From Ad Hoc to Repeatable: A Knowledge-centric Framework for Introducing API Performance Testing

Inga Zilinskiene
Mykolas Romeris University, Vilnius, Lithuania

inga.zilinskiene@mruni.eu

Abstract: API performance testing is often introduced informally and remains dependent on individual expertise, which limits reuse and makes results difficult to repeat. This paper presents a lightweight framework and repository approach. It helps organizations establish a repeatable API performance testing process and capture knowledge for reuse. Using a Design Science Research (DSR) approach, a Knowledge-Centric API Performance Testing Framework with six components was developed: Scope & SLOs, Environment Setup, Data Preparation, Execution & Metrics, Analysis & Reporting, and Knowledge Capture & Sharing. A key distinguishing feature of the framework is its explicit inclusion of non-technical roles (e.g., Business Analysts and Product Owners), enabling them to contribute to performance testing through structured involvement in scope definition, governance, and knowledge capture. The framework was applied in a global IT organization during an initiative to validate API performance after a data source change. The case shows that a structured workflow and shared artefacts reduce repeated effort and support cross-functional collaboration. The paper contributes (1) a practical framework for organizations introducing API performance testing from scratch and (2) a knowledge management framing that treats performance testing outputs as reusable organizational assets.

Keywords: API performance testing, Knowledge management, Organizational learning, Framework, Design science research

1. Introduction

1.1 Problem and Motivation

In modern cloud-native and microservices-based architectures, APIs are critical components that connect systems, enable integrations, and directly affect user experience and business continuity. If APIs fail or degrade, business operations may be disrupted. API performance testing requires technical expertise, clear rules, and continuous organisational learning. Team members often develop personal know-how. Without structured documentation and shared repositories, this knowledge remains tacit. It is often lost when roles change or projects end. As a result, performance bottlenecks are often discovered only in production. This can lead to instability, missed service targets, and user dissatisfaction. In addition, the lack of reusable scripts, benchmarks, and reporting standards forces each initiative to start from zero. This leads to repeated mistakes and unnecessary effort.

Research literature mainly treats API performance testing as a technical activity. It gives limited attention to the knowledge management challenges behind fragmented practices. There is little guidance on how organizations with no established performance testing practice can introduce a repeatable approach and retain the resulting knowledge so it can be reused over time. Therefore, this study addresses the following research question: “How can an organization with no established API performance testing practice introduce a repeatable testing process and retain the resulting knowledge for reuse?”

1.2 Objectives and Contribution

The proposed framework addresses inconsistent and person-dependent testing practices. It introduces a structured and repeatable process that stabilises test conditions and produces predictable metrics. It reduces knowledge loss by treating performance testing as a knowledge-creation activity. Insights, scripts, and decisions are captured and reused across teams. The framework reduces role fragmentation by defining clear responsibilities across teams. This strengthens performance ownership and accountability. It also links performance testing with organisational governance. Results are integrated into SLO monitoring, risk assessment, and continuous improvement.

The framework was designed using the Design Science Research (DSR) paradigm, which focuses on solving real-world problems through the creation and iterative refinement of useful artefacts. Applying DSR ensures methodological rigor, problem relevance, and the integration of theoretical and empirical knowledge throughout the framework’s development.

The contributions of this paper are twofold:

- Practical contribution: a lightweight, structured methodology that guides teams in establishing performance testing from scratch, reducing ambiguity, and institutionalizing learning.
- Theoretical contribution: a knowledge management perspective that reconceptualizes API performance testing as an organizational learning process, transforming a technical task into a reusable knowledge asset.

The remainder of this paper is organized as follows. Section 2 reviews related work on API performance testing, knowledge management, and the Design Science Research paradigm. Section 3 presents the proposed framework and its six components. Section 4 describes the demonstration through a case study in a global IT organization. Section 5 provides evaluation results and lessons learned, followed by conclusions and future research directions in Section 6.

2. Related Work and Theoretical Background

2.1 Performance Testing in Complex API Landscapes

API performance testing in modern cloud-native systems is inherently complex and requires explicit scoping, planning, and coordination. Introducing this practice from scratch requires a shared and structured understanding. Without it, teams face blind spots and duplicated effort. API performance testing is complex. This complexity comes from technical, data, governance, and organisational factors. The review synthesised recurring themes related to performance testing practices. To synthesise prior research, studies published between 2015 and 2025 were reviewed across major databases (IEEE Xplore, Scopus, ACM DL), focusing on API and microservices performance testing. The most frequently cited challenges and corresponding solution patterns are summarised in Table 1.

Table 1: API performance testing: complexity, challenges, solutions and key references

Factors	Complexity of API Performance Testing	Challenges Still Identified in Literature	How to Transform Complexity into Clarity	Key References
Technical	Distributed architectures (microservices/SOA/cloud-native) with many interacting APIs.	End-to-end effects and service dependencies make endpoint-level tests insufficient.	Integrated methodologies that combine performance + reliability testing in DevOps pipelines (e.g., MIPaRT).	(Camilli et al., 2022); (Mingxuan Hui, 2025)
	Dynamic, elastic environments (containers/serverless/public cloud).	High result variability and limited reproducibility across runs.	Reproducible benchmarking/reporting practices and CI-based automation with clear provenance.	(Papadopoulos et al., 2021; Straesser et al., 2023)
Data-related	Data realism (synthetic vs. production-like).	Privacy/GDPR and lack of representative datasets undermine validity.	Metric standardization (e.g., p95/p99 latency, error %) and dashboards; causal analysis across services.	(Silva et al., 2022)
	Variable workloads (bursts/peaks/long sessions).	Capturing realistic user behavior is costly and effort-intensive.	Workload modeling and operational profiles to exercise peak and steady-state regimes.	(Feitelson, 2015) (Muhlbauer et al., 2023)
Governance	Lack of standardized processes, unclear ownership	Performance testing remains informal, inconsistent, and dependent on individual initiative rather than defined processes.	Define formal roles, responsibilities, and process steps; integrate performance testing into release and change management governance.	(Waseem et al., 2021); (Leite et al., 2021)
	Weak linkage to SLAs, risk, and decision-making	Performance insights rarely reach decision-makers; no structured escalation or acceptance criteria.	Introduce SLA-aligned performance thresholds, risk-based dashboards, governance gates for releases.	(Papadopoulos et al., 2021); (Eismann et al., 2020)

Factors	Complexity of API Performance Testing	Challenges Still Identified in Literature	How to Transform Complexity into Clarity	Key References
	Lack of metric governance and reporting standards	Fragmented reporting, incomparable metrics across environments and teams.	Implement standardized metrics (latency percentiles, throughput, error budgets), reporting templates, and governed telemetry.	(Thalheim et al., 2017); (Waseem et al., 2021)
	Cross-layer metrics (latency, throughput, errors) at scale	Fragmented metrics and weak governance impede analysis.	Metric standardization (e.g., p95/p99 latency, error %) and dashboards; causal analysis across services.	(Eismann et al., 2020; Thalheim et al., 2017; Waseem et al., 2021)
Organizational	"QA-only" ownership and silos	Limited collaboration/governance ; business not involved; low shared understanding of API performance testing	Cross-functional structures and shared telemetry/dashboards, as observed in industry studies of microservices delivery.	(Waseem et al., 2021) (Leite et al., 2021)
	Knowledge fragmentation (tacit results; lessons not institutionalized)	Repeated mistakes and low reuse.	Capturing tacit knowledge into repositories, reusable scripts	(Alves et al., 2024; De Souza et al., 2015; Nogeste & Walker, 2006; Wnuk & Garrepalli, 2018)

The literature highlights several recurring themes. Technically, distributed microservices architectures and dynamic cloud environments introduce variability and make endpoint-level testing insufficient. From a data perspective, realistic workloads and production-like datasets are difficult to prepare and reproduce. Governance challenges include unclear ownership and fragmented metric reporting. Organizationally, performance testing often remains siloed within QA teams, and knowledge generated during testing is rarely shared across the organization. The existing gaps justify the need for a framework that unifies technical, governance, and learning perspectives. Design Science Research (DSR) is therefore an appropriate paradigm for developing such a framework.

2.2 Knowledge Management and Organizational Learning in Testing

Studies show that applying KM to software testing improves effectiveness, efficiency, and reuse. Many testing activities depend heavily on tacit expertise that often remains undocumented (De Souza et al., 2015; Alves et al., 2024). When such knowledge is not captured, teams repeatedly face the same issues, resulting in duplicated effort, weak knowledge transfer, and "reinventing the wheel" across projects (Nogeste & Walker, 2006; Wnuk & Garrepalli, 2018). Prior studies indicate that KM mechanisms-such as reusable test assets, repositories, shared templates, and structured lessons-learned practices-significantly improve test efficiency and reduce waste in large-scale software development (Alves et al., 2024; Leite et al., 2021). In addition, reuse of historical insights, patterns, and root-cause analyses enhances test coverage and quality (Waseem et al., 2021).

Applied specifically to performance testing, KM practices enable the systematic documentation of test scripts, environment configurations, performance thresholds, and result interpretations. This shifts knowledge from individual memory to shared artefacts. Over time, this codification improves consistency, communication, and decision-making. From a KM perspective, this transition - from tacit, person-dependent practices to explicit, reusable knowledge assets - can be understood as organizational learning.

2.3 Design Science Research as the Chosen Paradigm

Design Science Research (DSR) focuses on developing and evaluating artefacts that address real-world problems (Hevner et al., 2004; Aparicio et al., 2023). It is particularly suitable when the goal is to design structured solutions-such as frameworks or methods-that can be applied in practice while contributing to theory. DSR emphasizes iterative cycles of problem identification, artefact design, demonstration, and evaluation.

In this study, DSR was selected because the research objective was not only to analyze API performance testing practices, but to design and evaluate a practical framework that organizations can use to introduce

testing from scratch. The Knowledge-Centric API Performance Testing Framework represents the primary artefact of this study. It was developed, applied in a real organizational context, and refined based on practical observations.

Using DSR ensures that the framework is grounded in both empirical practice and relevant literature. This approach supports the dual aim of the study: providing a usable solution for organisations and contributing a structured knowledge management perspective to API performance testing research.

2.4 Research Gap and Positioning

Existing API performance testing frameworks predominantly focus on technical execution by developers and QA engineers. This study extends prior work by explicitly incorporating non-technical roles (e.g., Business Analysts and Product Owners) into the performance testing lifecycle. The framework provides these roles with structured visibility and responsibility across planning, governance, and knowledge capture activities, thereby repositioning performance testing as a cross-functional organisational process rather than a purely technical task.

Table 2: Position of a Knowledge-Centric Framework through the lenses of the existing research

Aspect	Existing Performance Testing Frameworks	KM in Software Testing	A Knowledge-Centric Framework
Focus	Technical execution	Knowledge reuse	Integrated technical + governance + KM
Knowledge handling	Implicit / weak	General (not performance-specific)	Embedded in testing lifecycle
Reusability	Limited	Conceptual	Operational (scripts, SLOs, dashboards)
Governance	Weak	Not central	Core component
Target context	Mature orgs	Generic	Low-maturity orgs

This study extends KM literature by demonstrating how knowledge capture can be embedded into operational processes, rather than implemented as a parallel activity:

- With the tacit-to-explicit knowledge transformation perspective, it embeds knowledge codification directly into performance testing workflows rather than treating it as a separate KM activity.
- It integrates feedback loops (Plan–Execute–Reflect–Refine), enabling continuous refinement of testing practices.
- It represents a Level 2 design artefact (method) that contributes prescriptive guidance for organisations introducing performance testing in low-maturity contexts.

3. Research Methodology

In line with DSR principles, the research process in this study comprised the following stages:

- **Problem Identification and Motivation.** The initial step involved identifying the practical problem: API performance testing being conducted in an ad hoc, person-dependent manner with limited governance and knowledge reuse. Empirical observations in a global IT organization provided additional evidence of fragmented practices, tacit knowledge, and weak performance ownership.
- **Definition of Objectives for a Solution.** Based on the problem analysis, the study defined four main objectives for the solution:

1. to establish a structured and repeatable API performance testing process;
2. to transform testing into a knowledge-creation and knowledge-sharing activity followed by extended knowledge repository;
3. to strengthen cross-functional collaboration and performance ownership; and
4. to link testing outcomes with governance mechanisms such as SLOs, risk management, and continuous improvement.

- **Design and Development of the Artefact.** The framework was designed as the main artefact of the study. Based on the literature and case context, three core dimensions were identified: technical, governance, and learning. These dimensions guided the design of six framework components: Scope & SLOs, Environment Setup, Data Preparation, Execution & Metrics, Analysis & Reporting, and

Knowledge Capture & Sharing. The framework's three-layer architecture and governance cycle (Plan-Execute-Reflect-Refine) were iteratively refined using practitioner feedback and insights from early testing activities.

- **Demonstration in a Real-World Context.** The framework was applied in a global IT organization during an enterprise initiative to validate API performance after a critical data source change.
- **Evaluation and Validation.** In accordance with early-stage DSR, evaluation relied on qualitative and observational methods rather than statistical experiments.
- **Communication of Results.** This paper presents the DSR outcomes in four parts: the problem context, the framework design, the case study, and the evaluation with conclusions and future work.

By following this DSR process, the study ensures that the proposed framework is both practically relevant and theoretically grounded. The artefact responds directly to an observed organizational problem while contributing a knowledge-centric perspective on API performance testing to the information systems and software engineering literature.

4. Design and Development of the Knowledge-Centric API Performance Testing Framework

4.1 Purpose and Design Logic

The purpose of the framework is to help organizations transition from ad hoc, fragmented API performance testing practices to a structured, transparent, and repeatable process. It integrates technical activities with knowledge management principles. It reframes performance testing not only as a technical activity but as a knowledge creation process. This helps organisations reduce complexity, manage risks, and build long-term capability. Without such structure, testing remains tacit, person-dependent, and error-prone - leading to repeated mistakes and hidden risks.

4.2 Design Principles

The framework was developed following Design Science Research (DSR) principles, ensuring alignment between the identified problem, the defined objectives, and the structure of the artefact. Iterative build-evaluate-refine cycles were used to adjust the framework based on literature insights and practitioner feedback.

4.3 Design Requirements

Analysis of literature and case observations revealed three necessary dimensions for introducing a sustainable performance testing practice:

- Technical Layer: *Components:* Environment Setup; Data Preparation; Execution & Metrics, *Purpose:* Ensure reliable and reproducible test execution.
- Governance Layer: *Components:* Scope & SLOs; Analysis & Reporting, *Purpose:* Define performance targets, decision criteria, and accountability.
- Organizational Learning Layer: *Component:* Knowledge Capture & Sharing, *Purpose:* Convert testing outcomes into reusable organizational artefacts.

This three-layer architecture treats performance testing as an integrated organisational process rather than an isolated technical activity. The technical layer enables consistent execution, the governance layer structures decision-making, and the learning layer supports reuse and long-term capability development.

The framework operates through a simple **governance cycle**:

- Plan - Define scope, SLOs, and required data.
- Execute - Run tests and collect metrics.
- Reflect - Analyze results and identify insights.
- Refine - Update documentation, scripts, and repository artefacts.

Two feedback mechanisms connect the layers:

- Upward flow: test results and lessons learned are codified into shared artefacts.
- Downward flow: accumulated knowledge informs future planning and threshold definition.

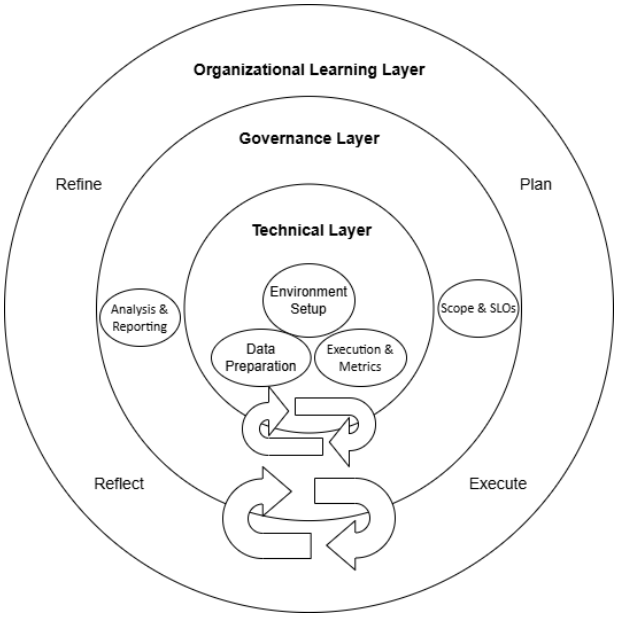

Figure 1: The Framework: Knowledge-Centric API Performance Testing

This cycle supports continuous improvement while reinforcing repeatability and knowledge reuse.

4.4 Expected Outcomes

Application of the framework produces outcomes at two levels:

Technical outcomes	Organizational outcomes
1. Reproducible test execution	1. Reusable scripts and documentation
2. Stable environment configuration	2. Reduced onboarding effort
3. Standardized metric definitions	3. Clearer cross-functional roles
4. Clear performance targets (SLOs)	4. Improved confidence in performance-related decisions

By embedding structured knowledge capture into performance testing, the framework supports the transition from one-off validation activities to a repeatable organizational capability.

5. Case study: Application of the Framework in an Enterprise API Performance Initiative

The case study describes the organisation's first structured attempt to introduce API performance testing. The goal was to move from an ad hoc activity to a governed, knowledge-driven process. The framework was derived and refined through this implementation.

5.1 The Context

The Knowledge-Centric API Performance Testing Framework was applied in a global IT organization that had no prior structured approach to API performance testing. No shared procedures, repositories, metrics, or documentation existed. The implementation took place during an enterprise initiative to validate API performance following a critical data source change. As the organization transitioned toward API-based integrations, ensuring that system changes did not degrade performance became increasingly important.

5.2 Framework Application

Each framework component guided the team through a structured sequence - from defining scope to capturing reusable knowledge. Business Analysts and Product Owners contributed to scope definition, SLO formulation, and interpretation of performance results, ensuring alignment with business priorities and governance requirements.

Table 3: The framework components and their details

Framework Component	Key artefacts produced	Primary roles involved	Example from Case
1. Scope & SLOs	Performance testing strategy; endpoint selection criteria; SLO definition template	PO, BA, QA, QA Lead, Security	Selected endpoints by call volume and response time; executed Normal/Peak/Soak tests; used average P95 across runs + 20% buffer to set initial SLOs.
2. Environment Setup	Documented production-like test environment	Developers, DevOps, QA, Infrastructure, Security	Database resources aligned with production capacity and configuration
3. Data Preparation	Data generation, validation procedures	PO, BA, QA, DevOps, DBA, Legal, Security	Realistic datasets prepared while preserving performance-relevant patterns
4. Execution & Metrics	Standardised test execution plan	DevOps, QA	Core metrics defined: average latency, P95/P99 latency, max latency, throughput, error rate
5. Analysis & Reporting	Structured reporting templates	QA, DevOps, DB teams	Two reporting types: SLO definition (pre- and post-migration) and SLO validation
6. Knowledge Capture & Sharing	Central repository of artefacts	Infrastructure, QA, cross-functional teams	Scripts, configurations, logs, and lessons learned stored for reuse

Through this structured approach, testing moved from irregular and undocumented activities to a reproducible process. Performance discussions expanded beyond QA and DevOps, involving developers, database specialists, and business analysts.

5.3 The Outputs

The framework produced a set of tangible artefacts that did not previously exist in the organization:

- Defined scope and SLO documents
- Standardized test scenarios and execution plans
- Realistic test datasets and data anonymization procedures
- Metrics dashboards (latency, throughput, errors, resource usage)
- Identified performance bottlenecks and improvement recommendations
- Lessons-learned documentation and troubleshooting notes
- Reusable JMeter scripts and environment configurations
- A central knowledge repository for performance testing

These artefacts represent explicit organizational knowledge that can be reused across projects.

5.4 The Observed Effects

The application of the framework resulted in observable changes at both technical and organizational levels.

Technical improvements

- Test results became reproducible and comparable across runs.
- Environment configuration was stabilized and documented.
- Metric definitions were standardized and governed.

Organizational improvements

- The first documented set of SLOs for critical endpoints was established.
- Cross-team collaboration increased.
- Tacit know-how was captured explicitly for the first time.
- A reusable set of scripts reduced setup time for new tests.
- The knowledge repository became a shared reference point for decisions.

Performance testing shifted from an individual activity to a coordinated, multi-role process supported by shared artefacts.

5.5 Lessons Learned

Several lessons emerged from introducing performance testing in a low-maturity context:

- Start small and document consistently. Even small testing activities generate valuable knowledge when captured systematically.
- Treat testing artefacts as reusable assets. Scripts, dashboards, and documentation quickly become core organizational resources.
- Integrate governance early. Without a structured Plan-Execute-Reflect-Refine cycle, teams risk reverting to ad hoc practices.
- Allow flexibility within structure. Components can be adapted to local needs, provided the overall process remains consistent.

In addition to the positive outcomes, several unexpected challenges were observed. First, despite structured documentation, initial adoption was uneven, as some technical team members perceived knowledge capture activities as overhead. This required additional alignment and communication efforts. Second, early SLO definitions proved unstable and required multiple iterations, highlighting the difficulty of translating empirical performance data into governance thresholds.

6. Evaluation and Validation

The evaluation assessed whether the framework achieved its objectives: improving repeatability, enabling knowledge reuse, strengthening collaboration, and supporting decision-making. In Design Science Research (DSR), evaluation focuses on assessing the usefulness and applicability of the artefact in addressing the identified problem (Hevner et al., 2004).

Because the framework was introduced in an organization with no prior structured testing practice, qualitative evaluation methods were appropriate. The assessment used observations, artefact inspection, and practitioner feedback. The focus was on practical use and observable process changes, not statistical measurement. (Table 4)

Table 4: The evaluation criteria and corresponding evidence

Objective	Before	After	Evaluation Focus	Evidence from case
1. Repeatable process	No structured testing process, ad hoc	Defined workflow (Plan–Execute–Reflect–Refine)	Existence of a structured workflow that can be reused	Strategy template, execution checklist, defined test types (Normal/Peak/Soak)
2. Knowledge capture and reuse	Tacit, undocumented	Repository	Artefacts documented and accessible for future use	Repository containing scripts, configurations, data preparation notes, and reporting templates
3. Cross-functional collaboration	QA-only	Cross-functional involvement (QA lead, architects)	Participation beyond QA	Involvement of DevOps, DB, Security, BA/PO in scoping, data preparation, and analysis
4. Decision support	Undefined responsibilities	Defined responsibilities across the teams and stakeholders	Results used in governance discussions	SLO definition and validation reports shared with stakeholders
5. Metric standardisation	Inconsistent	Standardized	Consistent interpretation and reporting of results	Defined latency percentiles (P95/P99), throughput, error rate, and structured reporting format

Following the objectives stated in the Section 1, the table 5 presents the results obtained:

Table 5: The objectives, evidence and achieved results

Objective	Evidence	Achieved
Repeatability	Structured templates, reusable scripts, execution workflows	Yes
Knowledge reuse	Repository with scripts, reports, and lessons learned	Partial
Collaboration & Decision support	Multi-role involvement across QA, DevOps, BA, PO, architects, team leads	Yes

The evaluation results indicate that all primary design requirements - repeatability, knowledge capture, collaboration, and governance integration - were addressed. While knowledge reuse remains partially achieved, the presence of a structured repository and reusable artefacts provides a foundation for future improvement. A key implication of this study is that performance testing should not be treated solely as a technical validation activity but as a cross-functional organisational capability supported by structured knowledge flows.

7. Conclusion, Limitations, and Future Work

This paper addressed the question of how an organization with no established API performance testing practice can introduce a repeatable process while retaining knowledge for reuse. It presented the Knowledge-Centric API Performance Testing Framework, which integrates technical execution, governance structure, and systematic knowledge capture. Developed using the Design Science Research (DSR) approach, the framework structures performance testing into six components: Scope & SLOs, Environment Setup, Data Preparation, Execution & Metrics, Analysis & Reporting, and Knowledge Capture & Sharing.

The case study showed that structured workflows and shared artefacts can replace ad hoc practices. This makes performance testing more repeatable and transparent. The findings show that embedding knowledge capture in testing supports reuse, cross-functional coordination, and consistent decision-making.

Several limitations must be acknowledged. First, the framework was derived and evaluated in a single organizational context, which limits generalizability. Additional applications across different industries and technical environments would strengthen external validity. Second, evaluation relied on qualitative evidence, including artefact inspection and practitioner observation. While appropriate for early-stage DSR, future research could complement these findings with quantitative performance indicators or comparative studies. Third, the case organization was at an early stage of testing maturity; further investigation is needed to assess applicability in highly automated or advanced DevOps settings.

Future research can extend the framework in three directions. First, replication in additional organizations would enable refinement and validation across contexts. Second, more detailed guidance on repository design, template standardization, and tool integration could support deeper operationalization. Third, expanding the framework toward continuous performance testing and performance engineering practices may contribute to a broader organizational model of reliability and learning.

Overall, the study contributes to a structured approach for introducing API performance testing in low-maturity contexts and frames performance testing outputs as reusable organizational knowledge assets rather than isolated technical results.

Ethics declaration: This research did not require formal ethical approval, as it did not involve human subjects, personal data, or sensitive information. The study was conducted using organisational processes and artefacts, ensuring no identifiable individuals or confidential data were disclosed.

AI declaration: ChatGPT was used to enhance the structure and refine the language of the paper.

References

Alves, D. dos S., Smek, D. J., Souza, É. F. de, Felizardo, K. R., & Vijaykumar, N. L. (2024). Updating a Systematic Literature Review on Knowledge Management Diagnostics in Software Development Organizations. *Proceedings of the XXIII Brazilian Symposium on Software Quality*, 125-135. https://doi.org/10.1145/3701625.3701652

Aparicio, J. T., Aparicio, M., & Costa, C. J. (2023). Design Science in Information Systems and Computing. *Lecture Notes in Networks and Systems*, *614 LNNS*. https://doi.org/10.1007/978-981-19-9331-2_35

Brocke, J. vom, Weber, M., & Grisold, T. (2022). Design science research of high practical relevance: dancing through space and time. *… : A Design Science Research …*.

Camilli, M., Guerriero, A., Janes, A., Russo, B., & Russo, S. (2022). Microservices Integrated Performance and Reliability Testing. *Proceedings - 3rd ACM/IEEE International Conference on Automation of Software Test, AST 2022*. https://doi.org/10.1145/3524481.3527233

De Souza, É. F., Falbo, R. D. A., & Vijaykumar, N. L. (2015). Knowledge management initiatives in software testing: A mapping study. *Information and Software Technology*, *57*(1). https://doi.org/10.1016/j.infsof.2014.05.016

Eismann, S., Bezemer, C. P., Shang, W., Okanović, D., & Van Hoorn, A. (2020). Microservices: A performance tester's dream or nightmare? *ICPE 2020 - Proceedings of the ACM/SPEC International Conference on Performance Engineering*. https://doi.org/10.1145/3358960.3379124

Feitelson, D. G. (2015). Workload modeling for computer systems performance evaluation. In *Workload Modeling for Computer Systems Performance Evaluation*. https://doi.org/10.1017/CBO9781139939690

Hevner, A. R., March, S. T., Park, J., & Ram, S. (2004). Design science in information systems research. *MIS Quarterly: Management Information Systems*, *28*(1). https://doi.org/10.2307/25148625

Leite, L., Pinto, G., Kon, F., & Meirelles, P. (2021). The organization of software teams in the quest for continuous delivery: A grounded theory approach. *Information and Software Technology*, *139*. https://doi.org/10.1016/j.infsof.2021.106672

Mingxuan Hui, L. W. (2025). Unveiling the microservices testing methods, challenges, solutions, and solutions gaps: A systematic mapping study. *Journal of Systems and Software*.

Muhlbauer, S., Sattler, F., Kaltenecker, C., Dorn, J., Apel, S., & Siegmund, N. (2023). Analysing the Impact of Workloads on Modeling the Performance of Configurable Software Systems. *Proceedings - International Conference on Software Engineering*. https://doi.org/10.1109/ICSE48619.2023.00176

Nogeste, K., & Walker, D. H. t. (2006). Using knowledge management to revise software-testing processes. *Journal of Workplace Learning*, *18*(1). https://doi.org/10.1108/13665620610641283

Papadopoulos, A. V., Versluis, L., Bauer, A., Herbst, N., Kistowski, J. Von, Ali-Eldin, A., Abad, C. L., Amaral, J. N., Tuma, P., & Iosup, A. (2021). Methodological Principles for Reproducible Performance Evaluation in Cloud Computing. *IEEE Transactions on Software Engineering*, *47*(8). https://doi.org/10.1109/TSE.2019.2927908

Silva, P., Gonçalves, C., Antunes, N., Curado, M., & Walek, B. (2022). Privacy risk assessment and privacy-preserving data monitoring. *Expert Systems with Applications*, *200*. https://doi.org/10.1016/j.eswa.2022.116867

Straesser, M., Eismann, S., Von Kistowski, J., Bauer, A., & Kounev, S. (2023). Autoscaler Evaluation and Configuration: A Practitioner's Guideline. *ICPE 2023 - Proceedings of the 2023 ACM/SPEC International Conference on Performance Engineering*. https://doi.org/10.1145/3578244.3583721

Thalheim, J., Rodrigues, A., Akkus, I. E., Bhatotia, P., Chen, R., Viswanath, B., Jiao, L., & Fetzer, C. (2017). Sieve: Actionable Insights from Monitored Metrics in Microservices. In *Middleware '17*.

Tuunanen, T., Winter, R., & vom Brocke, J. (2024). DEALING WITH COMPLEXITY IN DESIGN SCIENCE RESEARCH: A METHODOLOGY USING DESIGN ECHELONS1. *MIS Quarterly: Management Information Systems*, *48*(2), 427-458. https://doi.org/10.25300/MISQ/2023/16700

Waseem, M., Liang, P., Shahin, M., Di Salle, A., & Márquez, G. (2021). Design, monitoring, and testing of microservices systems: The practitioners' perspective. *Journal of Systems and Software*, *182*. https://doi.org/10.1016/j.jss.2021.111061

Wnuk, K., & Garrepalli, T. (2018). Knowledge management in software testing: A systematic snowball literature review. In *E-Informatica Software Engineering Journal* (Vol. 12, Issue 1). https://doi.org/10.5277/e-Inf180103