Mapeamento Sistemático de Comparativos de Ferramentas de Análise Estática de Código com Foco na Segurança

Francisco Jean S. Sousa[1], João Gustavo I. Braga[1], Anna Beatriz V. Sousa[1], Valéria Lelli L. Dantas[1], Rossana M. C. Andrade[1], Ismayle S. Santos[2]

[1]Departamento de Computação – Universidade Federal do Ceará (UFC)

[2]Centro de Ciências e Tecnologia – Universidade Estadual do Ceará (UECE)

Grupo de Redes de Computadores, Engenharia de Software e Sistemas (GREat)

{franciscojean,gustavoirineu,biavieiras}@alu.ufc.br

{valerialelli,rossana}@ufc.br, ismayle.santos@uece.br

Resumo. *A garantia de segurança é peça fundamental no processo de desenvolvimento de soluções de software, sendo crucial supervisionar esse aspecto no ciclo de vida da aplicação, desde a concepção da ideia. A análise estática, aplicada na etapa de implementação, utiliza ferramentas automatizadas para prevenir falhas no código-fonte. Assim, a escolha das ferramentas de análise estática que sejam mais eficazes e abrangentes torna-se um fator crítico para identificar e corrigir problemas, por exemplo, relacionados à segurança. Este estudo realiza um mapeamento sistemático para identificar quais ferramentas de análise estática de segurança (SAST, Static Application Security Testing) são melhores avaliadas em estudos comparativos. Foram inicialmente coletados 258 estudos, dos quais 15 foram selecionados, utilizando como fonte três bases de dados, resultando em 64 ferramentas analisadas, cobrindo seis linguagens de programação avaliadas com base em 18 métricas distintas.*

Abstract. *Security assurance is a fundamental element in the development process of software solutions, and it is crucial to supervise this aspect throughout the application's life cycle, starting from the initial idea. Static analysis, applied during the implementation phase, uses automated tools to prevent flaws in the source code. Thus, the choice of static analysis tools that are more effective and comprehensive becomes a critical factor in identifying and correcting issues, such as those related to security. This study conducts a systematic mapping to identify which static application security testing (SAST) tools are best evaluated in comparative studies. Initially, 258 studies were collected, from which 15 were selected, using three different databases as sources, resulting in 64 tools analyzed, covering six programming languages evaluated based on 18 distinct metrics.*

1. Introdução

As aplicações de *software* de mundo real exigem um cuidado especial com a segurança [Chess and McGraw 2004] e, para isso, é necessário o desenvolvimento de técnicas para prevenção de vulnerabilidades. Uma técnica de verificação bem utilizada para inspecionar código-fonte é a análise estática de código, também conhecida por teste estático

[ISO/IEC/IEEE 29119-1 2022]. Essa técnica envolve examinar o código-fonte com base em um conjunto de regras de qualidade sem executá-lo.

Nessa direção, o teste estático de segurança de aplicações (*SAST, Static Application Security Testing*) é uma das técnicas mais populares existentes para detecção e ponderamento de erros que podem levar a falhas de segurança nas aplicações em processo de construção [Li et al. 2023]. Com a adesão de diversos prestadores de serviços de implementação de *software* ao conceito de *Shift-Left*, que defende a ideia de que os testes devem acontecer previamente para acelerar a detecção de falhas e fragilidades [Charoenwet et al. 2024a], essa técnica se difundiu e inúmeras propostas de ferramentas para automação desse processo surgiram, visando uma melhor cobertura do código, num horizonte mais amplo de linguagens de programação e sob boas métricas de avaliação.

Tendo em vista esse aumento da necessidade de controle da qualidade das aplicações desenvolvidas, a decisão de qual *software* de análise estática com foco em segurança utilizar no decorrer de um projeto se torna cada vez mais desafiadora. A fim de direcionar a escolha dessas ferramentas, o presente estudo procura levantar, a partir de estudos já existentes, resultados concretos a respeito da eficiência, da qualidade e da abragência de ferramentas SAST. Foram selecionados, inicialmente, 258 estudos em três bases de dados consolidadas na comunidade científica, a partir de uma *string* de busca bem definida. Esses estudos passaram por três rodadas de refinamento, nas quais a partir de critérios de inclusão e exclusão, os 15 estudos mais aptos permaneceram e auxiliaram a responder as três questões de pesquisas investigadas neste trabalho.

Sob o conteúdo levantado nos estudos selecionados, foi realizado um apanhado de SASTs, das linguagens de programação compatíveis à essas ferramentas e das métricas mais utilizadas nos comparativos para mensurar a qualidade de uma ferramenta em relação às demais. A partir disso, foram selecionadas para análise, ao todo, 64 SASTs diferentes aplicáveis a 6 linguagens de programação de mercado e avaliadas através de 18 métricas variadas. Os resultados obtidos revelam que uma possível combinação entre SASTs pode levar a uma melhor cobertura de fragilidades. Em um âmbito mais individual, a ferramenta CodeQL, bastante citada, foi reconhecida pela sua capacidade de identificar vulnerabilidades e pela boa taxa de *true positives* nas detecções.

Este artigo está estruturado da seguinte forma: A seção 2 apresenta conceitos importantes para um bom entendimento do trabalho realizado; A seção 3 apresenta trabalhos relacionados a este; A seção 4 descreve a metodologia utilizada; A seção 5 detalha os resultados obtidos; E por fim, a seção 6 coclui a discussão e apresenta possibilidades de trabalhos futuros.

2. Fundamentação Teórica

Esta seção apresenta conceitos fundamentais para o bom entendimento do escopo no qual a pesquisa está inserida, abordando temas relacionados as ferramentas SASTs e as métricas mais utilizadas para compara-lás, segundo os estudos selecionados.

2.1. Análise estática Automatizada

A análise estática é uma técnica de verificação estática também conhecida como teste estático [ISO/IEC/IEEE 29119-1 2022]. Essa técnica busca identificar potenciais problemas em representações imutáveis de sistemas de *software* — e.g., especificação de re-

quisitos, código-fonte, e documentação de testes — averiguando se o produto está sendo construído levando em conta padrões de qualidade [D 2024]. Tal processo é composto por várias técnicas, que podem ser executadas de forma manual, por um inspetor ou por ferramentas automatizadas, que realizam uma varredura sistemática na estrutura dos artefatos de entrada, que no contexto da segurança procura por amostras que evidenciem a presença de vulnerabilidades. [et. al. 2004]. Para facilitar o processo de análise estática, é comum utilizar ferramentas que implementam regras de qualidade e verificam automaticamente se essas regras estão sendo violadas em um artefato de software, por exemplo, o código-fonte.

Segundo abordado por Sommerville [SOMMERVILLE 2011], os analisadores estáticos são ferramentas de *software* que automatizam a varredura do texto-fonte de um programa, explorando sua estrutura com o objetivo de identificar indícios de defeitos. Cada defeito encontrado gera o que é descrito como alertas [Pashchenko et al. 2017], que especificam cada erro encontrado.

2.2. Avaliação Comparativa de Ferramentas SASTs

O processo de avaliar o desempenho de ferramentas SASTs pode ser conduzido a partir de benchmarks, que têm por objetivo fornecer *insights* valiosos para contribuir na escolha de quais ferramentas serão utilizadas em atividades de análise estática de código com foco na segurança [Pashchenko et al. 2017].

Como fonte de informação da comparação, as métricas de avaliação são variáveis utilizadas para representar aspectos distintos do desempenho das ferramentas de análise estática de código na identificação automatizada de vulnerabilidades de segurança [Alqaradaghi and Kozsik 2024]. A seguir, estão listadas a descrição ou a fórmula para cada uma das métricas mais citadas nos estudos mapeados:

- **True Positive (TP)**: O total de alertas gerados pela ferramenta onde foi atestada a existência do problema no local indicado. [Brito et al. 2023]
- **False-Positive (FP)**: A quantidade de alertas onde um trecho de código seguro é sinalizado como vulnerável. [Li et al. 2024]
- **False Negative (FN)**: A quantidade de casos onde a ferramenta falhou em gerar alertas para vulnerabilidades reais presentes no código. [Li et al. 2024]
- **Recall / True-Positive-Rate (TPR)**: $\frac{TP}{TP + FN}$ [Brito et al. 2023]
- **Precision**: $\frac{TP}{TP + FP}$ [Pashchenko et al. 2017]
- **F1-Score**: $2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$ [Esposito et al. 2024]

3. Trabalhos Relacionados

Esta seção apresenta uma visão geral de estudos secundários que investigam ferramentas de análise estática de código (SASTs) com foco em segurança, bem como pesquisas que abordam metodologias de avaliação comparativa dessas ferramentas.

[Stefanović et al. 2020] realizaram uma revisão sistemática da literatura sobre ferramentas de análise estática, com foco em identificar categorias, características e limitações dessas ferramentas. [de Mendonça et al. 2013] realizaram um mapeamento sistemático com o objetivo de organizar as técnicas e ferramentas de análise estática, classificando-as por tipo de técnica utilizada e domínio de aplicação. O trabalho destaca a evolução das técnicas ao longo dos anos e sugere a necessidade de mais estudos empíricos comparativos entre as ferramentas. Mais recentemente, Bhutani et al.[Bhutani et al. 2024] investigaram o comportamento prático de ferramentas populares de análise de código-fonte, identificando diferenças significativas na detecção de defeitos e na qualidade dos relatórios gerados.

Com base nessas contribuições, o presente trabalho busca complementar a literatura existente ao realizar um mapeamento sistemático com foco específico na comparação entre ferramentas de análise estática voltadas à segurança, considerando sua eficácia na detecção de vulnerabilidades. Isso se faz necessário pois não se encontrou um estudo que mapeasse e sintetizasse resultados de variados comparativos, dando uma visão geral do atual estado da arte do nível das ferramentas SASTs amplamente utilizadas.

4. Metodologia

Nesta seção é descrita a abordagem para o planejamento e condução da pesquisa com base nas diretrizes de mapeamento sistemático [Kitchenham et al. 2009]. Para a organização deste processo, foi utilizada a ferramenta *Parsifal*[1].

4.1. Planejamento

O planejamento envolveu a definição dos objetivos, das questões de pesquisa bem como a seleção das bases de dados e definição dos critérios de seleção. A descrição detalhada do planejamento resultou no protocolo que foi seguido durante a pesquisa.

4.1.1. Objetivo e Questões de Pesquisa

O objetivo deste estudo é mapear pesquisas comparativas de ferramentas de análise estática de código com foco na segurança, buscando identificar.

As Questões de Pesquisa (QPs) alvo deste mapeamento foram definidas com o objetivo de concentrar, em um mesmo estudo, informações gerais importantes relacionadas as SASTs, gerando assim, uma fonte de informação útil na procura por qual ferramenta utilizar. A Tabela 1 apresenta as QPs utilizadas neste trabalho.

Tabela 1. Questões de Pesquisa.

QP1	Quais ferramentas de análise estática de código voltadas à segurança são melhores classificadas em estudos científicos com teor comparativo?
QP2	Quais critérios de classificação são utilizados para a comparação de SASTs?
QP3	Quais as linguagens de programação mais presentes em estudos que comparam ferramentas SASTs?

[1]https://parsif.al

4.1.2. PICO

A presente pesquisa secundária foi estruturada utilizando o *framework* PICOC. Esse método define:

- ***Population (P):*** Ferramentas de análise estática de código.
- ***Intervention (I):*** Segurança.
- ***Comparison (C):*** Comparação entre diferentes SASTs.
- ***Outcome (O):*** Ferramentas, métricas, benchmarking e comparativos.
- ***Context (C):*** Estudos secundários com teor comparativo.

4.1.3. String de Busca

A *String* de Busca foi definida com base em palavras-chave comumente utilizadas em estudos relacionados a SASTs conforme ilustrada na Tabela 2. A estrutura da *String* é montada utilizando operadores de conjunção (*AND*), para unificar termos com mesmo próposito e operadores de disjunção (*OR*), para tornar possível a identificação de termos similares.

Tabela 2. String de Busca.

"static application security testing"OR (("static analysis tool"OR "code static analyser"OR "code static analyzer"OR "modern code review"OR "static test") AND (security OR secure)) AND ("tools comparasion"OR "tool comparison"OR "tool comparative"OR "tools comparative"OR "tools benchmark"OR "tools benchmarking"OR "tools evaluation"OR "tools rank"OR "tools ranking")

4.1.4. Critérios de Seleção

Para a seleção dos estudos foram definidos, ao todo, dois critérios de inclusão e cinco critérios de exclusão conforme apresentado na Tabela 3.

Tabela 3. Critérios de inclusão e exclusão utilizados no estudo.

ID	**Critérios de Inclusão (CI)**
CI1	Estudo em que há um comparativo entre SASTs com foco em segurança
CI2	Estudos em inglês ou português
	Critérios de Exclusão (CE)
CE1	Análise fora do escopo de código
CE2	Estudo fora do escopo de segurança
CE3	Impossibilidade de acesso ao artigo
CE4	O estudo é um resumo de um artigo completo
CE5	O estudo é um índice de conferência, tutorial ou *keynotes*

4.1.5. Base de Dados

Para a condução da pesquisa, foram selecionadas três bases de dados populares na área da computação, por motivos como confiabilidade e boa indexação de conferências. Essas foram: *ACM Digital Library* [2], *IEEE Xplore*[3] e *Scopus*[4], onde, em cada uma, a *String* de Busca foi submetida.

4.1.6. Formulário de Extração

O formulário de extração, descrito na Tabela 4, foi definido buscando fomentar informações suficientes para respondar as questões de pesquisa investigadas neste trabalho. Para evitar a perda de informação relevante para os resultados do presente mapeamento devido algum aspecto único de cada estudo que não se enquadrasse nos itens 1 a 8 do formulário, foi adicionado um campo de observação, para informações extras.

Tabela 4. Itens do Formulário de Extração.

	Descrição
1.	Resumo do artigo
2.	Qual(is) o(s) autor(es)?
3.	Ano e local de publicação
4.	Em qual contexto o estudo foi realizado? (e.g., aplicações web, aplicações móveis)
5.	Quais as linguagens de programação dos códigos avaliados?
6.	Quais ferramentas são comparadas?
7.	Quais as métricas de classificação das ferramentas? (e.g., precisão, recall)
8.	Quais ferramentas se destacaram para as métricas estabelecidas no estudo?
9.	Observação

4.2. Condução

Para econduzir o mapeamento, foram selecionados termos populares na área de análise estática de código e no segmento de segurança de *software*, que foram utilizados para a criação da *string* de busca (ver subseção 4.1.3), a qual foi submetida nas bases selecionadas (ver subseção 4.1.5). O processo de seleção dos estudos está descrito na Figura 1, com as respectivas quantidade de artigos removidos e selecionados em cada etapa.

Os estudos passaram por três etapas de refinamento: a primeira etapa foi a seleção por meio da *string* de busca; a segunda foi a leitura dos resumos dos artigos; e a terceira e a última etapa foram realizadas a leitura completa dos estudos. Além dos artigos excluídos por não estarem nos critérios de inclusão ou se enquadrarem nos critérios de exclusão, 10 foram retirados da catalogação por estarem duplicados. Após a etapa de seleção, os trabalhos selecionados foram lidos por completo e seus dados foram adicionados ao formulário

[2]https://dl.acm.org/about
[3]https://ieeexplore.ieee.org
[4]https://www.scopus.com

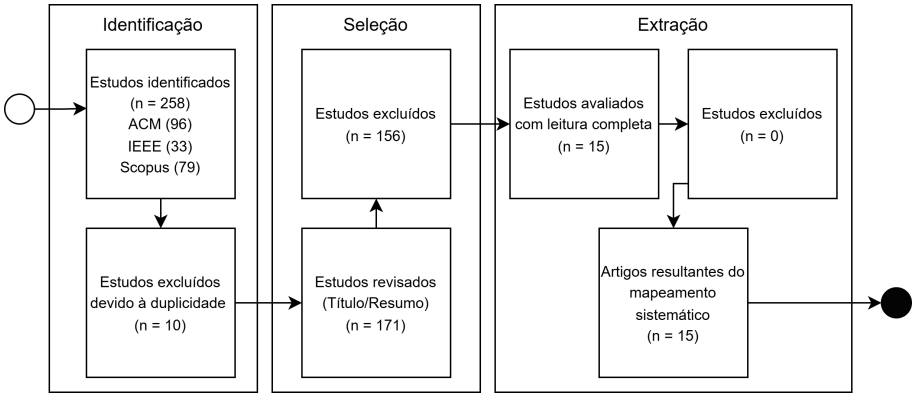

Figura 1. Diagrama de fluxo PRISMA adaptado de [Moher et al. 2009].

de extração. A leitura foi realizada de maneira cuidadosa, focando em captar nos textos, em sua totalidade, as SASTs, as linguagens de programação e as métricas de avaliação abordadas nas publicações.

5. Resultados

Analisando a extração dos 15 artigos selecionados (Tabela 5), 64 ferramentas para análise estática de segurança, sendo as 18 mais citadas e bem avaliadas apresentadas na Tabela 6 com suas respectivas quantidade de citações, referências e usos recomendados, seis linguagens de programação e 17 métricas de avaliação distintas. Nos estudos analisados, diversas ferramentas de análise estática se destacaram em diferentes contextos e métricas, com resultados variados dependendo da linguagem de programação, tipo de aplicação e métricas utilizadas na comparação. A seguir, estão sendo respondidas as 3 questões de pesquisa elaboradas, de forma que os resultados avaliados estejam sintetizados.

Tabela 5. Artigos selecionados.

Artigo	**Referência**
Evaluating C/C++ Vulnerability Detectability of Query-Based Static Application Security Testing Tools	[Li et al. 2024]
A Comprehensive Study on Static Application Security Testing (SAST) Tools for Android	[Zhu et al. 2024]
Evaluation of static analysis tools for software security	[AlBreiki and Mahmoud 2014]
An Empirical Study of Static Analysis Tools for Secure Code Review	[Charoenwet et al. 2024b]
Detecting Exception Handling Bugs in C++ Programs	[Zhang et al. 2023]
Comparison and Evaluation on Static Application Security Testing (SAST) Tools for Java	[Li et al. 2023]
Test-driving static analysis tools in search of C code vulnerabilities	[Chatzieleftheriou and Katsaros 2011]
Study of JavaScript Static Analysis Tools for Vulnerability Detection in Node.js Packages	[Brito et al. 2023]
New Tricks to Old Codes: Can AI Chatbots Replace Static Code Analysis Tools?	[Ozturk et al. 2023]
An Empirical Study of Rule-Based and Learning-Based Approaches for Static Application Security Testing	[Croft et al. 2021]
Semgrep*: Improving the Limited Performance of Static Application Security Testing (SAST) Tools	[Bennett et al. 2024]
An Extensive Comparison of Static Application Security Testing Tools	[Esposito et al. 2024]
Evaluation of Static Web Vulnerability Analysis Tools	[Tyagi and Kumar 2018]
Discovering software vulnerabilities using data-flow analysis and machine learning	[Kronjee et al. 2018]
Static analysis of source code security: Assessment of tools against SAMATE tests	[Díaz and Bermejo 2013]

Tabela 6. Ferramentas SAST, citações, sites oficiais e contextos de uso recomendados.

Ferramenta	Citações	Site oficial	Sumário de Contextos Recomendados
CodeQL	5	https://codeql.github.com/	Análise semântica em CI/CD para C/C++, Java, C, Python e JavaScript.
InsiderSec	4	https://github.com/insidersec	Análise de vulnerabilidades em apps web e mobile (Java, C, Swift, Kotlin).
Contrast	4	https://www.contrastsecurity.com/	Plataforma SAST+IAST para DevOps ágil com foco em Java e .NET.
Horusec	3	https://horusec.io/	Orquestrador open-source para centralizar análises de segurança multi-linguagem.
SpotBugs	3	https://spotbugs.github.io/	Encontra bugs e falhas de segurança em bytecode de projetos Java.
Semgrep	3	https://semgrep.dev/	Análise rápida com regras customizáveis para Go, Java, Python, Ruby, etc.
SonarQube	3	https://www.sonarsource.com/products/sonarqube/	Plataforma completa para inspeção contínua de qualidade e segurança do código.
NodeJsScan	2	https://github.com/ajinabraham/nodejsscan	Ferramenta especializada em encontrar vulnerabilidades em projetos Node.js.
ODGen	2	https://github.com/Song-Li/ODGen	Detecção específica de vulnerabilidades de ”desreferência de objeto”em C/C++.
Graudit	2	https://github.com/wireghoul/graudit	Auditoria rápida baseada em regex (grep) para encontrar falhas em código.
ESLint SSC	2	https://eslint.org/	Linter com plugins de segurança para código JavaScript e TypeScript.
DevSkim	2	https://github.com/microsoft/DevSkim	Plugin para IDEs que fornece feedback de segurança em tempo real.
Mosca	2	https://sourceforge.net/projects/mosca/	Análise de bugs e buffer overflows em código C/C++ de sistemas legados.
Drek	2	https://github.com/chrisallenlane/drek	Ferramenta leve baseada em grep para auditorias rápidas de vulnerabilidade.
FindSecBugs	2	https://find-sec-bugs.github.io/	Plugin de segurança para SpotBugs focado em vulnerabilidades Java.
VCG	2	https://github.com/nccgroup/VCG	Auditoria de código com visualização para C/C++, Java, C, VB e PL/SQL.
PMD	2	https://pmd.github.io/	Analisador de bugs comuns e código ”morto”para Java, JavaScript, etc.
MobSF	1	https://github.com/MobSF	Análise estática e dinâmica de segurança para aplicações Android e iOS.
QARK	1	https://github.com/linkedin/qark/	Encontra vulnerabilidades comuns em aplicações Android (código ou APK).
SUPER	1	https://github.com/SUPERAndroidAnalyzer/super	Automatiza a busca por vulnerabilidades conhecidas em apps Android via CLI.
CppCheck	1	https://cppcheck.sourceforge.io/	Analisador de bugs e código indefinido para garantir a robustez em C/C++.

5.1. QP1: Quais ferramentas de análise estática de código voltadas à segurança são melhores classificadas em estudos científicos com teor comparativo?

Estudos comparativos revelam que nenhuma ferramenta de análise estática de segurança se destaca de forma absoluta, sendo seu desempenho fortemente influenciado pelo contexto de uso, linguagem analisada e métricas avaliadas.

O trabalho de [Zhu et al. 2024] avaliou dez ferramentas para Android, incluindo MobSF, QARK, SUPER e outras sete menos eficazes. Embora nenhuma tenha se sobressaído de forma expressiva, QARK apresentou o menor *F1-score* (42,1%) e SUPER o menor *recall* (38,8%), indicando limitações gerais em precisão e *recall*. Em [Li et al. 2024], CodeQL obteve a melhor taxa de detecção entre as ferramentas testadas. Já em [Zhang et al. 2023], CppCheck manteve alta precisão na detecção de bugs em C++. O estudo de [Chatzieleftheriou and Katsaros 2011] evidenciou a precisão de Parasoft C++ Test e Frama-C, embora ambas sofram com elevado custo de desempenho em projetos maiores. No trabalho de [Díaz and Bermejo 2013], Prevent apresentou o melhor desempenho geral, combinando alta precisão e baixa taxa de falsos positivos. SCA, por sua vez, obteve o melhor *recall*. O estudo de [Charoenwet et al. 2024a] mostrou que CodeChecker teve o maior *recall*, enquanto CodeQL manteve a melhor precisão e a menor taxa de falsos alarmes, embora apenas 7% dos alertas fossem realmente relevantes para correção de vulnerabilidades.

Em aplicações PHP, [Tyagi and Kumar 2018] apontou que OWASP WAP superou o RIPS em precisão ao testar com aplicações deliberadamente vulneráveis (DVWA e bWAPP). Já [Kronjee et al. 2018] destacou o bom desempenho do RIPS em código não

vulnerável, além de resultados positivos com Pixy.

Para Java e .NET, [AlBreiki and Mahmoud 2014] indicou que Yasca teve maior efetividade que FindBugs e Microsoft Code Analysis Tool .NET (CAT.NET), beneficiando-se da integração com múltiplas ferramentas. Infer foi apontada por [Esposito et al. 2024] como a melhor ferramenta para essa linguagem, equilibrando bem precisão e *recall*, enquanto Snyk alcançou alto *recall*, mas com muitos falsos positivos. No estudo de [Li et al. 2023], Contrast e SpotBugs atingiram *F1-score* acima de 80% no benchmark OWASP, e Horusec e SpotBugs também se destacaram no Java CVE benchmark. Por fim, [Bennett et al. 2024] ressaltou o alto TPR da ferramenta FindSecBugs.

5.2. QP2: Quais critérios de classificação são utilizados para a comparação de SASTs?

Para a comparação, os estudos utilizam como insumo métricas diretamenta relacionadas com a qualidade com que as ferramentas conseguem identificar vulnerabilidades em trechos de código ou no código fonte de um sistema como um todo. As métricas encontradas totalizam 18, sendo evidenciadas os respectivos estudos que as citam na Tabela 7. Destas, precisão, *recall* e falsos negativos aparecem nove vezes. Em seguida, temos *F1-score*, detection rate e false positives, as quais apareceram 3 vezes. *F-measure* e *true positives* foram citadas 2 vezes. As demais métricas foram citadas apenas uma vez cada.

Tabela 7. Métricas utilizadas e respectivas referências.

Métrica	**Referências nos estudos selecionados**
Recall	[Esposito et al. 2024, Charoenwet et al. 2024a, Zhu et al. 2024, Li et al. 2024, Li et al. 2023, Alqaradaghi and Kozsik 2024, Díaz and Bermejo 2013, Chatzieleftheriou and Katsaros 2011, Charoenwet et al. 2024b]
Precision	[Esposito et al. 2024, Charoenwet et al. 2024a, Zhu et al. 2024, Li et al. 2024, Li et al. 2023, Alqaradaghi and Kozsik 2024, Díaz and Bermejo 2013, Chatzieleftheriou and Katsaros 2011, Charoenwet et al. 2024b]
False Negative	[Esposito et al. 2024, Charoenwet et al. 2024a, Zhu et al. 2024, Li et al. 2024, Li et al. 2023, Alqaradaghi and Kozsik 2024, Díaz and Bermejo 2013, Chatzieleftheriou and Katsaros 2011, Charoenwet et al. 2024b]
F1-Score	[Esposito et al. 2024, Charoenwet et al. 2024a, Díaz and Bermejo 2013]
Detection Rate	[Esposito et al. 2024, Li et al. 2024, Alqaradaghi and Kozsik 2024]
False Positive	[Charoenwet et al. 2024a, Díaz and Bermejo 2013, Li et al. 2024]
F-measure	[Bennett et al. 2024, Díaz and Bermejo 2013]
True Positive	[Charoenwet et al. 2024a, Díaz and Bermejo 2013]
Métrica Personalizada	[Zhu et al. 2024]
NPofBX	[Zhu et al. 2024]
Initial False Alarm	[Charoenwet et al. 2024a]
Computation Time	[Zhu et al. 2024]
Coef. Correlação de Matthews	[Li et al. 2024]
Warnings	[Charoenwet et al. 2024a]
Especificidade	[Díaz and Bermejo 2013]
Acurácia	[Díaz and Bermejo 2013]
Covered	[Li et al. 2023]
Cobertura OWASP Top 10	[Ozturk et al. 2023]

5.3. QP3: Quais as linguagens de programação mais presentes em estudos que comparam SASTs?

Com base na Figura 2, é possível observar que as linguagens mais citadas foram C e C++, cada uma com sete citações dentro dos 15 artigos. Em segundo lugar, aparece a linguagem Java, a qual foi citada cinco vezes. A linguagem PHP foi citada três vezes, ficando em terceiro lugar e, por fim, Javascript e .NET, as quais apareceram em apenas um estudo.

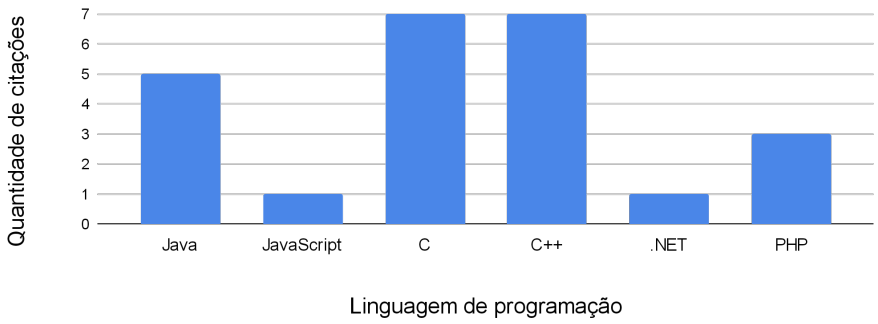

Figura 2. Quantidade de citações de cada linguagem de programação.

6. Conclusão e Trabalhos Futuros

Nesse estudo, foi realizado um mapeamento de trabalhos que comparam ferramentas SASTs, de forma a possibilitar uma escolha melhor embasada em diferentes situações em que deseja-se fazer análise estática com foco na segurança. Em síntese, percebeu-se que não existe unanimidade quanto à superioridade de determinada ferramenta em relação às demais. Entretanto, existem ferramentas com melhor desempenho em contextos específicos.

Os estudos analisados destacam diversas lacunas e oportunidades para pesquisas futuras no campo da análise estática de código e detecção de vulnerabilidades. Trabalhos futuros podem explorar técnicas para reduzir os falsos positivos, como a combinação de múltiplas ferramentas SAST ou a integração de abordagens baseadas em modelos de aprendizado de máquina. Esses modelos, que já se mostraram promissores em estudos como o de Croft et al. [Croft et al. 2021], podem oferecer uma precisão superior às ferramentas tradicionais, especialmente em cenários onde os padrões de vulnerabilidades não são previamente conhecidos. Além disso, a criação de benchmarks mais abrangentes e representativos, como proposto por Zhu et al. [Zhu et al. 2024], permitiria uma análise mais precisa da eficácia dos Analisadores Estáticos para segurança.

Agradecimentos

Ao CNPq pela bolsa de produtividade da Rossana M. C. Andrade 1D (306362/2021-0), a Funcap/SEPLAG pelo apoio financeiro na execução do projeto ”Governança da Informação na SEPLAG”, que custeia as bolsas de Inovação Tecnológica dos autores, e a Universidade Federal do Ceará/PREX, pelo fomento e apoio do projeto de extensão ”Fábrica de Testes de Segurança”.

Referências

AlBreiki, H. H. and Mahmoud, Q. H. (2014). Evaluation of static analysis tools for software security. pages 93–98.

Alqaradaghi, M. and Kozsik, T. (2024). Comprehensive evaluation of static analysis tools for their performance in finding vulnerabilities in java code. *IEEE Access*, 12:55824–55842.

Bennett, G., Hall, T., Winter, E., and Counsell, S. (2024). Semgrep*: Improving the limited performance of static application security testing (sast) tools. page 614–623.

Bhutani, V., Toosi, F. G., and Buckley, J. (2024). Analysing the analysers: An investigation of source code analysis tools. *Applied Computer Systems*, 29(1):98–111.

Brito, T., Ferreira, M., Monteiro, M., Lopes, P., Barros, M., Santos, J. F., and Santos, N. (2023). Study of javascript static analysis tools for vulnerability detection in node.js packages. *IEEE Transactions on Reliability*, 72(4):1324–1339.

Charoenwet, W., Thongtanunam, P., Pham, V.-T., and Treude, C. (2024a). An empirical study of static analysis tools for secure code review. page 691–703.

Charoenwet, W., Thongtanunam, P., Pham, V.-T., and Treude, C. (2024b). An empirical study of static analysis tools for secure code review.

Chatzieleftheriou, G. and Katsaros, P. (2011). Test-driving static analysis tools in search of c code vulnerabilities. pages 96–103.

Chess, B. and McGraw, G. (2004). Static analysis for security. *IEEE Security Privacy*, 2(6):76–79.

Croft, R., Newlands, D., Chen, Z., and Babar, M. A. (2021). An empirical study of rule-based and learning-based approaches for static application security testing. In *Proceedings of the 15th ACM / IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*, ESEM '21, New York, NY, USA. Association for Computing Machinery.

D, V. L. L. (2024). Análise estática. Disciplina Verificação, Validação e Teste de Software, Universidade Federal do Ceará.

de Mendonça, V. R. L., Rodrigues, C. L., de MN Soares, F. A. A., and Vincenzi, A. M. R. (2013). Static analysis techniques and tools: A systematic mapping study. *ICSEA*.

Díaz, G. and Bermejo, J. R. (2013). Static analysis of source code security: Assessment of tools against samate tests. *Information and Software Technology*, 55(8):1462–1476.

Esposito, M., Falaschi, V., and Falessi, D. (2024). An extensive comparison of static application security testing tools. page 69–78.

et. al., K. (2004). Vulnerability detection in software applications using static code analysis. *Journal of Theoretical and Applied Information TechnologyVolume 102, Issue 4, Pages 1307*, 102(4):1307.

ISO/IEC/IEEE 29119-1 (2022). ISO/IEC/IEEE 29119-1:2022 - Software and Systems Engineering — Software Testing — Part 1: Concepts and Definitions. Standard published by ISO/IEC/IEEE. Available at: `https://doi.org/10.1109/IEEESTD.2022.9698145`.

Kitchenham, B., Pearl Brereton, O., Budgen, D., Turner, M., Bailey, J., and Linkman, S. (2009). Systematic literature reviews in software engineering – a systematic literature review. *Information and Software Technology*, 51(1):7–15. Special Section - Most Cited Articles in 2002 and Regular Research Papers.

Kronjee, J., Hommersom, A., and Vranken, H. (2018). Discovering software vulnerabilities using data-flow analysis and machine learning.

Li, K., Chen, S., Fan, L., Feng, R., Liu, H., Liu, C., Liu, Y., and Chen, Y. (2023). Comparison and evaluation on static application security testing (sast) tools for java. page 921–933.

Li, Z., Liu, Z., Wong, W. K., Ma, P., and Wang, S. (2024). Evaluating c/c++ vulnerability detectability of query-based static application security testing tools. *IEEE Trans. Dependable Secur. Comput.*, 21(5):4600–4618.

Moher, D., Liberati, A., Tetzlaff, J., and Altman, D. G. (2009). Preferred reporting items for systematic reviews and meta-analyses: the prisma statement. *Bmj*, 339.

Ozturk, O. S., Ekmekcioglu, E., Cetin, O., Arief, B., and Hernandez-Castro, J. (2023). New tricks to old codes: Can ai chatbots replace static code analysis tools? page 13–18.

Pashchenko, I., Dashevskyi, S., and Massacci, F. (2017). Delta-bench: Differential benchmark for static analysis security testing tools. In *2017 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*, pages 163–168.

SOMMERVILLE, I. (2011). Engenharia de software. tradução ivan bosnic e kalinka g. de o.

Stefanović, D., Nikolić, D., Dakić, D., Spasojević, I., and Ristić, S. (2020). Static code analysis tools: A systematic literature review. In *Ann. DAAAM Proc. Int. DAAAM Symp*, volume 31, pages 565–573.

Tyagi, S. and Kumar, K. (2018). Evaluation of static web vulnerability analysis tools. pages 1–6.

Zhang, H., Luo, J., Hu, M., Yan, J., Zhang, J., and Qiu, Z. (2023). Detecting exception handling bugs in c++ programs. page 1084–1096.

Zhu, J., Li, K., Chen, S., Fan, L., Wang, J., and Xie, X. (2024). A comprehensive study on static application security testing (sast) tools for android. *IEEE Trans. Softw. Eng.*, 50(12):3385–3402.