розробки. Натомість JMeter демонструє вищу продуктивність у сценаріях високого та пікового навантаження, підкреслюючи його здатність моделювати паралельні дії користувачів і здійснювати стрес-тестування застосунків у масштабі. Крім того, різниця у показниках продуктивності, виявлена під час експериментів, свідчить, що JMeter надає більш точну модель застосунків у випадках, коли взаємодія користувачів спричиняє затримки, що робить його особливо корисним для подієво-орієнтованих або сесійних систем. Отримані результати підкреслюють, що жоден інструмент не є універсально оптимальним, а вибір має залежати від конкретних цілей тестування та операційного контексту. Надаючи емпіричні докази сильних і слабких сторін обох інструментів, це дослідження пропонує практичні рекомендації для розробників і QA-інженерів у виборі не лише відповідного інструмента, але й у проєктуванні ефективних стратегій тестування для забезпечення надійності та масштабованості системи.

Ключові слова: *тестування продуктивності, тестування API, навантажувальне тестування, стабільність системи, Postman, JMeter, аналіз часу відгуку, перцентильні метрики, оцінка продуктивності, прийняття рішень, тестування програмного забезпечення*

UDC 004.05
DOI https://doi.org/10.36059/978-966-397-538-2-13

PERFORMANCE PERCENTILE ANALYSIS FOR API-BASED TESTING

Ph.D. S. Khlamov[1][0000-0001-9434-1081], **M. Mendielieva** [2][0009-0002-4282-3147],
Ph.D. O. Vovk[3][0000-0001-9072-1634], **T. Trunova**[4][0000-0003-2689-2679],
Yu. Teslenko[5][0009-0009-8349-3683]
Kharkiv National University of Radio Electronics, Ukraine
EMAIL: [1]sergii.khlamov@gmail.com, [2]mariia.mendielieva@nure.ua,
[3]oleksandr.vovk@nure.ua, [4]tetiana.trunova@nure.ua, [5]yuliia.teslenko@nure.ua

Abstract. *This paper focuses on the systematic analysis of performance testing metrics, with particular attention to comparing the behavior of two widely used tools, Postman and JMeter, in the context of public application programming interfaces (APIs). API performance testing plays a critical role in evaluating the responsiveness and reliability of modern software systems; however, one persistent challenge lies in relying solely on average response times. Mean values can be easily skewed by anomalous outliers, which often mask significant response delays and lead to an incomplete picture of system performance. To overcome this limitation, the present study emphasizes the use of percentile-based analysis, which provides a more accurate and user-centered indicator of performance for the majority of requests. The methodology involved comparing average response time*

values with percentile distributions, with a primary focus on the 90th, 95th, and 99th percentiles. Additionally, the percentage deviation of percentile values from the average was calculated, serving as a measure of stability for the testing tools. This approach enabled a more reliable evaluation of the tool's behavior under different workloads. The experimental findings revealed distinct advantages for each tool. Postman demonstrated favorable results under medium load conditions for Create and Delete requests. At the same time, JMeter proved more effective for Read (GET) and Update operations, where stability and predictability are critical in complex systems. To support practical application, the study also proposes a decision-making diagram that guides the allocation of test scenarios between the two tools, ultimately improving testing efficiency and ensuring more reliable API performance assessments.

Keywords: *Performance testing, API testing, load testing, system stability, Postman, JMeter, response time analysis, percentile metrics, performance evaluation, decision-making, software testing*

1 Introduction

One of the challenges in analyzing performance test results is that the average response time (ART) represents the typical response time a user will experience while using a software application. On the other hand, the ART performance testing metric does not take into consideration outliers and large deviations in response times [1], which can be caused by issues such as memory leaks, thread contention, Input/Output (I/O) bottlenecks, and long-running SQL queries. A system may have an acceptable ART, but some users may still face significant delays, especially during load spikes or stress conditions [2].

Percentile response time, as a performance testing metric, provides a more reliable representation of system behavior for most users, capturing significant delays that the average response time would miss. A percentile is the value below which a percentage of the response times are completed successfully [3].

The concept of considering slow requests that exceed the threshold is crucial in performance testing, as it helps identify potential performance issues in applications [4]. It often indicates that the system may begin to degrade under high loads, especially in microservice environments and cloud-native applications [5-7].

Furthermore, performance analysts can investigate the internal behavior of various slow and fast web requests in greater detail, and by clustering and comparing their execution patterns, identify the factors that cause specific requests to perform slowly or exhibit unexpected behavior [8]. Additionally, service level agreements (SLAs), as contracts between a service provider and a customer, are often written with a specification of the maximum response time for the majority of users, rather than the average response time of the system [9].

Despite the emergence of modern performance tools and technologies in the performance testing process, a challenge remains in successfully applying testing tools to test microservices and cloud-native application APIs [10, 11]. The distributed nature of these systems creates complex testing scenarios where services

interact across network boundaries with varying degrees of reliability and latency [10].

The interconnected nature of microservices creates numerous dependencies, which increase the testing surface area. This leads to the problem of prolonged testing times in cloud environments, where costs directly correlate with resource allocation. Thus, there is a challenge of resource limitations [11] that is important for organizations with limited budgets or restricted access to resources.

Additionally, in the field of microservices API testing, it's essential to find an approach that can effectively validate complex service interactions while maintaining test stability. Effective API management in microservice architectures demands both the application of general strategic methods and the integration of reliable engineering practices that maintain system stability and reliability [12, 13].

Selecting the appropriate performance tools can be a challenging task, but they should be aligned with the project's requirements and the development team's expertise [11, 14].

In relation to the decision-making process regarding the most suitable performance test tool, it is essential to consider its stability, which is related to its reliability and robustness in consistently performing functions over time, as outlined in the ISO/IEC 25010 standard [15].

Postman and JMeter are popular tools for performance testing. However, they have architectural differences and offer different options for load generation, work collaboration and reporting. These differences raise questions about the efficient allocation of test scenarios between two performance testing tools, which can be aligned with the project's specific requirements. The results will provide IT professionals with objective insights for creating a successful performance test strategy.

The purpose of this work is to study the approach to the distribution of test scenarios between two testing tools, Postman and JMeter, based on performance metrics such as ART and percentile. The results will provide IT professionals with objective insights for developing an effective performance testing strategy and enhancing the flexibility and adaptability of the testing process.

The chapter is organized to emphasize the role of percentile metrics in evaluating system responsiveness and reliability. It begins with an introduction that explains the limitations of traditional average-based analysis and the need for percentile-oriented approaches in API testing. The literature review section examines prior research and current practices in the use of percentiles across performance engineering. The methodology outlines the selected APIs, test scenarios, and statistical models applied. The results section presents percentile distributions under varying loads, while the discussion interprets their significance. The chapter concludes with recommendations for engineers and future research directions.

2 Literature review

API testing is essential for software development teams to ensure the performance, scalability, and security of large-scale systems that handle millions of transactions daily. API testing is mandatory in the Continuous Integration and Continuous Delivery (CI/CD) phases [16]. In modern continuous development, the back and front-end interaction typically happens via methods and functions that the back end offers directly via API. Testing both the front-end and back-end allows for fast application quality feedback and enables the CI/CD pipeline to start automatically and promptly.

Performance testing is a subset of automatic tests and is designed to evaluate an application's behavior under particular load conditions. Effective load testing incorporates diverse scenarios, including peak traffic simulations, sustained operation periods, and irregular usage spikes, to comprehensively assess API resilience across operational conditions that distributed systems commonly encounter [17].

Apache JMeter is a performance testing tool that allows users to perform load tests on various protocols and technologies. It is one of the most widely used open-source tools for performance testing, particularly in the domains of API testing and web applications. The ability to create and execute complex testing scenarios is one of JMeter's most essential features. JMeter tool is multithreaded and can simulate a large number of VUs, enabling the simulation of a heavy load by distributing tests across multiple machines.

JMeter supports integration with external services and tools such as CI/CD pipelines, monitoring systems, and third-party performance analysis platforms. Despite its many advantages, JMeter also has certain limitations, as noted by authors [18, 19]. One of the main disadvantages is its relatively high memory consumption, especially when running large-scale tests or simulating a large number of users.

Additionally, JMeter lacks advanced features, such as real-time monitoring and has a high learning curve for setting up and configuring distributed tests.

Postman is a platform for API development and testing that has emerged as a leading tool for API development, boasting a very user-friendly interface [20]. Postman is not an open-source tool, and it has a paid version. Postman tests can be executed manually using the GUI or run automatically using the Collection Runner or the Newman command-line tool. Additionally, Postman enables collaboration with teammates by organizing, sharing, and communicating work with APIs. According to authors [20, 21], Postman can be used for API performance testing with a desktop application.

Performance tests can be executed using one of four load profiles: Fixed (the maximum number of virtual users is used throughout the test), Ramp-up (VUs gradually increase from initial load to the maximum), Spike (VUs increase from base load to maximum, then drop back to base load), and Peak (VUs increase from base load to maximum, stabilize, and then return to base load).

It is essential to highlight that during performance test execution in Postman, each virtual user runs the requests in the specified order within a repeating loop. Performance test execution can be monitored in real-time through the Postman Summary tab, which provides a summary of performance metrics available in both tabular and graphical forms.

Thus, analyzing the research results in the reviewed author's works [22, 23], it is worth noting that there are some limitations to running performance tests in Postman. Firstly, a limited number of performance runs can be used each month at no additional cost. Secondly, the number of VUs in a performance test depends on available system resources and the collection used for the test.

3 Methodology

Using performance test tools should help improve the chances of achieving performance testing goals (e.g., validating that the system can function under high load, ensuring the reliability and performance of a system, and checking the system's performance under everyday operational situations using the upper bound of performance).

The use of performance test tools and their integration into test automation frameworks can enhance testing performance by increasing test speed and efficiency, improving test accuracy, reducing test maintenance costs, and mitigating risks [24]. However, performance load testing often requires a significant amount of time, running from hours to even days [25].

Performance testing tools, CI/CD instruments and reporting tools can be parts of a test automation framework. They provide developers and QA engineers with feedback on system performance metrics, including response times and resource utilization. Furthermore, there is a growing trend of adopting a shift-left testing approach, combined with CI/CD practices, which implies that testing is done early and frequently throughout the project life cycle [26-29]. Additionally, automated API tests should be simple, fast, and stable [16]. Thus, the speed of performance test tools becomes an essential factor.

In the context of performance testing, response time is closely related to the perceived speed of the system. The lower the response time, the higher the perceived performance. Thus, there is a strong correlation between these characteristics in performance testing tasks. This allows for using response time as one of the leading performance indicators.

The ART is a metric that provides an overview of the general user experience, and it is also important as the ideal baseline response time, such that any lags that should be investigated or considered critical can be identified [30]. However, percentiles provide a more accurate representation of user experience by accounting for the distribution of response times.

This is crucial for identifying performance bottlenecks and ensuring that even the slowest responses are within acceptable limits. Percentiles should be used when focusing on responsiveness for the majority of users, identifying outliers, and ensuring consistent performance across all users. [31].

Percentiles are essential reliability metrics, capturing the threshold below which a given percentage of observations fall [32]. Performance test tools use percentile values, such as the 90th, 95th, and 99th percentiles (P90, P95, and P99, respectively).

SLAs in performance testing specify target metrics, such as response times or error rates [33, 34], typically using percentiles to represent acceptable performance levels. P99 helps identify performance bottlenecks; its high values indicate severe slowdowns affecting a small percentage of users. P95 and P90 values are commonly used in performance teams instead of ART when defining acceptable response times in SLA [35].

When comparing Postman and JMeter test tools, it is helpful to calculate the difference between the percentiles and the ART. If the ART and percentiles values are closer, then response times show little deviation, and there is confidence in the performance of your system.

It may also mean that the tool is more predictable in its performance [36], which is essential for high-load or response-critical applications [37 - 39]. On the other hand, if the ART is better for a tool, it means that, overall, this tool processes requests more quickly. This could be a result of more efficient request processing.

However, it is essential to consider that ART can be affected by extreme values (e.g., very slow requests). Considering factors such as the stability of a test tool, a larger difference between percentile and ART values may indicate that the tool is prone to performance fluctuations.

This may be especially important for testing real-time applications (e.g., gaming, streaming, and mobile applications, or high-traffic web applications as marketplaces) where minor delays can lead to false positive results (the application appears to be more productive than it actually is) and missed issues.

To understand the degree of stability in a test tool's performance, percentage deviation (D) from ART can be calculated using Formula 1. D value reflects the stability of a test tool, where a deviation below 20% indicates better stability:

$$P_{\text{failure}} = f(x_1, x_2, \ldots, x_n), \quad (1)$$

where P_p is the percentile value (e.g., P_{90} is the value of the 90th percentile).

A positive deviation indicates that the percentile value is significantly higher than the ART. This may indicate that some slow queries are contributing to the system's increased load.

A deviation of zero or close to zero indicates that the percentile value is close to the ART, meaning that the responses are relatively stable and slow queries do not significantly impact the system.

The following scale can be used for D values evaluation:

- low percentage deviation (less than 20%) that indicates stable performance of a system;

- moderate percentage deviation (20% - 50%), which suggests that the system has a noticeable number of slow queries that affect performance;
- high percentage deviation (more than 50%), which demonstrates significant performance degradation that impacts overall stability;
- critical deviation (more than 100%) that indicates there are non-optimal queries or unpredictable latencies, which have a significant impact on stability.

Thus, a test tool with a minor deviation between percentile and ART can be selected for performance testing as a more stable option, which is essential for highly loaded systems where predictability needs to be maintained. In contrast, if the testing goal is to reduce overall testing time, then a tool with a lower ART may be more suitable.

To evaluate deviation values (D) for Postman and JMeter test tools, performance testing of Hypertext Transfer Protocol (HTTP) requests was conducted on five public APIs. For each of the five public APIs, four types of HTTP requests were sent: GET, POST, PUT, and DELETE. Additionally, eight different test cases (TCs) were executed for each type of request in both test tools.

The TCs used different types of loads: Ramp Up, Spike, Peak, and Fixed in Postman and corresponding Thread Groups in JMeter, along with the number of virtual users (VU), think time or delay, and test duration.

The following types of loads were used in Postman:

- “Ramp Up” is used for gradual increase and scalability test;
- “Spike” is used for sudden surge and resilience test;
- “Peak” is used for sustained high load and endurance test;
- “Fixed” is used for stable load and baseline test.

“Ramp Up” load type in Postman simulates a gradual increase in traffic over time:

- *Example*: starting with 10 virtual users (VUs) and adding more users every few seconds until reaching 1,000 VUs;
- *Purpose*: tests how the system handles progressive load growth and whether it scales smoothly without errors or degradation;
- *Use case*: mimicking real-world traffic growth during product launches or normal adoption curves.

“Spike” load type in Postman introduces a sudden, extreme surge in traffic within a very short period:

- *Example*: jumping from 50 users to 5,000 users almost instantly;
- *Purpose*: evaluates stability and resilience when the system experiences an unexpected traffic spike;
- *Use case*: Black Friday sales, ticket bookings, flash sales, or viral traffic events.

“Peak” load type in Postman simulates sustained high traffic after a ramp-up period:

- *Example*: traffic ramps up to 2,000 users and then maintains that level for a long duration;

- *Purpose*: tests how the system performs under continuous heavy load without downtime or performance decline;
- *Use case*: streaming platforms during a popular live event, social media surges, or sustained API demand in production.

"Fixed" load type in Postman maintains a constant, steady number of users throughout the test:

- *Example*: running 500 users continuously for 30 minutes;
- *Purpose*: establishes a baseline performance benchmark for latency, error rates, and throughput under normal conditions;
- *Use case*: routine performance validation, regression testing, or comparing infrastructure changes.

ART values and values of the 90th, 95th, and 99th percentiles (P90, P95, and P99, respectively) were collected for each test case (TC) and type of request from performance test reports in both test tools.

Since percentile values were obtained for each test run, P90, P95, and P99 values were collected per test iteration to assess the stability and typical behavior of performance percentiles. To obtain an aggregate representation, we calculated the respective percentiles (e.g., 90th percentile of all P90 values) using *numpy.percentile()* function in Python.

This approach leverages the empirical distribution of performance metrics to identify representative or worst-case scenarios while mitigating the influence of outliers.

Calculation of percentage deviation D values for Postman and JMeter test tools was performed as the next step in evaluating the performance of a test tool's stability.

After that, mean values [40] of ART, P90, P95, P99, and D were calculated for each type of HTTP request, using Formula 2:

$$\text{Mean\_metric} = \frac{\sum_{k=1}^{n} m_k}{n}, \tag{2}$$

where m_k is the value of the performance metric for TC_k;
n is the number of test cases.

In the final phase, the aggregated performance data of ART and corresponding D values were analyzed for each type of HTTP request to assess the degree of deviation and draw conclusions about the stability of the testing tool.

4 Results

Research shows that techniques such as performance and regression testing frequently co-occur, combining enhancements to the application's performance with the absence of regression issues that appear during continuous deployment [41].

The inconsistency of the interaction chain, the independence of services, dynamic and frequent deployments, and the specific challenges of cloud-native

environments can influence the testing of microservices. Effective API testing should aim to strike a balance between thoroughness and resource efficiency, maximizing coverage while minimizing redundancy.

Additionally, Continuous Integration (CI) requires efficient regression testing to ensure software quality without significantly delaying its CI builds. This warrants the need for techniques to reduce regression testing time, such as Test Case Prioritization (TCP) techniques that prioritize the execution of test cases to detect faults as early as possible [42].

Thus, the approach of prioritizing test cases by test tools can be applied in the context of performance testing. To optimize resource utilization, minimize total test execution time, and use tool-specific strengths, a structured decision-making diagram is required.

The development of a decision-making diagram that supports the distribution of test cases between two performance testing tools must consider various factors. For Postman and JMeter test tools, important factors may include the number of VUs, think time, and the type of load planned for the test (e.g., high, medium, low), based on their differences in application during a planned test.

Additionally, statistical data on the performance of test tools need to be analyzed to identify other factors that can influence the decision on how to distribute tests among test tools. Percentile and D values for Postman and JMeter were calculated for each HTTP request method for 8 TCs and then averaged across all 5 APIs, as shown in Tables 1, 2, and 3. Percentile values exceeded ART for both tools, indicating the presence of performance degradation. The most significant values are highlighted in bold. Postman has a better ART and processes requests faster than JMeter.

Table 1

Aggregated mean values of ART for different requests in Postman and JMeter

Method	Tool	ART, sec
GET	Postman	**166,175**
GET	JMeter	**181,65**
POST	Postman	148,100
POST	JMeter	147,175
PUT	Postman	**144,925**
PUT	JMeter	**152,075**
DELETE	Postman	166,225
DELETE	JMeter	163,325

Table 2

Aggregated mean values of percentiles for different requests in Postman and JMeter

Method	Tool	P90, sec	P95, sec	P99, sec
GET	Postman	298,480	392,650	521,656
GET	JMeter	306,320	326,500	496,108
POST	Postman	188,820	219,400	328,572
POST	JMeter	194,040	207,600	385,248
PUT	Postman	193,460	224,800	643,546
PUT	JMeter	234,980	286,100	364,792
DELETE	Postman	200,040	210,690	434,812
DELETE	JMeter	212,300	232,660	581,036

The Postman shows minimal differences in P90 and P95 D for POST (6.855% and 0.112%) and DELETE requests (11.364% and 17.611%), indicating good stability. Moderate and high differences in P90 D and P95 D for GET (26.45% and 68.236%) requests suggest stability issues and performance fluctuations, which impact the user experience. Additionally, JMeter demonstrates better performance in P99 for PUT requests and in all scenarios for GET requests.

Table 3

Aggregated mean values of percentage deviation for different requests in Postman and JMeter

Method	Tool	P90 D, %	P95 D, %	P99 D, %
GET	Postman	**101,240**	158,613	311,005
GET	JMeter	**74,790**	90,376	267,864
POST	Postman	20,233	36,872	106,621
POST	JMeter	27,088	36,760	161,460
PUT	Postman	25,320	43,121	**399,459**
PUT	JMeter	53,631	82,779	**146,312**
DELETE	Postman	16,309	22,629	160,728
DELETE	JMeter	27,673	40,240	229,010

The calculated difference in average response time between Postman and JMeter (deltas) in each of the 8 TCs shows that Postman has higher performance in low to moderate load conditions. On the other hand, JMeter illustrates higher performance under the conditions of high load and request intensity (sharp increase in load to peak, followed by a gradual decrease with a short think time).

Thus, Postman with a lower ART improves response time and efficiency under average loads, but does not ensure stability in the presence of significant outliers. JMeter, with more minor deviations for GET and PUT requests, is more stable and

predictable, which is crucial for testing complex systems under fluctuating load conditions.

Moreover, applying a successful API performance testing strategy should also focus on the longevity and sustainability of the software's organizational structure. To improve such testing, it is critical to optimize developers' collaboration to ensure the proper allocation of responsibility and reduce ineffective communication [43]. As a result, collaboration becomes a crucial factor in performance testing.

n the context of performance testing tools, collaboration can be done in a built-in way, when the test tool itself has an option of sharing test cases, results and collections for API requests through UI and workspaces, or in an external way (using Git, version control or even manual sharing of test cases and test plans). For instance, Postman has a built-in collaboration option, and the external collaboration approach is mainly used in JMeter.

A critical difference between JMeter and Postman is their ability to be integrated into CI/CD pipelines, such as those built with Jenkins. The implementation of CI/CD in performance testing enables integrated, automated, and periodic execution of test processes [44].

It can also quickly respond to changes in parameter values. JMeter offers direct support for such integrations, making it well-suited for automated and continuous performance testing. Currently, Postman’s support for such automation workflows is comparatively limited, particularly in terms of its performance load profiles.

Thus, to address the challenge of efficiently allocating test scenarios across Postman and JMeter test tools, we propose a decision-making diagram based on a rule-based classification of test scenarios, as shown in Figure 1. The diagram incorporates both qualitative and quantitative criteria, serving as a practical guide for QA engineers and developers during the planning phase of performance test execution. Quantitative factors, such as the type of test load (number of VUs and request intensity), and the need to prioritize specific HTTP requests, provide objective benchmarks for selecting the right tool.

In contrast, qualitative aspects such as collaboration features, think-time flexibility, and CI/CD integration capabilities reflect practical considerations that influence QA or developer productivity and tool suitability. By combining these criteria, the diagram provides a balanced framework for allocating performance test scenarios.

The proposed decision-making diagram consists of four levels of decision logic. The diagram is designed to guide the allocation of performance test scenarios between two tools, JMeter and Postman, based on scenario attributes and tool capabilities.

At the first level, the diagram begins with a quantitative parameter that characterizes the test scenario, specifically the type of test load (required number of VUs and the expected request intensity). This parameter determines the initial classification of the scenario based on its load intensity.

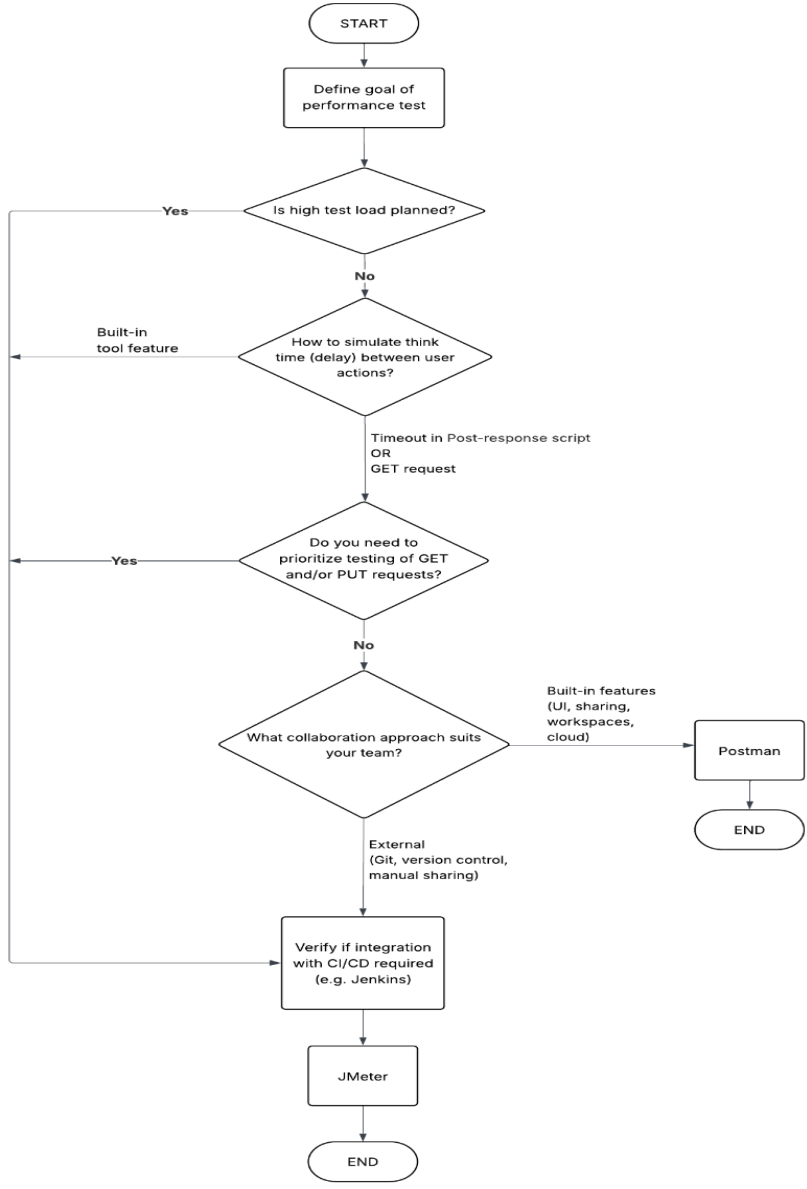

Figure 1. Decision-making diagram of the distribution of tests between test tools

The second level introduces qualitative criteria related to the flexibility of simulating think time. This includes factors such as the ease of implementing delays between user actions, either through built-in, low-overhead mechanisms (e.g., configurable timers) or using manual workarounds, such as post-response timeouts or additional GET requests intended to emulate pauses. This decision branch helps refine tool selection by assessing the functional (e.g., the need for realistic user behavior simulation) and operational requirements (e.g., efficiency, script simplicity, resource consumption (memory and CPU load)) of the test scenario.

At the third level, tool-specific technical considerations are evaluated, including the need for prioritizing specific HTTP requests (e.g., GET and PUT requests). This criterion serves to validate or adjust the preliminary tool assignment by comparing fine-grained compatibility between the scenario and the tool features.

The fourth level shows qualitative criteria related to the execution environment. This includes factors such as the presence of a collaboration feature that refines the tool selection by evaluating the functional requirements of the test scenario.

Each decision node represents a binary or categorical condition (e.g., high load, built-in think time needed, focus is needed for GET and/or PUT requests, collaboration using UI and shared workspaces required), which directs the flow to the following relevant decision node, leading to two terminal outcomes: assignment to JMeter or Postman test tool. This means that during the process of planning and executing performance tests, test cases can be split between two tools, each of which will be used for different types of tests, workloads or tasks, based on their strengths and weaknesses.

The diagram is constructed to be both extendable and interpretable, allowing new criteria to be incorporated without altering the core logic, and ensuring consistency in decision-making during the planning phase of performance testing.

For instance, in a scenario involving a complex API with high concurrency (over 500 simultaneous VUs), the need for realistic think time, and integration with Jenkins, the decision diagram leads to selecting JMeter due to its robust load-handling capabilities and native support for CI/CD pipelines. In contrast, for performance testing a lightweight API involving fewer than 50 users and simple GET/POST requests, with the ability to share results using a UI and cloud, the diagram suggests Postman as the more efficient choice.

The proposed decision-making diagram provides a structured approach for distributing performance test scenarios between two tools, Postman and JMeter, based on defined scenario parameters and tool capabilities. This contributes to reducing subjectivity in tool selection by replacing ad hoc decisions with transparent, criteria-driven logic.

In practical terms, the diagram enables efficient use of both tools: lightweight or low-load scenarios can be quickly executed using Postman. In contrast, more complex or high-load scenarios are directed to JMeter. This selective allocation helps reduce test execution time and unnecessary overhead, improves overall testing efficiency, especially in projects with limited time or resources.

5 Discussions

The proposed decision-making diagram addresses the lack of formal criteria for selecting performance testing tools in practical settings. By providing a structured approach based on scenario characteristics (load, think time requirements, collaboration needs, etc.), the diagram helps reduce subjectivity and improve the efficiency and consistency of test planning.

The developed approach can be helpful in development teams with mixed tool stacks (JMeter and Postman) where decisions on tool usage are often ad hoc. It can be embedded into test management workflows as a decision-support mechanism.

Unlike existing tool comparison matrices or benchmark studies, the proposed diagram integrates context-sensitive decision logic, allowing assessment based on test scenario attributes rather than static tool capabilities.

While the proposed decision-making approach provides a structured and repeatable approach for allocating performance test scenarios between different tools, several limitations should be acknowledged, such as limited tool scope (the approach currently considers only Postman and JMeter tools), the decision logic does not dynamically adjust the weight or priority of individual criteria based on project-specific needs or context, and the diagram does not incorporate empirical performance data (e.g., response time, error rate, CPU/memory usage of test execution) that could further inform the selection process.

6 Conclusions

The developed decision-making approach offers practical value by providing a structured and transparent framework for assigning performance test scenarios to the most suitable tools based on objective criteria. In practice, the selection of performance testing tools is often driven by subjective preferences, prior experience, or ad hoc judgments, which can lead to inconsistent tool usage, suboptimal test configurations, or even inaccurate test results.

The developed approach supports simple standardization of tool selection across teams and projects. This is especially beneficial in collaborative environments, where multiple stakeholders (e.g., QA engineers, DevOps specialists, project managers) participate in test planning and execution. The approach can serve as a reference, reducing ambiguity and supporting decisions aligned with scenario-specific requirements, thereby increasing both the transparency and reproducibility of the test process. Furthermore, the approach enables decision-making at an early stage during test planning, allowing better resource allocation and minimizing the risk of rework caused by choosing inappropriate performance test tools. The decision-making diagram facilitates the distribution of performance test scenarios between Postman and JMeter, enabling the rapid execution of simple tests using lightweight tools and delegating complex or high-load scenarios to more robust solutions, thereby improving overall testing speed and resource utilization.

7 Acknowledgements

The research was supported by the Ukrainian project of fundamental scientific research, "Development of computational methods for detecting objects with near-zero and locally constant motion by optical-electronic devices" (#0124U000259) from 2024 to 2026.

References

1. OctoPerf. (2020). *Statistical analysis in performance testing*. Retrieved from https://blog.octoperf.com/statistical-analysis-in-performance-testing
2. TestRail. (2025). *Performance testing metrics*. Retrieved from https://www.testrail.com/blog/performance-testing-metrics
3. BlazeMeter. (2020). *Key performance metrics*. Retrieved from https://help.blazemeter.com/docs/guide/performance-kpis-key-perf-test-metrics.htm
4. Soldani, J., & Brogi, A. (2022). Anomaly detection and failure root cause analysis in (micro) service-based cloud applications: A survey. *ACM Computing Surveys (CSUR), 55*(3), 1–39. https://doi.org/10.1145/3501297
5. Panahande, M., & Miller, J. (2023). A systematic review on microservice testing. *Research Square*. https://doi.org/10.21203/rs.3.rs-3158138/v1
6. Khader Basha, S., Purimetla, N. R., Roja, D., Vullam, N., Dalavai, L., & Vellela, S. S. (2023). A cloud-based auto-scaling system for virtual resources to back ubiquitous, mobile, real-time healthcare applications. *2023 3rd International Conference on Innovative Mechanisms for Industry Applications (ICIMIA)*, 1223–1230. https://doi.org/10.1109/ICIMIA60377.2023.10426107
7. Batubara, J., Sinta, R., & Panjaitan, F. (2024). Performance analysis of web-based e-commerce information systems using load testing method. *Idea, 2*(1), 18–27.
8. Fournier, Q., Ezzati-jivan, N., Aloise, D., & Dagenais, M. R. (2019). Automatic cause detection of performance problems in web applications. *2019 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW)*, 398–405. https://doi.org/10.1109/ISSREW.2019.00102
9. Chadwick, D., et al. (2008). *Using rational performance tester version 7*. IBM.
10. Dhandapani, A. (2025). Automation testing in microservices and cloud-native applications: Strategies and innovations. *Journal of Computer Science and Technology Studies, 7*(3), 826–836. https://doi.org/10.32996/jcsts
11. Pargaonkar, S. (2023). A comprehensive review of performance testing methodologies and best practices: Software quality engineering. *International Journal of Science and Research (IJSR), 12*(8), 2008–2014. https://doi.org/10.21275/SR23822111402
12. Ajiga, D., et al. (2024). Methodologies for developing scalable software frameworks that support growing business needs. *International Journal of Management and Entrepreneurship Research, 6*(8), 2661–2683. https://doi.org/10.51594/ijmer.v6i8.1413

13. Yaroshynskyi, M., Puchko, I., Prymushko, A., Kravtsov, H., & Artemchuk, A. (2025). Investigating the evolution of resilient microservice architectures: A compatibility-driven version orchestration approach. *Digital*. https://doi.org/10.3390/digital5030027

14. Molyneaux, I. (2014). *The art of application performance testing: From strategy to tools*. O'Reilly Media.

15. ISO Standards. (2020). Retrieved from https://iso25000.com/index.php/en/iso-25000-standards/iso-25010

16. Manukonda, K. R. R. (2024). *Optimizing performance: Designing API test automation frameworks*. JEC Publication.

17. Nagineni, S. (2025). Advancing software reliability through systematic API testing: A comparative analysis of modern automation frameworks and methodological implications for distributed systems. *JCSTS, 7*(8), 798–805. https://doi.org/10.32996/jcsts.2025.7.8.94

18. AutomateNow. (2023). *Advantages and disadvantages of using JMeter*. Retrieved from https://automatenow.io/advantages-and-disadvantages-of-using-jmeter

19. TestSigma. (2025). *JMeter vs Postman*. Retrieved from https://testsigma.com/blog/jmeter-vs-postman

20. Westerveld, D. (2024). *API testing and development with Postman: API creation, testing, debugging, and management made easy*. Packt Publishing.

21. Postman. (2024). *Performance test configuration*. Retrieved from https://learning.postman.com/docs/collections/performance-testing/performance-test-configuration

22. NashTech. (2024). *Performance testing with Postman: Is it worth?* Retrieved from https://blog.nashtechglobal.com/performance-testing-with-postman-is-it-worth

23. Susan Rini, V. S. (2024). When Postman goes that extra mile to deliver performance to APIs. *Software Testing Magazine*. Retrieved from https://www.softwaretestingmagazine.com/tools/when-postman-goes-that-extra-mile-to-deliver-performance-to-apis

24. Umar, M. A., & Chen, Z. (2019). A study of automated software testing: Automation tools and frameworks. https://doi.org/10.5281/zenodo.3924795

25. AlGhamdi, H. M., Bezemer, C.-P., Shang, W., Hassan, A. E., & Flora, P. (2023). Towards reducing the time needed for load testing. *Journal of Software: Evolution and Process*. https://doi.org/10.1002/smr.2276

26. Vaddadi, S. A., Thatikonda, R., Padthe, A., et al. (2023). Shift left testing paradigm process implementation for quality of software based on fuzzy. *Soft Computing*. https://doi.org/10.1007/s00500-023-08741-5

27. Mukherjee, R., & Patnaik, K. S. (2021). A survey on different approaches for software test case prioritization. *Journal of King Saud University - Computer and Information Sciences*, 1041–1054. https://doi.org/10.1016/j.jksuci.2018.09.005

28. Parmar, T. (2025). Implementing CI/CD in data engineering: Streamlining data pipelines for reliable and scalable solutions. https://doi.org/10.5281/zenodo.14762683

29. Rani, V. S., Babu, D. A. R., Deepthi, K., & Reddy, V. R. (2023). Shift-left testing in DevOps: A study of benefits, challenges, and best practices. *2023 2nd International Conference on Automation, Computing and Renewable Systems (ICACRS)*, 1675–1680. https://doi.org/10.1109/ICACRS58579.2023.10404436

30. Lawi, A., Panggabean, B. L. E., & Yoshida, T. (2021). Evaluating GraphQL and REST API services performance in a massive and intensive accessible information system. *Computers*. https://doi.org/10.3390/computers10110138

31. Godinho, A., Rosado, J., Sá, F. A., & Cardoso, F. (2023). Method for evaluating the performance of web-based APIs. *International Conference on Smart Objects and Technologies for Social Good*. https://doi.org/10.1007/978-3-031-52524-7_3

32. Smith, W. (2025). *Nobl9: Service level objectives in practice: The complete guide for developers and engineers*. HiTeX Press.

33. Amazon. (2020). *What is SLA*. Retrieved from https://aws.amazon.com/what-is/service-level-agreement

34. Seifert, M. (2021). Analysis of public cloud service level agreements: An evaluation of leading software as a service providers. *CIISR@Wirtschaftsinformatik*, 22–35.

35. Qian, W., et al. (2025). Learning unified system representations for microservice tail latency prediction.

36. Anderstedt, H., & Wifvesson, M. (2025). Benchmarking and load testing a dynamic CRM architecture (Bachelor's thesis). Lund University, Helsingborg, Sweden.

37. Hendayun, M., Ginanjar, A., & Ihsan, Y. (2023). Analysis of application performance testing using load testing and stress testing methods in API service. *Jurnal Sisfotek Global, 13*(1), 28–34. https://doi.org/10.38101/sisfotek.v13i1.2656

38. Ramu, V. B. (2023). Performance testing and optimization strategies for mobile applications. *International Journal of Performance Testing and Optimization*. https://doi.org/10.14445/22492615/IJPTT-V13I2P401

39. Gorantla, V. K. C. (2021). A hybrid WebSocket-REST approach for scalable real-time API design. *IJETCSIT*, 60–69. https://doi.org/10.63282/3050-9246.IJETCSIT-V2I3P107

40. Savanevych, V., et al. (2023). Mathematical methods for an accurate navigation of the robotic telescopes. *Mathematics, 11*(10), 2246. https://doi.org/10.3390/math11102246

41. Miao, T., Shaafi, A. I., & Song, E. (2025). Systematic mapping study of test generation for microservices: Approaches, challenges, and impact on system quality. *Electronics*. https://doi.org/10.3390/electronics14071397

42. Yaraghi, A. S., Bagherzadeh, M., Kahani, N., & Briand, L. C. (2023). Scalable and accurate test case prioritization in continuous integration contexts.

IEEE Transactions on Software Engineering, 1615–1639. https://doi.org/10.1109/TSE.2022.3184842

43. Li, X., Calefato, F., Lenarduzzi, V., & Taibi, D. (2024). Toward collaboration optimization in microservice projects based on developer personalities. *2024 IEEE 21st International Conference on Software Architecture Companion (ICSA-C)*, 95–99. https://doi.org/10.1109/ICSA-C63560.2024.00024

44. Pratama, M. R., & Kusumo, D. S. (2021). Implementation of continuous integration and continuous delivery (CI/CD) on automatic performance testing. *2021 9th International Conference on Information and Communication Technology (ICoICT)*, 230–235. https://doi.org/10.1109/ICoICT52021.2021.9527496

АНАЛІЗ ПЕРЦЕНТИЛІВ ПРОДУКТИВНОСТІ ДЛЯ ТЕСТУВАННЯ НА ОСНОВІ API

Ph.D. С. Хламов[1][0000-0001-9434-1081], М. Мендієлєва[2][0009-0002-4282-3147],
Ph.D. О. Вовк[3][0000-0001-9072-1634], Т. Трунова[4][0000-0003-2689-2679],
Ю. Тесленко[5][0009-0009-8349-3683]

Харківський національний університет радіоелектроніки, Україна
EMAIL: [1]sergii.khlamov@gmail.com, [2]mariia.mendielieva@nure.ua, [3]oleksandr.vovk@nure.ua, [4]tetiana.trunova@nure.ua, [5]yuliia.teslenko@nure.ua

Анотація. *У статті зосереджено увагу на системному аналізі метрик тестування продуктивності з особливим акцентом на порівняння поведінки двох широко застосовуваних інструментів — Postman та JMeter — у контексті публічних прикладних програмних інтерфейсів (API). Тестування продуктивності API відіграє критично важливу роль в оцінюванні відгукуваності та надійності сучасних програмних систем; однак однією з постійних проблем залишається опора виключно на середній час відгуку. Середні значення легко спотворюються аномальними викидами, які часто маскують суттєві затримки відповіді та призводять до неповної картини продуктивності системи. Щоб подолати це обмеження, у даному дослідженні наголошується на використанні аналізу на основі перцентилів, що забезпечує точніший і орієнтований на користувача показник продуктивності для більшості запитів. Методологія передбачала порівняння середніх значень часу відгуку з розподілами перцентилів, з основною увагою до 90-го, 95-го та 99-го перцентилів. Додатково було обчислено відсоткове відхилення значень перцентилів від середнього, що слугувало мірою стабільності інструментів тестування. Такий підхід дозволив більш надійно оцінити поведінку інструментів за різних навантажень. Експериментальні результати виявили виразні переваги кожного інструмента. Postman продемонстрував кращі результати за середнього рівня навантаження для запитів Create та Delete. Водночас JMeter виявився більш ефективним для операцій Read (GET) та Update, де стабільність і передбачуваність є критичними у складних системах. Для підтримки практичного застосування*

у роботі також запропоновано діаграму прийняття рішень, що допомагає розподіляти тестові сценарії між двома інструментами, зрештою підвищуючи ефективність тестування та забезпечуючи більш надійну оцінку продуктивності API.

Ключові слова: *тестування продуктивності, тестування API, навантажувальне тестування, стабільність системи, Postman, JMeter, аналіз часу відгуку, перцентильні метрики, оцінювання продуктивності, прийняття рішень, тестування програмного забезпечення*

УДК 69:628.4:681.5:004
DOI https://doi.org/10.36059/978-966-397-538-2-14

ІНТЕЛЕКТУАЛЬНА ІНФОРМАЦІЙНА ТЕХНОЛОГІЯ ІНВЕНТАРИЗАЦІЇ ТА ВИКОРИСТАННЯ БУДІВЕЛЬНИХ ВІДХОДІВ ПРИ РЕКОНСТРУКЦІЇ ПОШКОДЖЕНИХ ІНФРАСТРУКТУРНИХ ОБ'ЄКТІВ

Dr.Sci. О. Арсірій[1[0000-0001-8130-9613]], **Ph.D. О. Іванов**[2[0000-0002-8620-974X]], **Ph.D. N. Cudecka-Purina**[3[0000-0002-5736-7730]], **К. Бєляєв**[4[0009-0001-7135-3562]]

[1,2,4]Національний університет «Одеська політехніка», Україна,
[3]BA School of Business and Finance, Riga, Latvia,
[3]EKA University of Applied Sciences, Riga, Latvia
EMAIL: [1]e.arsiriy@gmail.com, [2]lesha.ivanoff@gmail.com,
[3]natalija.cudecka-purina@ba.lv, [4]kirillbelyaev2921@gmail.com

Анотація. *Ця робота розглядає потенціал підвищення ефективності інвентаризації та утилізації відходів від будівництва та демонтажу (ВБД) для створення та застосування як вторинних ресурсів під час відбудови пошкодженої інфраструктури, зокрема об'єктів, зруйнованих війною. Це досягається шляхом розробки інтелектуальної інформаційної технології (ІІТ). Актуальність вирішення цього питання підкреслюється тим, що великомасштабна відбудова пошкодженої інфраструктури вимагає значних витрат вітчизняних та інвестиційних первинних ресурсів. З огляду на неминучий дефіцит цих ресурсів, розвиток ринку вторинної сировини та можливість вилучення цінних матеріалів з потоків відходів для їх ефективного повторного використання стають надзвичайно важливими. Аналіз досвіду європейських колег щодо ефективного впровадження бізнес-моделі циркулярної економіки для подолання обмежень у використанні первинних ресурсів підкреслює необхідність розробки ІІТ для інвентаризації та утилізації ВБД. Попередній досвід розробників вказує на те, що ІІТ може бути реалізована як геоінформаційна система (ГІС), що дозволяє ідентифікувати об'єкти інфраструктури через базу даних. Розроблена база*