CAFE: A Context-Aware Framework for Selecting Third-Party Software Tools – An Industrial Case Study of Web API Fuzzing*

Alma Oručević-Alagić[1][0000−0001−5639−8167], Karl Alemo[1], Fredrik Orheim[1], Per Runeson[1][0000−0003−2795−4851], and Victoria Vucic[2]

[1] Department of Computer Science, Lund University, Lund, Sweden
{alma.orucevic-alagic,per.runeson}@cs.lth.se
{ka5261al-s,fr3214or-s}@student.lu.se
[2] Axis Communications, Lund, Sweden
Victoria.Vucic@axis.com

Abstract. Background: Industrial adoption of third-party software tools requires more than technical performance evaluation, yet selection methods often underrepresent integration and organizational constraints. **Objective:** We develop CAFE, a context-aware framework for eliciting, assessing, and synthesizing tool selection requirements. **Method:** Following a Design Science Research approach, CAFE was developed using literature and evidence from an industrial web API fuzzing tool selection project in an embedded systems organization. The case involved stakeholder interviews, focus groups, benchmarking, proof of concept integration, documentation and security reviews, and practitioner feedback. **Results:** CAFE structured requirements across technical, integration and operational, and organizational and contextual dimensions. Selection depended not only on technical performance but also on project maturity, security, maintainability, and integration feasibility. **Conclusions:** CAFE supports traceable and context sensitive tool selection by linking stakeholder requirements to heterogeneous evidence. The study highlights the limitations of performance centric evaluation in security and safety critical environments.

Keywords: Design Science Research · software evaluation framework · third-party software tools · software ecosystems · context-aware evaluation · web fuzzing · software testing

1 Introduction

The evaluation and selection of third-party software components is a fundamental and ongoing activity in industrial software development, driven by the

* Preprint version. Accepted for publication in the Industry Papers Track of the 27th International Conference on Product-Focused Software Process Improvement (PROFES 2026), Karlskrona, Sweden, November 30–December 2, 2026.

continuous evolution of software systems and the emergence of new technological alternatives [8, 19].

Although prior works address technology adoption, quality attributed trade-offs, technical benchmarking, software selection, and socio-technical context, they do not provide an integrated process for linking context-specific stakeholder requirements to heterogeneous evidence when selecting third-party tools.

To address this gap, we present a context-aware approach for evaluating third-party software components in industrial software ecosystems. The approach is derived in the context of an industrial case study on the evaluation of web API fuzz testing tools in a large-scale embedded systems environment [2], where both technical performance and organization specific factors, such as integration feasibility, usability, security considerations, and alignment with existing processes, are systematically considered. The need for context-aware evaluation is particularly acute for AI-enabled software engineering tools [12]. Public benchmarks capture a part of their industrial suitability, while adoption depends on integration with existing workflows, data governance, privacy and intellectual-property constraints, security, operational cost, human oversight, and the ability to monitor changing model behavior.

Using a Design Science Research (DSR) approach [24], we combine literature analysis, stakeholder interviews, and empirical benchmarking to identify evaluation criteria and assess multiple open source fuzz testing tools against real-world APIs. The results highlight that, while performance remains an important factor, the suitability of a tool is heavily influenced by contextual constraints in the software ecosystem, including workflow compatibility and organizational requirements. These findings reinforce the need for structured and context-aware evaluation methodologies when selecting third-party technology solutions, and provide a concrete foundation for the approach explored in this work. Our primary contribution is the context-aware evaluation framework (CAFE), targeting the selection and integration of third-party software components within an organization's software ecosystem, consisting of the following elements:

I. A structured framework connecting context, requirements, assessment, methods, evidence and selection decisions.
II. An industrially grounded process for eliciting technical and non-technical requirements.
III. An application of the framework showing how heterogeneous evidence changes a tool-selection decision.
IV. Lessons about the limitations of performance centric tool evaluation.

Hence, this paper offers a reusable process framework for context-aware selection of third-party software tools in general.

This paper is organized as follows. Section 2 reviews related work. Section 3 outlines research methodology grounded in DSR. Section 4 presents the proposed evaluation framework as the main design artifact. Section 5 presents the case study application of the framework. Section 6 discusses the findings and Section 7 concludes the paper.

2 Background and Related Work

Selection of third-party software components takes place in industrial software ecosystems where technical systems, organizational structures, development processes, and human factors are tightly interdependent [8]. Research has shown that coordination and communication structures within organizations influence software design and evolution [8, 9, 23]. As a result, decisions regarding technology adoption and system design must consider organizational context, workflows, and integration constraints.

Research on fuzzing-tool evaluation often prioritizes measurable attributes e.g., performance, scalability, or feature completeness. A systematic review of 150 fuzzing papers by Schloegel et al. [26] focuses on evaluation practices such as targets, metrics, baselines, statistical analysis, and reproducibility, without addressing the organizational criteria that shape selection and adoption of fuzzing tools in practice. Although these technical criteria are important, they do not fully capture the suitability of a technology in its intended context. Hence, decisions inherently involve trade-offs across multiple quality attributes and must account for broader contextual constraints [6, 8, 17].

Existing approaches to technology evaluation in software engineering, such as the technology acceptance model [28], largely address individual aspects of the problem in isolation. Architecture evaluation methods such as ATAM [17] and related work on software architecture [6] provide systematic ways to reason about quality attributes and trade-offs, but are primarily focused on system-level design decisions rather than the selection of third-party components in evolving software ecosystems. Similarly, empirical studies on software tools typically emphasize performance metrics, e.g. fault detection and coverage, through controlled benchmarking [20]. While these approaches provide valuable insights, they offer limited support for evaluating how third-party tools suit existing workflows and organizational contexts.

Related work that concerns software tool selection includes DESMET [18], which provides a context aware methodology for evaluating software engineering methods and tools. AHP[16] and TOPSIS[14] are multi-criteria decision analysis techniques for explicit weighting of predefined criteria and the ranking of alternatives. Open Source Software selection frameworks, such as FOSSES[1] provide structured procedures for assessment of candidate software. Bjarnason et al. [7] further present a systematic model for software selection in large scale software engineering based on interactive rapid reviews, emphasizing structured and evidence based decision making in industrial contexts. These approaches differ in whether they focus on evaluation methodology, evidence acquisition, formal aggregation, or OSS specific candidate assessment.

However, the complementary lines of research do not provide an integrated process for linking stakeholder derived, context-specific requirements to heterogeneous evidence in third party tool selection.

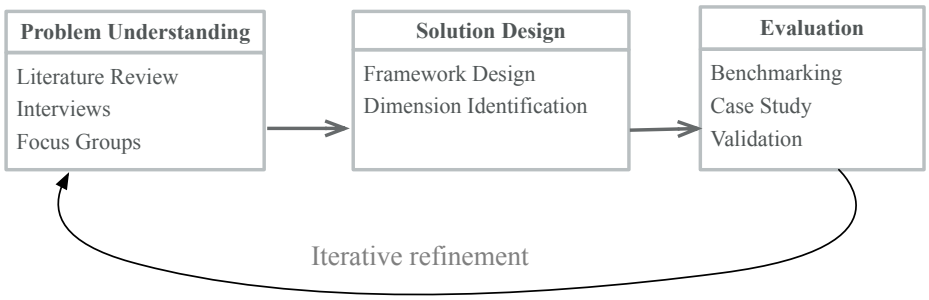

Fig. 1. Overview of the DSR-based research process used in this study, consisting of iterative phases of problem understanding, solution design, and evaluation. Adapted from Runeson et al. [24].

3 Research Methodology

To address this gap, we adopt a Design Science Research (DSR) approach [24], which provides a methodological framework for addressing complex, real-world problems through the design and evaluation of artifacts. In contrast to purely descriptive research, DSR focuses on developing solutions that are theoretically grounded and practically relevant, and enables the systematic construction of an evaluation artifact that integrates technical performance metrics with organizational and contextual factors identified in industrial practice.

3.1 Research Approach

The research process is illustrated in Figure 1, showing the iterative phases of problem understanding, solution design, and evaluation. The objective of this study is to design a framework that supports systematic and context-aware evaluation of third-party technologies. In line with DSR principles, the research process consists of iterative cycles of problem identification, artifact design, and evaluation. The problem addressed in this work is the lack of structured approaches for evaluating third-party software components in industrial contexts, where decisions are often based on limited technical criteria and insufficient consideration of organization and ecosystem related factors. The proposed artifact is a context-aware evaluation framework that incorporates both technical and socio-technical dimensions.

3.2 Industrial Case Study Context

The development of the framework is grounded in an industrial case study focusing on the evaluation and integration of open source API fuzzing tools in a

large-scale embedded systems environment. The study builds on work conducted as part of a master's thesis [2] which provides additional implementation details.

The study aims to identify key aspects required for adopting testing tools in practice, including performance characteristics, integration feasibility, usability, security considerations, and alignment with existing organizational processes. These aspects span both technical properties of the tools and software ecosystem related criteria, such as workflow compatibility, integration constraints, and organizational requirements. The work is conducted in a complex industrial setting characterized by established development workflows, system constraints, and high requirements on software quality and security.

3.3 Industrial Case Context

This work applies the proposed framework to the selection of fuzz testing tools for web APIs. Fuzz testing, or fuzzing, automatically generates and mutates inputs to expose faults and vulnerabilities [20]. Web API fuzzing tools generate HTTP requests based on interface specifications, such as the OpenAPI, and evaluate system responses for unexpected behavior or violations of the specification [13]. Although this enables large scale and automated exploration of input spaces, tools differ in capability, maturity, and integration needs, and are evaluated using technical benchmarks such as fault detection, code coverage, and response anomalies [20, 21]. These metrics provide only a partial basis for industrial selection since adoption also depends on usability, workflow integration, security, and organizational fit.

3.4 Data Collection and Analysis

The research combines multiple data collection methods to derive and structure the evaluation framework, following a DSR process, as illustrated in Figure 1.

In the *Problem Understanding* phase, a literature review was conducted to identify relevant concepts, evaluation criteria, and existing approaches related to software testing, technology evaluation, and software ecosystems. Stakeholder input was gathered initially through semi-structured interviews, followed by focus group discussions involving practitioners in software development, testing and security roles. These activities provided insights into practical requirements, constraints, and decision-making processes in the case company context. In the *Solution Design* phase, the collected insights were used to identify key evaluation dimensions and structure the proposed framework. In the *Evaluation* phase, an empirical evaluation was performed by benchmarking multiple open-source web API fuzzing tools against APIs of the case company. The evaluation considered not only technical performance metrics, such as fault detection and coverage, but also aspects related to usability, integration effort, and compatibility with existing workflows in the case company. The collected data was analyzed to identify key evaluation dimensions and recurring challenges, which informed the design of the proposed framework.

Table 1. Evidence sources used in the industrial study and their roles in the development and application of CAFE.

Evidence source	Participants or objects	Purpose	Role in CAFE
Interviews	Five stakeholders: two QA testers, one developer, one software architect, and one security expert.	Elicit testing practices, adoption constraints, and desired tool properties.	Basis for organizational, contextual, usability, and integration requirements.
Focus group	Eighteen employees divided into three groups, representing development, testing, security, architecture, and project management.	Review and refine the interview findings and discuss benefits, uses, and adoption challenges.	Triangulation and refinement of the elicited requirements.
Document study	Working standards, educational materials, QA processes, security measures, testing guidelines, and implementation resources.	Characterize the company environment and identify workflow, security, and legal constraints.	Case-context characterization and validation of requirements.
Tool screening	Twenty-five initial open-source candidates assessed for maintenance, security posture, OpenSSF Scorecard results, and licensing.	Remove unsuitable candidates before detailed technical evaluation.	Security, maturity, and licensing evidence for candidate selection.
Benchmarks	Five fuzzers across 10 APIs, eight camera units, five hardware models, and 1,000 runs representing approximately 583 CPU hours.	Assess server-error detection, schema violations, and API coverage.	Quantitative evidence for the technical dimension.
Proof of concept	EvoMaster integrated into the existing CI/CD pipeline and embedded testing environment.	Examine automation, configurability, reproducibility, and integration feasibility.	Application of the integration and operational requirements.
Practitioner feedback	Three representatives covering security-testing, development, and QA-testing perspectives, participating in weekly evaluation sessions.	Evaluate the PoC and identify improvements through iterative feedback.	Application-level evaluation and refinement of the PoC.

Table 1 summarizes the empirical evidence sources used during the industrial study and explains how each source contributed to the development or application of CAFE. The industrial case served both as an empirical basis for developing the framework and as an illustrative application of it [2].

4 Evaluation Framework

The framework is presented as a conceptual model, as illustrated in Figure 2. It organizes the evaluation process into a set of dimensions, each comprising a collection of requirements that are assessed using appropriate assessment methods, producing evaluation results.

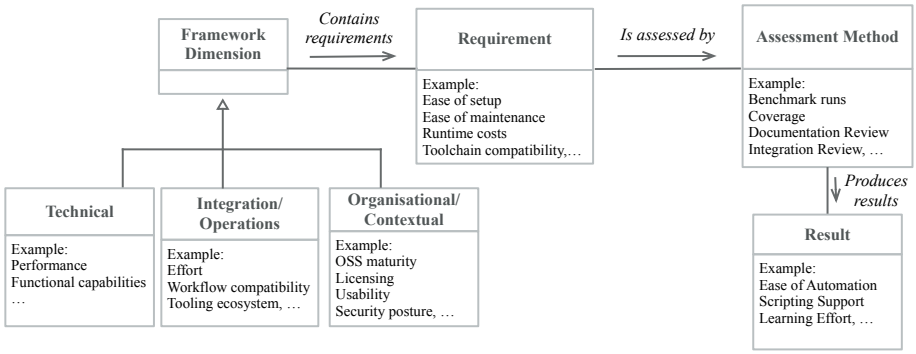

Fig. 2. Conceptual model of the Context-Aware Framework for Evaluation (CAFE). The framework organizes evaluation into dimensions, each containing requirements that are assessed using appropriate methods, producing evaluation results.

4.1 Framework Structure

At the highest level, the framework consists of a set of evaluation dimensions, which represent distinct perspectives from which a software component is assessed. Each dimension contains a set of evaluation requirements, capturing specific properties or criteria relevant for decision-making. These requirements are evaluated using one or more assessment methods, which generate evaluation results. The evaluation results are then synthesized across dimensions to support selection decisions among evaluated third-party components. Formally, the framework follows the structure: 1. Framework Dimension, 2. Requirement, 3. Assessment Method, and 4. Result. This structure reflects the need to systematically connect high-level evaluation concerns with concrete evidence obtained through empirical and qualitative assessment.

4.2 Evaluation Dimensions

The framework distinguishes between three primary dimensions, reflecting the multifaceted nature of technology evaluation in an industrial context.

The **Technical Dimension** captures intrinsic properties of the evaluated component, focusing on its performance characteristics and functional capabilities. Typical requirements include performance, e.g. fault detection capability, runtime efficiency, and functional capabilities, e.g. support for API specification. These requirements are typically evaluated using empirical methods such as benchmarking and test execution. **Integration and Operational Dimension** reflects how well the component integrates into the existing software ecosystem and development workflows. It captures requirements related to integration effort, workflow compatibility, and tooling ecosystem and interoperability. Assessment methods in this dimension often involve proof of concept (PoC) integration assessment, pipeline testing, and analysis of compatibility with existing tools and

processes. **Organizational and Contextual Dimension** captures factors that influence adoption within the specific organizational setting. These include: usability and learning effort, security posture, open source maturity and licensing constraints. These requirements are evaluated using qualitative methods such as expert assessment, documentation review, and analysis of community activity.

The specific requirements within each dimension are not fixed, but are derived from the evaluation context. In practice, these requirements are elicited through a combination of stakeholder input, analysis of system constraints, and consideration of organizational processes. This way, the framework can be adapted to different industrial settings while maintaining a consistent evaluation structure.

4.3 Assessment Methods and Evaluation Results

Each requirement is evaluated using one or more assessment methods, which may be quantitative, qualitative, or a combination of both, e.g., benchmarking assesses performance and fault detection, documentation review evaluates usability and maturity, and integration testing assesses compatibility with existing workflows.

The application of these methods provides evidence for assessing each requirement. These results may take different forms, e.g. quantitative metrics like coverage, number of faults, or qualitative assessments like ease of use, integration complexity, etc. Some assessment methods and results may be associated with multiple requirements reflecting the interdependent nature of evaluation criteria in real-world settings. CAFE intentionally does not specify a weighting schema for requirements, as their priorities and trade-offs depend on the organizational, operational and technical context. The proposed framework incites practitioners to assess the heterogeneous evidence per requirement through explicit and traceable rationale rather than through a mandatory score.

5 Case Study: Application of CAFE Framework

The CAFE framework is applied through a structured evaluation process that connects context-specific requirements with empirical and qualitative evidence. The application typically involves the following steps:

(1) **Context identification**: Identify system specific and organizational constraints that influence the evaluation, such as integration requirements, security policies, and development workflows.
(2) **Requirement selection**: Select and refine relevant evaluation requirements within each dimension based on the identified context.
(3) **Assessment**: Apply appropriate assessment methods to evaluate each requirement, using a combination of benchmarking, PoC integration assessment, and qualitative analysis.
(4) **Result synthesis**: Analyze the resulting evidence across dimensions, identifying trade-offs and relative strengths of candidate components.

(5) **Decision support**: Use the synthesized results to support informed selection decisions aligned with both technical requirements and organizational priorities.

The CAFE framework supports evaluation by structuring the assessment process across multiple dimensions and linking requirements to concrete evidence. Thus, CAFE guides companies to consider trade-offs across technical, integration, and organizational factors. This integrated perspective supports a multi-dimensional view of software under evaluation, and in this way may support informed selection decisions. Each step is summarized below and reported in more detail in the technical report [2].

5.1 Context Identification

The case company environment imposed several technical and organizational constraints influencing the evaluation process. The evaluated systems consisted of embedded Linux based products exposing multiple internal and external APIs. Hence, the evaluation required tools capable of generating tests from OpenAPI specifications, executing them automatically, and detecting faults related to robustness and security. In addition to technical aspects, the organization maintained mature CI/CD workflows and strict policies regarding software security and open source adoption. Therefore, the evaluation needed to take into consideration not only a fuzzing tool's effectiveness, but also operational integration, maintainability, licensing constraints, and long term sustainability of candidate tools.

The contextual factors were identified through a combination of literature analysis, internal document studies, semi-structured interviews, and focus group discussions. Five semi-structured interviews were conducted with stakeholders from different organization roles, including software developers, QA personnel, software architects, and security specialists. The interview material was then analyzed using thematic analysis, where statements were coded, grouped into themes, and further refined into higher level meta themes, representing recurring organizational and operational concerns.

The resulting themes highlighted several requirements extending beyond technical fuzzing performance. Practitioners emphasized CI/CD integration, setup complexity, reproducibility, operational stability, scripting support, usability, and transparency of generated test cases. Furthermore, organizational concerns related to open source governance, maintenance activity, software security posture, and licensing compatibility were identified as important decision factors.

Figure 3 illustrates the iterative DSR based workflow, including the relationship between requirement elicitation, benchmarking, proof of concept implementation, and evaluation activities.

5.2 Requirements Selection

The elicited requirements were structured according to the three dimensions of CAFE. From a **technical perspective**, requirements included fault detec-

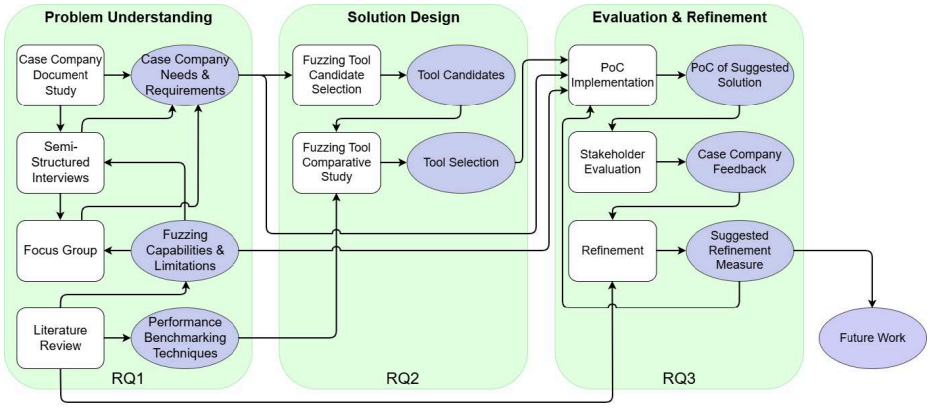

Fig. 3. Overview of the DSR-based research process used in the study at the case company. Adapted from the technical report [2].

tion capability, schema violation detection, API coverage, runtime efficiency, and support for OpenAPI specifications. Coverage evaluation was based on the Test Coverage Model (TCM) [21]. From an **integration and operational perspective**, requirements included containerization support, automation capabilities, CI/CD compatibility, runtime stability, reproducibility of executions, and operational setup complexity. From an **organizational and contextual perspective**, requirements included usability, documentation quality, open source maturity, maintenance activity, licensing constraints, and software security posture. Security related evaluation additionally considered OpenSSF Scorecard [22] analysis and inspection of licensing models, including permissive and copyleft licenses. Table 1 provides evidence sources used in the industrial study and their roles in the development and application of CAFE, while Table 2 links the elicited requirements and their corresponding framework dimensions.

5.3 Assessment and Benchmarking

The assessment phase combined empirical benchmarking with qualitative operational evaluation. Based on literature analysis and ecosystem exploration, several open source web API fuzzing tools were selected as candidates for evaluation, including EvoMaster [4], RESTler [5], and Schemathesis [15]. In order to support systematic benchmarking, the project extended RESTgym benchmarking framework [10]. Each of the selected candidate tools was benchmarked repeatedly against representative APIs from the industrial environment to account for the nondeterministic nature of fuzzing. Benchmark execution collected multiple performance indicators, including: number of server error responses, schema violations, black-box coverage metrics [21], Test Coverage Levels (TCL) [21], runtime stability and execution consistency.

Coverage analysis was performed using the Restats framework [11], which analyzed collected HTTP interactions against the OpenAPI specifications of the

Table 2. Examples of elicited requirements and their corresponding framework dimensions and assessment methods.

Framework Dimension	Example Requirement	Requirement Source	Assessment Method
Technical	Fault detection capability	Literature review, interviews	Empirical benchmarking
Technical	API coverage	Literature review	Restats / TCL analysis
Integration & Operational	CI/CD compatibility	Interviews, focus group	PoC integration evaluation
Integration & Operational	Runtime stability	Benchmarking activities	Repeated execution analysis
Organizational & Contextual	OSS maturity	Internal document study	OpenSSF Scorecard analysis
Organizational & Contextual	License compatibility	Internal policies	License inspection
Organizational & Contextual	Usability	Practitioner feedback	Hands-on PoC evaluation

evaluated APIs. The study applied non-parametric statistical methods recommended in fuzzing evaluation research. Friedman tests were used to determine whether statistically significant differences existed between candidate tools across benchmark scenarios, while Wilcoxon signed-rank tests were used for pairwise comparisons of observed performance differences.

The empirical benchmarking results showed substantial differences between candidate tools. RESTler demonstrated strong performance in several benchmark scenarios, especially in detection capability and API exploration depth. However, the tool also exhibited limitations related to setup complexity, execution management, and integration effort. Schemathesis provided comparatively strong automation support and ease of execution, but demonstrated lower consistency in coverage and fault detection across evaluated APIs. EvoMaster achieved competitive coverage and fault-detection performance while at the same time providing stronger operational characteristics with respect to reproducibility, integration support and maintainability.

5.4 Organizational and Operational Evaluation

Beyond benchmarking performance, candidate tools were evaluated with respect to operational feasibility and organizational suitability. Open source maturity and security posture were assessed through documentation review, analysis of maintenance activity, inspection of issue-management practices and OpenSSF Scorecard analysis.

The evaluation revealed substantial variation in project maturity and sustainability between candidate tools. Some tools demonstrated strong technical performance, but lacked operational maturity, exhibiting incomplete documentation, unstable releases, or limited maintenance activity. In several cases, con-

Table 3. Synthesized evaluation results across technical, integration and operational, and organizational dimensions.

Evaluation Aspect	RESTler	Schemathesis	EvoMaster
Fault Detection	High	Medium	Medium–High
TCL Coverage	High	Medium	High
Integration & Automation	Medium	High	High
Operational Complexity	High	Low	Medium
OSS Maturity & Maintainability	Medium	Medium	High
Final Outcome	Rejected	Rejected	Selected

figuration complexity and unclear operational workflows introduced additional barriers for long term industrial adoption.

To further evaluate practical operational feasibility, a proof of concept integration was implemented using EvoMaster, which was selected as the most suitable candidate based on the synthesized evaluation results. Practitioners from development, QA, and security teams interacted with the proof of concept solution and provided qualitative feedback regarding usability, learning effort, transparency of generated tests, operational stability, and CI/CD integration feasibility. The practitioner feedback revealed several concerns that were not fully visible through isolated benchmarking metrics alone. In particular, ease of setup, reproducibility of executions, maintainability of generated artifacts, and integration into existing testing workflows were identified as decisive practical adoption factors.

Table 3 summarizes the synthesized evaluation results across technical, operational, and organizational dimensions. The labels *High*, *Medium*, and *Low* indicate each tool's relative fit with the case specific requirements and with other candidates.

5.5 Result Synthesis and Decision Support

The application of CAFE enabled synthesis of heterogeneous evidence across framework dimensions. The evaluation demonstrated that the technically strongest performing tools were not necessarily the most suitable for industrial adoption. While RESTler achieved strong benchmarking performance, operational limitations and integration complexity reduced its practical suitability within the industrial environment. Schemathesis provided advantages regarding execution simplicity and automation support, but demonstrated weaker technical consistency across evaluated APIs. EvoMaster, while not always outperforming competing tools in isolated benchmarking metrics, provided the strongest overall balance across technical, operational, and organizational dimensions.

As a result, the final selection decision balanced technical effectiveness against integration feasibility, operational sustainability, maintainability, security posture and organizational compatibility. The case study therefore illustrates how CAFE supports context-aware technology evaluation by structuring and synthesizing multiple forms of empirical and qualitative evidence into a unified industrial decision support process.

The industrial application produced three principal lessons. **Context must precede assessment**, as the evaluation requirements need to be derived from organizational constraints and stakeholder needs rather than adopted directly from benchmarks. **Different requirements require different evidence**, e.g., technical benchmarking was necessary but insufficient. Subsequent PoC integration assessment, documentation and security reviews, and practitioner feedback revealed adoption barriers that performance measurements did not capture. **Selection requires explicit trade-off analysis** as the technically strongest candidate is not necessarily the most suitable for adoption. Integration feasibility, maintainability, security posture, and organizational compatibility materially affected the final decision.

5.6 Threats to Validity

The primary contribution of this paper is the CAFE framework, which is a design artifact, and as such "field-tested and grounded exemplars of how a problem can be solved" [24]. Thus, the main validation of *the CAFE framework* is assessed through the case study and its integrated analytical reflection and design iterations, reported above.

Design validation was conducted following a Design Science Research approach, under which the three CAFE dimensions and their associated requirements were iteratively developed and refined using literature and evidence from an industrial web API fuzzing tool selection project in an embedded systems organization. The case included stakeholder interviews, focus groups, benchmarking, proof of concept integration, documentation and security reviews, and practitioner feedback. This triangulation supports the relevance and coverage of the dimensions in the studied context. However, since the empirical evidence was obtained from a single industrial case, a future application domain may identify an evaluation concern that is not adequately represented by the current three dimensions. Such concern may require an additional requirement within an existing dimension or, if it cannot be classified consistently, an additional dimension.

Regarding the validity of the *use* of the CAFE framework, we discuss it in terms of construct, internal, reliability and external validity, for the study being conducted under the pragmatism paradigm [25].

Construct validity concerns whether indicators adequately measure the abstract constructs they are intended to represent [27]. In CAFE, construct validity considers whether CAFE's three dimensions and their elicited evaluation requirements adequately represent the domain of technology evaluation, and whether the selected assessment methods provide appropriate evidence for those

requirements. CAFE structures the evaluation across the Technical, Integration and Operational, and Organizational and Contextual dimensions. Within each dimension, requirements were elicited from documented sources and assessed using the methods shown in Table 2. For example, OSS maturity is an Organizational and Contextual requirement elicited from an internal document study and assessed through OpenSSF Scorecard analysis. A threat arises if the three dimensions omit a relevant category of evaluation, if a requirement does not adequately represent the concern it is intended to address, or if its assessment method does not provide appropriate evidence for that requirement. **Internal validity** concerns case-effect relationships, i.e. whether an observed effect actually depends on the studied factors. When using the CAFE framework, the outcomes depend on assumed cause-effect relationships in the underpinning literature, as well as the empirical observations in assessment phase. The empirical evaluation may be affected by environmental and methodological factors specific to the case.For example, benchmark settings may unintentionally favor certain tools over others. Mitigation actions include applying systematic benchmarking procedures across candidate tools, and documenting the evaluation process throughout the study to improve transparency, consistency, and reproducibility.

Reliability concerns the conclusions drawn from the collected data and evaluation results. The CAFE framework combines multiple forms of evidence, including empirical benchmarking, PoC integration assessment, documentation review, and practitioner feedback. This triangulation of quantitative and qualitative assessment methods reduces the risk of conclusions being derived from isolated metrics or single perspectives.

External validity concerns the generalizability of the findings. While the findings of the case study as such are not claimed to be directly generalizable, the overall framework structure is defined at a higher level of abstraction. Software intensive organizations operate within similar software ecosystems characterized by standardized development workflows, CI/CD and DevOps practices, integration constraints, organizational processes, open source governance, and legal or security related requirements. Consequently, we claim that the proposed framework would be applicable across a broader range of industrial contexts, while further cases are needed to assess whether the three specific dimensions are adequate for other domains.

6 Discussion

The proposed CAFE framework provides a structured approach to assessing a component's suitability based not only on its technical characteristics, but also on its alignment with the broader socio-technical environment. However, its effective application depends on the ability of practitioners to interpret and adapt evaluation criteria to their specific context. Thus, rather than prescribing a rigid process, CAFE serves as a flexible guide that supports systematic and context-aware decision making. The presented case study illustrates one instance of the framework in practice. While the specific requirements and outcomes

are context dependent, the overall approach is applicable to a wide range of technology evaluation scenarios.

The growing adoption of AI-enabled software engineering tools calls for context-aware evaluation, particularly as software intensive systems grow in complexity, incorporating CI/CD pipelines, cyber-physical components, and machine learning driven processes [3]. Benchmarks alone overlook data governance, privacy, intellectual property, integration costs, human oversight, vendor dependence, and evolving model behavior. CAFE structures these concerns across its technical, operational, and organizational dimensions. Since organizations cannot evaluate all alternatives exhaustively, selection requires informed compromises. In this setting, it becomes critical not only to assess candidate tools, but also to understand the risks associated with these compromises and their potential long term impact. Thus, context-aware evaluation is not merely a methodological choice, but a necessary capability. It enables organizations to account for their own processes, constraints, and existing assets when selecting new components. By supporting a multidimensional perspective, CAFE helps mitigate the risk of overlooking critical factors, enabling sustainable and informed technology adoption decisions.

7 Conclusion

The evaluation and selection of third-party software tools is a critical activity in industrial software development, yet it is often conducted with a primary focus on isolated technical metrics. This work shows that such an approach is insufficient in complex, software intensive organizations, where integration constraints, organizational factors, and contextual requirements significantly influence the suitability of candidate solutions.

We propose the Context-Aware Framework for Evaluation (CAFE), a multidimensional evaluation framework that integrates technical, integration, and organizational perspectives. The application of CAFE in an industrial case study demonstrated how a structured, context-aware approach reveals trade-offs that are not visible in performance based evaluation alone. Hence, CAFE supports more informed selection decisions aligned with real-world constraints.

The findings reinforce the view of software intensive organizations as socio-technical systems, where successful technology adoption depends not only on the properties of a tool, but also on its fit within existing processes, capabilities and constraints. In this context, evaluation becomes a process of balancing trade-offs rather than identifying a single optimal solution. This observation is particularly relevant to AI tools, whose suitability depends on both model capability and the environment in which the tool is introduced. The present study does not empirically validate CAFE for AI-tool selection, and hence, applying and refining it in such settings is therefore an important direction for future work. Future work should, thus, further refine the framework by adapting it to specific domains and technologies.

Disclosure of Interests The authors have no competing interests to declare that are relevant to the content of this article.

References

1. Adewumi, A., Misra, S., Omoregbe, N.A., Fernández-Sanz, L.: FOSSES: framework for open-source software evaluation and selection. Softw. Pract. Exp. **49**(5), 780–812 (2019). https://doi.org/10.1002/SPE.2682, `https://doi.org/10.1002/spe.2682`
2. Alemo, K., Orheim, F.: Efficient Fuzzing of Web APIs in Embedded Environments. Master's thesis, Lund University (2026)
3. Amershi, S., Begel, A., Bird, C., DeLine, R., Gall, H.C., Kamar, E., Nagappan, N., Nushi, B., Zimmermann, T.: Software engineering for machine learning: a case study. In: Proceedings of the 41st International Conference on Software Engineering: Software Engineering in Practice, ICSE (SEIP) 2019, Montreal, QC, Canada, May 25-31, 2019. pp. 291–300. IEEE/ACM (2019). https://doi.org/10.1109/ICSE-SEIP.2019.00042
4. Arcuri, A.: EvoMaster: Evolutionary Multi-context Automated System Test Generation. In: 11th International Conference on Software Testing, Verification and Validation (ICST). pp. 394–397. IEEE, Los Alamitos, CA, USA (Apr 2018). https://doi.org/10.1109/ICST.2018.00046
5. Atlidakis, V., Godefroid, P., Polishchuk, M.: RESTler: stateful REST API fuzzing. In: Proceedings of the 41st International Conference on Software Engineering. p. 748–758. ICSE '19, IEEE (2019). https://doi.org/10.1109/ICSE.2019.00083
6. Bass, L., Clements, P., Kazman, R.: Software Architecture in Practice. Addison-Wesley, 3rd edn. (2012)
7. Bjarnason, E., Åberg, P., Ali, N.B.: Software selection in large-scale software engineering: A model and criteria based on interactive rapid reviews. Empir. Softw. Eng. **28**(2), 51 (2023). https://doi.org/10.1007/S10664-023-10288-W
8. Cataldo, M., Wagstrom, P., Herbsleb, J.D., Carley, K.M.: Identification of coordination requirements: implications for the design of collaboration and awareness tools. In: Proceedings of the 2006 ACM Conference on Computer Supported Cooperative Work, CSCW 2006, Banff, Alberta, Canada, November 4-8, 2006. pp. 353–362. ACM (2006). https://doi.org/10.1145/1180875.1180929
9. Conway, M.E.: How do committees invent? Datamation **14**(4), 28–31 (1968)
10. Corradini, D., Pasqua, M., Ceccato, M.: RESTgym: A flexible infrastructure for empirical assessment of automated REST API testing tools. In: 2025 IEEE Conference on Software Testing, Verification and Validation (ICST). pp. 757–761 (2025). https://doi.org/10.1109/ICST62969.2025.10988956
11. Corradini, D., Zampieri, A., Pasqua, M., Ceccato, M.: Restats: A test coverage tool for RESTful APIs. In: International Conference on Software Maintenance and Evolution (ICSME). pp. 594–598 (2021). https://doi.org/10.1109/ICSME52107.2021.00063
12. Davila, N., Wiese, I., Steinmacher, I., da Silva, L.L., Kawamoto, A., Favaro, G.J.P., Nunes, I.: An industry case study on adoption of AI-based programming assistants. In: Proc. 46th International Conference on Software Engineering: Software Engineering in Practice. pp. 92–102. ICSE-SEIP, ACM, Lisbon, Portugal (2024). https://doi.org/10.1145/3639477.3643648

13. Golmohammadi, A., Zhang, M., Arcuri, A.: Testing RESTful APIs: A survey. ACM Trans. Softw. Eng. Methodol. **33**(1) (Nov 2023). https://doi.org/10.1145/3617175
14. Hanine, M., Boutkhoum, O., Agouti, T., Tikniouine, A.: Application of an integrated multi-criteria decision making ahp-topsis methodology for etl software selection. SpringerPlus **5**, 263 (Mar 2016). https://doi.org/10.1186/s40064-016-1888-z
15. Hatfield-Dodds, Z., Dygalo, D.: Deriving semantics-aware fuzzers from web api schemas. In: Proceedings of the ACM/IEEE 44th International Conference on Software Engineering: Companion Proceedings. p. 345–346. ICSE '22, Association for Computing Machinery, New York, NY, USA (2022). https://doi.org/10.1145/3510454.3528637
16. Jadhav, A.S., Sonar, R.M.: Evaluating and selecting software packages: A review. Inf. Softw. Technol. **51**(3), 555–563 (2009). https://doi.org/10.1016/J.INFSOF.2008.09.003
17. Kazman, R., Klein, M., Clements, P.: ATAM: Method for architecture evaluation. Tech. Rep. CMU/SEI-2000-TR-004, Software Engineering Institute, Carnegie Mellon University (Aug 2000), `https://www.sei.cmu.edu/library/atam-method-for-architecture-evaluation/`
18. Kitchenham, B.A.: A methodology for evaluating software engineering methods and tools. In: Experimental Software Engineering Issues: Critical Assessment and Future Directions, International Workshop Dagstuhl Castle, Germany, September 14-18, 1992, Proceedings. Lecture Notes in Computer Science, vol. 706, pp. 121–124. Springer (1992). https://doi.org/10.1007/3-540-57092-6_110
19. Lehman, M.M.: Programs, life cycles, and laws of software evolution. Proceedings of the IEEE **68**(9), 1060–1076 (1980). https://doi.org/10.1109/PROC.1980.11805
20. Manès, V.J.M., Han, H., Han, C., Cha, S.K., Egele, M., Schwartz, E.J., Woo, M.: The art, science, and engineering of fuzzing: A survey. IEEE Trans. Software Eng. **47**(11), 2312–2331 (2021). https://doi.org/10.1109/TSE.2019.2946563
21. Martin-Lopez, A., Segura, S., Ruiz-Cortés, A.: Test coverage criteria for RESTful web APIs. In: Vos, T.E.J., Prasetya, W., Getir, S. (eds.) Proceedings of the 10th ACM SIGSOFT International Workshop on Automating TEST Case Design, Selection, and Evaluation, A-TEST@ESEC/SIGSOFT FSE 2019, Tallinn, Estonia, August 16-17, 2019. pp. 15–21. ACM (2019). https://doi.org/10.1145/3340433.3342822
22. OpenSSF: OpenSSF Scorecard: Security health metrics for open source. `https://scorecard.dev` (2026), accessed: 2026-05-26
23. Orucevic-Alagic, A., Höst, M.: A two phase case study on implementation of open source development practices within a company setting. In: Proceedings Software Engineering and Knowledge Engineering (SEKE 2016). pp. 63–70. Knowledge Systems Institute (KSI) (Jun 2016). https://doi.org/10.18293/SEKE2016-088
24. Runeson, P., Engström, E., Storey, M.: The design science paradigm as a frame for empirical software engineering. In: Contemporary Empirical Methods in Software Engineering, pp. 127–147. Springer (2020). https://doi.org/10.1007/978-3-030-32489-6_5
25. Runeson, P., Höst, M., Rainer, A., Regnell, B.: Case Study Research in Software Engineering – Guidelines and Examples. Wiley (2012). https://doi.org/10.1002/9781118181034
26. Schloegel, M., Bars, N., Schiller, N., Bernhard, L., Scharnowski, T., Crump, A., Ebrahim, A.A., Bissantz, N., Muench, M., Holz, T.: Sok: Prudent evaluation practices for fuzzing. In: IEEE Symposium on Security and Privacy, SP 2024, San Francisco, CA, USA, May 19-23, 2024. pp. 1974–1993. IEEE (2024)

27. Sjøberg, D.I.K., Bergersen, G.R.: Construct validity in software engineering. IEEE Transactions on Software Engineering **49**(3), 1374–1396 (2023). https://doi.org/10.1109/tse.2022.3176725
28. Venkatesh, V., Davis, F.D.: A theoretical extension of the technology acceptance model: Four longitudinal field studies. Management science **46**(2), 186–204 (2000). https://doi.org/10.1287/mnsc.46.2.186.11926