes *DevSecOps*? 57
3.3.3 Quais medidas são comumente usadas para aferição de segurança? 58
3.3.4 Como as práticas de segurança são avaliadas? 59
3.3.5 De que maneiras as medidas de segurança são avaliadas nas organizações? . 60

3.3.6 Que tipos de vulnerabilidades são mitigados pelas práticas de *DevSecOps*? . . . 60
3.3.7 Quais ferramentas e tecnologias são usadas no *DevSecOps*? . . . 60
3.3.8 Qual é o volume anual de publicações sobre *DevSecOps* de 2009 a 2025? . . . 61
3.3.9 Quantos artigos foram publicados em periódicos acadêmicos? . . . 62
3.3.10 Quantos estudos têm validação experimental? . . . 63
3.3.11 Caso o estudo tenha validação experimental, qual foi o tipo? . . . 63

4 PLANEJAMENTO DO ESTUDO DE CASO . . . 65
4.1 Definição . . . 65
4.2 Objetivo . . . 66
4.3 Caso . . . 66
4.4 Trabalhos Relacionados . . . 69
4.5 Questão de Pesquisa . . . 70
4.6 Fonte de Dados . . . 71
4.7 Unidades de Análise e Procedimentos . . . 71
4.8 Análise de Dados . . . 72
4.9 Instrumentação . . . 73

5 EXECUÇÃO DO ESTUDO DE CASO . . . 75
5.1 Configuração do *Pipeline* de CI/CD Localmente . . . 75
5.2 Elaboração dos *Scripts* para Automação da Coleta de Dados . . . 76
5.3 Consolidação dos Dados Coletados . . . 78
5.4 Aplicação do Questionário . . . 80

6 RESULTADOS DO ESTUDO DE CASO . . . 81
6.1 Análise dos Dados . . . 81
6.1.1 Identificação de Vulnerabilidades de Segurança . . . 81
6.1.2 Análise Automática da Segurança e Tomada de Decisões . . . 86
6.2 Análise do Indicador . . . 87
6.3 Ameaças à Validade do Estudo . . . 87
6.3.1 Ameaças à Validade do Constructo . . . 88
6.3.2 Ameaças à Validade Interna . . . 88
6.3.3 Ameaças à Validade Externa . . . 89
6.3.4 Ameaças à Confiabilidade . . . 89

7 CONCLUSÃO . . . 91
7.1 Resultados Obtidos . . . 91
7.2 Atividades Concluídas . . . 92
7.3 Objetivos Específicos . . . 92
7.4 Limitações . . . 93

7.5 Trabalhos Futuros 94

REFERÊNCIAS 95

1 Introdução

Neste capítulo são apresentados os conceitos fundamentais que norteiam este trabalho: segurança de produtos de *software*; Modelos de Qualidade; *DevOps* e; por fim; *DevSecOps*. Adicionalmente, são apresentados o escopo do problema, a questão de pesquisa que guia essa investigação e a estrutura geral do documento.

1.1 Contexto

Avaliar os fatores de qualidade de um *software* é de suma importância no desenvolvimento de *software* e, para isso, faz-se necessária a utilização de um modelo que guie a avaliação, de forma a sistematizar o processo e reduzir subjetividades (SIAVVAS et al., 2021a). Nesse sentido, o modelo proposto por McCall, Richards e Walters (1977a), foi o primeiro modelo proposto para analisar a qualidade de produto de *software*. Nesse trabalho, foram identificados aspectos e propriedades do produto de *software*, chamados de fatores e seus respectivos subfatores. Também foi proposto um método de avaliação quantitativa, baseada nos fatores, critérios e métricas. Nesse modelo, aspectos relacionados à segurança, foi percebido como um subfator de Integridade. Tratou-se portanto, de um modelo hierárquico. Subsequentemente, o modelo proposto por Boehm (1978a) evoluiu as ideias de McCall, e ambos se tornaram modelos seminais, servindo de referência para os modelos subsequentes. Além de propor novas propriedades e aspectos, renomeou os fatores e subfatores para características e subcaracterísticas. Essa terminologia é utilizada até hoje, em modelos de referência de qualidade contemporâneos.

A partir da ISO/IEC-9126 (2001), estabeleceu-se uma norma padrão internacional para analisar a qualidade de produto de *software*. Essa norma foi baseada nos modelos de McCall, Richards e Walters (1977a) e Boehm (1978a) além de outros, como por exemplo o modelo de Dromey (1995), que incorporava aspectos e propriedades específicas de código-fonte baseado no paradigma orientado a objetos (OO). Na ISO/IEC-9126 (2001) a segurança foi definida como uma subcaracterística de Funcionalidade. A ISO/IEC-25010 (2010) surgiu como uma evolução da ISO 9126, expandindo seus conceitos e adaptando o modelo para a realidade da qualidade de produto de *software* modernos. Uma relevante alteração foi transformação da até então, subcaracterística de qualidade para característica, destacando-a como um dos pilares da qualidade do produto. Além de reorganizar as características e suas respectivas subcaracterísticas na visão de qualidade interna, também foi incorporada ao modelo a visão da qualidade em uso. Tem-se ainda, a ISO/IEC-27001 (2022), que define requisitos para implementar, manter e melhorar continuamente, a segurança da informação em uma organização; além da ISO/IEC-27034 (2011), que fornece

diretrizes para as organizações sobre como integrar a segurança em suas aplicações, ao longo de todo o ciclo de vida de desenvolvimento, buscando mitigar as potenciais ameaças cibernéticas. Contudo, nenhuma dessas normas ISO/IEC, fornecem um método para avaliar a segurança de *software* de maneira quantitativa ou qualitativa. Assim, torna-se necessário recorrer a outros modelos e ferramentas que ofereçam uma abordagem para operacionalizar os conceitos abstratos definidos nessas normas, com suas respectivas fórmulas de cálculo, medidas e métricas, valores de referência, ponderação e agregação numérica, possibilitando o cálculo de indicadores quantitativos e qualitativos de segurança. Entretanto, é importante destacar que existem alguns *frameworks* e práticas que auxiliam a observação de aspectos técnicos da segurança, quanto organizacionais, como por exemplo:

- O modelo de maturidade *DevSecOps Maturity Model* (DSOMM) produzido pela *Open Web Application Security Project* (OWASP) (OWASP, 2020) e a lista das dez vulnerabilidades de maior ocorrência *OWASP Top 10* (OWASP, 2021);
- O catálogo de vulnerabilidades *Common Weakness Enumeration* (CWE), mantido por uma comunidade que envolve a indústria, academia e governo norte americano (Mitre Corporation, 1999)

Nos dias atuais, a segurança de sistemas e produtos de *software* é um dos fatores mais críticos. Os ataques cibernéticos explorando vulnerabilidades de produtos de *software* têm provocado impactos negativos, em ordem mundial. Recentemente, por exemplo, o React Team (2025) divulgou uma falha catalogada no sistema CVE *(Common Vulnerabilities and Exposures)*, lista padrão da indústria para riscos de segurança. Trata-se da CVE-2025-55182, que permite execução não autenticada de códigos-fonte remotamente em aplicações que utilizam a biblioteca React nas versões 19.0, 19.1.0, 19.1.1, and 19.2.0. Portanto, torna-se imperativo promover a cultura de desenvolvimento seguro, além de incorporar práticas e técnicas de segurança, que tornem as análises e decisões mais sistemáticas, ao longo de todo ciclo de desenvolvimento.

Nas últimas três décadas, a Engenharia de *Software* passou por uma profunda transformação na maneira de como se desenvolver produtos de *software*. Isso se deu por meio dos métodos ágeis, a filosofia de desenvolvimento *Lean* e práticas de comunidades de *software* livre (FITZGERALD; STOL, 2017) (LóPEZ et al., 2022). Atualmente, a cadência do ciclo de desenvolvimento acontece em um pequeno intervalo de tempo (dias, semanas). Isso fez com que entrega e implantação de versões de produtos de *software* passassem a ser disponibilizadas e implantadas continuamente. Além disso, os antigos sistemas "monolito", transformaram-se em subsistemas com responsabilidades específicas, especializadas e que se comunicam entre si. Assim, as versões dos produtos passaram a ser cada vez "menores"e independentes do ponto de vista da implantação. Para lidar com essa nova realidade de escala no desenvolvimento de produtos de *software*, é necessário dispormos de tecnologias

que nos auxiliem com a automação e sistematização das análises, com efeito na tomada de decisão sobre novas implantações.

Na esteira da transformação ágil, em 2009, Patrick Debois propôs o termo *DevOps*. Esse termo abarcou um conjunto de nuances que, em essência, buscava remover a barreira existente entre os times de desenvolvimento e operações, quando da implantação. A compreensão sobre esse termo passa por diferentes facetas como: cultura, automação; métodos e procedimentos de desenvolvimento e operações contínuos e integrados; colaboração (FRANçA; JUNIOR; TRAVASSOS, 2016) (DíAZ et al., 2022) (LUZ; PINTO; BONIFÁCIO, 2019). Um dos principais objetivos é reduzir o ciclo de vida do desenvolvimento de *software*, permitindo entregas mais frequentes e automatizadas ou semi-automatizadas. As principais práticas de *DevOps* são a Integração Contínua (CI), que consiste em integrar o código-fonte desenvolvido na ramificação principal com validação de *build* e testes de forma automática para detectar falhas, e a Entrega/Implantação Contínua (CD), que consiste em deixar o *software* pronto para entrar em produção e realizar o seu lançamento de forma automatizada (RAJAPAKSE et al., 2022).

Já o *DevSecOps* integra os princípios e práticas do *DevOps*, adicionando o time de segurança ao processo. Esse paradigma implementa uma abordagem de segurança chamada *Shift-Left*, na qual os processos de segurança são realizados desde o início do desenvolvimento, com o objetivo de evitar problemas decorrentes de uma avaliação tardia. Além disso, é formado por práticas de segurança como treinamento da equipe, testes de segurança automatizados e *feedback* contínuo (RAJAPAKSE et al., 2022).

1.2 Problema

Medir a segurança de *software* representa um grande desafio (RAJAPAKSE et al., 2022). Embora já exista muito conhecimento acumulado na área, ainda carecemos de modelos que apresentem formas sistematizadas de avaliação da segurança. Frequentemente, os métodos são baseados em critérios subjetivos, como a opinião de especialistas e não possuem validação empírica, o que afeta a confiabilidade dos resultados (SIAVVAS et al., 2021a).

No contexto do desenvolvimento e implantação contínua, esse desafio se torna ainda mais difícil. Métodos tradicionais de análise de segurança são impraticáveis devido à velocidade das entregas (RAJAPAKSE et al., 2022). A medição da segurança se torna ainda mais desafiadora ao lidar com ciclos contínuos de lançamentos de versões de produtos. A segurança é uma propriedade multifacetada, emergente e dependente do contexto, o que complica sua quantificação (KUDRIAVTSEVA; GADYATSKAYA, 2024). Em essência, a falta de compreensão sobre cultura e práticas de desenvolvimento contínuo e seguro, pode comprometer ou mesmo inviabilizar a estratégia de negócio de várias organizações

mundiais.

1.3 Questão de Pesquisa

A definição da questão de pesquisa foi elaborada utilizando a abordagem *Goal Question Metric* (GQM). Essa é uma abordagem que tem como objetivo definir, de maneira *top-down* e hierárquica, os objetivos a serem alcançados, as perguntas a serem respondidas para cumprir tais objetivos e as métricas necessárias para responder a cada pergunta de forma quantitativa, como mostra a Figura 1. Essa estrutura foi adaptada para o contexto da pesquisa, conforme o Quadro 1, resultando na seguinte questão:

Quadro 1: GQM Adaptado

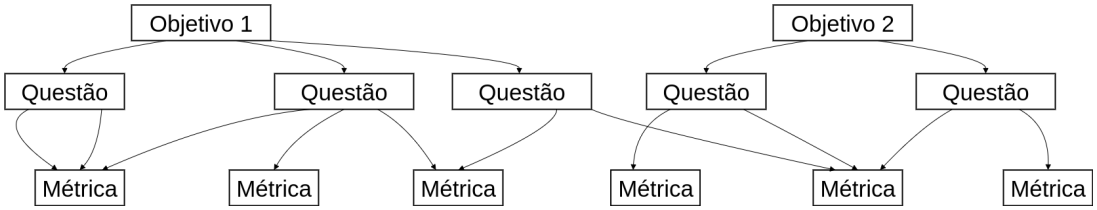
Característica	**Valor**
Analisar	a característica de qualidade de produto de *software* com foco na segurança e suas respectivas subcaracterísticas, nas visões interna e externa.
Com o propósito de	Caracterizar
Com respeito ao	desenvolvimento contínuo de produtos de *software* seguros (*DevSecOps*)
Do ponto de vista de	Pesquisador
No contexto do	desenvolvimento de aplicações *web* seguras e de código-fonte aberto

Fonte: Adaptado de Basili, Caldiera e Rombach (1994)

Considerando essa perspectiva definimos a seguinte questão principal de pesquisa deste estudo:

Como analisar a característica de segurança no desenvolvimento contínuo de sistemas web, considerando as visões de qualidade interna e externa?

Figura 1 – Abordagem GQM



Fonte: Adaptado de Basili, Caldiera e Rombach (1994)

1.4 Objetivos

O objetivo geral deste trabalho consiste em analisar a adoção de práticas de *DevSecOps* no ciclo de vida de desenvolvimento do produto de *software* livre Brasil Participativo[1]. Para alcançar este propósito, foram definidos os seguintes objetivos específicos:

- Fundamentar teoricamente os conceitos de *DevSecOps*, modelos de segurança e metodologias de desenvolvimento seguro.
- Incorporar um conjunto de práticas *DevSecOps* ao ciclo de desenvolvimento do produto de *software* sob análise.
- Planejar um estudo de caso focado na observação das práticas implementadas.
- Conduzir o estudo de caso, realizando a coleta e a análise das medidas e métricas de segurança, que apoiem a discussão dos resultados alcançados.
- Apresentar as conclusões e os *insights* resultantes desta investigação.

1.5 Estrutura do Trabalho

A seguir, são apresentados os capítulos que compõem a estrutura deste trabalho.

- Introdução: apresenta a contextualização do trabalho, o problema de pesquisa, a definição da questão de pesquisa e dos objetivos do trabalho. Por fim, descreve a estrutura das atividades realizadas.
- Referencial Teórico: estabelece a fundamentação teórica da monografia, abordando os tópicos centrais da pesquisa: *DevSecOps*, qualidade de *software*, segurança de *software* e modelos de avaliação de maturidade.
- Revisão Estruturada da Literatura: explicita o processo empregado para seleção que fundamentam este trabalho incluindo o protocolo de pesquisa e filtragem dos estudos e os resultados obtidos.
- Planejamento do Estudo de caso: descreve o protocolo utilizado para a condução do estudo de caso, detalhando seus objetivos, perguntas de pesquisa e atividades.
- Execução do Estudo de caso: apresenta a execução do planejamento, incluindo as etapas realizadas para a coleta e análise dos dados.
- Resultado do Estudo de Caso: explicita os resultados obtidos com a execução do estudo de caso, incluindo a análise dos dados coletados e a discussão dos achados.

[1] *Link* para o repositório do projeto

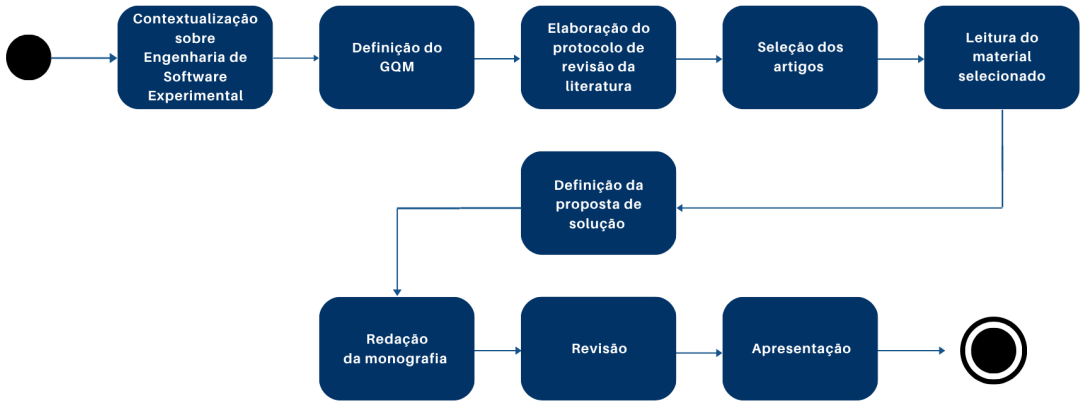
- Conclusão: consolida os achados obtidos parcial ou integralmente e como esses resultados respondem à questão de pesquisa principal, bem como as limitações do estudo e as possibilidades de aprofundamento de trabalhos futuros.

1.6 Cronograma e Atividades

1.6.1 Primeira Etapa

Esta subseção detalha as atividades desenvolvidas na primeira etapa da monografia. A Figura 2 ilustra o fluxo das atividades, enquanto a Figura 3 apresenta o cronograma correspondente.

Figura 2 – Atividades da Primeira Etapa



Fonte: Autor

- Contextualização sobre Engenharia de *Software* Experimental: Estudo sobre os métodos de pesquisa empírica em Engenharia de *Software*, como revisão sistemática da literatura, *survey*, experimentos e estudo de caso. Conceitos esses, fundamentais para a condução do trabalho proposto (WOHLIN et al., 2012).
- Definição do GQM: Aplicação da abordagem *Goal Question Metric* (GQM) (BASILI et al., 1994) para a construção da questão principal de pesquisa, suas subquestões e as medidas a serem analisadas.
- Elaboração do Protocolo de Revisão da Literatura: Estruturação e adaptação do protocolo de revisão sistemática da literatura proposto por Kitchenham, Charters

Figura 3 – Cronograma da Primeira Etapa



Fonte: Autor

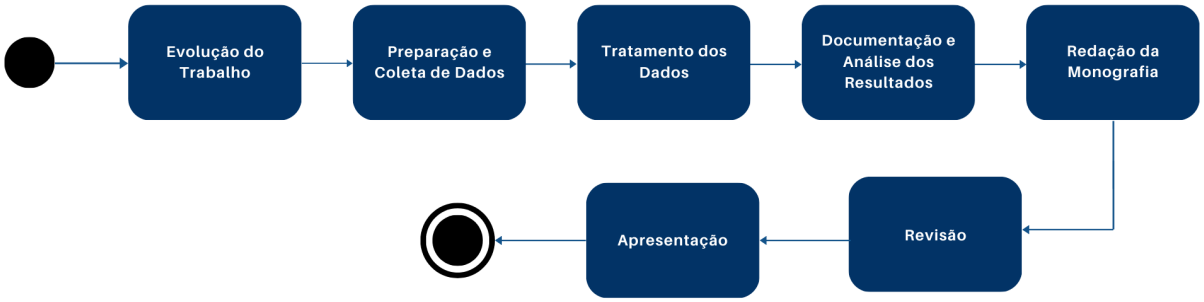
et al. (2007). Essa sistematização organizou nosso processo de revisão da literatura pertinente. Este processo inclui a definição da *string* de busca baseada no protocolo *Population, Intervention, Comparison, Outcome* (PICO), adaptado da área da medicina (PAI et al., 2004a). Por meio desse protocolo definimos sinônimos de termos, critérios de inclusão e exclusão de estudos, além do método de análise e extração de dados.

- Seleção dos Artigos: Execução da filtragem dos estudos por meio da leitura de títulos, resumos e palavras-chave, com a aplicação dos critérios de inclusão e exclusão definidos.
- Análise do Material Selecionado: Leitura completa dos artigos selecionados para aprofundar o conhecimento sobre o estado da arte em métodos de avaliação de segurança de sistemas *web* e práticas de desenvolvimento seguro.
- Definição da Proposta de Solução: Definição dos modelos de segurança, as ferramentas e as atividades do estudo de caso.
- Redação da Monografia: Escrita do texto da monografia, conforme a estrutura presente na Seção 1.5.
- Revisão: Realização das correções e dos ajustes solicitados pelo orientador.
- Defesa: Preparação do material e apresentação do trabalho final para a banca examinadora.

1.6.2 Segunda Etapa

Esta subseção, de forma semelhante à anterior, detalha as atividades realizadas na segunda etapa da monografia. A Figura 4 define o fluxo das atividades e a Figura 5 o cronograma.

Figura 4 – Atividades da Segunda Etapa



Fonte: Autor

Figura 5 – Cronograma da Segunda Etapa



Fonte: Autor

- Evolução do Trabalho: Revisão e aprimoramento do trabalho desenvolvido na primeira etapa, com base no *feedback* recebido durante a defesa. Além disso, foram adicionadas as atividades parcialmente concluídas na etapa anterior.
- Preparação e Coleta de Dados: Configuração do ambiente e das ferramentas necessárias para a coleta dos dados do estudo de caso.
- Tratamento dos Dados: Processamento e organização dos dados coletados, preparando-os para a análise.
- Documentação e Análise dos Resultados: Elaboração dos resultados, incluindo a análise dos dados e a discussão dos *insights* obtidos com a realização do estudo de caso.
- Redação da monografia: Continuação da escrita da monografia, de acordo com a estrutura da Seção 1.5.
- Revisão: Correção final do trabalho.
- Apresentação: Defesa final do trabalho concluído.

2 Referencial Teórico

O referencial teórico é composto pelos estudos e trabalhos que foram utilizados como base para a realização da pesquisa. Desse modo, este capítulo apresenta os conceitos-chave, teorias, modelos e metodologias que guiaram a execução do estudo de caso proposto.

2.1 Engenharia de *Software* Experimental

A engenharia de *software* é uma área bastante influenciada pelo comportamento humano, portanto, diferentemente de áreas como a física, não é possível a formulação de regras ou leis universais, exceto para certos aspectos estritamente técnicos. No entanto, é necessário realizar escolhas entre diferentes abordagens de desenvolvimento, ferramentas, técnicas, práticas e métodos, avaliar ou validar propostas já existentes. Assim, a engenharia de *software* experimental fornece as ferramentas necessárias para a realização de experimentos e análises, permitindo a validação de propostas e a comparação de diferentes abordagens com base científica. É através da engenharia de *software* experimental que se torna possível compreender e identificar a relação entre diferentes fatores ou variáveis, pois seus resultados são objetivos e estatisticamente significativos. Dessa forma, a engenharia de *software* experimental oferece uma forma sistemática, controlada e quantificável de avaliar as atividades da área, mesmo que elas sejam baseadas em comportamentos humanos (WOHLIN et al., 2024).

O estudo de caso é uma investigação empírica de um fenômeno contemporâneo, ou várias pequenas ocorrências dele, em seu contexto real. Esse tipo de estudo é adequado para avaliar métodos e ferramentas de engenharia de *software*, pois permite ao pesquisador observar o fenômeno em tempo real. Devido a esse contexto, seu planejamento, coleta e análise de dados seguem uma abordagem flexível, o que significa que essas etapas acontecem de forma iterativa e são evoluídas com a maturidade da pesquisa (WOHLIN et al., 2024).

Já as *surveys*, são pesquisas utilizadas para coletar informações sobre pessoas. Com elas é possível descrever, comparar ou explicar suas opiniões e comportamentos, possibilitando uma compreensão ampla do contexto em análise. Sua execução geralmente acontece por meio de questionários ou entrevistas, sendo os questionários a forma principal de coleta de dados. Eles podem coletar dados qualitativos, por meio de perguntas abertas, e quantitativos, usando perguntas fechadas, e têm como característica a possibilidade de alcance de um grande número de participantes, fornecendo um panorama geral. Por sua vez, as entrevistas envolvem o contato direto entre o pesquisador e os participantes, isso permite ao pesquisador fazer perguntas com alto grau de profundidade, o que leva a

uma compreensão mais minuciosa do fenômeno. Apesar disso, os custos para realizar uma pesquisa são significativamente maiores em relação ao questionário (WOHLIN et al., 2024).

Para fundamentar as bases de uma pesquisa baseada na engenharia de *software* experimental utilizam-se as revisões sistemáticas da literatura. O principal objetivo desse processo é identificar, analisar e interpretar as evidências disponíveis relacionadas à questão de pesquisa definida pelo pesquisador, de forma livre de viés e através de um protocolo que possibilite a reprodutibilidade. Para alcançar tal objetivo, é fundamental elaborar um planejamento da revisão, especificando as questões de pesquisa e o protocolo detalhado de revisão. Após essa etapa acontece a condução da revisão, etapa em que os estudos são identificados pela estratégia de busca definida, para que, posteriormente, seja feita a seleção do material com base nos critérios de inclusão e exclusão. Finalmente, a qualidade dos estudos é avaliada para que ao final da revisão os pesquisadores apresentem os achados obtidos com a revisão (WOHLIN et al., 2024).

2.1.1 Estudo de Caso

Um estudo de caso é uma metodologia de pesquisa empírica que se concentra na investigação de um fenômeno contemporâneo dentro de seu contexto da vida real, utilizando múltiplas fontes de evidência e bastante adequado para avaliação industrial de métodos e ferramentas de engenharia de *software* (WOHLIN et al., 2024).

Ao conduzir um estudo de caso, existem cinco etapas principais a serem percorridas: definição, preparação para a coleta de dados, coleta de evidências, análise de dados coletados e relato dos resultados (YIN, 2015).

2.1.1.1 Definição

Define os conceitos fundamentais do estudo de caso que compõem o planejamento do estudo a ser realizado.

- Objetivo: O que se pretende alcançar. Pode ser descobrir algo sobre algum fenômeno, retratar alguma situação, tentar aprimorar algum aspecto de fenômeno, entre outros. O objetivo tende a ser mais geral no início e evolui durante o estudo.
- Caso: O objeto de estudo. Pode ser um projeto de desenvolvimento de *software*, um grupo de pessoas, um processo, um produto, uma política, um papel na organização, uma tecnologia, entre outros.
- Trabalhos Relacionados: São as referências que constituem a base teórica do estudo. Eles são obtidos pela realização de uma revisão da literatura e ajudam o pesquisador a conduzir o estudo.

- Perguntas de Pesquisa: O que precisa ser conhecido para cumprir o objetivo do estudo. Assim como o objetivo, as perguntas de pesquisa evoluem e são refinadas durante o estudo.
- Métodos: As decisões principais sobre os métodos de coleta de dados são definidas nesta fase.
- Seleção: Escolha do caso e das unidades de análise que serão analisados.

2.1.1.2 Preparação e Coleta de Dados

Estabelece quais dados serão coletados e de que forma isso acontecerá. É necessário estabelecer a triangulação dos dados para aumentar a confiabilidade dos resultados obtidos.

- Primeiro grau: O pesquisador tem contato direto com os participantes.
- Segundo grau: O pesquisador coleta dados brutos diretamente sem interagir com os sujeitos durante a coleta, como o monitoramento automático de ferramentas de engenharia de *software*.
- Terceiro grau: O pesquisador utiliza dados já disponíveis e tratados.

2.1.1.3 Análise de Dados Coletados

Os dados coletados, sejam qualitativos ou quantitativos, são analisados visando obter informações relevantes para o estudo de caso.

- Análise de Dados Quantitativos: Inclui estatísticas descritivas, análise de correlação, desenvolvimento de modelos preditivos e teste de hipóteses.
- Análise de Dados Qualitativos: Visa extrair conclusões a partir dos dados.

2.1.1.4 Relato dos Resultados

Aqui são apresentados os resultados que os pesquisadores conseguiram obter com a aplicação do estudo de caso.

- Contexto: apresenta a visão geral do projeto e informações pertinentes relacionadas ao ambiente onde o estudo de caso foi realizado.
- Procedimentos: aborda as etapas realizadas ao decorrer do estudo.
- Resultados: apresenta os resultados de forma clara e com evidências sólidas.

- Discussão: os resultados são interpretados e são apresentadas as limitações do estudo e suas implicações.

- Conclusões: Sintetiza os achados da pesquisa.

2.2 Modelos de Qualidade de *Software*

O modelo de McCall, Richards e Walters (1977b) é considerado pioneiro na avaliação da qualidade de *software*. O modelo propõe uma estrutura hierárquica para avaliar a qualidade do *software* a partir de fatores de qualidade, ou seja, ele permite que a qualidade de um determinado sistema seja avaliada de maneira mensurável. Os fatores são agrupados nas categorias principais de operação, revisão e transição, porém cada categoria de fator é decomposta em vários subfatores para tornar possível a sua análise. A categoria de operação está relacionada com os fatores referentes ao uso efetivo do *software*, já a revisão está ligada à capacidade do *software* de ser mantido e modificado, por fim, a transição à adaptação do *software* a novos ambientes (MCCALL; RICHARDS; WALTERS, 1977b).

Posteriormente, Boehm (1978b) expandiu o modelo de McCall, Richards e Walters (1977b). Ele propôs uma abordagem hierárquica, mas com uma estrutura e terminologia diferente, terminologia essa que se tornou o padrão na engenharia de *software*. Os fatores de McCall, Richards e Walters (1977b), agora chamados de características de alto nível, que representam o valor do *software*, características de nível intermediário, que detalham as características de alto nível, e as características primitivas, que podem ser avaliadas diretamente no código-fonte (BOEHM, 1978b).

Com o avanço da engenharia de *software*, novos modelos de qualidade foram propostos. Entre eles, destacam-se o modelo de Dromey (1995), que enfatiza a importância de vincular as características de qualidade diretamente aos componentes do *software*. Ele utiliza a abordagem *bottom-up*, ou seja, os componentes de *software* de baixo nível determinam diretamente as características de qualidade de alto nível do sistema (DROMEY, 1995).

No processo de desenvolvimento de *software*, é fundamental garantir o uso consistente e correto de terminologias, métricas e métodos para a avaliação de qualidade. Nesse sentido, a norma ISO/IEC 25010 (2023) se estabelece como o documento que determina o padrão internacional que orienta a avaliação de sistemas de *software*. Esta norma integra a família ISO/IEC 25000 (2014), conhecida como SQuaRE (Requisitos e Avaliação de Qualidade de Sistemas e *Software*). O objetivo da série SQuaRE é estruturar a avaliação da qualidade de *software* por meio de um modelo de qualidade hierárquico e adaptável ao contexto de cada organização (ISO/IEC 25010, 2023).

A ISO/IEC 25010 (2023) substitui a versão anterior do modelo de qualidade, a

ISO/IEC 9126 (2001), e define a qualidade de um produto de *software* com base em oito características principais, que por sua vez são subdivididas em subcaracterísticas. As características de qualidade de produto são: adequação funcional, desempenho, compatibilidade, usabilidade, confiabilidade, segurança, manutenibilidade e portabilidade. Adicionalmente, a norma também define as características de qualidade em uso, que avaliam o impacto do produto da perspectiva do usuário. São elas: eficácia, eficiência, satisfação, liberdade de risco e contexto de cobertura (ISO/IEC-25010, 2010).

O modelo SQALE proposto por (LETOUZEY; COQ, 2010), inspirado por McCall, Richards e Walters (1977b) e pela ISO/IEC 9126 (2001), deu um passo além de apenas medir, pois ele fornece um caminho para aumentar a qualidade do *software*. Para atingir esse objetivo, o SQALE define um conjunto de práticas que orientam a avaliação e a melhoria da qualidade do *software*. Elas fornecem diretrizes técnicas aos desenvolvedores apontadas por um sistema de pontuação adaptável que enfatiza com notas baixas as partes mais problemáticas do sistema (LETOUZEY; COQ, 2010).

O Quamoco é um metamodelo desenvolvido por Wagner et al. (2012) baseado na ISO/IEC 25010 (2023) que utiliza um método de avaliação de qualidade baseado na Teoria de Utilidade de Múltiplos Atributos/Valores (MAUT/MAVT). Por ser um metamodelo, ele permite que sejam construídos modelos adaptados à realidade de uma organização, em vez de aplicar uma análise rígida como visto nos outros modelos. Ademais, o modelo possui validação empírica em relação à detecção de diferenças de qualidade e visibilidade de melhorias ao longo do tempo (WAGNER et al., 2012).

Buscando atender às necessidades do desenvolvimento moderno López et al. (2018) desenvolveram o modelo Q-Rapids. Apesar de baseado no Quamoco, ele incorpora aspectos adicionais relacionados ao desenvolvimento ágil e *DevOps*. O modelo Q-Rapids integra dados de diversas fontes, como repositórios de código-fonte, ferramentas de análise estática, sistemas de rastreamento de defeitos, ferramentas de integração contínua e o resultado de testes para fornecer uma visão abrangente da qualidade do *software* ao longo do ciclo de vida do desenvolvimento (LóPEZ et al., 2018).

Também baseado na norma ISO/IEC 25010 (2023), o modelo QATCH Siavvas, Ampatzoglou e Stamelos (2017) fornece uma abordagem de avaliação de qualidade de *software* que leva em conta a natureza subjetiva da qualidade de *software*. Segundo os autores, o modelo deve ser capaz de ser ajustado às necessidades específicas dos *stakeholders*, permitindo a personalização dos critérios de avaliação e a adaptação às particularidades do contexto em que o *software* será utilizado. Para isso, ele utiliza análise estática em repositórios de referência e opinião de especialistas para definir os limiares e pesos (SIAVVAS; AMPATZOGLOU; STAMELOS, 2017).

2.3 Segurança de *Software*

A segurança de *software* é um aspecto fundamental da qualidade de *software*, sendo um dos atributos principais do modelo de qualidade ISO/IEC-25010 (2010). Ela é definida como o grau em que um sistema ou produto de *software* protege informações e dados, armazenados ou em trânsito, de forma a impedir o acesso não autorizado por diferentes entes, sejam pessoas, programas ou outros sistemas, de acordo com os seus respectivos níveis de permissão, assim contribuindo para a confiança do sistema (ISO/IEC-25010, 2010).

Para possibilitar a sua análise detalhada, a ISO/IEC-25010 (2010) define cinco subcaracterísticas de segurança fundamentais. A primeira delas é a confidencialidade, que se refere à capacidade do sistema de proteger informações contra acessos não autorizados. A segunda é a integridade, que diz respeito à capacidade do sistema de proteger informações contra modificações não autorizadas. A terceira subcaracterística é o não-repúdio, que se refere à capacidade do sistema de garantir que as ações realizadas por um usuário ou sistema não possam ser negadas posteriormente. A quarta subcaracterística é a responsabilidade, que diz respeito à capacidade do sistema de rastrear e registrar as ações realizadas por usuários ou sistemas. Por fim, a quinta subcaracterística é a autenticidade, que se refere à capacidade do sistema de garantir que os usuários ou sistemas são quem dizem ser.

Por outro lado, a família de normas ISO/IEC 27000 (2018) é um conjunto de padrões internacionais que fornecem diretrizes e melhores práticas para a gestão da segurança da informação. Elas definem termos e definições, estabelecem requisitos para um sistema de gestão de segurança da informação (SGSI) e fornecem orientações para a implementação e manutenção de controles de segurança da informação.

Desse modo, a norma ISO/IEC 27001 (2022) é a referência internacional para gestão da segurança da informação. Ela especifica os requisitos para estabelecer, implementar, manter e melhorar continuamente um SGSI dentro do contexto organizacional, ajudando a proteger as informações de maneira eficaz. A norma também define confidencialidade, integridade e disponibilidade como os três pilares da segurança da informação. Segundo a ISO/IEC 27000 (2018), a confidencialidade garante que as informações sejam acessíveis apenas por pessoas autorizadas, a integridade assegura que as informações sejam precisas e completas, e a disponibilidade garante que as informações estejam acessíveis quando necessário.

Atualmente, monitorar, medir, analisar e avaliar a eficácia da segurança da informação é uma tarefa essencial para a engenharia de *software*. Visando atender essa necessidade, a ISO/IEC 27004 (2016) estabelece como realizar esses procedimentos, além de fornecer orientações para verificar o desempenho dos mesmos. Com o apoio da norma

ISO/IEC 27004 (2016), as organizações conseguem desenvolver e selecionar métricas, estabelecer processos de medição e avaliação, e implementar melhorias contínuas na segurança da informação.

A OWASP (*Open Web Application Security Project*) é uma fundação internacional sem fins lucrativos que tem como objetivo conscientizar desenvolvedores, promover boas práticas e melhorar a segurança de *software* através da elaboração de documentos, ferramentas, *frameworks* e pesquisas disponibilizadas de maneira gratuita (OWASP Foundation, 2025). Um dos documentos mais importantes produzidos pela OWASP é o OWASP Top 10, que lista as dez principais vulnerabilidades de segurança em aplicações *web*. A lista é atualizada periodicamente e serve como um guia para desenvolvedores e profissionais de segurança, facilitando identificar e mitigar os riscos comuns nos sistemas de *software* (OWASP, 2021).

De maneira complementar, o CWE (*Common Weakness Enumeration*) feito pela Mitre Corporation (1999), é uma lista categorizada de tipos de fraquezas de *software* e *hardware* que estabelece uma linguagem padrão para descrever as causas das vulnerabilidades. Já o CVE (*Common Vulnerabilities and Exposures*) é um dicionário público de vulnerabilidades de segurança conhecidas, também mantido pela (The MITRE Corporation, 1999), descreve as vulnerabilidades encontradas nos sistemas de maneira individual. Por último, o CVSS (*Common Vulnerability Scoring System*) é um padrão aberto para avaliar a gravidade das vulnerabilidades de segurança em sistemas de *software*, fornecendo uma pontuação numérica que ajuda as organizações a priorizar suas ações de mitigação, por exemplo, as vulnerabilidades são enumeradas de zero à dez e acompanhadas de um nível de severidade (Nenhum, Baixo, Médio, Alto, Crítico) (Forum of Incident Response and Security Teams (FIRST), 2015).

2.4 *DevOps*

A origem do termo e do movimento *DevOps* remonta a dois eventos distintos no ano de 2009. O primeiro deles aconteceu em julho daquele ano, quando John Allspaw e Paul Hammond apresentaram a palestra "10 *deploys* por dia: cooperação entre Dev & Ops no Flickr"na conferência Velocity. Poucos meses depois, Patrick Debois organizou a conferência "*DevOps Days*"em Ghent, Bélgica, que é considerada o primeiro evento oficial do movimento *DevOps*. Desde então, o termo e o movimento *DevOps* ganharam popularidade e se tornaram uma abordagem amplamente adotada para o desenvolvimento e operação de *software* (Google, 2024).

Em suma, *DevOps* é um conjunto de práticas e valores culturais que visa encurtar o ciclo de vida de desenvolvimento de *software*, aumentar a qualidade do *software* produzido e facilitar a sua evolução contínua. Para alcançar esses objetivos, o *DevOps*

promove a colaboração e a comunicação entre as equipes de desenvolvimento e operações (ALJOHANI; ALQAHTANI, 2023).

Sinan, Shahin e Gondal (2025) definem como práticas basilares a automação e o monitoramento. A automação é essencial para realizar tarefas repetitivas de forma eficiente e consistente, reduzindo erros humanos e aumentando a velocidade de entrega. Já o monitoramento é fundamental para garantir o funcionamento correto, desempenho e segurança dos sistemas em produção, permitindo a identificação e resolução rápida de problemas. Ademais, essas duas práticas se beneficiam mutuamente, uma vez que o monitoramento possibilita observar o comportamento da automação e a automação permite que o processo de monitoramento seja mais eficiente.

Segundo Sinan, Shahin e Gondal (2025), o *pipeline* de CI/CD é a implementação prática dos princípios de automação e monitoramento. Ele funciona como uma esteira de produção de *software*, onde o código-fonte é continuamente integrado, testado e entregue de forma automatizada. O *pipeline* de CI/CD é composto por duas etapas principais, a Integração Contínua (CI) e a Entrega Contínua (CD).

A primeira fase é a Integração Contínua, aqui o código-fonte é frequentemente integrado em um repositório central compartilhado, onde é automaticamente testado para garantir que ele funcione corretamente. A etapa seguinte é a Entrega Contínua. Nessa fase, o código-fonte que passou pelos testes na etapa de CI é automaticamente implantado no ambientes de produção e entregue aos usuários finais (SINAN; SHAHIN; GONDAL, 2025).

2.5 *DevSecOps*

Nos modelos tradicionais de desenvolvimento de *software*, a segurança é frequentemente tratada como uma etapa separada, realizada por uma equipe isolada e realizada no fim do ciclo de vida do desenvolvimento. Isso pode levar a atrasos na entrega do *software*, aumento dos custos e vulnerabilidades de segurança não detectadas. Essa situação é ainda mais acentuada em ambientes *DevOps*, onde a velocidade e a integração contínua são priorizadas e essa metodologia acaba representando um gargalo no processo de desenvolvimento (SINAN; SHAHIN; GONDAL, 2025).

Para superar esses desafios, surgiu o conceito de *DevSecOps*, que integra práticas de segurança diretamente no ciclo de vida do desenvolvimento de *software*. O objetivo do *DevSecOps* é garantir que a segurança seja uma responsabilidade compartilhada entre as equipes de desenvolvimento, operações e segurança, promovendo uma cultura de colaboração e comunicação contínua (SINAN; SHAHIN; GONDAL, 2025).

Assim, a implementação de práticas *DevSecOps* consiste na inserção de ferramentas e processos de segurança em diferentes etapas do *pipeline* de CI/CD. Essa prática ficou

conhecida como deslocar a segurança para a esquerda (*shift-left security*), que significa incorporar a segurança já nas fases iniciais do desenvolvimento.

Como primeira barreira de defesa, a análise estática de código-fonte (*Static Application Security Testing* - SAST) é utilizada para identificar vulnerabilidades diretamente no código-fonte, abordagem conhecida como caixa-branca. Ferramentas SAST analisam o código-fonte em busca de padrões inseguros, práticas de codificação inadequadas e possíveis falhas de segurança.

Além disso, a análise dinâmica de código-fonte (*Dynamic Application Security Testing* - DAST) é empregada para testar a aplicação em execução, simulando ataques reais e identificando vulnerabilidades que podem ser exploradas por agentes maliciosos. Essa abordagem é conhecida como caixa-preta, pois ela testa a aplicação em execução (SINAN; SHAHIN; GONDAL, 2025).

O modelo de maturidade de segurança OWASP DSOMM (*DevSecOps Maturity Model*) é uma importante ferramenta na construção e avaliação de projetos *DevSecOps*. Por ter sido construído com base na norma ISO/IEC 27001 (2022), ele define atividades, métricas e tecnologias que devem ser usadas para construir um ambiente *DevSecOps*, além de proporcionar o acompanhamento da maturidade da segurança do projeto em cinco dimensões: *Build and Deployment*, *Culture and Organization*, *Implementation*, *Information Gathering* e *Test and Verification* (LANGE; KUNZ, 2024).

3 Revisão Estruturada da Literatura

Este capítulo destina-se a documentar o processo realizado para selecionar o conjunto de artigos científicos que compõe a bibliografia desta monografia. Para isso, a base de dados Scopus foi escolhida devido à sua característica de indexar diversos artigos da área da computação, muitos publicados nos principais meios de divulgação científica (ELSEVIER, 2024).

3.1 Protocolo

O protocolo utilizado para realizar a revisão estruturada da literatura foi baseado no modelo proposto por Kitchenham, Charters et al. (2007). Seu objetivo é tornar possível que outros pesquisadores, partindo do mesmo ponto, cheguem aos mesmos resultados, facilitando a replicabilidade em estudos futuros e permitindo a conferência dos resultados obtidos.

3.1.1 *String* de Busca

A busca em bases de dados acadêmicas requer o uso de um protocolo, pois elas indexam grande quantidade de artigos de várias áreas do conhecimento. Seu uso incorreto pode acarretar em um número excessivo de artigos sem relação com o tema da pesquisa ou, inversamente, retornar um volume insuficiente de estudos para responder à questão de pesquisa.

Por essa razão, o protocolo PICO foi utilizado para guiar a elaboração de uma *string* de busca adequada às necessidades desta monografia. O protocolo, no entanto, precisou ser adaptado, pois sua origem é na medicina e nem todos os seus elementos se adequam à área da Engenharia de *Software* (PAI et al., 2004a). Uma representação visual que facilita a compreensão do protocolo pode ser vista na Figura 6.

Figura 6 – Protocolo PICO



Fonte: Adaptado de Pai et al. (2004b)

Ao adaptar o modelo PICO para a Engenharia de *Software*, suas definições adquirem um novo significado no contexto da Engenharia de *Software*. Por exemplo, o termo **Paciente**, outrora usado para indicar o perfil do paciente, passa a representar a área de aplicação, ou amostra, neste caso, o desenvolvimento de *software*. A **Intervenção** também sofre adaptação, deixando de significar "tratamento médico"para se referir às diferentes metodologias, métodos, técnicas ou procedimentos avaliados. Não obstante, **Resultados** mantém seu sentido original, referindo-se aos efeitos ou consequências observadas. Por fim, a **Comparação**, que está relacionada a investigar como a intervenção proposta se relaciona com outras propostas de intervenção, não pôde ser adotada, devido a estar muito mais alinhada com os objetivos da medicina que da engenharia de *software*. Assim, as definições de cada um dos elementos usados do modelo estão apresentadas no Quadro 2.

Quadro 2: PICO Adaptado

Elementos	**Termo Central**	**Sinônimos e Termos Relacionados**
População	*software development*	*software systems, online systems, software applications, systems development, application development*
Intervenção	*DevSecOps*	*cybersecurity practices, security automation, secure software development, CI/CD, continuous integration, continuous deployment, continuous delivery, DevOps, security development culture*
Resultados	*security quality*	*software security, application security, security improvement, security assurance, vulnerability reduction, protection against threats, system security, owasp, cwe, common weakness enumeration, security quality characteristic, injection, cross site request forgery, XSRF, broken authentication, data exposure, external entities, broken access control, XSS, XXE, Out-of-bounds Write, SQL Injection, CSRF, Path Traversal, Directory Traversal, OS Command Injection, UAF, ISO 25010*

Fonte: Autor

Desta maneira, a *string* de busca foi construída usando o operador lógico *OR* entre os termos de cada elemento, com o objetivo de englobar todos os termos da pesquisa. Já o operador *AND* foi usado para conectar os diferentes elementos do protocolo PICO, assim restringindo a busca apenas aos estudos que apresentam os termos necessários para responder às perguntas de pesquisa. Ademais, foram adicionados os termos próprios da base de dados Scopus para a realização da consulta, resultando na seguinte *string* de busca:

TITLE-ABS-KEY (("software development"OR "software developments"OR "software system"OR "software systems"OR "online system"OR "online systems"OR "software

application"OR "software applications"OR "system development"OR "systems development"OR "application development"OR "applications development") AND ("DevSecOps"OR "cybersecurity practice"OR "cybersecurity practices"OR "security automation"OR "security automations"OR "secure software development"OR "secure software developments"OR "CI/CD"OR "continuous integration"OR "continuous integrations"OR "continuous deployment"OR "continuous deployments"OR "continuous delivery"OR "continuous deliveries"OR "DevOps"OR "security development culture"OR "security development cultures") AND ("security quality"OR "software security"OR "application security"OR "security improvement"OR "security assurance"OR "vulnerability reduction"OR "protection against threats"OR "system security"OR "owasp"OR "cwe"OR "common weakness enumeration"OR "security quality characteristic"OR "injection"OR "cross site request forgery"OR "XSRF"OR "broken authentication"OR "data exposure"OR "external entities"OR "broken access control"OR "XSS"OR "XXE"OR "Out-of-bounds Write"OR "SQL Injection"OR "CSRF"OR "Path Traversal"OR "Directory Traversal"OR "OS Command Injection"OR "UAF"OR "ISO 25010"))

3.2 Seleção dos Artigos

Com a *string* de busca criada, foram estabelecidos os critérios de inclusão e exclusão visando selecionar apenas os artigos mais pertinentes ao contexto desta pesquisa. Além disso, em um primeiro momento, foram lidos o título, resumo e palavras-chave de todos os artigos resultantes da execução da *string* de busca; isso se deu para selecionar com maior rigor aqueles estudos que porventura não correspondessem ao objetivo do trabalho. Posteriormente, os artigos aprovados pelos critérios de escolha foram lidos e os dados relevantes para responder às perguntas de pesquisa secundárias foram extraídos em um formulário. O Quadro 3 contém o protocolo completo.

Ao todo foram analisados 291 artigos resultantes da *string* de busca, onde 38 desses foram selecionados e lidos de maneira integral. Esses artigos foram obtidos na base de dados Scopus no dia 7 de maio de 2025. A fim de complementar os artigos selecionados por meio da *string* de busca, foi realizada uma busca manual com o objetivo de responder de maneira mais assertiva às perguntas de pesquisa, resultando em mais dois artigos selecionados, *Accelerate State of DevOps 2024* (Google, 2024) e *Quantitative DevSecOps Metrics for Cloud-Based Web Microservices* (ZHANG; ZHANG, 2024), usados para compor o referencial teórico, totalizando 40 artigos. Os artigos selecionados estão dispostos no Quadro 4. Para facilitar a compreensão do protocolo de seleção de artigos pode-se observar a Figura 7, que ilustra todas as etapas explanadas anteriormente.

Quadro 3 – Protocolo de Busca

Item	**Descrição**
Questões Secundárias de Pesquisa	1. Como o aspecto de segurança no desenvolvimento contínuo de sistemas *web* pode ser analisado considerando perspectivas de qualidade interna e externa? 2. Quais práticas de *DevSecOps* são mais prevalentes nas organizações? 3. Quais modelos de segurança são mais comumente empregados em ambientes *DevSecOps*? 4. Quais medidas são comumente usadas para avaliação de segurança? 5. Como as práticas de segurança são avaliadas? 6. De que maneiras as medidas de segurança são avaliadas nas organizações? 7. Quais tipos de vulnerabilidades são mitigados pelas práticas de *DevSecOps*? 8. Quais ferramentas e tecnologias são utilizadas em *DevSecOps*? 9. Qual é o volume anual de publicações sobre *DevSecOps* de 2009 a 2025? 10. Quantos artigos foram publicados em revistas acadêmicas? 11. Quantos estudos possuem validação experimental? 12. Se o estudo possui validação experimental, de que tipo?
String de Busca	Vide Tabela 2
Critérios de Inclusão	1. Aborda o uso de práticas de desenvolvimento seguro 2. Enfatiza a qualidade de segurança de sistemas *web* 3. Avalia modelos de qualidade com foco em segurança 4. Concentra-se em produtos de *software* 5. Inclui validação experimental
Critérios de Exclusão	1. Artigos em idiomas diferentes de inglês ou português 2. Publicações duplicadas 3. Publicações com data de lançamento anterior a 2009 4. Estudos com foco em *hardware*, dispositivos móveis, segurança de IoT ou outros tópicos não relacionados a sistemas *web*

Quadro 3 – Continuação

Item	**Descrição**
Formulário de Extração	1. Título 2. Resumo 3. Ano de Publicação 4. Fonte de Publicação 5. Autores 6. Palavras-chave 7. Práticas de *DevSecOps* Prevalentes 8. Modelos de Segurança Empregados 9. Medidas de Avaliação de Segurança 10. Avaliação de Práticas de Segurança 11. Análise de Modelos de Segurança 12. Avaliação de Segurança Organizacional 13. Vulnerabilidades Mitigadas 14. Ferramentas e Tecnologias 15. Publicado em Revista Acadêmica 16. Validação Experimental 17. Tipo de Validação Experimental 18. Pesquisa Secundária

Fonte: Autor

Quadro 4 – Artigos Selecionados

№ N	**Título**	**Publicado em Revista**	**Validação Experimental**
1	Development of Secure Software Based on the New DevSecOps Technology	Sim	Não
2	Automating Security in a Continuous Integration Pipeline	Não	Não
3	Extensive Review of Threat Models for DevSecOps	Sim	Não
4	Implementing and Automating Security Scanning to a DevSecOps CI/CD Pipeline	Sim	Não
5	Automating Static Code Analysis Through CI/CD Pipeline Integration	Sim	Sim
6	Design and Practice of Security Architecture via DevSecOps Technology	Sim	Sim

Quadro 4 – Continuação

№	**Título**	**Publicado em Revista**	**Validação Experimental**
7	Implementation of DevSecOps by Integrating Static and Dynamic Security Testing in CI/CD Pipelines	Sim	Não
8	Research of Static Application Security Testing Technique Problems and Methods for Solving Them	Sim	Não
9	A Large-scale Fine-grained Empirical Study on Security Concerns in Open-source Software	Sim	Sim
10	Evolution of secure development lifecycles and maturity models in the context of hosted solutions	Não	Não
11	Automation and DevSecOps: Streamlining Security Measures in Financial System	Sim	Não
12	Securing the development and delivery of modern applications	Sim	Não
13	You cannot improve what you do not measure: A triangulation study of software security metrics	Sim	Sim
14	On DevSecOps and Risk Management in Critical Infrastructures: Practitioners' Insights on Needs and Goals	Sim	Sim
15	Container Security in Cloud Environments: A Comprehensive Analysis and Future Directions for DevSecOps	Não	Sim
16	Microservices-based DevSecOps Platform using Pipeline and Open Source Software	Não	Não
17	Securing the Digital Frontier: A Proactive Approach to Software Development	Sim	Não
18	A Secure Software Development Methodology for Enterprise Business Applications	Sim	Sim
19	Building Resilient CICD Pipelines: A DevOps Security-First Framework	Sim	Não
20	Review of Techniques for Integrating Security in Software Development Lifecycle	Não	Não

Quadro 4 – Continuação

№	**Título**	**Publicado em Revista**	**Validação Experimental**
21	A hierarchical model for quantifying software security based on static analysis alerts and software metrics	Sim	Sim
22	A preventive secure software development model for a software factory: A case study	Sim	Sim
23	Security impacts of sub-optimal DevSecOps implementations in a highly regulated environment	Sim	Sim
24	A survey and comparison of secure software development standards	Sim	Sim
25	Continuous Security Testing: A Case Study on Integrating Dynamic Security Testing Tools in CI/CD Pipelines	Sim	Sim
26	Infiltrating Security into Development: Exploring the World's Largest Software Security Study	Não	Sim
27	Challenges and solutions when adopting DevSecOps: A systematic review	Sim	Não
28	Systematic Mapping Study on Security Approaches in Secure Software Engineering	Sim	Não
29	Systematic Literature Review on Security Risks and its Practices in Secure Software Development	Sim	Não
30	BP: Security concerns and best practices for automation of software deployment processes: An industrial case study	Sim	Sim
31	Static analysis for web service security - Tools & techniques for a secure development life cycle	Sim	Não
32	Security characterization for evaluation of software architectures using ATAM	Sim	Sim
33	Software security	Sim	Não
34	Using the ISO/IEC 27034 as reference to develop an application security control library	Sim	Sim

Quadro 4 – Continuação

№	Título	Publicado em Revista	Validação Experimental
35	Hunting for aardvarks: Can software security be measured?	Não	Não
36	Francois Raynaud on DevSecOps	Sim	Não
37	Integrating application security into software development	Sim	Não
38	Busting a myth: Review of agile security engineering methods	Sim	Não
39	Accelerate State of DevOps 2024	Não	Sim
40	Quantitative DevSecOps Metrics for Cloud-Based Web Microservices	Sim	Não

Fonte: Autor

Figura 7 – Seleção dos Artigos



Fonte: Autor

3.3 Resultados

Após a leitura dos artigos, os dados foram extraídos utilizando o formulário de extração da plataforma Parsifal. Essa plataforma foi usada para conduzir a revisão da literatura devido ao seu conjunto de funcionalidades que permitem executar a revisão da literatura de forma eficiente. Depois de realizada a extração dos dados, os mesmos

foram consolidados em uma planilha [1] disponível publicamente, sendo que as suas linhas representam cada um dos artigos selecionados e as colunas representam os campos do formulário de extração anteriormente apresentado na Tabela 3.

Nas subseções a seguir são apresentadas as respostas para as questões secundárias de pesquisa, específicas desta revisão.

3.3.1 Quais práticas de *DevSecOps* são mais prevalentes nas organizações?

- Metodologias e Cultura
  - *Shift Security Left*: É o princípio fundamental do *DevSecOps*. Ele consiste em deslocar as práticas de segurança para o início do processo de desenvolvimento do projeto, em vez de deixá-las para o final, como acontece na maioria dos casos. Essa postura eleva a prioridade da segurança no projeto, permitindo que as vulnerabilidades sejam tratadas mais cedo (RAJAPAKSE et al., 2022).
  - *Continuous Vulnerability Assessment*: Juntamente com *Shift Left*, a Avaliação Contínua de Vulnerabilidades forma os pilares do *DevSecOps*. Nessa prática a segurança do *software* é continuamente verificada, não somente durante o desenvolvimento, mas também após a implantação do *software*, pois é necessário monitorar a segurança do sistema durante todo o seu ciclo de vida (RAJAPAKSE et al., 2022).
  - CI/CD: *Continuous Integration* (CI) e *Continuous Delivery/Deployment* (CD) integram o *DevSecOps*, que se trata de uma evolução do *DevOps*. Como esses conceitos são herdados, é importante defini-los para o completo entendimento da metodologia. CI se refere à prática de realizar continuamente integrações de código-fonte na *branch* principal do código-fonte. Como essa atividade envolve a alteração da ramificação principal, ela é validada por meio de verificadores de *build* e testes automatizados. Já o CD visa à implantação automática e contínua das atualizações de código-fonte no ambiente de produção. Para isso, o novo código-fonte passa por verificações de qualidade e, se tudo estiver de acordo com a política estabelecida, é publicado sem intervenção humana por meio do *pipeline* (RAJAPAKSE et al., 2022).
  - *Continuous Feedback*: Obter opinião de maneira contínua e rápida é vital em ambientes de entrega contínua. Nessa abordagem, os problemas são rapidamente identificados, possibilitando que as informações cheguem depressa às equipes para que as medidas necessárias sejam tomadas. Os métodos tradicionais de coleta de dados e opinião são lentos demais para a agilidade requisitada

[1] Planilha com o resultado da revisão: <https://docs.google.com/spreadsheets/d/1sGjU2aB8CTqvINyPxsv-z_gWZCGzmErr/edit?usp=sharing&ouid=103637687294157542932&rtpof=true&sd=true>

em ambientes *DevSecOps*, e essa lentidão afeta diretamente a velocidade de localização e resolução dos problemas (RAJAPAKSE et al., 2022).

- Testes de Segurança
  - SAST, DAST, IAST: Ferramentas SAST executam testes estáticos, ou seja, são realizados apenas sobre o código-fonte, podendo indicar potenciais vulnerabilidades e más práticas, conhecidas como *code smells*. Por outro lado, as ferramentas DAST analisam o *software* em execução, o que possibilita testar o comportamento do sistema em uso. As ferramentas de *Interactive Application Security Testing* (IAST), por sua vez, fornecem uma abordagem híbrida, incorporando características da análise estática e dinâmica; são modernas e possuem boa integração com ambientes de desenvolvimento contínuo (RAJAPAKSE et al., 2022).
  - *Fuzz Testing*: O teste de *fuzzing*, como também é conhecido, destaca-se entre as práticas de teste de segurança. Ele consiste em fornecer à aplicação entradas aleatórias, inválidas e inesperadas, de forma a verificar possíveis casos de borda não testados (MASOOD; JAVA, 2015).
  - BDST: Os testes BDST *Behavior Driven Security Testing* são baseados em *Behavior Driven Development* (BDD), porém aplicados ao contexto de testes de segurança. Como os testes descrevem seu comportamento em linguagem natural, pessoas de fora da área de desenvolvimento e segurança de *software* conseguem compreender os testes realizados (RANGNAU et al., 2020).
- Infraestrutura
  - IaC: Essa prática consiste em definir a infraestrutura do projeto como código. É altamente usada em contextos *DevSecOps*, pois possibilita a rápida configuração da infraestrutura. Como está definida em código, pode-se realizar seu versionamento, teste e implantação de forma ágil e adequada a ambientes complexos que necessitam de segurança robusta (RAJAPAKSE et al., 2022).
- Acompanhamento
  - *Monitoring* e *Logging*: Muitas vezes, o registro e a documentação dos eventos relacionados à segurança são negligenciados pelas equipes, o que consiste em um grande erro. Registrar e monitorar fornece um *feedback* valioso que pode ser utilizado para tomar decisões estratégicas ou realizar auditorias (RAJAPAKSE et al., 2022).

3.3.2 Quais modelos de segurança são mais comumente aplicados em ambientes *DevSecOps*?

- OWASP SAMM: O *Software Assurance Maturity Model* é um modelo prescritivo de maturidade de segurança de *software*, ou seja, ele informa quais atividades preci-

sam ser realizadas, diferentemente de um modelo descritivo, que apenas descreve as atividades. Sua estrutura é adequada para diferentes tipos e tamanhos de empresas, bem como para distintas metodologias de desenvolvimento, como cascata e ágil (LANGE; KUNZ, 2024).

- OWASP DSOMM: Embora baseado no SAMM, o *DevSecOps Maturity Model* nasce devido à necessidade de um modelo adequado para ambientes *DevOps*, onde a segurança é parte essencial do ciclo de vida. Também é prescritivo, mas, diferentemente do SAMM, suas atividades são definidas em um nível mais próximo do programador do que da gestão. Desse modo, ele fornece com detalhes técnicos os requisitos necessários para atingir cada um dos níveis de maturidade em suas diferentes dimensões (LANGE; KUNZ, 2024).
- BSIMM: Diferentemente dos outros, o *Building Security In Maturity Model* (BSIMM) é um modelo descritivo, ou seja, ele descreve atividades sem exigir que sejam implementadas. Outra diferença fundamental é que se trata de um modelo proprietário, mantido pela Synopsys. A aplicação do modelo, bem como sua metodologia de avaliação, só estão disponíveis mediante contratação dos serviços da empresa (LANGE; KUNZ, 2024).

3.3.3 Quais medidas são comumente usadas para aferição de segurança?

- Métricas de Zhang: Segundo Zhang e Zhang (2024), medir de forma eficaz as características de *softwares web* é fundamental, porém pouco explorado. Para resolver este problema, eles realizaram uma revisão sistemática da literatura e definiram doze métricas voltadas a atender às necessidades dos sistemas *web* que usam *DevSecOps*. Elas permitem quantificar o desempenho do serviço, a segurança e a eficiência da operação, apoiando, assim, as tomadas de decisão e a melhoria contínua das práticas.
  - *Non-Comment Lines of Code*: Tamanho do código-fonte, excluindo comentários e linhas em branco.
  - *Design Defect Ratio*: Proporção de defeitos de *design* em relação às linhas de código-fonte não comentadas.
  - *Shared or Unknown Library Ratio*: Proporção de bibliotecas compartilhadas ou não verificadas em um serviço.
  - *Technical Debt Ratio*: Compara o custo de resolução de um débito técnico com o custo total do código-fonte.
  - *Continuous Deployment Cycles Score*: Pontuação dos ciclos de implantação contínua.
  - *Mean Change Lead Time*: Tempo médio que uma mudança leva desde o *commit* até chegar à produção.
  - *Mean Time to Recover*: Tempo médio para recuperação de falhas causadas nos

pipelines de CI/CD.
  - *Mean Number of Test Cases Per Parameter*: Média de casos de teste por parâmetro.
  - *Points of Environmental Risk*: Total de riscos de segurança não resolvidos em produção.
  - *Time for Response*: Tempo médio que as equipes de desenvolvimento levam para solucionar incidentes de segurança.
  - *Throughput*: Mede a capacidade de processamento de um serviço.
  - *Errors Per Time Unit*: Taxa de erros em determinada unidade de tempo.
- Métricas DORA: O *DevOps Research and Assessment* (DORA) é um dos principais programas de pesquisa do mundo na área de *DevOps*. Esse programa, que faz parte do Google, chegou, após anos de estudos, a quatro métricas-chave para medir os aspectos de *DevOps* de um projeto. Suas métricas passam por uma avaliação estatística rigorosa para possibilitar o entendimento da relação entre as medições e o sucesso das organizações (Google, 2024).
  - *Change lead time*: Tempo que uma alteração leva para chegar à produção.
  - *Deployment frequency*: Frequência com que as alterações chegam à produção.
  - *Change fail percentage*: Percentual de implantações que causam falhas em produção.
  - *Failed deployment recovery time*: Tempo necessário para se recuperar de uma implantação com falha.

3.3.4 Como as práticas de segurança são avaliadas?

É fundamental que as práticas de segurança sejam avaliadas para que seja possível analisar a eficácia das abordagens adotadas, evoluir as existentes ou adotar novas que se adequem melhor às necessidades da organização. Sendo assim, a avaliação de métricas representa uma forma eficiente e quantitativa de avaliar práticas de segurança, pois permite acompanhar a evolução das atividades e verificar os resultados de cada uma. Outra forma de analisar as práticas de segurança é por meio de auditorias, que verificam a adequação da organização aos padrões de segurança e conformidade (RAJAPAKSE et al., 2022).

Por fim, podem-se usar modelos de avaliação de maturidade, pois eles introduzem uma linha de base para comparação com a organização avaliada. Eles permitem analisar o estado atual da aplicação, além de identificar áreas de melhoria e traçar um plano para alcançar um nível de segurança mais elevado (LANGE; KUNZ, 2024).

3.3.5 De que maneiras as medidas de segurança são avaliadas nas organizações?

As medidas de segurança são avaliadas de diversas formas dentro das organizações. Uma das principais é a análise da repercussão das métricas de segurança nos *KPIs* da organização. *KPIs* (Indicadores-Chave de Desempenho) são as métricas centrais que medem a saúde dos projetos. É crucial criar alertas e *dashboards* para acompanhar o desenvolvimento das métricas e *KPIs*, pois, assim, pode-se obter *insights* de como as métricas de segurança impactam nos indicadores da empresa, além de possibilitar o rastreio das métricas durante todo o período em que foram monitoradas (JOSHI, 2024).

Outra maneira de avaliar as medidas de segurança está relacionada ao quanto elas se adequam ao *compliance* da organização ou a aspectos regulatórios. Muitas vezes, as empresas precisam seguir rígidos padrões de conformidade relacionados às métricas — como a cobertura de testes, que, dependendo da área, precisa ser extremamente alta —, bem como em setores financeiros, onde uma falha de segurança pode gerar um prejuízo bilionário (KUDRIAVTSEVA; GADYATSKAYA, 2024).

3.3.6 Que tipos de vulnerabilidades são mitigados pelas práticas de *DevSecOps*?

Existem diversos tipos de falhas de segurança que podem ocorrer durante o processo de desenvolvimento de *software*, entre elas, falhas que podem extrapolar o escopo do trabalho, como as relacionadas ao *hardware* utilizado. Por esse motivo, é necessário entender quais problemas de segurança são afetados pelas práticas *DevSecOps*, pois, desse modo, será possível analisar de forma mais assertiva a repercussão da adoção dessa metodologia na qualidade da segurança. No Quadro 5, foram listadas as principais vulnerabilidades impactadas por esse paradigma.

3.3.7 Quais ferramentas e tecnologias são usadas no *DevSecOps*?

- Monitoramento: Prometheus[2], Grafana[3] e Loki[4].
- Infraestrutura: Terraform[5], Kubernetes[6] e Docker[7].

[2] <https://prometheus.io>
[3] <https://grafana.com>
[4] <https://grafana.com/oss/loki>
[5] <https://www.terraform.io>
[6] <https://kubernetes.io>
[7] <https://www.docker.com>

Quadro 5: Vulnerabilidades Reduzidas

Vulnerabilidade	**Referência**
SQL Injection	Saeed et al. (2025)
Command Injection	Ramirez, Aiello e Lincke (2020)
Cross-Site Scripting (XSS)	Saeed et al. (2025)
XML External Entity (XXE)	Nocera et al. (2023)
Buffer Overflow	Ramirez, Aiello e Lincke (2020)
Cross-Site Request Forgery (CSRF)	Kushwaha, David e Suseela (2024)
Distributed Denial of Service (DDoS)	Saeed et al. (2025)
Man-in-the-Middle (MITM)	Nocera et al. (2023)
Broken Authentication	Saeed et al. (2025)
Broken Access Control	Saeed et al. (2025)
Security Misconfiguration	Saeed et al. (2025)
Session Hijacking	Kushwaha, David e Suseela (2024)
Server-Side Request Forgery (SSRF)	Nocera et al. (2023)

Fonte: Autor

- CI/CD: Jenkins[8], GitLab CI/CD[9], GitHub Actions[10], Tekton[11] e ArgoCD[12].
- Testes: SonarQube[13], FindBugs[14], Snyk[15], OWASP Dependency-Check[16], ZAP[17], Trivy[18], Detect Secrets[19], Asylo[20], StackHawk[21], JMeter[22] e Selenium[23].

3.3.8 Qual é o volume anual de publicações sobre *DevSecOps* de 2009 a 2025?

O volume anual de artigos está representado na Figura 8. Observa-se o crescimento das pesquisas sobre o tema, principalmente após o ano de 2020, chegando a seu pico em 2024. Em abril de 2025, mês em que a *string* de busca foi executada, a quantidade de estudos já se igualava ao volume de todo o ano de 2023, o que evidencia o crescimento e a importância da área nos meios acadêmico e profissional.

[8] <https://www.jenkins.io>
[9] <https://docs.gitlab.com/ee/ci>
[10] <https://github.com/features/actions>
[11] <https://tekton.dev>
[12] <https://argo-cd.readthedocs.io>
[13] <https://www.sonarsource.com/products/sonarqube>
[14] <http://findbugs.sourceforge.net>
[15] <https://snyk.io>
[16] <https://owasp.org/www-project-dependency-check>
[17] <https://www.zaproxy.org>
[18] <https://aquasecurity.github.io/trivy>
[19] <https://github.com/Yelp/detect-secrets>
[20] <https://asylo.dev>
[21] <https://www.stackhawk.com>
[22] <https://jmeter.apache.org>
[23] <https://www.selenium.dev>

Figura 8 – Volume Anual de Artigos entre 2009 e 2025

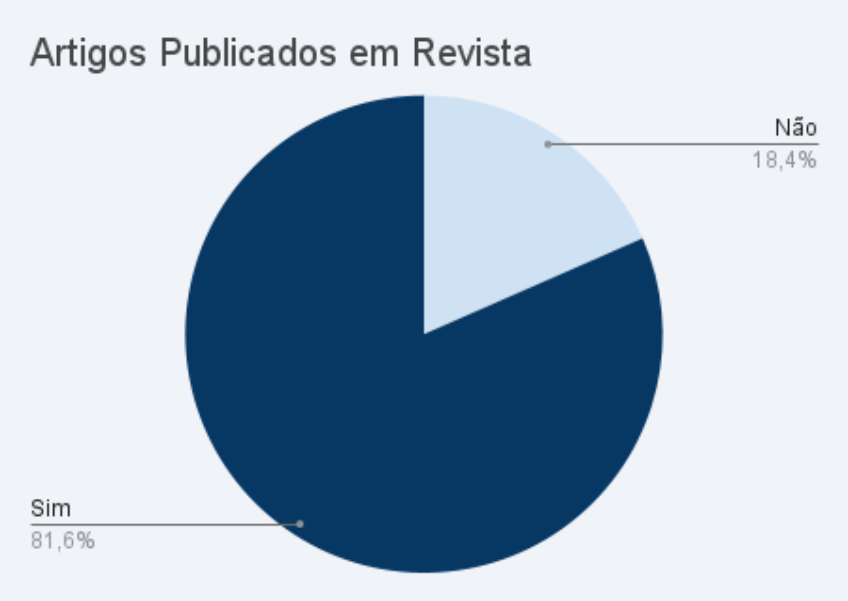


Fonte: Autor

3.3.9 Quantos artigos foram publicados em periódicos acadêmicos?

Como indicado na Figura 9, a grande maioria dos artigos foi publicada em revistas científicas. Desse modo, sabe-se que a maior parte dos artigos passou por um critério alto de revisão e análise de qualidade, elevando o nível de confiabilidade dos resultados da pesquisa.

Figura 9 – Artigos Publicados em Revistas

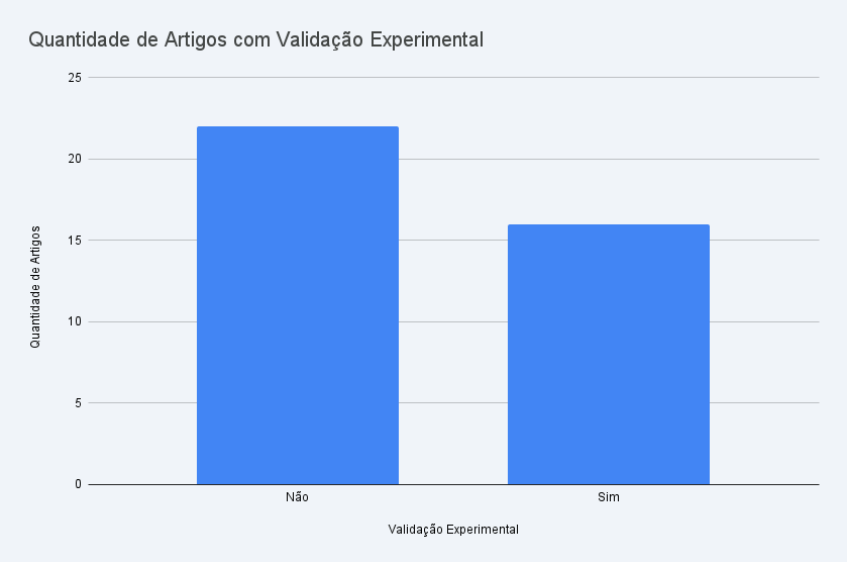


Fonte: Autor

3.3.10 Quantos estudos têm validação experimental?

Um total de dezesseis estudos, dos quarenta selecionados, ou seja, quarenta porcento, conforme pode ser observado na Figura 10, que apresenta a quantidade de estudos que possuem ou não essa validação. O baixo índice de validação experimental pode estar relacionado à característica emergente da área; por ser muito recente, ainda falta estabelecer e consolidar os métodos de pesquisa comumente usados em outras áreas para a validação dos estudos.

Figura 10 – Quantidade de Artigos com Validação Experimental

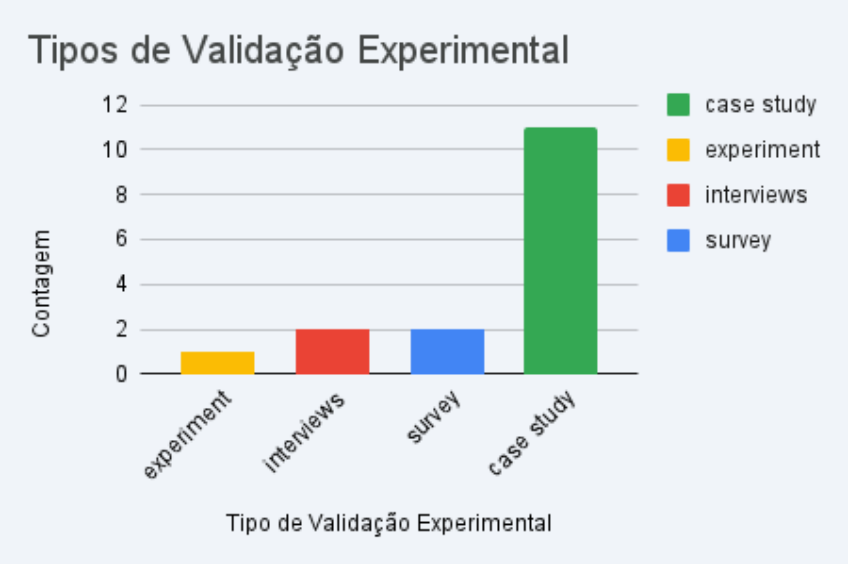


Fonte: Autor

3.3.11 Caso o estudo tenha validação experimental, qual foi o tipo?

Existem diferentes tipos de validação experimental, entre eles o estudo de caso, o experimento, as entrevistas e as pesquisas. Esses foram os métodos empíricos utilizados na validação de alguns dos estudos. A quantidade de cada tipo está representada no gráfico da Figura 11, do qual se depreende que a validação por estudo de caso é, com grande margem, o método mais utilizado. Isso se deve, principalmente, à característica do *DevSecOps* de estar ligado a muitos projetos reais, tanto na academia quanto na indústria, o que cria um ambiente propício para a aplicação de estudos de caso, pois eles possibilitam analisar e obter *insights* ao estudar o desenvolvimento realizado durante a construção de um produto.

Figura 11 – Tipos de Validação Experimental dos Artigos



Fonte: Autor

4 Planejamento do Estudo de Caso

Neste capítulo é apresentado o planejamento do estudo de caso, que foi conduzido na segunda etapa do trabalho, no qual são estabelecidos os conceitos e definições relacionados a esse método.

4.1 Definição

Yin (2015) define o estudo de caso como uma investigação empírica que analisa um determinado fenômeno da atualidade em seu contexto real, ou seja, o pesquisador se insere no ambiente cotidiano onde o objeto de estudo está sendo executado. Essa abordagem é particularmente relevante para a Engenharia de *Software*, pois frequentemente os fenômenos analisados nessa área são complexos e interligados, o que dificulta analisá-los isoladamente do seu ambiente de execução (RUNESON; HöST, 2009).

Ainda de acordo com Yin (2015), o estudo de caso é flexível e interativo. Isso significa que a estrutura do estudo pode se adaptar no decorrer da pesquisa, pois, o pesquisador, ao realizar as interações de coleta e análise dos dados, pode vir a perceber características do caso que não foram identificadas *a priori*.

Ademais, visando assegurar o rigor e a confiabilidade da investigação, os estudos de caso devem coletar dados de múltiplas fontes. Dessa forma, ao verificar que conclusões obtidas através de dados obtidos em diferentes locais apontam para o mesmo resultado, aumenta-se a robustez e confiabilidade dos resultados, pois esse processo de triangulação diminui a probabilidade de erro ou viés, além de fornecer uma visão mais ampla sobre o caso (YIN, 2015).

Também é necessário determinar se o estudo será holístico ou incorporado (YIN, 2015). Os estudos de caso holísticos analisam o caso como um todo, ou seja, o fenômeno é visto como um sistema único e integrado; portanto, o pesquisador obtém uma visão ampla e geral do caso. Já os estudos incorporados têm como principal característica o aprofundamento em múltiplas unidades de análise dentro de um mesmo caso, permitindo uma análise mais aprofundada.

Para além disso, (RUNESON; HöST, 2009) explicita a necessidade de considerar as ameaças à validade do estudo desde o início da pesquisa, pois ignorar tais fatores interfere na confiabilidade dos resultados. A primeira ameaça definida pelo autor está relacionada à validade do constructo, que representa em qual extensão as medidas operacionais realmente representam o que o pesquisador tem em mente e o que está sendo investigado de acordo com as perguntas de pesquisa. Já a ameaça à validade interna refere-se ao risco de

existir um fator não mapeado pelo pesquisador que cause interferência na relação causal entre os fatores selecionados. As ameaças externas, por outro lado, estão relacionadas à capacidade de generalização do estudo, ou seja, deseja-se que os achados do estudo tenham relevância para casos com características semelhantes. Por fim, a confiabilidade refere-se ao grau em que os dados e a análise dependem de pesquisadores específicos, isto é, se outro pesquisador conduzisse o mesmo estudo, o resultado deveria ser o mesmo.

4.2 Objetivo

Segundo Siavvas et al. (2021b), o desenvolvimento de *software* seguro é pautado na medição da qualidade da segurança de *software*, pois assim, é possível avaliar o nível da segurança do produto e conseguir traçar metas para guiar os processos de melhoria contínua do sistema. Porém, por diversas vezes são utilizados critérios de avaliação subjetivos ou que não possuem a devida validação, o que pode acarretar catástrofes relacionadas à segurança do produto. Situação essa que se agrava ao se tratar de práticas emergentes na indústria, como *DevSecOps*, que apesar de seu destaque no desenvolvimento ágil, por muitas vezes, carece de avaliação por metodologias apropriadas.

Assim, o objetivo desse estudo consistiu em investigar como as práticas *DevSecOps* afetam os aspectos relacionados à segurança de um projeto de *software* livre em desenvolvimento. Para tanto foram utilizados métodos, técnicas e ferramentas, observadas e propostos na literatura específica da área de segurança e em consonância com o estado da prática na indústria.

4.3 Caso

O caso em estudo neste trabalho é a plataforma Brasil Participativo [1]. Trata-se de uma plataforma digital de participação social, que contribui para a criação, monitoramento e aperfeiçoamento de políticas públicas brasileiras. A plataforma é de responsabilidade da Secretaria Nacional de Participação Social da Secretaria Geral da Presidência da República (SNPS/SGPR)[2]. Essa plataforma é um produto de *software* livre, disponibilizado nos termos da licença *GNU Affero General Public License*, versão 3 (GNU AGPL-3.0), e foi desenvolvida com o apoio da Dataprev[3], da comunidade Decidim-Brasil[4], do Ministério da Gestão e da Inovação em Serviços Públicos (MGI)[5] e da Universidade de Brasília (UnB), por meio do Laboratório Avançado de Produção, Pesquisa e Inovação

[1] https://brasilparticipativo.presidencia.gov.br/
[2] https://www.gov.br/secretariageral/pt-br/composicao/orgaos-especificos-singulares/snps
[3] https://www.dataprev.gov.br/
[4] https://decidim.org/pt-br/
[5] https://www.gov.br/gestao/pt-br

em *Software* (LAPPIS)[6]. O Brasil Participativo tem como objetivo promover a interação entre cidadãos e o governo, permitindo que os cidadãos e cidadãs participem de consultas públicas, conferências, planos e enquetes sobre políticas públicas. A versão em uso do produto fica hospedada na infraestrutura computacional da Dataprev[7]. O Brasil participativo tem como base o produto de *software* mundialmente utilizado, o Decidim[8] (Brasil. Presidência da República, 2025).

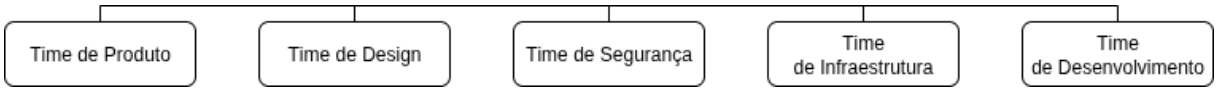
O Brasil Participativo possibilitou que 1.619.015 pessoas de diversas regiões do país contribuíssem com ideias de impacto nacional de forma simples, segura e transparente. Em acesso ao *site* da plataforma, no dia 10/11/2025, constava a informação de aproximadamente 1.619.015 participantes, 9.200.013 acessos e um total de 47 processos desde o seu lançamento. Esses números evidenciam seu papel no fortalecimento e apoio à democracia brasileira, ao promover a diversidade e a inclusão na formulação de políticas públicas (Brasil. Presidência da República, 2025).

O Decidim é um produto de *software* livre disponibilizado sob a licença GNU AGPL-3.0. Trata-se de um *framework* democrático, participativo, genérico, baseado em *Ruby on Rails*. Ele foi desenvolvido pelo governo de Barcelona para promover a participação cidadã e a democracia participativa. Ele permite que qualquer organização faça uma adaptação de seus componentes para a própria realidade, desse modo, tornou-se uma referência em tecnologia cívica (Decidim, 2025).

O Brasil Participativo [9] é desenvolvido por várias equipes multidisciplinares, compostas principalmente pelos perfis de jornalistas, redatores publicitários, *designers* gráficos, desenvolvedores, especialistas em segurança da informação, especialistas em infraestrutura, gerentes de projeto e líderes técnicos. Cada um dos times tem um líder responsável por coordenar as atividades e garantir a qualidade do trabalho realizado. Em acesso ao repositório do código-fonte da plataforma, no dia 10/11/2025, havia 4.192 *commits*[10] e 103 versões de produto (*releases*)[11] disponibilizadas.

A Figura 12 apresenta o organograma das equipes envolvidas no projeto. A equipe de desenvolvimento é composta por 9 desenvolvedores, enquanto a equipe de segurança conta com 5 especialistas dedicados a garantir a integridade e proteção do sistema.

Figura 12 – Organograma das equipes do projeto Brasil Participativo



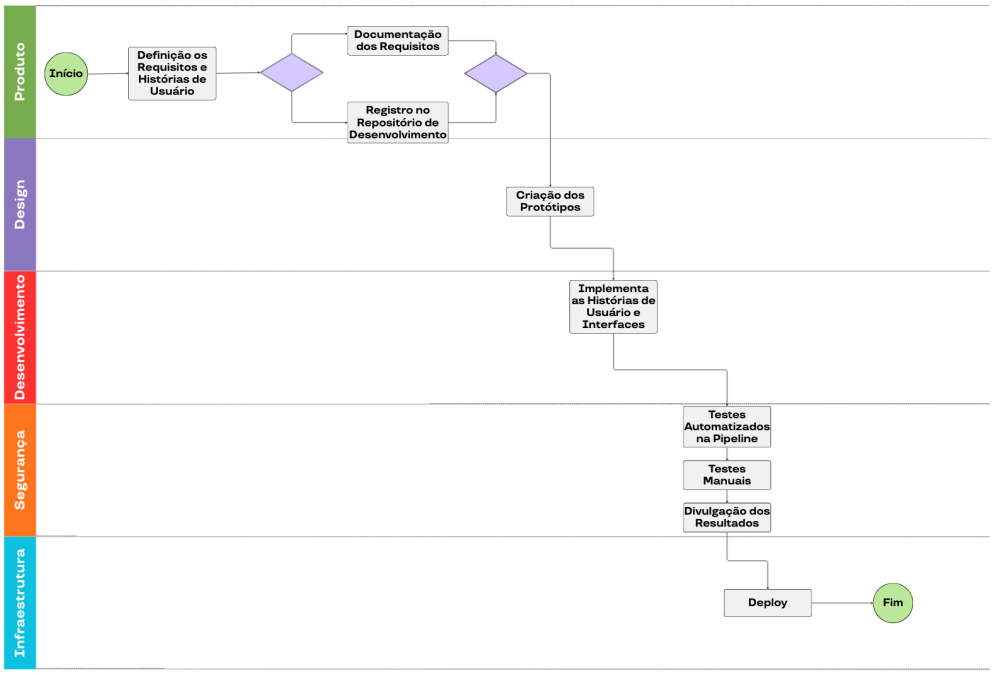
Fonte: Autor

[6] https://www.lappis.rocks/
[7] https://www.dataprev.gov.br/
[8] https://decidim.org/
[9] https://gitlab.com/lappis-unb/decidimbr/decidim-govbr
[10] https://gitlab.com/lappis-unb/decidimbr/decidim-govbr/-/commits/main
[11] https://gitlab.com/lappis-unb/decidimbr/decidim-govbr/-/tags?page=5

Já a Figura 13 apresenta uma visão em alto nível do processo de desenvolvimento do Brasil Participativo utilizando a notação *Business Process Model and Notation* (BPMN). Estão representadas na figura as atividades envolvidas no processo de melhoria contínua do projeto, dado que esse é o foco deste trabalho. O início do fluxo se dá com a definição dos novos requisitos a serem implementados, sendo que os mesmos são escritos como histórias de usuário. Posteriormente, essas histórias são documentadas e repassadas para o time de desenvolvimento, que é responsável por implementar as mudanças no código-fonte. Após a implementação, o código-fonte é submetido a uma série de testes automatizados e manuais para garantir sua qualidade e segurança. Após os testes serem executados, o time de segurança divulga os resultados para o time de desenvolvimento, que é responsável por corrigir as vulnerabilidades encontradas. Por fim, o código-fonte é integrado ao sistema principal pelo time de infraestrutura, concluindo o ciclo.

Este estudo de caso concentrou-se nas etapas posteriores ao desenvolvimento e anteriores à entrada em produção. Nesse estágio, ferramentas de segurança foram executadas por meio do *pipeline* de CI/CD para realizar a análise estática do código-fonte, bem como a análise dinâmica, que depende da implantação (*deploy*), pois considera aspectos do comportamento em uso do produto.

Figura 13 – BPMN do processo de desenvolvimento do Brasil Participativo



Fonte: Autor

Esse projeto foi escolhido por conveniência da observação do fenômeno. Trata-se de um produto de *software* livre que conta com equipes de segurança e desenvolvimento acessíveis e ativas, o que facilita a comunicação e a colaboração durante o estudo de caso. Além disso, o LAPPIS possui um ambiente de desenvolvimento ágil, cultura *DevSecOps*

e entregas contínuas, o que é fundamental para a avaliação das práticas implementadas, além de o *software* ser de grande relevância social, o que reforça a importância de garantir sua segurança e confiabilidade.

Porém, apesar de já possuir o Brakeman[12], uma ferramenta SAST, implementada em seu *pipeline* de CI/CD, foi identificado que ela só analisa três diretórios em todo o projeto. Nesse sentido, foi fundamental a conversa com o time de segurança para mudar a abordagem de análise da ferramenta para o contexto do projeto como um todo, o que apesar de representar uma mudança na cultura do projeto, é fundamental para uma análise quantitativa de fato representativa.

Ademais, outro aspecto fundamental para a realização do estudo foi optar por marcar o *pipeline* como sinalizado ao encontrar uma vulnerabilidade, ou seja, o *pipeline* executa até o fim mesmo se encontrar problemas de segurança, porém fica marcado com uma sinalização de aviso. Essa decisão foi tomada, pois ao implementar novas ferramentas que analisam pontos até então não abarcados por testes de segurança é esperado que muitas vulnerabilidades sejam descobertas, e levando em consideração o volume de trabalho a que o time está submetido, falhar o *pipeline* significaria bloquear a produtividade nas demandas atuais.

4.4 Trabalhos Relacionados

Alguns trabalhos obtidos por meio da revisão da literatura foram utilizados como referência para a realização desta pesquisa. Esses artigos foram selecionados, pois abordam os temas pertinentes e correlatos, para o planejamento e execução da pesquisa deste estudo de caso.

O estudo realizado por Siavvas et al. (2021b) foi o estudo central para a elaboração da monografia. Sua relevância se dá, pois ele apresenta um modelo hierárquico de avaliação de segurança que quantifica a segurança interna do *software* com base em alertas de análise estática (SAST) e métricas de *software*. O autor demonstrou que é possível avaliar a segurança de forma confiável e quantitativa usando modelos de qualidade e análise estática.

O trabalho de (ZHANG; ZHANG, 2024), apresenta doze métricas quantitativas de *DevSecOps* especificamente projetadas para microsserviços *web* baseados em nuvem. O foco é avaliar a qualidade, segurança e eficiência operacional, com o intuito de auxiliar na tomada de decisões informadas e na melhoria contínua.

A revisão sistemática feita por Rajapakse et al. (2022), identifica os desafios e soluções na adoção do *DevSecOps*, incluindo seleção de ferramentas, avaliação contínua

[12] https://brakemanscanner.org/

de segurança e o equilíbrio entre velocidade e segurança.

Saeed et al. (2025) aborda técnicas para integrar a segurança no ciclo de desenvolvimento de *software*, enfatizando a necessidade de uma abordagem colaborativa e ferramentas automatizadas para análise de ameaças e testes de segurança.

4.5 Questão de Pesquisa

De forma análoga ao procedimento adotado no planejamento da revisão da literatura, empregou-se a abordagem GQM Basili, Caldiera e Rombach (1994) para a definição das questões específicas, derivadas da questão principal, bem como das métricas correspondentes. Essa abordagem visa assegurar a aderência ao objetivo central do estudo e possibilitar a avaliação quantitativa ou qualitativa de cada questão de pesquisa secundária, que atualmente orienta a condução do estudo de caso.

A ISO/IEC 25010 (2023) define a confidencialidade como a capacidade do sistema de impedir o acesso não autorizado às informações, assim, impedindo que os dados privados sejam visíveis para quem não possui as permissões necessárias. Ela é uma subcaracterística da característica de segurança e foi analisada neste estudo de caso por meio da análise de vulnerabilidades detectadas no *pipeline* de CI/CD. Para corroborar com essa análise e fazer a triangulação da coleta de dados, uma avaliação qualitativa com os membros do time é necessária para observar os impactos dessas novas práticas no processo de desenvolvimento.

À vista disso, foram elaboradas as seguintes questões específicas deste estudo:

- Questão Específica 1 (QE-1): A aplicação de práticas *DevSecOps* permitiu identificar vulnerabilidades de segurança sob as perspectivas da qualidade interna e externa do produto?
  - Métrica 1.1: Média de vulnerabilidades encontradas por ferramentas SAST, por nível de severidade, por *merge* (ISO/IEC 27004, 2016).
  - Métrica 1.2: Média de vulnerabilidades encontradas por ferramentas DAST, por nível de severidade, por *merge* (ISO/IEC 27004, 2016).
  - Métrica 1.3: Proporção de vulnerabilidades em bibliotecas encontradas por ferramentas SCA *Software Composition Analysis*, por nível de severidade, por *merge* (ZHANG; ZHANG, 2024).
  - Métrica 1.4: *Security Incidents Trend* pelas ferramentas SAST e DAST, por nível de severidade, por *merge* (ISO/IEC 27004, 2016).
- Questão Específica 2 (QE-2): A análise automática da segurança do *pipeline* e as métricas coletadas ajudaram na tomada de decisões relacionadas ao projeto?
  - Métrica 2.1: *Change Fail Rate*, por *merge* (Google, 2024).
  - Métrica 2.2: Opinião do time obtida por um questionário sobre os impactos

das novas práticas.

4.6 Fonte de Dados

Para coletar os dados necessários para a posterior avaliação das métricas foram necessárias diferentes fontes de dados. Primeiramente, as ferramentas SAST e SCA foram executadas diretamente no código-fonte. Outra fonte de dados foi o sistema em uso, que foi usado para a obtenção dos dados referentes às ferramentas DAST que analisam o *software* em execução. Adicionalmente, o orquestrador de CI/CD atuou como fonte de dados para coletar as tentativas falhas de integração do código-fonte, evidenciando a identificação de falhas pelas ferramentas.

Por fim, a equipe técnica é identificada como fonte de dados dos formulários de avaliação ao final do estudo de caso, permitindo a análise qualitativa e triangulação dos resultados.

4.7 Unidades de Análise e Procedimentos

Para atender à complexidade das questões de pesquisa deste trabalho, optou-se pelo método de estudo de caso incorporado ou embutido, fundamentado por Yin (2015). Esta abordagem foi escolhida, pois permite que o caso principal seja investigado por meio da análise aprofundada de múltiplas subunidades de análise. Especificamente, o estudo se debruça sobre diferentes faces do projeto, como o impacto na qualidade da segurança interna e externa do produto, e a percepção da equipe sobre as mudanças implementadas. Ao examinar cada um desses componentes seguindo o protocolo de estudo de caso, é possível construir uma visão detalhada e triangulada que fortalece a validade das conclusões e oferece uma resposta mais completa ao problema investigado.

A unidade de análise primária envolve a avaliação quantitativa da segurança do *software*. As métricas de segurança foram coletadas automaticamente nos *Merge Requests* (MRs) integrados na *branch* principal (*main*) do projeto. Dessa forma, foi possível monitorar a evolução da segurança do *software* ao longo do tempo, permitindo a identificação de tendências e padrões relacionados à detecção de vulnerabilidades. Os MRs foram adotados como unidade de análise, pois eles fornecem a granularidade ideal para o tempo previsto para o estudo ao mesmo tempo que conseguem transmitir a situação das principais alterações no código-fonte. O ideal seria ter a *release* como a unidade de análise. Contudo, durante o período de execução desse estudo, não foi gerada uma quantidade de *releases* que figurasse como uma amostra estatística significativa. Por isso, optou-se por um grão menor da versão do produto, no caso, as versões geradas após a revisão e aceite do código-fonte, no fechamento de um determinado MR.

A outra unidade de análise envolve a equipe técnica responsável pelo desenvolvimento e manutenção do *software*, cuja percepção sobre as mudanças foi avaliada por meio de um questionário. O objetivo foi coletar a opinião da equipe sobre o impacto das práticas *DevSecOps* no processo de desenvolvimento, identificando benefícios, desafios e sugestões de melhoria. Esta unidade de análise fornece dados qualitativos que complementaram as análises quantitativas das outras unidades.

4.8 Análise de Dados

A análise dos dados quantitativos foi fundamentada em estatísticas descritivas e em análise de tendência, visando avaliar a evolução da segurança da aplicação ao longo do tempo. Para as métricas de detecção de vulnerabilidades por ferramentas SAST e DAST, Métrica 1.1 e Métrica 1.2, referentes a métrica B.34 - *Security Incidents Trend* da norma ISO/IEC 27004 (2016), foi calculado um indicador de tendência para permitir avaliar o volume de vulnerabilidades de criticidade média ou superior, por nível de severidade.

O cálculo do indicador foi obtido através da razão entre a média de vulnerabilidades das últimas duas *sprints* concluídas e a média das últimas seis *sprints* concluídas, sendo que, um valor inferior a 1,0 indica uma tendência de melhoria, entre 1,0 e 1,3 sinaliza uma tendência de estabilidade e superior a 1,3 indica uma piora significativa.

Para a análise de tendência, definimos a média recente (M_{rec}) e a média histórica (M_{hist}).

A média recente (M_{rec}) é definida como a soma das últimas duas observações dividida por dois, o que pode ser expresso pela seguinte equação:

$$M_{rec} = \frac{1}{2} \sum_{i=n-1}^{n} x_i; \tag{4.1}$$

Enquanto a média histórica (M_{hist}) é definida como a soma das últimas seis observações dividida por seis, conforme a seguinte equação:

$$M_{hist} = \frac{1}{6} \sum_{i=n-5}^{n} x_i. \tag{4.2}$$

Nas duas equações anteriores, temos que x_i representa o valor da métrica no intervalo de tempo i e n é o intervalo atual.

Essas equações fornecem os valores necessários para calcular a razão (R) que será usada para definir a tendência das vulnerabilidades encontradas. Ela é dada por:

$$R = \frac{M_{rec}}{M_{hist}}. \tag{4.3}$$

Para a Métrica 1.3, adaptada da métrica *Shared or Unknown Library Ratio* proposta por Zhang e Zhang (2024), foi calculada a proporção de bibliotecas com vulnerabilidades em relação ao total. Zhang e Zhang (2024) argumentam que o uso de bibliotecas de terceiros aumenta o vetor de ataque e dependência de terceiros no código da aplicação. Porém, o foco desse estudo foi avaliar a segurança do *software* em si, portanto, a métrica foi adaptada para considerar apenas as bibliotecas que apresentam vulnerabilidades de criticidade média ou superior. A métrica foi calculada conforme a Equação 4.4:

$$P_{vuln} = \frac{N_{vuln_libs}}{N_{total_libs}} \times 100. \tag{4.4}$$

Por sua vez, a Métrica 2.1, uma adaptação da métrica *Change Fail Rate* elaborada pelo Google (2024), foi calculada pela taxa percentual de *merges* sinalizados por vulnerabilidades de criticidade média ou superior. Se trata de uma adaptação, pois a métrica original considera falhas em produção, o que não é o foco deste estudo. A métrica foi calculada conforme a Equação 4.5:

$$CFR = \frac{N_{merges_sinalizados}}{N_{total_merges}} \times 100. \tag{4.5}$$

Para a análise de frequência de respostas do questionário, Métrica 2.2, a frequência das respostas de cada pergunta foi registrada, de modo a possibilitar o cálculo da moda, pois ao ter a opinião da maioria dos participantes sobre o tópico solicitado foi possível obter a percepção geral do impacto das atividades realizadas.

4.9 Instrumentação

A instrumentação se refere às ferramentas que foram utilizadas para a realização do estudo de caso, aqui estão definidas as ferramentas usadas para a análise da segurança do sistema, controle de implementação do código-fonte, versionamento do código-fonte do *pipeline* e coleta de informações da equipe.

O Brakeman é uma ferramenta *open source* de avaliação da qualidade e segurança do código-fonte. Ele realiza análise estática para detectar *bugs*, vulnerabilidades e *code smells* em várias linguagens. Ele fez parte das ferramentas *white-box* integradas ao *pipeline* realizando a análise estática de segurança de aplicação (SAST). Sua escolha deu-se em função da equipe já possuir experiência prévia com a ferramenta.

O Trivy [13] é um *scanner* de segurança de código-fonte aberto. Ele é utilizado na análise de composição de *software*, verificando as bibliotecas de terceiros do projeto, garantindo que componentes externos ao projeto não insiram vulnerabilidades no sistemas.

[13] https://trivy.dev/latest/

Além disso, ele é capaz de buscar vulnerabilidades em *containers* e configurações de infraestrutura como código. Ele complementou o SonarQube como ferramenta *white-box*.

O Zed Attach Proxy (ZAP) [14] foi desenvolvido para encontrar problemas de segurança em aplicações *web* em execução. Ele foi empregado para realizar a Análise Dinâmica de Segurança de Aplicação (DAST) em um ambiente de testes.

O GitLab CI/CD [15] é uma ferramenta integrada ao GitLab que permite a automação das etapas do ciclo de vida do *software*, através dele que as ferramentas de segurança foram executadas automaticamente e ele serviu para sinalizar o *build*/*deploy* caso vulnerabilidades fossem encontradas.

O Git [16] é um sistema de controle de versão e o GitHub [17] é uma plataforma de hospedagem de código-fonte para controle de versão com Git. Eles foram utilizados para o versionamento e armazenamento dos artefatos por este estudo de caso.

O Google Forms[18] permite a criação rápida e fácil de formulários *online*, além de permitir a gestão e análise dos resultados. Ele é propício para a aplicação do questionário que coleta os dados qualitativos da equipe.

[14] https://www.zaproxy.org/
[15] https://docs.gitlab.com/install/
[16] https://git-scm.com/downloads
[17] https://github.com/
[18] https://docs.google.com/forms/u/0/

5 Execução do Estudo de Caso

Aqui serão apresentados os processos e etapas, bem como as ferramentas construídas para a realização do estudo de caso proposto nesta pesquisa. Adicionalmente, serão detalhadas as atividades desenvolvidas para a coleta e análise dos dados necessários para a validação dos objetivos estabelecidos, bem como uma discussão sobre o que foi possível observar durante o estudo. Todos os códigos-fonte citados podem ser encontrados no repositório [1] destinado a guardar os artefatos produzidos durante essa etapa.

5.1 Configuração do *Pipeline* de CI/CD Localmente

Primeiramente, é importante ressaltar a necessidade de desenvolver um método para executar o *pipeline* de CI/CD localmente, visto que o projeto original não contempla uma etapa de *deploy* em sua esteira. Essa etapa é crucial para a análise proposta, pois a ferramenta de análise dinâmica requer uma versão do projeto em execução para ser aplicada. Com o intuito de suprir essa lacuna, a biblioteca *gitlab-ci-local* [2] foi utilizada para viabilizar a execução local do *pipeline*. Essa ferramenta permite que as etapas definidas no arquivo `.gitlab-ci.yml` sejam reproduzidas em um ambiente local, de maneira análoga ao GitLab CI.

Dessa forma, elaborou-se um *script Shell* que executa individualmente cada etapa do *pipeline*, uma vez que certas fases demandam adaptações para operarem localmente. Por exemplo, a etapa de análise estática exige variáveis e caminhos específicos devido ao uso de *Docker in Docker* na configuração original. Já para a análise dinâmica, foi necessário implementar um *health-check* para assegurar que a ferramenta inicie apenas quando o *build* da versão analisada estiver ativo. Adicionalmente, o *script* é responsável por organizar os resultados das análises em diretórios específicos, facilitando a posterior coleta de dados. Vale destacar que o *script* também realiza os *logs* de cada uma das etapas, armazenando-os em arquivos que auxiliam na identificação de falhas e na análise dos resultados.

A Listagem 5.1 apresenta os principais trechos do *script* desenvolvido para a execução local do *pipeline* de CI/CD.

```
1 #!/bin/bash
2 # --- SAST ---
3 run_and_capture "SAST" \
```

[1] https://github.com/chaydson/tcc-code
[2] https://github.com/firecow/gitlab-ci-local

```
    gitlab-ci-local --volume "/var/run/docker.sock:/var/run/docker.sock"
    --variable "DOCKER_HOST=unix:///var/run/docker.sock" SAST
if [ $? -ne 0 ]; then
    echo "Job 'SAST' falhou, mas continuando..."
    OVERALL_STATUS=1
fi

# --- SCA ---
run_and_capture "SCA" \
  gitlab-ci-local SCA
if [ $? -ne 0 ]; then
  echo "Job 'SCA' falhou, mas continuando..."
  OVERALL_STATUS=1
fi

# --- DAST ---
if [ "$APP_IS_READY_FOR_DAST" = true ]; then
echo "Iniciando DAST."
run_and_capture "DAST (ZAP Baseline)" \
  sudo docker run --network="host" --rm \
  -v $(pwd):/zap/wrk/:rw -t \
  ghcr.io/zaproxy/zaproxy:stable \
  zap-baseline.py \
    -t http://localhost:3000/ \
    -r "$REPORTS_TOOLS_DIR/zap-report.html" \
    -J "$REPORTS_TOOLS_DIR/zap-report.json" \
    -l WARN
```

Listing 5.1 – Shell Script para *Pipeline* Local

5.2 Elaboração dos *Scripts* para Automação da Coleta de Dados

Para viabilizar a coleta do grande volume de dados gerado pela execução do *pipeline* na unidade de análise escolhida, os *Merge Requests* (MRs) integrados ao ramo principal do repositório, fez-se necessária a criação de *scripts* que extraíssem automaticamente as informações de cada MR. Dessa forma, foram desenvolvidos *scripts* em *Python* para cada uma das ferramentas (SAST, SCA e DAST) utilizadas no *pipeline*, visto que cada uma gera relatórios em formatos distintos, sendo preciso padronizar a extração para o cálculo das métricas.

Esses *scripts* seguem uma lógica comum, que consiste em acessar o diretório onde os relatórios estão armazenados, contabilizar a quantidade de vulnerabilidades por severidade (crítico, alto, médio e baixo) em cada *merge* e consolidar essas informações em um *DataFrame* Pandas. A Listagem 5.2 exemplifica esse processo por meio do *script* desenvolvido para a extração dos dados do SAST.

```
def parse_brakeman(base_path):
    print("--- Processando Brakeman ---")
    files = glob.glob(os.path.join(base_path, "*", "artifacts", "brakeman-report.json"))
    results = []

    for filepath in files:
        try:
            path_parts = os.path.normpath(filepath).split(os.sep)
            commit_hash = path_parts[-3]

            with open(filepath, 'r', encoding='utf-8') as f:
                data = json.load(f)

            row = {'commit_hash': commit_hash}
            warnings = data.get("warnings", [])

            if warnings:
                df_temp = pd.DataFrame(warnings)
                if "confidence" in df_temp.columns:
                    counts = df_temp['confidence'].value_counts().to_dict()
                    row.update({f"brakeman_{k}": v for k, v in counts.items()})

            row['brakeman_total'] = data.get("scan_info", {}).get("security_warnings", 0)
            results.append(row)
        except Exception:
            continue
```

Listing 5.2 – Script de Extração (SAST)

Entretanto, para coletar os dados referentes ao resultado da execução do *pipeline* o processo é diferente. Nesse caso, um *script Python* acessa os *logs* gerados por cada execução do *pipeline* e verifica se as etapas executaram com sucesso ou falha, bem como se vulnerabilidades foram encontradas. Esses dados são então organizados em um *DataFrame* Pandas para facilitar a análise posterior. A Listagem 5.3 ilustra o processo de extração dos dados referentes ao resultado da execução.

```
def parse_pipeline_logs(base_path):
    print("--- Processando Logs da Pipeline (Warn/Fail) ---")
    search_pattern = os.path.join(base_path, "*", "logs_pipeline", "pipeline_summary.txt")
    files = glob.glob(search_pattern)

    results = []

```

```
    for filepath in files:
        try:
            path_parts = os.path.normpath(filepath).split(os.sep)
            commit_hash = path_parts[-3]

            with open(filepath, 'r', encoding='utf-8') as f:
                content = f.read()

            has_fail = "FAIL" in content
            has_warn = "WARN" in content

            status_code = 1 if (has_fail or has_warn) else 0

            results.append({
                'commit_hash': commit_hash,
                'pipeline_has_issue': status_code
            })

        except Exception:
            continue

    if not results:
        return pd.DataFrame(columns=['commit_hash', 'pipeline_has_issue'
    ])

    return pd.DataFrame(results)
```

Listing 5.3 – Script de Extração (Logs)

5.3 Consolidação dos Dados Coletados

Conforme preconiza a medida *Security Incidents Trend* - B.34 da norma ISO/IEC 27004 (2016), é necessário definir um intervalo temporal para observar o fenômeno estudado, calculando a média de cada período e comparando os dois mais recentes com os seis anteriores. Com base nessa diretriz, definiu-se que a janela de análise seria de 14 dias, contados retroativamente a partir do último dia da coleta (22/11/2025), todos os períodos estão representados no Quadro 6. A quantidade de dias foi escolhida depois de um alinhamento com o time de desenvolvimento, que indicou que ciclos quinzenais seriam mais adequados para observar as variações nas métricas de segurança, considerando o ritmo de desenvolvimento, demandas atuais e a possibilidade de analisar e, se possível, iniciar a correção das vulnerabilidades encontradas.

Posteriormente, os dados foram consolidados em uma linha do tempo que agrega as informações coletadas por cada um dos *scripts* desenvolvidos. Para tal, um *script* em *Python* percorre os diretórios onde os dados extraídos foram armazenados e identifica a

Quadro 6: Períodos Analisados

Grupo	Intervalo
1	17/08/25 até 30/08/25
2	31/08/25 até 13/09/25
3	14/09/25 até 27/09/25
4	28/09/25 até 11/10/25
5	12/10/25 até 25/10/25
6	26/10/25 até 08/11/25
7	09/11/25 até 22/11/25

Fonte: Autor

data de cada alteração. Em seguida, realiza-se o agrupamento das entregas em ciclos de 14 dias, utilizando a divisão inteira entre a data da alteração e a data de término da coleta.

Outro ponto a ser destacado é que, para cada *merge*, foi extraída a quantidade total de bibliotecas do projeto, conforme demonstra a Listagem 5.4. Esse dado é fundamental para calcular a proporção entre as bibliotecas vulneráveis e o total de bibliotecas utilizadas, viabilizando assim a aferição da métrica relacionada ao SCA (*Software Composition Analysis*).

```
def parse_dependency_counts(repo_path, commit_hashes):
    print("Contando Bibliotecas")

    results = []
    gem_regex = re.compile(r'^\s{4}[\w\-_]+ \(')

    for commit_hash in commit_hashes:
        total_libs = 0
        gems_count = 0
        npm_count = 0

        try:
            cmd = ["git", "show", f"{commit_hash}:Gemfile.lock"]
            output = subprocess.check_output(cmd, cwd=repo_path, stderr=
    subprocess.DEVNULL).decode('utf-8')
            gems_count = len([line for line in output.splitlines() if
    gem_regex.match(line)])
        except subprocess.CalledProcessError:
            pass

        try:
            cmd = ["git", "show", f"{commit_hash}:package-lock.json"]
            output = subprocess.check_output(cmd, cwd=repo_path, stderr=
    subprocess.DEVNULL).decode('utf-8')
            pkg_lock = json.loads(output)

            if "packages" in pkg_lock:
```

```
                npm_count = len([k for k in pkg_lock["packages"].keys() if k != ""])
            elif "dependencies" in pkg_lock:
                npm_count = len(pkg_lock["dependencies"])

        except (subprocess.CalledProcessError, json.JSONDecodeError):
            try:
                cmd = ["git", "show", f"{commit_hash}:package.json"]
                output = subprocess.check_output(cmd, cwd=repo_path, stderr=subprocess.DEVNULL).decode('utf-8')
                pkg_json = json.loads(output)
                npm_count = len(pkg_json.get('dependencies', {})) + len(pkg_json.get('devDependencies', {}))
            except:
                pass

        total_libs = gems_count + npm_count

        results.append({
            'commit_hash': commit_hash,
            'total_libs': total_libs,
            'ruby_libs': gems_count,
            'npm_libs': npm_count
        })

    return pd.DataFrame(results)
```

Listing 5.4 – Script de Extração das Bibliotecas

5.4 Aplicação do Questionário

Infelizmente, devido a limitações de tempo e recursos, não foi possível aplicar o questionário planejado para coletar dados qualitativos sobre a opinião dos desenvolvedores envolvidos no projeto. A aplicação do questionário ou de outro método de coleta de dados qualitativos exigiria um planejamento detalhado, o que não pôde ser realizado dentro do cronograma estabelecido para este estudo de caso. Ademais, o time de desenvolvimento não recebia os dados já tratados por parte do time de segurança de forma a orientar a tomada de decisão com mais clareza, o que demandaria um grande trabalho por parte da equipe que já se encontrava em um ciclo de entrega intenso. Portanto, a análise se concentrou exclusivamente nos dados quantitativos coletados através das ferramentas automatizadas descritas anteriormente.

6 Resultados do Estudo de Caso

Neste capítulo, são apresentados os resultados obtidos a partir do estudo de caso realizado. A análise dos dados coletados é detalhada, destacando as métricas calculadas e os principais *insights* relevantes obtidos com o estudo.

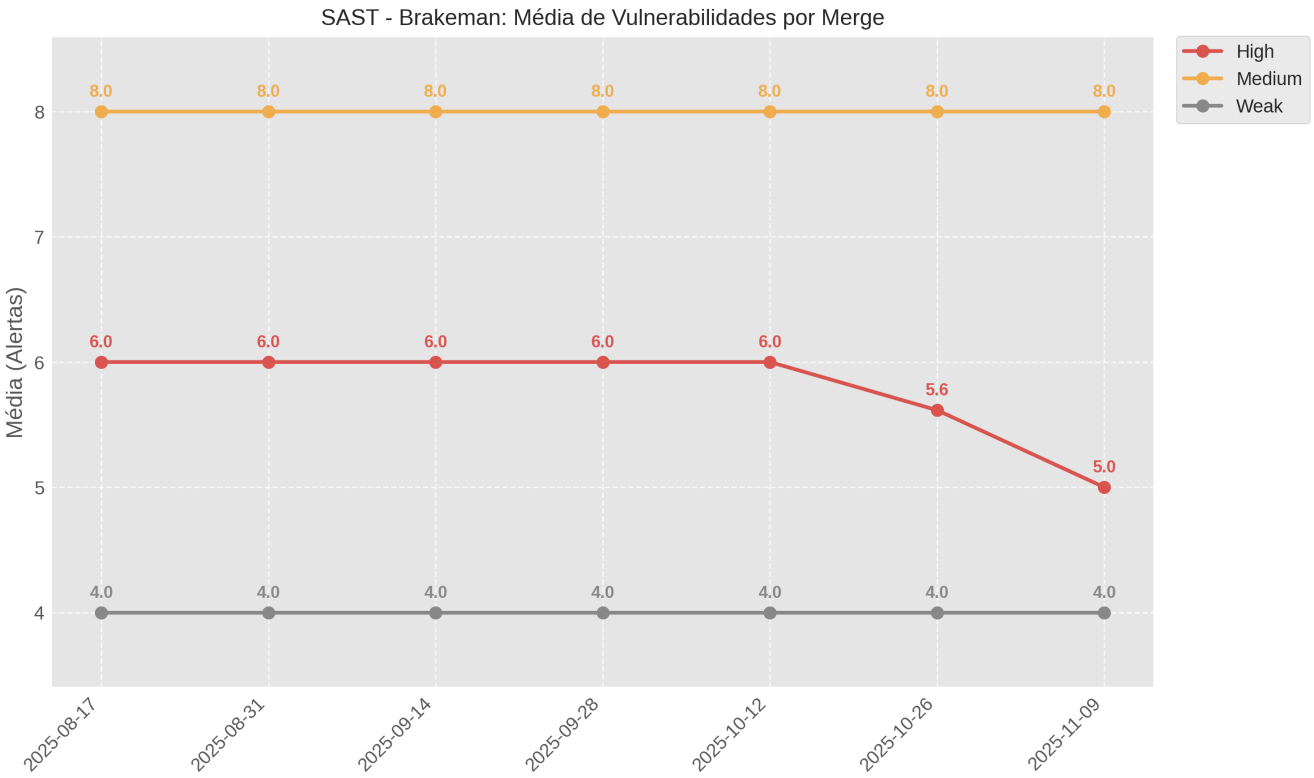
6.1 Análise dos Dados

6.1.1 Identificação de Vulnerabilidades de Segurança

Retomando a questão específica QE-1 definida na Subseção 4.5: "A aplicação de práticas *DevSecOps* permitiu identificar vulnerabilidades de segurança sob as perspectivas da qualidade interna e externa do produto?"

Com a Figura 14, é possível observar a média de vulnerabilidades identificadas pelo Brakeman (ferramenta SAST) por *merge* ao longo do período analisado. Ela representa a Métrica 1.1 - Média de vulnerabilidades encontradas por ferramentas SAST, por nível de severidade, por *merge*, definida na Subseção 4.5. As vulnerabilidades de criticidade *Weak* estão representadas no gráfico, porém elas não são consideradas para o cálculo do indicador, conforme detalhado na Subseção 4.5. Isso se deu para permitir uma análise mais abrangente das vulnerabilidades identificadas pela ferramenta, mesmo que elas não sejam consideradas relevantes para o indicador.

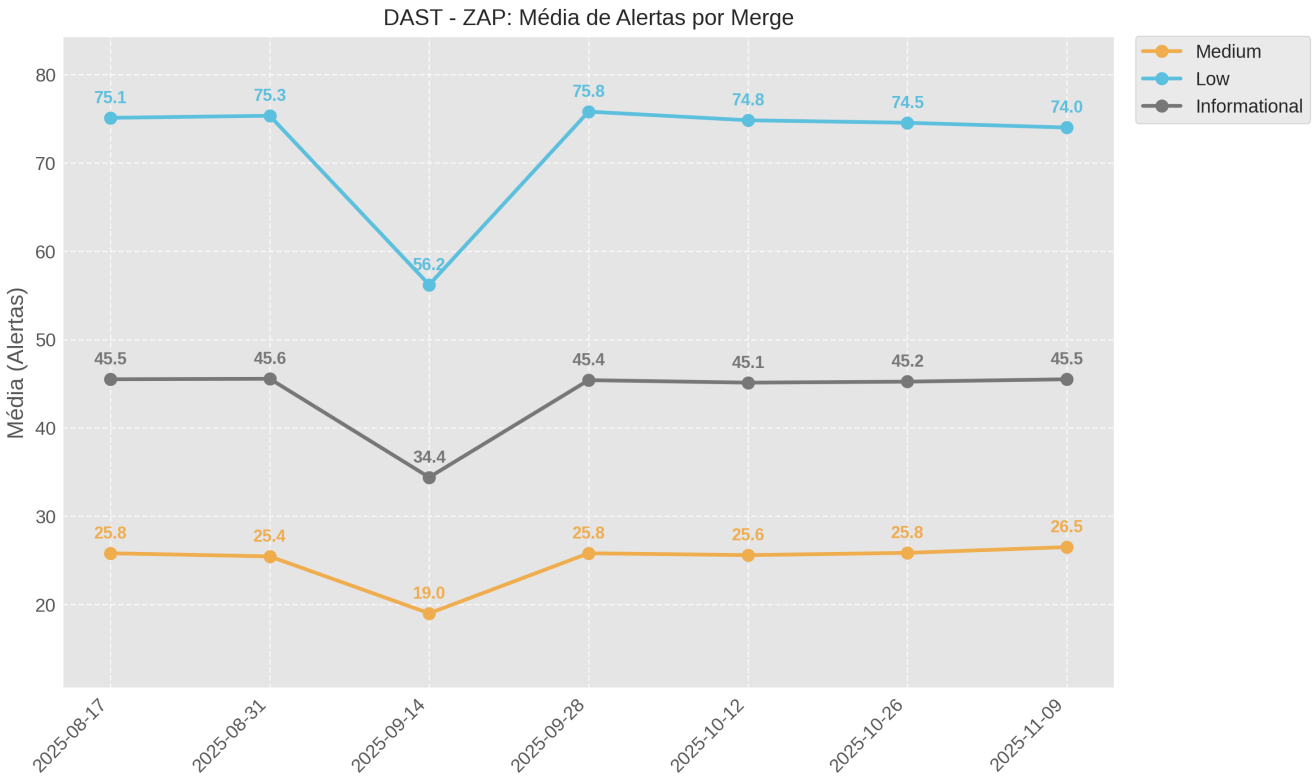
Figura 14 – SAST - Média de Vulnerabilidades por *Merge*



Fonte: Autor

A Figura 15 está relacionada à Métrica 1.2 - Média de vulnerabilidades encontradas por ferramentas DAST, por nível de severidade, por *merge*, e apresenta a média de vulnerabilidades identificadas pelo ZAP (ferramenta DAST) por *merge* ao longo do período analisado. Nesse caso, não foram encontradas vulnerabilidades de criticidade alta, porém a ferramenta identificou vulnerabilidades de criticidade baixa e de caráter informativo. Essas vulnerabilidades também estão representadas no gráfico, mas não são consideradas para o cálculo do indicador, conforme detalhado na Subseção 4.5. Assim como no caso do Brakeman, essa inclusão visa proporcionar uma visão mais completa das vulnerabilidades detectadas pela ferramenta.

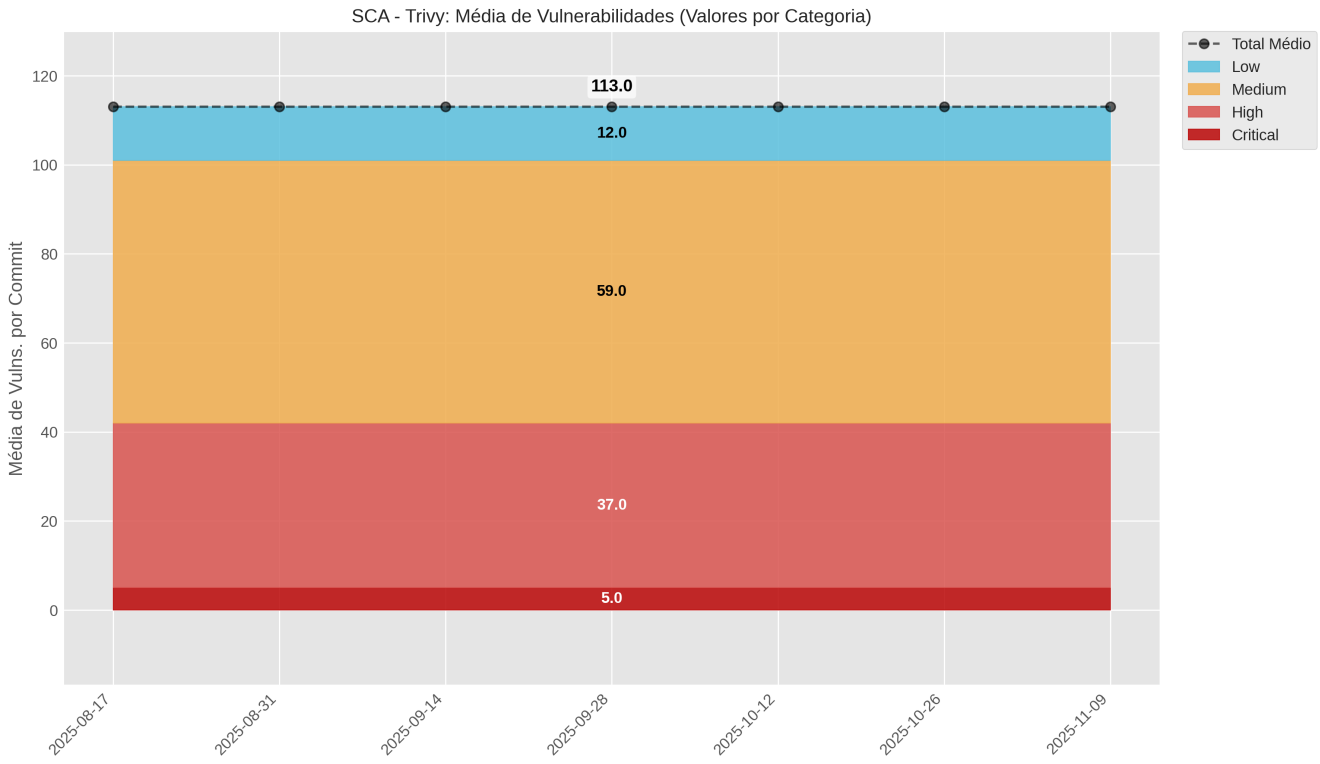
Figura 15 – DAST - Média de Vulnerabilidades por *Merge*



Fonte: Autor

Diferentemente das outras figuras, a Figura 16 não está diretamente relacionada a uma métrica. Ela apresenta a média de vulnerabilidades identificadas pelo Trivy (ferramenta SCA) por *merge* ao longo do período analisado. Essa análise é importante para complementar a compreensão das vulnerabilidades presentes no sistema, uma vez que por meio dele é possível identificar a quantidade de vulnerabilidades por diferentes níveis de criticidade nas bibliotecas utilizadas no projeto.

Figura 16 – SCA - Média de Vulnerabilidades por *Merge*

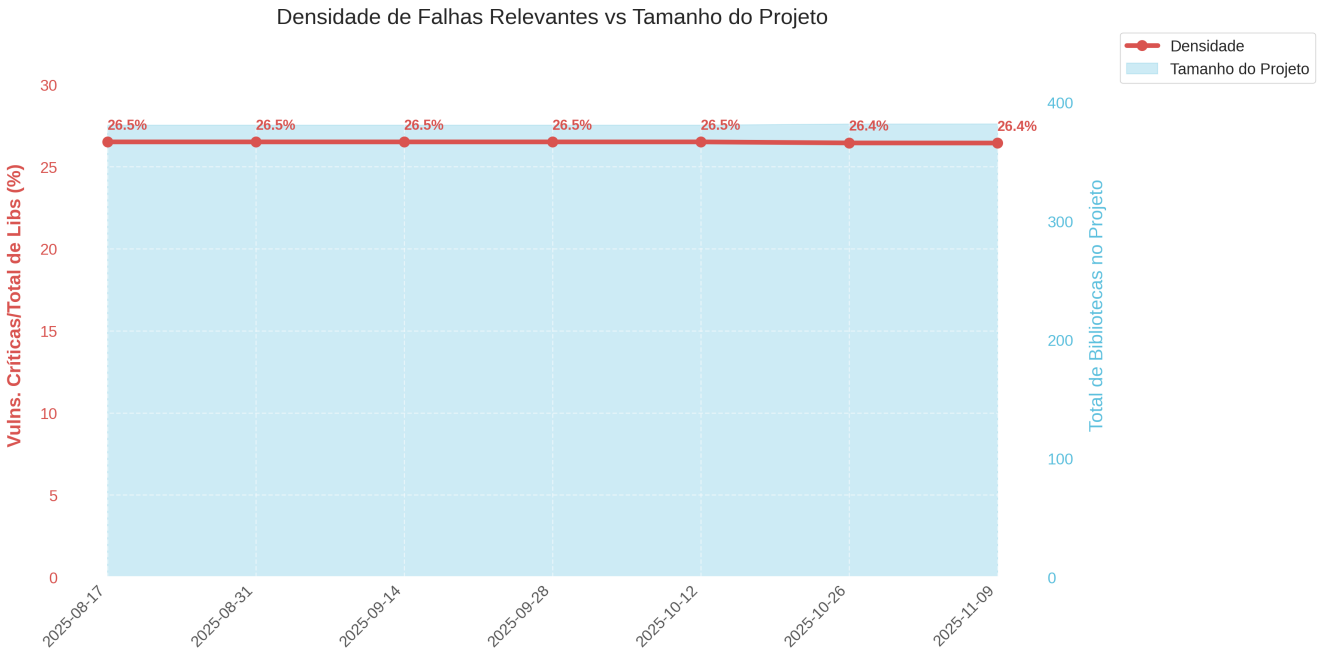


Fonte: Autor

Já a Figura 17, relacionada à Métrica 1.3 - Proporção de vulnerabilidades em bibliotecas encontradas por ferramentas SCA, por nível de severidade, por *merge*, apresenta a proporção de bibliotecas com vulnerabilidades em relação ao total de bibliotecas utilizadas no projeto por *merge* ao longo do período analisado. A linha vermelha representa a porcentagem de bibliotecas com vulnerabilidades de criticidade média ou superior, enquanto a área azul representa o total de bibliotecas do projeto.

A análise dessa métrica ao longo do tempo permite avaliar a eficácia das práticas de gerenciamento de dependências e a adoção de medidas para mitigar riscos associados a bibliotecas vulneráveis. Apesar da pequena redução, 0,1%, observada na proporção de bibliotecas vulneráveis ao longo do período analisado, é importante destacar que a métrica ainda indica uma presença significativa de vulnerabilidades nas dependências do projeto: aproximadamente 26,4% das bibliotecas foram marcadas pelo Trivy com vulnerabilidades médias ou superiores.

Figura 17 – Proporção de Bibliotecas com Vulnerabilidades por *Merge*



Fonte: Autor

Observa-se que, apesar de algumas variações, as médias de vulnerabilidades por *merge* tendem a se manter relativamente estáveis ao longo do tempo. Isso indica que as práticas *DevSecOps* implementadas estão contribuindo para a identificação consistente de vulnerabilidades, tanto na qualidade interna (SAST e SCA) quanto na qualidade externa (DAST) do produto. Um dos motivos para essa estabilidade foi a priorização do time em relação às vulnerabilidades encontradas, pois não foi possível observar a *sprint* em que as correções foram realizadas, apenas o momento em que foram detectadas.

Além disso, as vulnerabilidades não foram analisadas pelo time de segurança, apenas reportadas, o que pode ter impactado a priorização e a correção das mesmas. Isso reforça a importância de uma abordagem contínua e integrada de segurança no ciclo de desenvolvimento, alinhada aos princípios *DevSecOps*, para garantir a identificação e mitigação eficaz de vulnerabilidades ao longo do tempo. Outro fator impactado por esse fato é a quantidade de falsos positivos, que não foi possível analisar, mas que pode ter influenciado os dados e, posteriormente, a resolução desses problemas de segurança. Ainda sobre falsos positivos, vale destacar que a escolha de continuar com o uso da ferramenta Brakeman partiu do time de segurança, pois, segundo a experiência empírica da equipe, por ela ser específica para aplicações desenvolvidas em *Ruby on Rails*, a quantidade de falsos positivos seria menor.

6.1.2 Análise Automática da Segurança e Tomada de Decisões

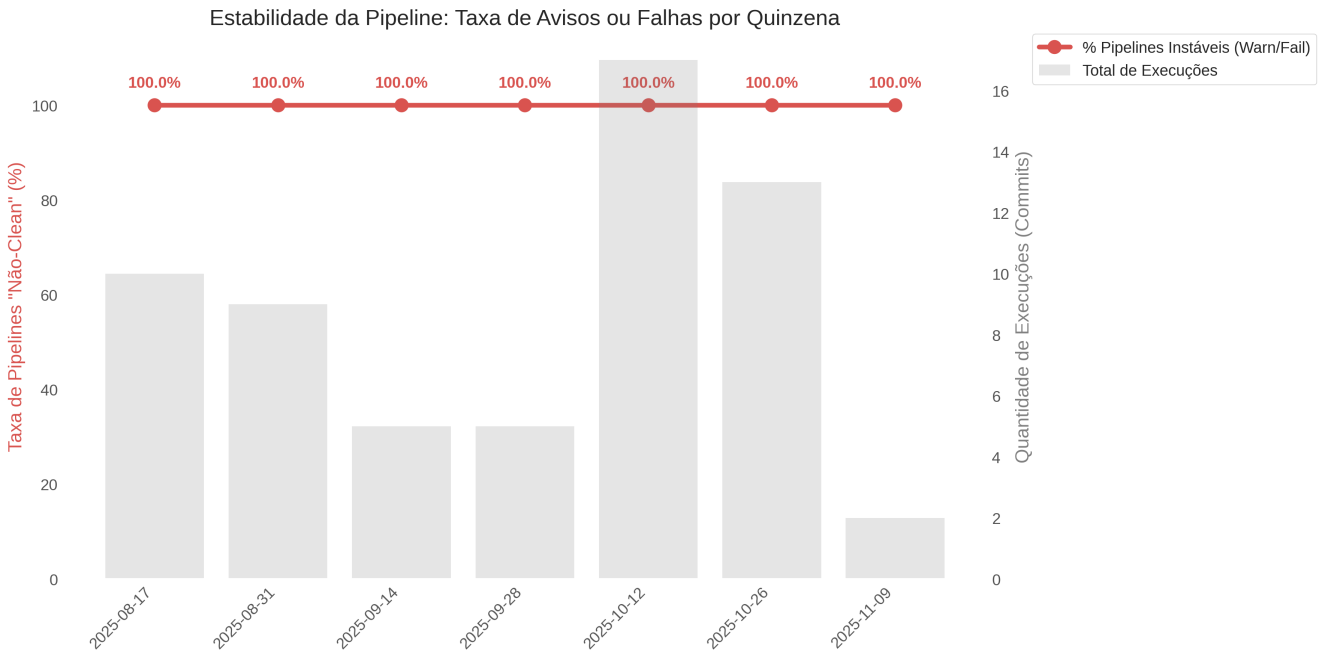
Na Seção 4.5, foi levantada a questão específica QE-2: "A análise automática da segurança do *pipeline* e as métricas coletadas ajudaram na tomada de decisões relacionadas ao projeto?"

Para responder a essa questão, foi analisada a Métrica 2.1, relacionada à estabilidade do *pipeline* de CI/CD, conforme detalhado a seguir. A Figura 18 apresenta as taxas de aviso ou falha no *pipeline* por quinzena ao longo do período analisado.

Como foram implementadas ferramentas que analisaram aspectos do sistema que até então não tinham sido submetidos a esse tipo de análise — SCA com Trivy, DAST com ZAP e SAST no projeto inteiro com o Brakeman — e o time de desenvolvimento não teve tempo hábil para corrigir todas as vulnerabilidades encontradas, é possível observar que o *pipeline* encontrou falhas de segurança em todas as suas execuções. No entanto, é importante destacar que essas variações refletem o processo de adaptação do time de desenvolvimento às novas práticas *DevSecOps* implementadas e que, conforme a cultura do time for se integrando ao novo modo de enxergar a segurança, é provável que eles consigam reduzir esses índices.

De todo modo, o conjunto de métricas *DevOps* DORA (Google, 2024) indica que uma taxa de falhas em torno de 20% é considerada de média *performance*. Com isso, observa-se que o *pipeline* do projeto Brasil Participativo está classificado como de baixa *performance*, o que reforça a necessidade de melhorias contínuas no processo de desenvolvimento e na integração das práticas *DevSecOps*.

Figura 18 – Taxas de Aviso ou Falha por Quinzena



Fonte: Autor

6.2 Análise do Indicador

Retomando a questão específica QE-1: "A aplicação de práticas *DevSecOps* permitiu identificar vulnerabilidades de segurança sob as perspectivas da qualidade interna e externa do produto?"

As métricas 1.1 e 1.2 fornecem os dados necessários para realizar o cálculo da métrica B.34 - *Security Incidents Trend* da norma ISO/IEC 27004 (2016), que fornece um método e valores de referência para avaliar a tendência da segurança de um sistema ao longo do tempo. Observando as informações das Figuras 14 e 15 é possível extrair os dados necessários para realizar o cálculo dessas métricas.

Os critérios de classificação de tendência foram definidos conforme a ISO/IEC 27004 (2016):

- **Melhora (Verde):** se $R < 1,00$
- **Estabilidade (Amarelo):** se $1,00 \leq R \leq 1,30$
- **Degradação (Vermelho):** se $R > 1,30$

Tabela 1 – Análise de Tendência de Vulnerabilidades - SAST (Brakeman)

Categoria	**Média Hist. (6)**	**Média Rec. (2)**	**Razão (R)**	**Status**
High	5,77	5,30	0,92	**Verde**
Medium	8,00	8,00	1,00	**Amarelo**

Fonte: Autor

Tabela 2 – Análise de Tendência de Alertas - DAST (ZAP)

Categoria	**Média Hist. (6)**	**Média Rec. (2)**	**Razão (R)**	**Status**
Medium	24,68	26,15	1,06	**Amarelo**

Fonte: Autor

Pode-se perceber que a análise de tendência das vulnerabilidades identificadas pelas ferramentas SAST e DAST, respectivamente ilustrada pela Tabela 1 e Tabela 2, indica uma estabilidade na segurança do *software* ao longo do tempo, sendo que as vulnerabilidades de alta criticidade identificadas pelo Brakeman apresentaram uma ligeira queda. Esses resultados fornecem *insights* valiosos para a equipe de desenvolvimento e segurança, permitindo que eles monitorem e priorizem as demandas de segurança do *software*.

6.3 Ameaças à Validade do Estudo

Conforme destacado por Runeson e Höst (2009), é essencial considerar as ameaças à validade desde o início da pesquisa para garantir a confiabilidade dos resultados.

Nesta seção, são identificadas as principais ameaças à validade deste estudo de caso e as estratégias adotadas para sua mitigação.

Há quatro preocupações centrais para avaliar a qualidade dos resultados obtidos em uma pesquisa. São conhecidas como testes às ameaças: de constructo, interna, externa e de confiabilidade. Ao conduzir estudos de caso, podemos aplicar diversas estratégias para atender a esses testes. Contudo, nem todas essas estratégias são usadas apenas na fase de planejamento formal. Algumas delas são aplicadas durante a coleta e análise dos dados, ou mesmo nas etapas de estruturação da pesquisa (YIN, 2015).

6.3.1 Ameaças à Validade do Constructo

A ameaça à validade do constructo diz respeito à capacidade do pesquisador interpretar as alterações e variações observadas na execução do estudo de caso de forma a garantir fidedignamente a representação da realidade. Em outras palavras, o pesquisador precisa garantir que os eventos observados *in loco* realmente refletem o fenômeno de forma inequívoca, ou se aconteceram apenas com base nas impressões do pesquisador.

Para realizar esse teste, o pesquisador precisa assegurar o cumprimento de duas etapas:

1. selecionar os tipos específicos de mudanças que devem ser estudadas;
2. demonstrar que as medidas operacionais selecionadas fornecem uma percepção quantitativa dessas mudanças e que são corretas para os conceitos que estão sob estudo.

Dessa maneira, as medidas selecionadas podem não capturar completamente os aspectos de segurança que se pretende avaliar. Para mitigar essa ameaça, foram selecionadas medidas baseadas em modelos e normas de referência na literatura, como a ISO/IEC 27004 (2016), e estudos que apresentam evidências experimentais dos seus "achados", como discutido por Zhang e Zhang (2024).

Além disso, para garantir esse alinhamento, foi adotada a abordagem GQM detalhada na Seção 1.3. Com isso, há uma questão de pesquisa principal, norteadora de todo o trabalho e há outras questões específicas, secundárias, da revisão estruturada da literatura e do estudo de caso. Essas questões possuem suas respectivas métricas que, por sua vez, auxiliam a quantificar ou qualificar as percepções da segurança, por meio os dados coletados e analisados.

6.3.2 Ameaças à Validade Interna

As ameaças à validade interna tratam das relações de causalidade. Portanto, analisam a relação de causa e efeito entre as variáveis observadas.

Por se tratar de um estudo do tipo exploratório, que procura observar e compre-

ender um fenômeno específico ocorrendo em seu ambiente real, o foco é a percepção de todo o contexto. Por isso, esse teste não se aplica a esta pesquisa (YIN, 2015).

6.3.3 Ameaças à Validade Externa

A validade externa, por sua vez, refere-se ao grau de generalização dos resultados observados. Estudos de caso costumam ser criticados em virtude do baixo poder de generalização dos seus resultados, principalmente em comparação a análises quantitativas. No entanto, essa analogia, que considera amostra e população, é equivocada quando se fala em estudos de caso (YIN, 2015).

Análises puramente quantitativas utilizam a generalização estatística em que uma amostra, se bem representada, pode ser facilmente estendida a uma população maior. Já nos estudos de caso, o objetivo do pesquisador é usar um conjunto específico de resultados para corroborar ou expandir uma teoria mais abrangente, em vez de aplicá-los diretamente a um universo estatístico (YIN, 2015).

Este estudo está limitado a um caso único, o projeto Brasil Participativo, que possui características específicas que podem não ser compatíveis ou aplicáveis em outros sistemas de *software* ou contextos. Embora as conclusões sejam específicas deste caso, como forma de mitigar essa ameaça, toda a descrição metodológica e documentação necessária para a compreensão das singularidades do caso Brasil Participativo estarão disponíveis em repositórios públicos. Dessa forma, outros pesquisadores podem analisar a viabilidade da replicação dos resultados em seus contextos específicos.

6.3.4 Ameaças à Confiabilidade

Por fim, a ameaça à confiabilidade refere-se à capacidade de reprodução dos procedimentos por outros pesquisadores. Com isso, verifica-se se os resultados observados são consistentes, caso a reprodução obtenha os mesmos resultados (YIN, 2015) (WOHLIN et al., 2012).

Variações na forma como os dados são coletados e analisados podem afetar os resultados. Para mitigar essa ameaça, serão fornecidas informações detalhadas sobre o produto, o processo, o contexto, o time de desenvolvimento, o conjunto de dados coletados e analisados, além do código-fonte e *scripts* utilizados na construção do tratamento proposto. Todas essas informações estão disponíveis em repositórios públicos. Estão estabelecidos procedimentos claros e automatizados para a coleta de dados, minimizando a intervenção manual e sistematizando a análise. Além disso, todas as ferramentas utilizadas na instrumentação são produtos de *software* de código-fonte livre ou aberto.

7 Conclusão

Neste capítulo, será apresentado o resultado do trabalho desenvolvido, destacando os resultados obtidos e as atividades concluídas. Além disso, serão discutidos os objetivos específicos alcançados, as limitações do estudo e sugestões para trabalhos futuros.

7.1 Resultados Obtidos

Retomando a questão de pesquisa principal definida no início deste trabalho: *como analisar a característica de segurança no desenvolvimento contínuo de sistemas web, considerando as visões de qualidade interna e externa?* Para responder a essa questão, foi realizada uma revisão da literatura com o objetivo de obter embasamento teórico sobre *DevSecOps*, segurança em desenvolvimento contínuo e métricas de segurança. Depois de obter um panorama geral sobre o assunto, foi proposto um estudo de caso em um projeto de *software* livre desenvolvido pela Universidade de Brasília.

O estudo de caso foi estruturado usando a abordagem GQM (*Goal Question Metric*) para definir metas, questões e métricas necessárias para avaliar as práticas analisadas. Assim, foram apresentadas na Seção 4.5 duas questões específicas, QE-1 e QE-2, respectivamente: *a aplicação de práticas DevSecOps permitiu identificar vulnerabilidades de segurança sob as perspectivas da qualidade interna e externa do produto?* E como segunda questão específica: *a análise automática da segurança do pipeline e as métricas coletadas ajudaram na tomada de decisões relacionadas ao projeto?*

Então, para fundamentar a escolha e análise das métricas coletadas e permitir uma análise com valor de referência, a ISO/IEC 27004 (2016) foi utilizada como referência para a definição das métricas, pois ela estabelece um conjunto de métricas para avaliar a segurança de *software* durante todo o seu ciclo de vida e em diferentes aspectos. Já os estudos Zhang e Zhang (2024) e Google (2024), foram usados como referência visto que apresentam métricas relacionadas ao desenvolvimento contínuo seguro de projetos de *software*. Foi a partir deles que as métricas relacionadas ao contexto de integração contínua foram definidas.

Nesse sentido, foram construídas e integradas ao *pipeline* de CI/CD do projeto as ferramentas para coletar as métricas propostas. Em primeiro momento, o *pipeline* oficial do projeto teve o funcionamento da ferramenta SAST alterada para permitir analisar o projeto como um todo e também foi adicionada a ferramenta de SCA para identificar vulnerabilidades em bibliotecas de terceiros. Posteriormente, foi implementado um *pipeline* paralelo para realizar análises DAST, dada a impossibilidade da integração de testes

dinâmicos no *pipeline* do projeto devido à ausência de integração automática em um ambiente de homologação. Dessa forma, foi possível coletar as métricas propostas para responder às questões do estudo de caso.

Desse modo, conclui-se da análise que tanto na perspectiva de qualidade interna quanto na externa, a aplicação das práticas *DevSecOps* permitiu identificar vulnerabilidades de segurança no projeto analisado. Na perspectiva de qualidade interna, as ferramentas SAST e SCA integradas ao *pipeline* de CI/CD possibilitaram a detecção precoce de vulnerabilidades no código-fonte e nas bibliotecas utilizadas.

Já na perspectiva de qualidade externa, o estudo de caso indicou que a análise automática da segurança do *pipeline* e as métricas coletadas fornecem parâmetros quantitativos para a tomada de decisões relacionadas ao projeto. A taxa de *pipelines* sinalizados devido à descoberta de vulnerabilidades forneceu *insights* valiosos para o time de desenvolvimento, permitindo priorizar correções e melhorias no código-fonte, além de permitir obter uma visão geral da segurança do projeto ao longo do tempo.

Enquanto na perspectiva de qualidade em uso, a ferramenta DAST permitiu identificar vulnerabilidades que poderiam ser exploradas em um ambiente de produção, proporcionando uma visão mais abrangente da segurança do sistema.

Apesar de não ter sido possível coletar a opinião das equipes em relação às novas práticas implementadas por meio do questionário, durante as reuniões de alinhamento com os times foi relatado, em caráter pessoal, o contentamento com as práticas *DevSecOps* implementadas que serão de grande valia para o projeto.

7.2 Atividades Concluídas

As atividades planejadas para este trabalho foram definidas na primeira etapa do estudo e podem ser encontradas na Seção 1.6. O Quadro 7 lista todas as atividades, além de apresentar a situação atual de cada uma e onde os resultados de cada atividade podem ser encontrados.

7.3 Objetivos Específicos

Os objetivos do trabalho definidos na Seção 1.4 foram atingidos com sucesso. O Quadro 8 registra cada um deles e indica a situação atual e onde os resultados foram registrados.

Quadro 7: Atividades Concluídas

Atividade	**Situação**	**Resultados**
Contextualização sobre Engenharia de *Software* Experimental	Concluída	2.1
Definição do GQM	Concluída	1.3
Elaboração do Protocolo de Revisão da Literatura	Concluída	3.1
Seleção dos Artigos	Concluída	3.2
Análise do Material Selecionado	Concluída	3.3
Definição da Proposta de Solução	Concluída	4
Redação da Monografia	Concluída	-
Revisão	Concluída	-
Primeira Defesa	Concluída	-
Evolução do Trabalho	Concluída	-
Preparação e Coleta de Dados	Concluída	5
Tratamento dos Dados	Concluída	5
Documentação e Análise dos Resultados	Concluída	6
Redação da monografia	Concluída	-
Revisão	Concluída	-
Apresentação Final	Agendada	-

Fonte: Autor

Quadro 8: Objetivos Concluídos

Objetivo	**Situação**	**Resultados**
Fundamentar teoricamente os conceitos de *DevSecOps*, modelos de segurança e metodologias de desenvolvimento seguro	Alcançado	3 e 2
Incorporar um conjunto de práticas *DevSecOps* ao ciclo de desenvolvimento do produto de *software* sob análise	Alcançado	5
Planejar um estudo de caso focado na observação das práticas implementadas	Alcançado	4
Conduzir o estudo de caso, realizando a coleta e a análise das medidas e métricas de segurança, que apoiem a discussão dos resultados alcançados	Alcançado	5 e 6
Apresentar as conclusões e os *insights* resultantes desta investigação.	Alcançado	6

Fonte: Autor

7.4 Limitações

Apesar dos resultados positivos obtidos neste trabalho, algumas limitações foram identificadas durante o desenvolvimento do estudo de caso. Primeiramente, a análise foi realizada em um único projeto de *software* livre, o que pode limitar a generalização dos resultados para outros contextos e tipos de projetos. Além disso, a ausência de um ambiente de homologação automatizado impediu a integração direta da ferramenta DAST no *pipeline* oficial do projeto, o que poderia ter proporcionado uma análise mais integrada e

contínua.

Ademais, a coleta de opinião do time por meio de um questionário não foi possível, o que limitou a avaliação qualitativa das percepções dos desenvolvedores em relação às práticas *DevSecOps* implementadas. Essa limitação pode ter impactado a compreensão completa dos benefícios e desafios enfrentados pela equipe durante a adoção dessas práticas.

Outrossim, a necessidade de adaptação das métricas propostas para o contexto específico do projeto analisado pode ter influenciado os resultados obtidos, uma vez que algumas métricas podem não ter sido totalmente adequadas para avaliar a segurança no desenvolvimento contínuo do sistema em questão.

7.5 Trabalhos Futuros

Os trabalhos subsequentes, além de superar as limitações identificadas, podem explorar diversas direções para aprofundar a compreensão e a aplicação das práticas *DevSecOps* no desenvolvimento contínuo de sistemas *web*. Primeiramente, usar diferentes ferramentas de análise de segurança, tanto SAST quanto DAST, pode proporcionar uma visão mais abrangente das vulnerabilidades presentes no código-fonte e no ambiente de execução. A comparação entre diferentes ferramentas também pode ajudar a identificar aquelas que melhor se adaptam às necessidades específicas do projeto.

Também é importante que seja elaborado um relatório detalhado das vulnerabilidades encontradas, visando identificar falsos positivos e ajudar na priorização das correções. Esse relatório pode servir como base para futuras análises e melhorias no processo de desenvolvimento seguro.

Adicionalmente, um trabalho futuro poderia desenvolver um *dashboard* centralizado para o monitoramento contínuo e visualização das vulnerabilidades detectadas ao longo do ciclo de vida do software. Essa visibilidade é fundamental não apenas para a equipe técnica, mas também para a gestão, pois transforma dados brutos de segurança em inteligência estratégica, facilitando a governança sobre o nível de exposição do sistema.

Outra direção promissora seria a realização de estudos de caso adicionais em diferentes tipos de projetos e contextos organizacionais. Isso permitiria validar os resultados obtidos neste trabalho e identificar práticas, ferramentas e métricas que sejam mais eficazes em diferentes cenários.

Referências

ALJOHANI, M. A.; ALQAHTANI, S. S. A unified framework for automating software security analysis in devsecops. In: . [s.n.], 2023. Cited by: 9. Disponível em: <https://www.scopus.com/inward/record.uri?eid=2-s2.0-85153866654&doi=10.1109%2fICSCA57840.2023.10087568&partnerID=40&md5=ddab1036dfd21896853bbfa52e8bf3ad>. Citado na página 44.

BASILI, V. et al. Goal question metric (gqm) approach. In: _____. [S.l.: s.n.], 1994. ISBN 9780471028956. Citado na página 32.

BASILI, V. R.; CALDIERA, G.; ROMBACH, H. D. The Goal Question Metric Approach. In: MARCINIAK, J. J. (Ed.). Encyclopedia of Software Engineering. [S.l.]: Wiley, 1994. v. 1, p. 528–532. Citado 2 vezes nas páginas 30 e 70.

BOEHM, B. W. Characteristics of Software Quality. North-Holland, 1978. Disponível em: <https://books.google.com.br/books?id=jLFXswEACAAJ>. Citado na página 27.

BOEHM, B. W. Characteristics of Software Quality. [S.l.]: North-Holland, 1978. (TRW Series of Software Technology). Citado na página 40.

Brasil. Presidência da República. Plataforma Brasil Participativo. 2025. Disponível em: <https://brasilparticipativo.presidencia.gov.br/processes/brasilparticipativo/f/1400/>. Citado na página 67.

Decidim. About Decidim - Decidim Docs. 2025. Disponível em: <https://docs.decidim.org/en/develop/understand/about>. Citado na página 67.

DROMEY, R. G. A model for software product quality. IEEE Trans. Softw. Eng., IEEE Press, v. 21, p. 146–162, 2 1995. ISSN 0098-5589. Disponível em: <http://dx.doi.org/10.1109/32.345830>. Citado 2 vezes nas páginas 27 e 40.

DíAZ, J. et al. Harmonizing devops taxonomies - a grounded theory study. 08 2022. Citado na página 29.

ELSEVIER. Banco de dados de resumos e citações organizado por especialistas. 2024. Disponível em: <https://www.periodicos.capes.gov.br/>. Citado na página 47.

FITZGERALD, B.; STOL, K.-J. Continuous software engineering: A roadmap and agenda. The Journal of Systems & Software, v. 123, p. 176–189, 2017. Citado na página 28.

Forum of Incident Response and Security Teams (FIRST). Common Vulnerability Scoring System. 2015. Acesso em: 06 de outubro de 2025. Disponível em: <https://www.first.org/cvss/>. Citado na página 43.

FRANçA, B. B.; JUNIOR, H. J.; TRAVASSOS, G. Characterizing devops by hearing multiple voices. In: . [S.l.: s.n.], 2016. Citado na página 29.

Google. DORA. [S.l.], 2024. Acessado em: 20 de julho de 2025. Disponível em: <https://dora.dev>. Citado 7 vezes nas páginas 43, 50, 59, 70, 73, 86 e 91.

ISO/IEC 25000.
ISO/IEC 25000:2014 Systems and software engineering — Systems and software Quality Requirements a
Geneva, CH, 2014. Citado na página 40.

ISO/IEC-25010. ISO/IEC 25010 System and software quality models. [S.l.], 2010.
Citado 3 vezes nas páginas 27, 41 e 42.

ISO/IEC 25010.
ISO/IEC 25010:2023 Systems and software engineering — Systems and software Quality Requirements a
Geneva, CH, 2023. Citado 3 vezes nas páginas 40, 41 e 70.

ISO/IEC 27000.
ISO/IEC 27000:2018 Information technology — Security techniques — Information security management
Geneva, CH, 2018. Citado na página 42.

ISO/IEC-27001. Information technology – Security techniques – Information security management systems – Requirements. 2022. Available at: https://www.iso.org/standard/70032.html. Citado na página 27.

ISO/IEC 27001.
ISO/IEC 27001:2022 Information security, cybersecurity and privacy protection — Information security
Geneva, CH, 2022. Citado 2 vezes nas páginas 42 e 45.

ISO/IEC 27004.
ISO/IEC 27004:2016 Information technology — Security techniques — Information security management
Geneva, CH, 2016. Citado 8 vezes nas páginas 42, 43, 70, 72, 78, 87, 88 e 91.

ISO/IEC-27034. Information technology – Security techniques – Application security – Part 1: Overview and concepts. 2011. ISO/IEC 27034-1:2011. Citado na página 27.

ISO/IEC 9126.
ISO/IEC 9126-1:2001 Software engineering — Product quality — Part 1: Quality model.
Geneva, CH, 2001. Esta norma foi descontinuada e substituída pela ISO/IEC 25010:2023.
Citado na página 41.

ISO/IEC-9126. ISO/IEC 9126. Software engineering – Product quality. [S.l.]: ISO/IEC, 2001. Citado na página 27.

JOSHI, H. A secure software development methodology for enterprise business applications. In: . [s.n.], 2024. p. 7 – 12. Cited by: 1. Disponível em: <https://www.scopus.com/inward/record.uri?eid=2-s2.0-85217554758&doi=10.1109%2fICODSE63307.2024.10829891&partnerID=40&md5=aa77a0827de37592c910cb0f32cbb0fc>. Citado na página 60.

KITCHENHAM, B.; CHARTERS, S. et al. Guidelines for performing systematic literature reviews in software engineering. Keele, UK, 2007. Citado 2 vezes nas páginas 33 e 47.

KUDRIAVTSEVA, A.; GADYATSKAYA, O. You cannot improve what you do not measure: A triangulation study of software security metrics. In: . [s.n.], 2024. p. 1223 – 1232. Cited by: 6; All Open Access, Hybrid Gold Open Access. Disponível em: <https://www.scopus.com/inward/record.uri?eid=2-s2.0-85194836591&doi=10.1145%2f3605098.3635892&partnerID=40&md5=0beecf2d8290ab7b673bdda3b6300a57>. Citado 2 vezes nas páginas 29 e 60.

KUSHWAHA, M. K.; DAVID, P.; SUSEELA, G. Automation and devsecops: Streamlining security measures in financial system. In: . [s.n.], 2024. Cited by: 0. Disponível em: <https://www.scopus.com/inward/record.uri?eid=2-s2.0-85205770194&doi=10.1109%2fCONECCT62155.2024.10677271&partnerID=40&md5=6dd76e3d104de49d5559093b3f58c303>. Citado na página 61.

LANGE, F.; KUNZ, I. Evolution of secure development lifecycles and maturity models in the context of hosted solutions. Journal of Software: Evolution and Process, v. 36, n. 12, 2024. Cited by: 0; All Open Access, Hybrid Gold Open Access. Disponível em: <https://www.scopus.com/inward/record.uri?eid=2-s2.0-85200057398&doi=10.1002%2fsmr.2711&partnerID=40&md5=9b0595cd71d28171da66b90852ed7301>. Citado 3 vezes nas páginas 45, 58 e 59.

LETOUZEY, J.-L.; COQ, T. The sqale method for evaluating technical debt. In: 14th European Conference on Software Maintenance and Reengineering (CSMR). [S.l.]: IEEE, 2010. p. 195–198. Citado na página 41.

LUZ, W. P.; PINTO, G.; BONIFÁCIO, R. Adopting devops in the real world: A theory, a model, and a case study. J. Syst. Softw., v. 157, 2019. Disponível em: <https://doi.org/10.1016/j.jss.2019.07.083>. Citado na página 29.

LóPEZ, L. et al. Quality measurement in agile and rapid software development: A systematic mapping. Journal of Systems and Software, v. 186, p. 111187, 2022. ISSN 0164-1212. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0164121221002661>. Citado na página 28.

LóPEZ, L. et al. Q-rapids tool prototype: Supporting decision-makers in managing quality in rapid software development. In: Forum at the 30th International Conference on Advanced Information Systems Engineering (CAiSE). [S.l.]: Springer, 2018. (Lecture Notes in Business Information Processing, v. 317), p. 157–164. Citado na página 41.

MASOOD, A.; JAVA, J. Static analysis for web service security - tools & techniques for a secure development life cycle. In: . [S.l.: s.n.], 2015. Citado na página 57.

MCCALL, J.; RICHARDS, P. K.; WALTERS, G. F. Factors in software quality. volume i, ii and iii. concepts and definitions of software quality. US Rome Air Development Center Reports, US Department of Commerce, USA, p. 168, 1977. Citado na página 27.

MCCALL, J. A.; RICHARDS, P. K.; WALTERS, G. F. Factors in Software Quality. [S.l.], 1977. Citado 2 vezes nas páginas 40 e 41.

Mitre Corporation. Common Weakness Enumeration (CWE™). 1999. Acessado em: 24-07-2025. Disponível em: <http://web.archive.org/web/20080207010024/http://www.808multimedia.com/winnt/kernel.htm>. Citado 2 vezes nas páginas 28 e 43.

NOCERA, S. et al. A large-scale fine-grained empirical study on security concerns in open-source software. In: . [s.n.], 2023. p. 418 – 425. Cited by: 3. Disponível em: <https://www.scopus.com/inward/record.uri?eid=2-s2.0-85183330517&doi=10.1109%2fSEAA60479.2023.00069&partnerID=40&md5=cedd775ef204cc60b75b12f44bdda858>. Citado na página 61.

OWASP. OWASP Software Assurance Maturity Model (SAMM). 2020. Https://owaspsamm.org/. Acessado: 24-07-2025. Citado na página 28.

OWASP. OWASP Top Ten 2021. 2021. <https://owasp.org/TopTen/>. OWASP Top Ten 2021 Report. Citado 2 vezes nas páginas 28 e 43.

OWASP Foundation. OWASP Foundation, the Open Source Foundation for Application Security. 2025. Acesso em: 06 de outubro de 2025. Disponível em: <https://owasp.org/>. Citado na página 43.

PAI, M. et al. Clinical research methods. THE NATIONAL MEDICAL JOURNAL OF INDIA, v. 17, n. 2, 2004. Citado 2 vezes nas páginas 33 e 47.

PAI, M. et al. Systematic reviews and meta-analyses: an illustrated, step-by-step guide. National Medical Journal of India, v. 17, n. 2, p. 86–95, mar-apr 2004. PMID: 15141602. Citado na página 48.

RAJAPAKSE, R. N. et al. Challenges and solutions when adopting devsecops: A systematic review. Information and Software Technology, v. 141, 2022. Cited by: 91; All Open Access, Green Open Access. Disponível em: <https://www.scopus.com/inward/record.uri?eid=2-s2.0-85114377924&doi=10.1016%2fj.infsof.2021.106700&partnerID=40&md5=8d37a7b4dea325db0f7ecd9a9ed17a0e>. Citado 5 vezes nas páginas 29, 56, 57, 59 e 69.

RAMIREZ, A.; AIELLO, A.; LINCKE, S. J. A survey and comparison of secure software development standards. In: . [s.n.], 2020. Cited by: 17. Disponível em: <https://www.scopus.com/inward/record.uri?eid=2-s2.0-85100614059&doi=10.1109%2fCMI51275.2020.9322704&partnerID=40&md5=8b4d0a69a6b627d3d34f5be03aa2c6ba>. Citado na página 61.

RANGNAU, T. et al. Continuous security testing: A case study on integrating dynamic security testing tools in ci/cd pipelines. In: . [s.n.], 2020. p. 145 – 154. Cited by: 57; All Open Access, Green Open Access. Disponível em: <https://www.scopus.com/inward/record.uri?eid=2-s2.0-85096513818&doi=10.1109%2fEDOC49727.2020.00026&partnerID=40&md5=c40a98ef4d8eff4fb8b08e234290f023>. Citado na página 57.

React Team. Critical Security Vulnerability in React Server Components. 2025. Acesso em: 07 dez. 2025. Disponível em: <https://react.dev/blog/2025/12/03/critical-security-vulnerability-in-react-server-components>. Citado na página 28.

RUNESON, P.; HöST, M. Guidelines for conducting and reporting case study research in software engineering. Empirical Softw. Engg., Kluwer Academic Publishers, USA, v. 14, n. 2, p. 131–164, abr. 2009. ISSN 1382-3256. Disponível em: <https://doi.org/10.1007/s10664-008-9102-8>. Citado 2 vezes nas páginas 65 e 87.

SAEED, H. et al. Review of techniques for integrating security in software development lifecycle. Computers, Materials and Continua, v. 82, n. 1, p. 139 – 172, 2025. Cited by: 2; All Open Access, Gold Open Access. Disponível em: <https://www.scopus.com/inward/record.uri?eid=2-s2.0-85214507365&doi=10.32604%2fcmc.2024.057587&partnerID=40&md5=16c958450978504c24f727dbb66e06d5>. Citado 2 vezes nas páginas 61 e 70.

SIAVVAS, M.; AMPATZOGLOU, A.; STAMELOS, I. Qatch: A framework for software product quality assessment using fuzzy multi-criteria decision making. Expert Systems with Applications, Elsevier, v. 86, p. 350–366, 2017. Citado na página 41.

SIAVVAS, M. et al. A hierarchical model for quantifying software security based on static analysis alerts and software metrics. Software Quality Journal, Kluwer Academic Publishers, USA, v. 29, n. 2, p. 431–507, jun. 2021. ISSN 0963-9314. Disponível em: <https://doi.org/10.1007/s11219-021-09555-0>. Citado 2 vezes nas páginas 27 e 29.

SIAVVAS, M. et al. A hierarchical model for quantifying software security based on static analysis alerts and software metrics. Software Quality Journal, v. 29, n. 2, p. 431 – 507, 2021. Cited by: 21; All Open Access, Green Open Access. Disponível em: <https://www.scopus.com/inward/record.uri?eid=2-s2.0-85106207514&doi=10.1007%2fs11219-021-09555-0&partnerID=40&md5=b0ecf71985d4ef4c10438eabe73da349>. Citado 2 vezes nas páginas 66 e 69.

SINAN, M.; SHAHIN, M.; GONDAL, I. Integrating security controls in devsecops: Challenges, solutions, and future research directions. Journal of Software: Evolution and Process, v. 37, n. 6, 2025. Cited by: 0; All Open Access, Hybrid Gold Open Access. Disponível em: <https://www.scopus.com/inward/record.uri?eid=2-s2.0-105007632373&doi=10.1002%2fsmr.70029&partnerID=40&md5=cbcf15029b0648ce3192ffaeaf651643>. Citado 2 vezes nas páginas 44 e 45.

The MITRE Corporation. CVE - Common Vulnerabilities and Exposures. 1999. Acesso em: 06 de outubro de 2025. Disponível em: <https://www.cve.org/>. Citado na página 43.

WAGNER, S. et al. The quamoco product quality modeling and assessment approach. In: Proceedings of the 34th International Conference on Software Engineering (ICSE). [S.l.]: IEEE Press, 2012. p. 1133–1142. Citado na página 41.

WOHLIN, C. et al. Experimentation in Software Engineering. Berlin, Heidelberg: Springer Berlin Heidelberg, 2024. ISBN 978-3-662-69306-3. Disponível em: <https://link.springer.com/book/10.1007/978-3-662-69306-3>. Citado 2 vezes nas páginas 37 e 38.

WOHLIN, C. et al. Experimentation in Software Engineering. [S.l.]: Springer Publishing Company, Incorporated, 2012. ISBN 3642290434, 9783642290435. Citado 2 vezes nas páginas 32 e 89.

YIN, R. Estudo de Caso - 5.Ed.: Planejamento e Métodos. Bookman Editora, 2015. ISBN 9788582602324. Disponível em: <https://books.google.com.br/books?id=EtOyBQAAQBAJ>. Citado 5 vezes nas páginas 38, 65, 71, 88 e 89.

ZHANG, J. Y.; ZHANG, Y. Quantitative devsecops metrics for cloud-based web microservices. IEEE Access, v. 12, p. 160317 – 160342, 2024. Cited by: 2; All Open Access, Gold Open Access. Disponível em: <https://www.scopus.com/inward/record.uri?eid=2-s2.0-85208091278&doi=10.1109%2fACCESS.2024.3486314&partnerID=40&md5=cc74abbeeef3802bac7d726302f6c5e2>. Citado 7 vezes nas páginas 50, 58, 69, 70, 73, 88 e 91.