**Storozhuk Yuriy,** Postgraduate student, Vinnytsia National Technical University, Vinnytsia, Ukraine, https://orcid.org/0009-0002-1066-1358

# METHODOLOGY FOR CONTINUOUS PERFORMANCE TESTING OF EDUCATIONAL INFORMATION SYSTEMS BASED ON K6 AND DYNAMIC QUALITY GATES

*Abstract*

Educational information systems operate under uneven and often unpredictable workloads caused by examination periods, mass access to learning materials, online assessment activities, and administrative operations. Traditional episodic performance testing is often insufficient for such systems because performance regressions may appear during continuous software delivery and remain undetected until deployment. This paper proposes a methodology for continuous performance testing of educational information systems based on k6 and dynamic Quality Gates. The methodology combines realistic educational workload scenarios, automated execution of tests in a CI/CD pipeline, baseline-oriented regression detection, and release validation using service-level objectives. A low-cost experimental environment is proposed to make the approach reproducible for universities and small development teams. The environment may be implemented on a single virtual machine or free-tier cloud instance and is intended primarily for regression-oriented validation rather than full-scale capacity testing. The results show that k6-based continuous performance testing can support early detection of performance degradation, improve release decision-making, and provide a practical path for gradual implementation of performance quality control in educational digital ecosystems.

The study shows that performance testing of educational systems should be based on realistic user scenarios rather than only on generic technical request

patterns. Critical workflows such as authentication, course access, online test start, answer submission, and test completion should be represented as repeatable k6 scenarios with clearly defined metrics and acceptance criteria.

A key feature of the proposed methodology is the use of dynamic Quality Gates that combine thresholds, baselines, and service-level objectives. This makes it possible to detect both absolute violations of performance requirements and relative degradation compared with previously accepted behavior. As a result, performance testing becomes not only a diagnostic activity, but also a release control mechanism.

*Keywords:* continuous performance testing, educational information systems, k6, dynamic Quality Gates, performance regression, CI/CD, low-cost environment, baseline testing

*Problem statement.* Educational information systems have become a critical component of modern digital learning environments. They support course delivery, access to learning materials, assessment procedures, communication between participants of the educational process, and administrative operations. In practice, such systems operate under uneven load conditions caused by academic calendars, examination sessions, admission campaigns, simultaneous access of large numbers of users, and short-term activity spikes during critical learning events. Because of this, performance stability becomes an essential requirement for the continuity and quality of educational services.

*Analysis of recent research and publications.* Traditional performance testing is usually applied before release or during isolated infrastructure validation. Although such testing remains important, it is often insufficient in rapidly evolving software delivery processes. In CI/CD environments, changes in code, configuration, dependencies, and infrastructure may introduce performance regressions long before they are discovered by conventional testing practice. As

a result, systems may formally pass functional verification while still degrading in response time, stability, or throughput after deployment. Continuous delivery practice emphasizes early and repeated verification of quality attributes, including performance [1; 2].

This issue is especially important for educational systems, where performance degradation directly affects critical user's scenarios. Delays during authentication, slow course loading, unstable online testing sessions, or failures during answer submission may disrupt the educational process and reduce user trust in the platform. Therefore, performance validation should be integrated into the software delivery lifecycle and should support release-oriented decision-making rather than remain an isolated post-development activity.

The main contribution of this paper is a methodology for continuous performance testing of educational information systems based on k6 and dynamic Quality Gates. k6 is selected as the execution layer because it supports scenario-based workload modeling, threshold-based pass/fail validation, and integration with CI/CD and monitoring environments [3; 4]. The proposed methodology combines realistic educational workload scenarios, automated test execution, and adaptive release validation based on thresholds, baselines, and service-level objectives. Modern distributed architecture also increases the need for observability and systematic monitoring. Site Reliability Engineering emphasizes service-level indicators and service-level objectives as a basis for reliability-oriented engineering decisions [5]. Systems performance engineering highlights the importance of latency, throughput, saturation, and resource behavior as interrelated indicators rather than isolated metrics [6]. Microservice-based systems additionally require scenario-level and service-level validation because a single user operation may depend on several backend components [7]. The use of k6 is relevant in this context because it supports script-based scenario definition, threshold-based validation, and integration into automated pipelines.

However, the scientific and applied problem is not the adoption of k6 itself, but the development of a methodology in which k6 acts as an execution layer for continuous, release-oriented performance validation. This interpretation is consistent with recent work on performance regression testing, self-service performance testing platforms, and integration of performance checks into DevOps pipelines. However, the scientific and applied problem is not the adoption of k6 itself, but the development of a methodology in which k6 acts as an execution layer for continuous, release-oriented performance validation [8; 9].

*Purpose of the article.* This article presents a methodology for continuous performance testing of educational systems characterized by fluctuating workloads and multiple critical user services.

*Presentation of the main material of the study.* Educational information systems have their own performance risks. Their load is not evenly distributed over time: examination sessions, admission campaigns, mass testing, and synchronous educational activities may create short but critical peaks. In such systems, testing should not only measure infrastructure behavior but also verify the stability of critical educational workflows. Previous stress testing of JetIQ and related educational services showed the practical importance of checking learning management workflows under exceptional usage conditions [10; 11].

It is based on four principles: scenario realism, automated repeatability, release-oriented validation, and low-cost reproducibility.

Scenario realism requires that performance tests reflect actual educational workflows instead of abstract request loops. In this study, the workload model is centered around critical scenarios: user authentication, course access, online test session start, answer submission, and test completion. These scenarios are selected because they directly affect continuity of the educational process and user trust in the platform. In k6, they can be represented as separate scenarios

with different virtual user profiles, ramp-up phases, pacing, and acceptance criteria.

Automated repeatability requires that performance tests be executable in a CI/CD pipeline without manual reconfiguration. In the proposed methodology, k6 acts as the workload generation and execution layer, while the pipeline provides orchestration, reporting, and integration with monitoring. Test scripts are versioned together with application code, which makes it possible to compare release candidates under the same controlled workload.

Release-oriented validation means that the output of performance testing should influence release control. For this purpose, the methodology introduces dynamic Quality Gates based on thresholds, baselines, and service-level objectives. Thresholds define explicit acceptance limits for metrics such as response time, error rate, or throughput. Baselines represent historical performance behavior for the same scenario. Service-level objectives define the expected quality targets for critical educational operations. This logic is consistent with the approach to target performance indicators developed in earlier work on service-oriented performance evaluation.

Low-cost reproducibility means that continuous performance testing should be applicable not only to enterprise-level infrastructure, but also to educational environments with limited resources. Therefore, the methodology separates regression-oriented validation from full-scale capacity testing. Regression detection can be performed in a minimal environment because it evaluates relative degradation compared with a stable baseline. Full-scale capacity testing requires larger infrastructure and is executed less frequently.

The reference environment may be implemented on a cloud instance with 1-2 vCPU, 2-4 GB RAM, and 20-30 GB SSD storage. It includes Docker Compose, the application under test, a lightweight database container, k6 as the

load generator. This configuration is sufficient for regression-oriented continuous performance testing.

In practical terms, the methodology includes six stages: identification of critical educational scenarios; implementation of k6 scripts; integration of test execution into CI/CD; collection and export of results; comparison with thresholds and baselines; and generation of a Quality Gate decision for the release.

As shown in Figure 1, the methodology begins with the identification of critical educational scenarios.

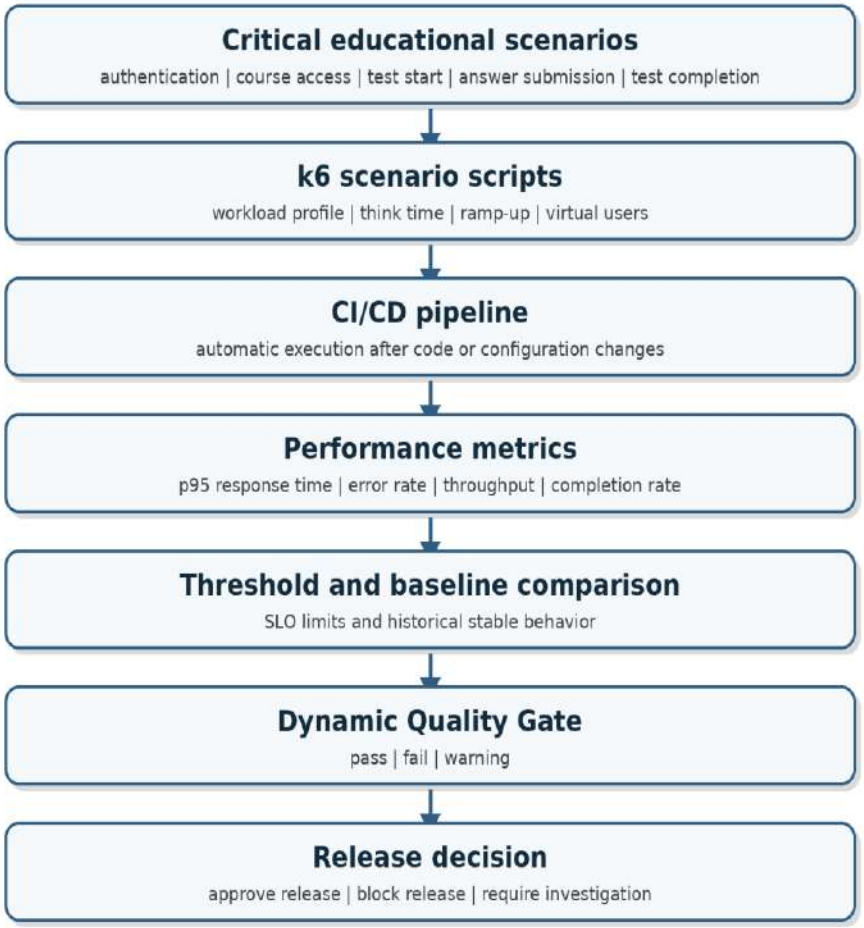



Figure 1. Architecture of k6-based continuous performance testing with dynamic Quality Gates

These scenarios are transformed into k6 scripts that reproduce user behavior under controlled workload conditions. The scripts are executed automatically in the CI/CD pipeline, and the collected metrics are compared with predefined thresholds and baseline values. The final output is a dynamic Quality Gate decision that determines whether the release candidate can be approved, blocked, or marked for additional investigation.

The Dynamic Quality Gate is defined as a release control mechanism that evaluates the results of continuous performance testing against acceptance rules. Unlike static pass/fail checks based only on fixed thresholds, the dynamic gate combines static criteria with baseline-aware comparison and scenario criticality.

Let m denote a performance metric observed during execution of scenario s. The gate decision is based on the comparison of the observed value $V_s(m)$ with a threshold $T_s(m)$ and a historical baseline $B_s(m)$. A simplified gate condition is represented as follows:

$$QG_s(m) = \begin{cases} 1, & \text{if } V_s(m) \le T_s(m) \text{ and } \Delta_s(m) \le \delta_s \\ 0, & \text{otherwise} \end{cases} \quad (1)$$

where $QG_s(m)$ is the Quality Gate result for metric m in scenario s; value 1 means that the gate passes, while value 0 means that it fails. $V_s(m)$ is the observed metric value, $T_s(m)$ is the allowed threshold, $\Delta_s(m)$ is the deviation from the baseline, and $\delta_s$ is the maximum acceptable baseline deviation for the scenario.

The deviation from the baseline is expressed as:

$$\Delta_s(m) = \frac{V_s(m) - B_s(m)}{B_s(m)} \quad (2)$$

where $B_s(m)$ is the baseline value of metric m for scenario s. Formula (2) makes it possible to capture regressions even when the absolute threshold is not violated, but the service behaves significantly worse than in previous stable runs.

The model assumes that different scenarios have different gate sensitivity. For example, answer submission in an online testing service should be treated as more critical than access to a non-essential informational page. Therefore, stricter thresholds and smaller allowable deviations should be assigned to educationally critical operations. Within k6, threshold logic can be directly encoded in test scripts, while baseline comparison can be implemented as an additional evaluation step in the CI/CD pipeline.

To illustrate the applicability of the proposed methodology, an educational information system similar to JetIQ is considered as a representative case of a distributed digital learning environment. The selected case reflects the operational logic of an E-University ecosystem, where several critical services support user authentication, access to learning materials, online assessment, and administrative workflows.

The case study focuses on five performance-sensitive scenarios: user authentication, course or workspace opening, online test session start, answer submission during an active test, and test session completion. These scenarios directly affect continuity of the educational process. In particular, answer submission and test completion are critical operations because their degradation may lead to loss of user progress or interruption of assessment activities.

Within the proposed methodology, each critical scenario is implemented as a dedicated k6 script or as a coordinated set of k6 scenarios. Scenario configuration includes virtual user behavior, request sequence, think time, ramp-up profile, expected thresholds, and baseline-aware interpretation rules. For the online testing module, a script may simulate a user who authenticates, opens a test session, submits several answers, and completes the test. The primary

metrics include p95 response time for answer submission, request failure rate, authentication latency, session completion success rate, and selected infrastructure indicators.

The low-cost environment supports three levels of validation, shown in Table 1. This layered structure allows educational institutions to introduce continuous performance testing gradually without requiring expensive infrastructure from the beginning.

Table 1

Levels of k6-based performance validation in educational information systems

| Validation level | Purpose | Environment | Execution frequency | Main output |
|---|---|---|---|---|
| Smoke performance check | Detect obvious performance degradation | CI/CD runner or small VM | Each commit or merge request | Quick pass/fail result |
| Baseline regression test | Compare release candidate with stable baseline | Low-cost VM or staging instance | Before release approval | Dynamic Quality Gate decision |
| Extended load test | Evaluate behavior under expected academic peak load | Dedicated staging or cloud resources | Periodically or before exams | Capacity and bottleneck report |

The proposed methodology provides a practical mechanism for introducing continuous performance validation into educational information systems under limited infrastructure conditions. Its main result is the integration of k6-based scenario execution, baseline comparison, and dynamic Quality Gate logic into a repeatable CI/CD-oriented process. This makes it possible to detect performance degradation before deployment and to use testing results directly in release decision-making.

The use of a low-cost environment changes the interpretation of performance testing results. Its purpose is not to determine the absolute maximum capacity of an educational platform, but to detect relative performance regressions between software versions. A small test environment cannot fully reproduce production-scale traffic, but it can reveal whether a new release introduces slower response times, higher error rates, or unstable behavior under the same controlled workload.

The methodology separates two levels of validation. The first is regression-oriented continuous testing, which can be executed frequently in a low-cost CI/CD environment. The second is full-scale capacity testing, which requires more powerful infrastructure and should be executed periodically or before high-risk academic events such as examinations, admission campaigns, or mass online assessment sessions. This separation makes the approach realistic for educational institutions with limited computing resources.

The main advantage of the proposed approach is economic accessibility. Educational institutions and small development teams can introduce continuous performance testing without deploying a complex Kubernetes cluster or maintaining expensive observability infrastructure. At the same time, the methodology remains compatible with more advanced environments. If additional resources become available, the same k6 scripts and Quality Gate logic can be reused with Prometheus, Grafana, Kubernetes, cloud runners, or dedicated staging environments.

The limitations of the low-cost environment should also be recognized. It is not suitable for determining the final scalability limit of the system or validating the maximum number of simultaneous users. It is also sensitive to resource noise caused by shared infrastructure. Therefore, results should be interpreted primarily as regression indicators rather than absolute production capacity measurements.

*Conclusions.* This paper proposed a methodology for continuous performance testing of educational information systems based on k6 and dynamic Quality Gates. The methodology is aimed at early detection of performance regressions and release-oriented validation of critical educational services in CI/CD environments.

The proposed low-cost implementation demonstrates that continuous performance testing can be introduced gradually, even under limited infrastructure conditions. A small virtual machine or free-tier cloud instance is sufficient for regression-oriented validation if the goal is to compare releases under the same controlled workload. Full-scale capacity testing remains necessary for evaluating maximum throughput, but it can be performed less frequently and in a dedicated environment.

The practical value of the approach lies in improving reproducibility of performance tests, reducing the risk of deploying releases with hidden performance regressions, and supporting faster identification of bottlenecks in critical educational workflows. Future research may focus on formalizing dynamic baseline calculation, automating Quality Gate configuration, and validating the methodology on real educational platforms under different academic load profiles.

*References*

1. Humble, J. and Farley, D. (2010) *Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation*. Boston: Addison-Wesley.

2. Gregg, B. (2020) *Systems Performance: Enterprise and the Cloud.* 2nd edn. Boston: Addison-Wesley.

3. Newman, S. (2021) *Building Microservices: Designing Fine-Grained Systems.* 2nd edn. Sebastopol: O'Reilly Media.

4. Heinrich, R., van Hoorn, A., Knoche, H., Li, F., Lwakatare, L.E., Pahl, C., Schulte, S. and Wettinger, J. (2017) Performance Engineering for Microservices: Research Challenges and Directions, in Companion of the 8th ACM/SPEC *International Conference on Performance Engineering.* DOI: 10.1145/3053600.3053653.

5. Camilli, M., Guerriero, A., Janes, A., Russo, B. and Russo, S. (2022) Microservices Integrated Performance and Reliability Testing, in Proceedings of the *IEEE/ACM International Conference on Automation of Software Test.* DOI: 10.1145/3524481.3527233.

6. Vasilevskii, A., Kachur, O., Balsamo, S., Knottenbelt, W., Abad, C. and Shang, W. (2024) Self-Service Performance Testing Platform for Autonomous Development Teams, in Companion of the 15th ACM/SPEC *International Conference on Performance Engineering*. DOI: 10.1145/3629527.3652268.

7. Santos, L.B.R., de Souza, E.F., Endo, A.T., Trubiani, C., Pinciroli, R. and Vijaykumar, N.L. (2025) Performance regression testing initiatives: A systematic mapping, *Information and Software Technology*, 179, article 107641. DOI: 10.1016/j.infsof.2024.107641.

8. Fischer, S., Urbanke, P., Ramler, R., Steidl, M. and Felderer, M. (2024) An Overview of Microservice-Based Systems Used for Evaluation in Testing and Monitoring: A Systematic Mapping Study, in Proceedings of the *5th ACM/IEEE International Conference on Automation of Software Test*, pp. 1–11. DOI: 10.1145/3644032.3644445.

9. Ramu, V.B. (2023) Optimizing DevOps Pipelines with Performance Testing: A Comprehensive Approach, *International Journal of Computer Trends and Technology*, 71(6), pp. 35–41. DOI: 10.14445/22312803/IJCTT-V71I6P106.

10. Kovalenko, O.O. and Palamarchuk, Ye.A. (2020) Dynamic results of stress quarantine testing of the JetIQ VNTU learning management system and G

Suite for Education services', in Internet-Education-Science 2020: Proceedings of the *XII International Scientific and Practical Conference*, pp. 157–158.

11. Storozhuk, Yu.V. and Kovalenko, O.O. (2025) Method for determining target performance indicators of software, *Visnyk Vinnytskoho Politekhnichnoho Instytutu*, 4, pp. 108–117. DOI: 10.31649/1997-9266-2025-181-4-108-117.