

Automated Testing Tools and Software Reliability: A Systematic Mapping

Artur S. Farias
0009-0005-5576-124X

Computing Institute Federal University of Alagoas
in Maceio, Brazil
Email: asf2@ic.ufal.br

Rodrigo Rocha
0000-0003-1993-5044

Computing Institute Federal University of Alagoas
in Maceio, Brazil
Email: rodrigo.rocha@ufape.edu.br

Jamilson Dantas
0000-0001-9009-7659
Federal University of Pernambuco
in Recife, Brazil
Email: jrd@cin.ufpe.br

Abstract—Modern software systems operate in increasingly complex and heterogeneous environments, intensifying challenges related to reliability. Automated software testing (AST) helps address these demands, yet existing evidence on how widely used tools support reliability-oriented attributes remains dispersed. A consolidated view is needed to understand their contributions to coverage, fault detection, robustness, and execution reliability. This study conducts a systematic mapping of 184 primary works to identify the most frequently cited AST tools and to examine empirical evidence of their reliability-related capabilities. The analysis synthesizes findings on structural coverage, fault revelation, robustness under stress, and cross-platform consistency, revealing complementary strengths and gaps in current tool support. The results offer a structured perspective on how automation contributes to reliability evaluation, supporting software practitioners in selecting tools aligned with reliability goals and assisting researchers in identifying opportunities for advancing reliability-aware testing. In addition to the empirical synthesis, this study produces an evidence-based artifact in the form of a reliability-oriented attribute matrix, explicitly linking automated testing tools to traceable primary-study evidence and supporting informed, practical tool selection.

Index Terms—Automated Software Testing, Software Reliability, Fault Detection, Robustness Testing, Systematic Mapping Study, Test Automation Tools.

I. INTRODUCTION

MODERN software systems increasingly operate in heterogeneous execution environments and are developed under rapid release cycles, particularly in mobile and multi-platform contexts, which intensify challenges related to reliability and testability [1]. In such settings, failures may emerge from complex interactions, environmental variability, and tight operational constraints, making reliability assurance a continuous concern throughout the software development lifecycle. Automated software testing (AST) supports these demands through systematic behavior exploration, repeatable execution, and early fault detection [2]. Empirical studies indicate that the diversity of testing tools directly influences how organizations manage scalability, integration, and continuous delivery [3], [4], [5].

The ISO/IEC/IEEE 29119-1:2022 standard positions software testing as a disciplined evaluation activity and highlights the role of automation in improving coverage, effectiveness, and effort efficiency, particularly in agile and DevOps-oriented settings [6]. Despite these advances, empirical evidence consistently reports limitations related to tool adequacy, maintainability, and fault detection effectiveness, especially in distributed, interactive, or resource-constrained systems [7], [8], [9], [10]. These limitations indicate that increased automation alone does not guarantee reliable outcomes and that the contribution of testing tools must be interpreted more carefully.

Recent research emphasizes the need to evaluate software reliability under non-nominal conditions, where subtle failures and robustness issues frequently surface only under perturbation, stress, or fault-injection scenarios [11], [12], [13], [14]. Complementary approaches propose resilience-oriented evaluation techniques to expose vulnerabilities that remain undetected during nominal execution [13], [14]. Collectively, these findings highlight four reliability-oriented attributes as essential lenses for interpreting automated testing evidence: *coverage*, *fault detection*, *robustness*, and *execution reliability*.

Although prior studies have advanced the understanding of automated testing in specific domains, the available evidence remains fragmented across works that examine isolated aspects such as coverage and fault detection [15], [2], industrial challenges [16], [8], [10], domain-specific constraints [17], [18], [19], [20], [21], and robustness or fault-injection evaluation [11], [13], [14]. These contributions rarely converge toward a consolidated view of how widely used automated testing tools relate to reliability-oriented attributes in a systematic and cross-contextual manner.

To address this gap, this study conducts a systematic mapping of 184 primary works on automated software testing, synthesizing empirical evidence on tool usage and its relation to coverage, fault detection, robustness, and execution reliability. The results aim to support practitioners by clarifying how tool categories align with reliability objectives and to assist

researchers by revealing recurring patterns, methodological limitations, and opportunities for advancing reliability-aware automation strategies.

The remainder of this paper is organized as follows. Section II presents background concepts. Section III reviews related work. Section IV describes the mapping methodology. Section V reports and analyzes the results, and Section VI concludes the paper with final remarks and future directions.

II. BACKGROUND

Software reliability is a long-established quality attribute in software engineering, commonly associated with the ability of a system to perform its required functions under specified conditions for a defined period of time [6]. In contemporary software-intensive systems, reliability is increasingly shaped by distributed architectures, heterogeneous execution environments, and accelerated release cycles [1]. These characteristics amplify the likelihood of subtle failures, interaction faults, and environment-dependent behaviors, making reliability assurance a continuous concern throughout the development lifecycle rather than a post-deployment activity.

Automated software testing (AST) plays a central role in supporting reliability by enabling systematic behavior exploration, repeatable execution, and early fault detection [2]. The ISO/IEC/IEEE 29119-1:2022 standard positions testing as a disciplined evaluation activity and highlights the role of automation in improving coverage, effectiveness, and scalability, particularly in agile and DevOps-oriented contexts [6], [22], [23], [24]. Nevertheless, empirical studies consistently report limitations related to tool adequacy, maintainability, and fault detection effectiveness, especially in distributed, interactive, or resource-constrained systems [7], [8], [9], [10].

Recent research emphasizes that reliability-related failures frequently emerge under non-nominal conditions, such as environmental perturbations, partial faults, or unexpected interactions [11], [12]. Robustness issues and soft errors may remain latent during nominal execution and surface only when systems are subjected to stress, fault injection, or degraded operational contexts [12]. To address these challenges, complementary techniques such as resilience testing and fault-injection-based evaluation have been proposed to reveal weaknesses that are not detected through conventional testing strategies [13], [14]. These findings reinforce that reliability cannot be inferred from a single metric or execution outcome, but rather from a combination of attributes that capture different dimensions of system behavior.

Across the literature, four reliability-oriented attributes recurrently emerge as essential lenses for interpreting automated testing evidence: *coverage*, which reflects the extent to which system behaviors are exercised; *fault detection*, which captures the ability to expose defects; *robustness*, referring to the stability of test execution under variation and perturbation; and *execution reliability*, which concerns the consistency and predictability of test outcomes across repeated runs [14], [11]. Empirical investigations show that these attributes are interrelated but not interchangeable, and that improvements

in one dimension do not necessarily imply gains in others. Tool-centered empirical studies further indicate substantial variability in how automated testing tools support these attributes, revealing differences in detection capability, execution stability, and practical applicability across contexts [25].

The increasing adoption of continuous integration (CI) and DevOps practices further influences how reliability-oriented testing is operationalized in practice. CI environments promote frequent integration and automated quality checks, embedding testing activities directly into development workflows [26]. While this integration enables early feedback, it also introduces practical constraints related to execution time, resource availability, and pipeline stability, which may affect the feasibility and effectiveness of automated testing tools [7]. Recent empirical evidence highlights that tool behavior and testing outcomes in CI settings are strongly influenced by configuration choices, project characteristics, and execution environments, reinforcing the need for context-aware reliability assessment [27], [28], [29].

Taken together, these foundations indicate that understanding the role of automated testing in supporting software reliability requires a consolidated perspective that accounts for multiple reliability-oriented attributes and diverse empirical contexts. Rather than focusing on isolated metrics, prediction models, or individual execution scenarios, a broader synthesis of empirical evidence is necessary to clarify how automated testing tools contribute to coverage, fault detection, robustness, and execution reliability across heterogeneous settings. This perspective provides the conceptual grounding for the systematic mapping study presented in the following sections.

III. RELATED WORK

Research on software reliability and automated testing encompasses empirical, theoretical, and tool-centric perspectives, reflecting the multifaceted nature of reliability as a software quality attribute. Prior work has explored reliability from predictive, structural, and operational viewpoints, often focusing on models, metrics, or process-level interventions rather than on the concrete behavior of testing tools in practice.

Oveisi et al. [30] provide a comprehensive survey on software reliability prediction, synthesizing models based on fault data, operational profiles, and coverage-derived indicators. Their survey highlights the diversity of reliability modeling approaches and reinforces the relevance of reliability-oriented attributes such as fault density, failure rates, and coverage. However, these models abstract away from the concrete mechanisms through which automated testing tools contribute to reliability in real development environments, offering limited insight into tool behavior across heterogeneous empirical contexts.

The relationship between testing adequacy and reliability has been extensively investigated through coverage-based studies. Bach et al. [31] empirically examined the association between structural coverage and bug density in a large industrial system, providing evidence that covered code tends to exhibit fewer future defects. These findings establish coverage as a

meaningful adequacy indicator, yet the analysis is restricted to a single system and does not account for the variability introduced by different automated testing tools or execution contexts.

Other studies approach reliability from a structural or predictive perspective. Diwaker et al. [32], for instance, propose a soft computing model for predicting the reliability of component-based systems by combining factors such as reusability, complexity, component interaction, dependency, and failure rate using fuzzy logic and particle swarm optimization. While such approaches provide insight into high-level drivers of reliability, they do not analyze empirical evidence on automated testing tools or their contribution to reliability-oriented attributes such as coverage, fault detection, robustness, or execution stability.

The adoption of DevOps and continuous integration practices has introduced new perspectives on reliability assessment. Pietrantuono et al. [33] propose DevOpRET, an approach for continuous reliability testing in DevOps pipelines that leverages monitored operational profiles to estimate failure-on-demand and support reliability-driven release decisions. Although relevant to operational reliability assessment, this approach focuses on runtime behavior and monitored executions, rather than on how automated testing tools contribute to reliability attributes during development and testing activities.

Complementary work has examined the reliability of the testing infrastructure itself. Yang et al. [34] employ randomized differential testing to detect faults in code coverage tools, revealing inconsistencies and correctness issues in widely used coverage analyzers. This line of work highlights that testing tools are not neutral instruments and may introduce reliability risks, but it does not address how different tools support reliability attributes across diverse testing scenarios.

More closely related to the present study, empirical investigations have analyzed automated testing tools from a tool-centric and evidence-oriented perspective. Studies published in empirical software engineering venues have compared testing tools and practices across multiple contexts, examining aspects such as coverage support, fault detection capability, and practical applicability in real projects [28], [29]. These studies highlight substantial variability among tools and emphasize the importance of empirical evaluation for informed tool selection. However, they typically focus on specific tool sets or evaluation criteria and do not provide a consolidated mapping of reliability-oriented attributes across a broad body of empirical evidence, nor do they systematically synthesize how automated testing tools contribute to coverage, fault detection, robustness, and execution reliability.

Compared to these works, the present study contributes a tool-centered and evidence-oriented perspective by systematically mapping 184 primary studies on automated software testing. Rather than focusing on reliability prediction models, isolated adequacy metrics, or operational pipelines, this mapping synthesizes how commonly used testing tools support four reliability-oriented attributes—coverage, fault detection, robustness, and execution reliability—across a wide variety

of empirical settings. This consolidated view complements existing research by clarifying the strengths, limitations, and recurring patterns associated with automated testing tools. An overview of the evidence identified in the related literature is summarized in Table I.

TABLE I
FINDING EVIDENCE IN RELATED WORKS

	Approaches	Benefits	Best practices	Challenges	Phases	Selection criteria	Tools	Types
This Research	■	■	□	■	□	□	■	■
Oveisi et al. [30]	■	■	□	■	□	□	□	■
Bach et al. [31]	■	■	□	■	□	□	□	■
Diwaker et al. [32]	■	■	□	■	□	□	□	■
Pietrantuono et al. [33]	■	■	□	■	■	□	□	■
Yang et al. [34]	■	■	□	■	□	□	■	■
Jurvanen et al. [28]	■	■	□	■	□	■	■	□
Bertolino et al. [29]	■	■	□	■	□	■	■	□

In contrast to prior empirical comparisons that evaluate specific tools or contexts, this study provides a longitudinal and cross-contextual synthesis of reliability-oriented evidence, positioning automated testing tools within a unified reliability framework. This work contributes (i) a reliability-oriented classification of AST evidence, (ii) an empirical synthesis of tool behavior across four reliability attributes, and (iii) an evidence-based perspective to support tool selection and research planning.

IV. SYSTEMATIC MAPPING

Systematic mapping studies (SMS) provide a structured approach for organizing existing evidence in a research domain, enabling the identification of trends, concentration areas, and gaps requiring further investigation. Following the guidelines of Kitchenham et al. [35], this study adopts an SMS to consolidate evidence on automated software testing (AST), with emphasis on the role of automation tools in supporting robustness, coverage, fault detection, and reliability. A single research question guides this investigation: *What evidence exists in the literature regarding how automated testing tools support robustness, coverage, fault detection, and reliability?*

The methodological protocol applied in this study is derived from a previously conducted broad SMS on software testing [36] shown in Figure 1. That protocol defined the search strategies, screening criteria, and data extraction procedures, and remains publicly available at <https://11nk.dev/9WIsH>. The earlier SMS retrieved 341 primary studies from five major scientific repositories (IEEEExplore, ACM Digital Library, ScienceDirect, SpringerLink, and Scopus). These studies formed the initial evidence base for the refinement carried out in

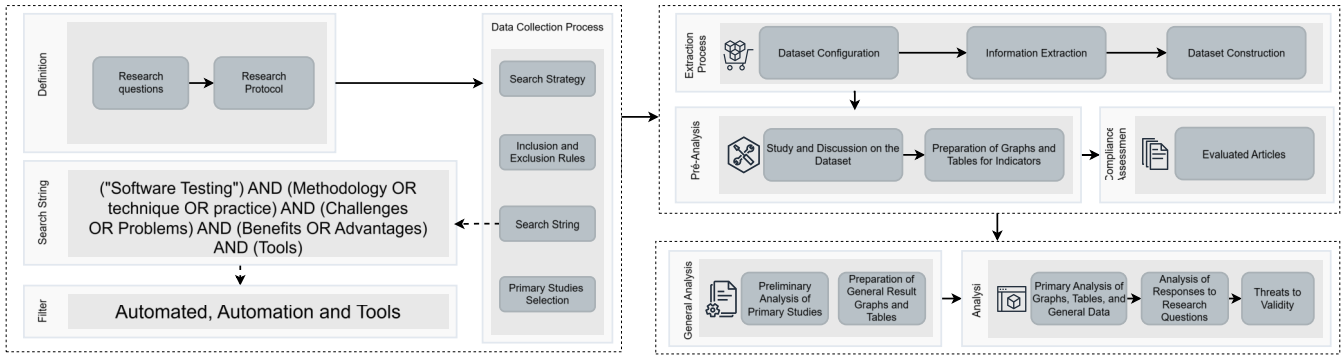


Fig. 1. Flow of activities in the adopted methodology.

the present mapping, leading to a detailed, evidence-based classification used to evaluate tool support for reliability-oriented attributes.

From this corpus, a focused screening process was performed to identify only studies directly addressing automated software testing. Titles, abstracts, and keywords were examined, together with the evidence previously collected, to determine whether automation played a substantial role in each study. Studies that did not describe automated procedures, tools, or automation-supported testing evidence were excluded. This filtering resulted in a final dataset of 184 primary studies specifically related to AST. The overall selection workflow is shown in Figure 2.

For each of the 184 included studies, a structured data extraction procedure was applied to capture information on the automated testing tools mentioned or evaluated, the types of evidence associated with robustness, coverage, fault detection, and reliability, and the methodological characteristics of each work. To ensure consistency and validity, a quality and compliance assessment was performed using five criteria adapted from the original protocol, with each study receiving a score of 1.0 (fully meets), 0.5 (partially meets), or 0.0 (does not meet). 59.78% of the studies received the maximum score (5), 6.52% scored 4.5, 21.19% scored 4, and 5.43% scored 3.5; The sum of the lowest values is 7.06%.

A. Classification Criteria for Reliability Attributes

The definition of classification criteria is a critical step in systematic mapping studies, as it determines how heteroge-

neous empirical evidence is interpreted and synthesized. In this work, the selection of reliability-oriented attributes was guided by both normative definitions of software reliability and empirical findings reported in the literature on robustness and fault manifestation under non-nominal conditions.

Software reliability is commonly defined as the ability of a system to perform its required functions under specified conditions for a defined period of time [6]. However, empirical studies consistently demonstrate that reliability-related failures often emerge from incomplete behavioral exploration, ineffective fault revelation, sensitivity to perturbations, and instability across executions or environments [14], [11]. From this perspective, reliability cannot be adequately characterized by a single metric or testing outcome, but rather by a combination of complementary attributes that reflect how systems behave under diverse operational and testing conditions.

Based on this rationale, four attributes were selected as analytical lenses in this study. *Coverage* captures the extent to which system behavior is exercised and remains a foundational adequacy indicator in automated testing research. *Fault detection* reflects the ability of testing strategies to expose defects, crashes, or unexpected behaviors beyond nominal execution paths. *Robustness* addresses system and tool behavior under stress, randomness, malformed inputs, or environmental perturbations, which are frequently associated with latent or interaction-level faults. Finally, *reliability*—interpreted here as execution reliability—captures the combined effect of stability, repeatability, and consistency of test outcomes across runs,

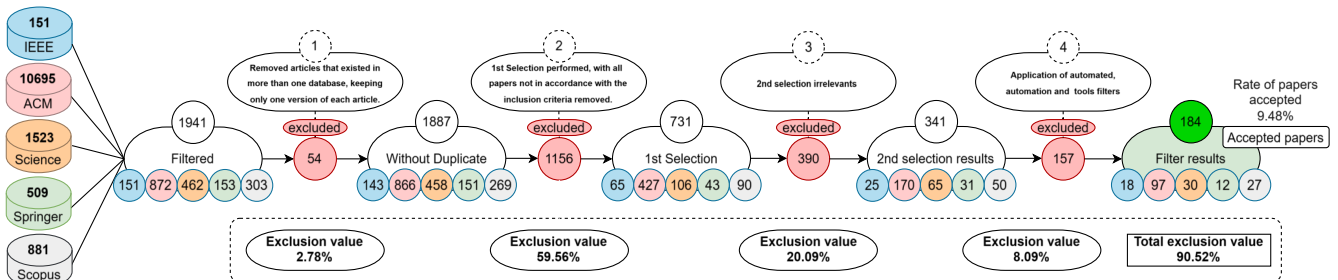


Fig. 2. Study selection process from the initial 341 studies to the final 184 automated testing works.

platforms, and configurations.

Other quality attributes frequently discussed in software engineering, such as performance, usability, or maintainability, were intentionally excluded from the classification framework. While relevant to overall software quality, these attributes are either orthogonal to reliability-oriented testing evidence or require different forms of empirical assessment not consistently reported in the analyzed studies. Focusing on a concise set of reliability-oriented attributes enables a more consistent synthesis of evidence across heterogeneous tools, systems, and experimental designs, while remaining aligned with established definitions and prior empirical work.

To support the analysis in Section VI, each tool was classified according to the degree of evidence supporting the four selected reliability-oriented attributes: coverage, fault detection, robustness, and execution reliability. The classification was based on explicit empirical findings described in the 184 primary studies, following a three-step procedure.

Attribute operationalization. Each attribute was defined as follows: *Coverage* refers to structural or behavioral exploration reported in terms of branch, line, path, or method coverage metrics. *Fault detection* refers to observed bug revelation capabilities, including mutation score improvements, crash detection, or discovery of unexpected failures. *Robustness* refers to tool support for stress, random, or fuzz-based execution capable of exposing platform- or input-sensitive fragilities. *Execution reliability* refers to the combined effect of coverage, stability, repeatability, and cross-platform consistency reported by the studies.

Scoring scheme. Each tool was assigned a score of ● (low support), ●● (baseline support), or ●●● (strong, empirically demonstrated support). A low ● was assigned when the amount of concrete evidence of contribution was equal to or less than half of the primary studies presenting evidence for the attribute. For each tool, evidence counts were normalized by the number of studies mentioning that tool. A baseline ●● required that the amount of evidence be greater than half of the primary studies reporting concrete contributions. A strong ●●● required multiple independent studies or consistent evidence across different contexts, experimental conditions, or evaluation metrics.

Reviewer agreement. The classification was conducted manually and guided by the predefined attribute criteria. Because the decisions were grounded in explicit empirical evidence reported in the primary studies, the coding process was generally straightforward. Although no formal inter-rater reliability metric was applied, the classifications were reviewed and refined iteratively to mitigate subjectivity. These procedures provide transparency for the assignment of attribute levels in Table V and support the interpretation presented in Section VI.

V. FINDINGS FROM THE SYSTEMATIC MAPPING

The systematic mapping resulted in 184 primary studies focused on automated software testing (AST). These works span the last eleven years (2014–2024), illustrating the sustained

interest of the research community in automation-related evidence. Figure 3 presents the annual distribution of included studies, showing a relatively stable publication volume with a notable increase in 2024, which recorded the highest number of contributions. This pattern reflects the growing relevance of AST in contemporary software engineering practices.

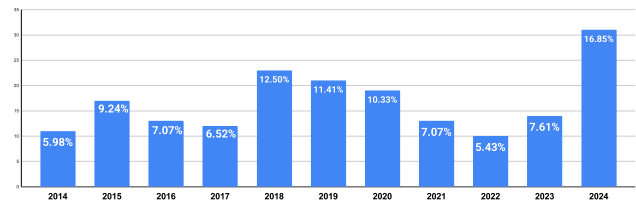


Fig. 3. Distribution of selected studies by publication year.

Across all selected papers, 575 distinct authors contributed to the body of evidence, indicating a broad and diverse research community. In the Table II Andrea Arcuri, Vahid Garousi, Zhenyu Chen, and Chunrong Fang appear as the most prolific contributors. Regarding the indexing sources of the selected studies, ACM and IEEE dominate the dataset, followed by ScienceDirect, SpringerLink, and a small set of additional repositories. This distribution reflects the centrality of major software engineering events in disseminating research related to automated software testing.

TABLE II
MAIN AUTHORS AMONG THE SELECTED PRIMARY STUDIES.

Author	Papers	%	Author	Papers	%
Andrea Arcuri	8	1.20	Vahid Garousi	7	1.05
Zhenyu Chen	5	0.75	Chunrong Fang	5	0.75
Gordon Fraser	4	0.60	Phil McMinn	4	0.60
Wasif Afzal	4	0.60	Emil Alégroth	4	0.60
Phil McMinn	4	0.60			

The 184 works were published across 84 distinct venues, including journals, conferences, and symposiums, the main ones can be seen in Table III. The journals *Information and Software Technology* and the *Journal of Systems and Software* accounted for the highest number of studies, followed by *ACM TOSEM*, *ISSTA*, *ICSE*, *ICST*, and *SBES*.

TABLE III
JOURNALS OR CONFERENCES OF THE SELECTED STUDIES.

Conference	N	%	Conference	N	%
Information and Software Technology	17	9.24	Journal of Systems and Software	15	8.15
TOSEM	12	6.52	ISSTA	8	4.35
ICSE	8	4.35	ICST	7	3.80
SBES	6	3.26			

This diversity indicates that AST research permeates both empirical software engineering and specialized software testing forums. As shown in Figure 4, the United States contributed the largest number of studies, followed by the Netherlands and Brazil. The dataset overall reflects a strong international distribution, signaling the global interest in advancing automated testing capabilities. To contextualize geographic distribution, the Scimago Journal & Country Rank

TABLE IV
INFORMATION COLLECTED FROM TESTING TOOLS.

Tool	Language	Platform	Testing
junit	java	desktop, backend/api, web, mobile	unit
evosuite	java	desktop, backend/api, web, mobile	unit
selenium	all (script)	web, mobile	functional, interface
randoop	java, .net	desktop, backend/api, web, mobile	unit
monkey	java, kotlin, dart, javascript	mobile	random, unit
appium	java, kotlin, dart, javascript	web, mobile	functional, non-functional
evomaster	java, kotlin, c#, javascript, python	desktop, web	system, regression, fuzzing
sikuli	java	desktop, web, mobile	web, interface

of the most cited tools are open source.

While these characteristics contextualize tool usage, they do not directly translate into reliability support. Therefore, Table V classifies each tool according to the degree to which evidence from the primary studies indicates support for the four reliability-oriented attributes. The classification is based on explicit empirical results, tool capabilities described in the studies, and the frequency and depth with which each attribute was associated with the tool.

The comparative analysis values for the Reliability Attributes of the tools studied. Regarding coverage, the tools

Evomaster, Evosuite, and Monkey exhibit high coverage levels, while Appium and Randoop show medium values; JUnit, Selenium, and Sikuli have the lowest values in this category. In terms of fault detection capability, Evomaster and Randoop again stand out, achieving high performance; Evosuite, JUnit, Monkey, and Sikuli obtained medium performance, whereas Appium and Selenium achieved low results. Regarding robustness, the tools Appium, Monkey, Randoop, Selenium, and Sikuli demonstrated greater resistance to variations, with high values, while Evosuite obtained medium values, and Evomaster and JUnit presented lower results. With respect to reliability, the results showed high reliability for most of the tools, with the exception of Randoop, which achieved only a medium performance.

Figure 6 shows the number of studies in which evidence was found regarding the eight main tools, followed by the amount of evidence identified in these studies for each of the Reliability Attributes. EvoSuite was the tool with the largest number of studies containing evidence, and it also had the highest amount of evidence for all four attributes. Several studies were also identified that reported Reliability Attributes for JUnit, Randoop, Selenium, and Monkey. Among these tools, JUnit stands out for presenting numerous pieces of evidence related to Reliability. Randoop excels particularly in the Robustness category. Both JUnit and Randoop show strong performance in Fault Detection. Additionally, Randoop and Monkey stand out for presenting significant evidence in the Coverage category. The remaining tools, EvoMaster, Sikuli, and Appium had few pieces of evidence, with Appium being the only tool with zero evidence in the Fault Detection

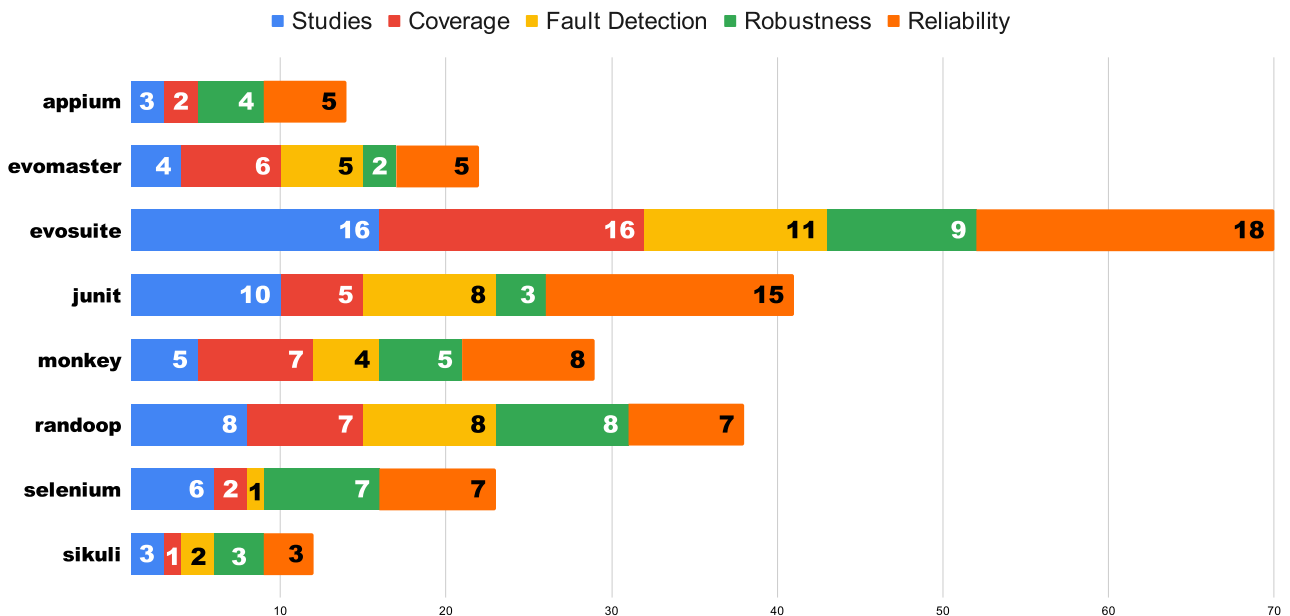


Fig. 6. Frequency of AST Tools evidence by attribute

TABLE V

EXECUTION RELIABILITY-ORIENTED ATTRIBUTES SUPPORTED BY THE MOST CITED AST TOOLS. LOW SUPPORT (●) DENOTES LIMITED OR INDIRECT EMPIRICAL EVIDENCE REPORTED IN THE PRIMARY STUDIES, WHICH MAY REFLECT EXECUTION STABILITY OR USAGE CONTEXT RATHER THAN EXPLICIT FAULT DETECTION CAPABILITY.

Tool	Coverage	Fault Detection	Robustness	Reliability
junit	●	●●	●	●●●
evosuite	●●●	●●	●●	●●●
selenium	●●	●●	●●●	●●●
randoop	●●	●●●	●●●	●●
monkey	●●●	●●●	●●●	●●●
appium	●●	●●	●●●	●●●
evomaster	●●●	●●●	●●	●●●
sikuli	●	●●	●●●	●●●

A. Interpretation Across Reliability Attributes

Table VI synthesizes how the extracted evidence characterizes the contribution of each tool to the four reliability-oriented attributes. Coverage is primarily associated with tools that employ systematic or high-volume exploration. EvoSuite, Randoop, Monkey, and EvoMaster exhibit the strongest empirical support, each accumulating several studies reporting high structural or behavioral coverage. Selenium also appears with consistent coverage evidence across cross-browser and UI-based scenarios, although Appium and JUnit show only limited coverage indications. Fault detection is most strongly supported by EvoSuite, Randoop, Monkey, and EvoMaster, which collectively account for the majority of fault-related evidence. Their underlying strategies—evolutionary search, feedback-directed randomness, event flooding, and API-level fuzzing—are recurrently associated with the revelation of unexpected behaviors and defects. Selenium offers only isolated fault-detection evidence, and the dataset indicates no explicit fault-detection support for Appium or significant robustness-related contributions from JUnit.

Robustness emerges most clearly in tools that introduce stress, variability, or malformed inputs. Monkey presents the largest concentration of robustness evidence, reflecting its use in stress-testing mobile interfaces with high event rates. EvoSuite, Randoop, and EvoMaster also show multiple robustness reports, particularly through boundary-value exposure, non-deterministic behavior exploration, and system-level perturba-

tion. While Selenium and Appium contribute to integration-level reliability through cross-platform execution consistency, the analyzed primary studies do not provide explicit empirical evidence of fault detection capabilities for Appium beyond execution stability and interface interaction support. Reliability, interpreted as the alignment of coverage, stability, and cross-platform consistency, is strongly represented for most tools with substantial empirical presence. EvoSuite, Randoop, Monkey, EvoMaster, Selenium, Appium, and Sikuli accumulate several references indicating execution reliability within their respective scopes. JUnit reported as stable execution baseline mainly due to its role as the foundational execution framework in many studies, despite limited evidence for robustness or fault detection beyond assertion-based checking.

Overall, the results indicate an ecosystem in which different automation strategies provide complementary reliability contributions. Tools based on evolutionary search, random exploration, stress testing, and API fuzzing offer the most comprehensive support across attributes, while UI-level frameworks contribute primarily to cross-platform reliability. No single tool addresses all reliability demands, reinforcing the need to combine diverse automation approaches. Taken together, these findings reveal systematic trade-offs across automated testing strategies and reliability-oriented attributes. Rather than converging toward a dominant solution, the evidence points to a fragmented landscape in which different tools address complementary aspects of reliability. This observation has

TABLE VI
EXECUTION RELIABILITY-ORIENTED ATTRIBUTES SUPPORTED BY AST TOOLS

Tool	Studies	Coverage	Fault Detection	Robustness	Reliability
appium	3	PS149 PS169		PS149 PS169	PS051 PS149 PS169
evomaster	4	PS102 PS106 PS139 PS147	PS102 PS106 PS139 PS147	PS147	PS102 PS106 PS139 PS147
evosuite	16	PS031 PS039 PS043 PS056 PS068 PS102 PS144 PS262 PS279 PS317 PS321 PS336	PS039 PS043 PS068 PS084 PS102 PS144 PS262 PS279 PS336	PS031 PS043 PS056 PS068 PS123 PS279 PS317 PS336	PS029 PS031 PS039 PS043 PS056 PS068 PS084 PS102 PS113 PS144 PS262 PS279 PS317 PS321 PS336
junit	10	PS029 PS162 PS231 PS317	PS039 PS094 PS162 PS184 PS231 PS339	PS162 PS231	PS029 PS039 PS078 PS094 PS162 PS184 PS231 PS300 PS317 PS339
monkey	5	PS009 PS090 PS195 PS229	PS090 PS174 PS195 PS229	PS090 PS174 PS195 PS229	PS009 PS090 PS174 PS195 PS229
randoop	8	PS013 PS031 PS113 PS256 PS317	PS013 PS084 PS113 PS175 PS256 PS317	PS013 PS030 PS031 PS113 PS256	PS013 PS084 PS113 PS256
selenium	6	PS027 PS189	PS027	PS027 PS170 PS178 PS189 PS294	PS027 PS078 PS170 PS178 PS189 PS294
sikuli	3	PS170	PS170 PS186	PS163 PS170 PS186	PS163 PS170 PS186

direct consequences for both empirical research agendas and practical testing strategies, which are discussed next.

B. Implications for Research and Practice

The evidence-based analysis presented in this section yields several implications for both software engineering research and industrial practice, particularly in contexts where automated testing is expected to contribute to software reliability under continuous integration constraints.

From a research perspective, the results highlight the need for empirical studies that move beyond isolated tool evaluations. A large proportion of the analyzed evidence focuses on single tools or narrowly scoped contexts, which limits the comparability and cumulative value of findings. Future work should prioritize cross-tool and cross-context investigations that explicitly examine how different automation strategies complement or trade off across reliability-oriented attributes. The analysis also exposes underexplored reliability challenges in backend services and API-level systems. Although tools such as EvoMaster and Randoop demonstrate strong empirical support in these contexts, the overall volume of evidence remains limited compared to UI-centric and unit-level testing. This imbalance suggests opportunities for targeted empirical studies addressing reliability in service-oriented and distributed architectures, where failures often emerge from interaction faults and unexpected input combinations. In addition, the absence of AI-driven or large language model (LLM)-based testing approaches in the primary evidence indicates a clear research gap. While recent work proposes AI-assisted test generation and maintenance, their impact on fault detection, robustness, and execution reliability has yet to be systematically evaluated. Empirical validation of these approaches represents a promising direction for advancing reliability-aware automation research.

For practitioners, the findings reinforce that no single automated testing tool provides comprehensive support for all reliability-oriented attributes. Tools that excel in coverage do not necessarily exhibit strong fault detection or robustness, and vice versa. As a result, practitioners should adopt a portfolio-based testing strategy that combines complementary tools according to reliability goals, system characteristics, and operational constraints. The results further caution against treating coverage metrics as a direct proxy for software reliability. Although coverage remains a useful adequacy indicator, the evidence shows that high coverage does not guarantee effective fault detection or robust execution behavior. Overreliance on coverage metrics may therefore lead to misleading conclusions about system reliability, particularly in CI-driven workflows.

Finally, the analysis underscores the importance of execution stability and reproducibility in continuous integration environments. Beyond detection capability, practitioners must consider tool robustness, flakiness, and execution overhead when integrating automated testing into CI pipelines. Evidence-based tool selection, informed by empirical reliability attributes rather than popularity or default usage, is

essential for sustaining reliable and maintainable testing infrastructures.

C. Threats to Validity

This mapping is subject to typical validity considerations. The study builds on a previously conducted systematic mapping, which defined the initial corpus of 341 studies. Although that protocol was broader in scope, all studies were re-evaluated using refined criteria focused on automated testing and reliability-related evidence, reducing the risk of selection bias. Subjectivity in data extraction is another potential source of variation, particularly in assigning support levels (•, ••, •••) to reliability attributes. To mitigate this, classifications were grounded in explicit empirical findings and iteratively reviewed for consistency. This study synthesizes evidence and does not establish causal relationships.

A further consideration relates to missing data: 65 studies from the original corpus were unavailable in full text and had to be excluded. Their proportion relative to the final set of 184 studies helps limit the impact of this constraint. Finally, the synthesis aggregates evidence from heterogeneous systems, tools, and domains. While this diversity supports broader applicability, certain patterns may not generalize uniformly. Even so, the structured protocol and transparent criteria contribute to the robustness of the conclusions.

VII. FINAL REMARKS

This systematic mapping consolidates eleven years of research on automated software testing (AST), synthesizing evidence from 184 primary studies to provide an integrated view of how automation tools contribute to reliability-oriented attributes. By examining the capabilities of the most frequently cited AST tools, the study highlights how different categories of automation support robustness evaluation, increase coverage, enhance fault detection, and contribute to dependable system behavior. The findings emphasize that no single tool addresses all reliability needs; instead, the evidence points to a complementary ecosystem in which systematic generation, UI automation, random event simulation, and fuzzing techniques play distinct and synergistic roles.

The mapping also reveals notable gaps. Backend and service-oriented architectures remain comparatively underserved by widely adopted automation tools, and explicit robustness-focused testing—such as stress, resilience, or fault-tolerance assessment—is still supported by only a limited subset of technologies. These gaps indicate promising opportunities for advancing automation techniques, particularly for systems that depend on distributed communication, heterogeneous interfaces, or unpredictable operational environments.

The contributions of this study lie in offering a structured, reliability-centered interpretation of AST evidence that clarifies the strengths and limitations of the current tool landscape and highlights areas requiring further methodological and technological progress. For practitioners, the results provide guidance on how different tool categories align with coverage goals, robustness requirements, and fault detection strategies.

For researchers, the findings outline a set of open challenges that may guide future developments in testability engineering and automated validation. A key outcome of this study is the generation of an explicit, evidence-based artifact in the form of a reliability-oriented attribute matrix. This matrix consolidates traceable empirical evidence from 184 primary studies and maps widely used automated testing tools to four reliability-oriented attributes—coverage, fault detection, robustness, and execution reliability. By grounding each classification in identifiable primary studies, the artifact enables transparent interpretation and practical reuse, supporting both researchers and practitioners in evidence-based tool selection and testing strategy design.

Future work may explore standardized decision frameworks for selecting AST tools according to reliability goals, empirical evaluations of tool combinations in real-world environments, and long-term studies on the sustainability and maintenance of automated test suites. Additional investigations into automation for DevOps pipelines, microservices, and AI-enabled systems may further expand the understanding of how AST tools can continue to support dependable software in increasingly complex operational contexts.

ACKNOWLEDGMENT

Financial support for this research was provided by the CAPES Social Demand Program and the Federal University of Alagoas (UFAL). We also thank the UFAL Institute of Computing (IC) and the DKD Lab from the Federal University of the Agreste of Pernambuco (UFPE).

Artificial intelligence was used to support the translation of this work into English.

REFERENCES

- [1] Bhagat and Shreya, "Review on mobile application development based on flutter platform," *International Journal for Research in Applied Science and Engineering Technology*, vol. 10, pp. 803–809, 01 2022.
- [2] J. Petrić, T. Hall, and D. Bowes, "Which software faults are tests not detecting?" in *Proceedings of the 24th International Conference on Evaluation and Assessment in Software Engineering*, ser. EASE '20. New York, NY, USA: Association for Computing Machinery, 2020. doi: 10.1145/3383219.3383236. ISBN 9781450377317 pp. 160–169.
- [3] F. Gurcan, G. G. M. Dalveren, N. E. Cagiltay, D. Roman, and A. Soyulu, "Evolution of software testing strategies and trends: Semantic content analysis of software research corpus of the last 40 years," *IEEE Access*, vol. 10, pp. 106 093–106 109, 2022. doi: 10.1109/ACCESS.2022.3211949
- [4] F. Pudlitz, F. Brokhausen, and A. Vogelsang, "What am i testing and where? comparing testing procedures based on lightweight requirements annotations," *Empirical Software Engineering*, vol. 25, no. 4, pp. 2809–2843, Jul 2020. doi: 10.1007/s10664-020-09815-w
- [5] K. L. Petry, E. O. Jr, and A. F. Zorzo, "Model-based testing of software product lines: Mapping study and research roadmap," *Journal of Systems and Software*, vol. 167, p. 110608, 2020. doi: 10.1016/j.jss.2020.110608
- [6] ISO/IEC/IEEE, *International Standard—Software and Systems Engineering Software Testing—Part 1: General Concepts*, ISO/IEC/IEEE Std. 29 119-1:2022(E), 2022.
- [7] V. Garousi, A. B. Keleş, Y. Balaman, Z. Özdemir Güler, and A. Arcuri, "Model-based testing in practice: An experience report from the web applications domain," *Journal of Systems and Software*, vol. 180, p. 111032, 2021. doi: 10.1016/j.jss.2021.111032
- [8] V. Stray, R. Florea, and L. Paruch, "Exploring human factors of the agile software tester," *Software Quality Journal*, vol. 30, no. 2, pp. 455–481, Jun 2022. doi: 10.1007/s11219-021-09561-2
- [9] Peischl, Bernhard, Tazl, Oliver, Wotawa, and Franz, "Testing anticipatory systems: A systematic mapping study on the state of the art," *Journal of Systems and Software*, vol. 192, p. 111387, 06 2022. doi: 10.1016/j.jss.2022.111387
- [10] M. Swillus and A. Zaidman, "Sentiment overflow in the testing stack: Analyzing software testing posts on stack overflow," *Journal of Systems and Software*, vol. 205, p. 111804, 2023. doi: 10.1016/j.jss.2023.111804
- [11] J. Lojda, R. Panek, J. Podivinsky, O. Cekan, M. Krcma, and Z. Kotasek, "Testing embedded software through fault injection: Case study on smart lock," in *2021 IEEE 22nd Latin American Test Symposium (LATS)*, 2021. doi: 10.1109/LATS53581.2021.9651770 pp. 1–6.
- [12] D. R. Falcó, A. Serrano-Cases, A. Martínez-Alvarez, and S. Cuenca-Asensi, "Soft error reliability predictor based on a deep feedforward neural network," in *2020 IEEE Latin American Test Symposium (LATS)*, 2020. doi: 10.1109/LATS49555.2020.9093688 pp. 1–5.
- [13] J. Podivinsky, J. Lojda, R. Panek, O. Cekan, M. Krcma, and Z. Kotasek, "Evaluation platform for testing fault tolerance: Testing reliability of smart electronic locks," in *2020 IEEE 11th Latin American Symposium on Circuits & Systems (LASCAS)*, 2020. doi: 10.1109/LASCAS45839.2020.9068977 pp. 1–4.
- [14] R. Natella, D. Cotroneo, and H. S. Madeira, "Assessing dependability with software fault injection: A survey," *ACM Comput. Surv.*, vol. 48, no. 3, Feb. 2016. doi: 10.1145/2841425. [Online]. Available: <https://doi.org/10.1145/2841425>
- [15] P. Kong, L. Li, J. Gao, K. Liu, T. F. Bissyandé, and J. Klein, "Automated testing of android apps: A systematic literature review," *IEEE Transactions on Reliability*, vol. 68, pp. 45–66, 03 2019. doi: 10.1109/TR.2018.2865733
- [16] V. Garousi, M. Felderer, M. Kuhrmann, K. Herkiloğlu, and S. Eldh, "Exploring the industry's challenges in software testing: An empirical study," *J. Softw. Evol. Process*, vol. 32, no. 8, Aug. 2020. doi: 10.1002/smr.2251
- [17] M. Waseem, P. Liang, M. Shahin, A. D. Salle, and G. Márquez, "Design, monitoring, and testing of microservices systems: The practitioners' perspective," *Journal of Systems and Software*, vol. 182, p. 111061, 08 2021. doi: 10.1016/j.jss.2021.111061
- [18] J. B. Minani, F. Sabir, N. Moha, and Y.-G. Guéhéneuc, "A systematic review of iot systems testing: Objectives, approaches, tools, and challenges," *IEEE Transactions on Software Engineering*, vol. 50, pp. 785–815, 2024. doi: 10.1109/TSE.2024.3363611
- [19] M. Dadkhah, S. Araban, and S. Paydar, "A systematic literature review on semantic web enabled software testing," *Journal of Systems and Software*, vol. 162, p. 110485, 2020. doi: 10.1016/j.jss.2019.110485
- [20] S. Piparia, D. Adamo, R. Bryce, H. Do, and B. Bryant, "Combinatorial testing of context aware android applications," in *2021 16th Conference on Computer Science and Intelligence Systems (FedCSIS)*, 2021. doi: 110.15439/2021F003
- [21] J. Bicevskis, Z. Bicevska, A. Nikiforova, and I. Oditis, "Data quality model-based testing of information systems," in *2020 15th Conference on Computer Science and Information Systems (FedCSIS)*, 2020. doi: 10.15439/2020F253 pp. 488–499.
- [22] I. Santos, E. F. Coutinho, and S. R. S. Souza, "Software testing ecosystems insights and research opportunities," in *SBES '20: Proceedings of the XXXIV Brazilian Symposium on Software Engineering*, 2020. doi: 10.1145/3422392.3422458 pp. 421–426.
- [23] M. Nass, E. Alégroth, and R. Feldt, "Why many challenges with gui test automation (will) remain," *Information and Software Technology*, vol. 138, p. 106625, 2021. doi: 10.1016/j.infsof.2021.106625
- [24] I. V. Kovalev, M. V. Saramud, V. V. Losev, N. A. Testoyedov, and M. V. Karaseva, "On the problem of predicting meantime between failures on the basis of data from automated testing of on-board software components," *Journal of Physics: Conference Series*, vol. 1679, p. 052037, 2020. doi: 10.1088/1742-6596/1679/5/052037
- [25] O. A. Lazzarini Lemos, F. Fagundes Silveira, F. Cutigi Ferrari, and A. Garcia, "The impact of software testing education on code reliability: An empirical assessment," *Journal of Systems and Software*, vol. 137, pp. 497–511, 2018. doi: 10.1016/j.jss.2017.02.042. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S0164121217300419>
- [26] I. Sommerville, *Software Engineering*, 10th ed. São Paulo: Pearson Universidades, 2019.
- [27] V. Pradhan, J. Dhar, and A. Kumar, "Testing coverage-based software reliability growth model considering uncertainty of operating environment," *Systems Engineering*, vol. 26, no. 4, pp. 449–

- 462, 2023. doi: <https://doi.org/10.1002/sys.21671>. [Online]. Available: <https://incose.onlinelibrary.wiley.com/doi/abs/10.1002/sys.21671>
- [28] P. Raulamo-Jurvanen, S. Hosio, and M. V. Mäntylä, “Practitioner evaluations on software testing tools,” in *Proceedings of the 23rd International Conference on Evaluation and Assessment in Software Engineering*, ser. EASE '19. New York, NY, USA: Association for Computing Machinery, 2019. doi: 10.1145/3319008.3319018. ISBN 9781450371452 p. 57–66. [Online]. Available: <https://doi.org/10.1145/3319008.3319018>
- [29] A. Bertolino, B. Miranda, R. Pietrantuono, and S. Russo, “Hybrid is better: Why and how test coverage and software reliability can benefit each other,” in *Web Information Systems and Technologies*. Cham: Springer International Publishing, 2019. ISBN 978-3-030-35330-8 pp. 25–38.
- [30] S. Oveisi, A. Moeini, S. Mirzaei, and M. A. Farsi, “Software reliability prediction: A survey,” *Quality and Reliability Engineering International*, vol. 39, pp. 412–453, 12 2022. doi: 10.1002/qre.3220
- [31] T. Bach, A. Andrzejak, R. Pannemans, and D. Lo, “The impact of coverage on bug density in a large industrial software project,” in *2017 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*, 2017. doi: 10.1109/ESEM.2017.44 pp. 307–313.
- [32] C. Diwaker, P. Tomar, A. Solanki, A. Nayyar, N. Z. Jhanjhi, and A. Abdullah, “A new model for predicting component-based software reliability using soft computing,” *IEEE Access*, vol. 7, pp. 147 191–147 203, 2019. doi: 10.1109/ACCESS.2019.2946862
- [33] R. Pietrantuono, A. Bertolino, G. D. Angelis, B. Miranda, and S. Russo, “Towards continuous software reliability testing in devops,” in *2019 IEEE/ACM 14th International Workshop on Automation of Software Test (AST)*, 2019. doi: 10.1109/AST.2019.00009 pp. 21–27.
- [34] Y. Yang, Y. Zhou, H. Sun, Z. Su, Z. Zuo, and L. Xu, “Hunting for bugs in code coverage tools via randomized differential testing,” in *2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE)*, 2019. doi: 10.1109/ICSE.2019.00061 pp. 488–499.
- [35] B. Kitchenham, O. P. Brereton, D. Budgen, and M. Turner, “Guidelines for performing systematic literature reviews in software engineering,” Keele University and Durham University Joint Report, Keele University, Tech. Rep. EBSE 2007-001, 2007.
- [36] A. Farias, R. Rocha, I. Vanderlei, J. Araujo, A. Araújo, and J. Dantas, “Software testing evidence: Results from a systematic mapping,” in *Proceedings of the 21st International Conference on Web Information Systems and Technologies - Volume 1: WEBIST, INSTICC*. SciTePress, 2025. doi: 10.5220/0013739100003985. ISBN 978-989-758-772-6 pp. 386–393.
- [37] SCIMAGO, “Scimago journal & country rank,” 2024, accessed on: November 16, 2024. [Online]. Available: <https://www.scimagojr.com/>