УДК 004.056.55:004.312.2 **doi: 10.26906/SUNZ.2026.2.090**

О. В. Запорожець, П. А. Калашников

Харківський національний університет радіоелектроніки, Харків, Україна

АВТОМАТИЗОВАНА СИСТЕМА ТЕСТУВАННЯ ПРОДУКТИВНОСТІ ПРОГРАМНИХ СИСТЕМ: АРХІТЕКТУРА, МЕТРИКИ ТА ІНТЕГРАЦІЯ В CI/CD

Анотація. Актуальність. Продуктивність є однією з ключових характеристик якості програмних систем, що суттєво впливає на досвід користувача, надійність сервісів та витрати на інфраструктуру. З огляду на модель ISO/IEC 25010 (performance efficiency: time behavior, resource utilization, capacity) та практики DevOps CI/CD актуальною є автоматизація регулярних і відтворюваних навантажувальних перевірок із формальним рішенням pass/fail та пояснюваною діагностикою регресій. **Об'єкт дослідження:** автоматизована система тестування продуктивності програмних систем у конвеєрі CI/CD, що поєднує навантажувальні тести та спостережуваність (метрики/логи/трейси). **Мета статті:** розробити й обґрунтувати концепцію такої системи, визначити її архітектуру, модулі, порядок виконання тестового циклу та мінімально достатній набір метрик і порогових критеріїв «quality gate» на основі інженерних SLO. **Результати дослідження**. Запропоновано модульну архітектуру, описано процедуру виконання тестового циклу та підхід до формування метрик і baseline-порівнянь (ISO/IEC 25023, термінологія ISTQB) для автоматичного виявлення регресій продуктивності. **Висновки.** Інтеграція тестування продуктивності в CI/CD забезпечує раннє виявлення деградацій і зменшує ризик їх потрапляння у промислове середовище, а поєднання навантаження зі спостережуваністю підвищує пояснюваність причин погіршення.

Ключові слова: продуктивність; тестування продуктивності; навантажувальне тестування; CI/CD; метрики; спостережуваність; пороги; регресія продуктивності.

Вступ

Тестування продуктивності визначається як тестування, що виконується для визначення продуктивності (performance) програмного продукту; у термінології воно напряму пов'язане з оцінкою «performance efficiency» компонента або системи. Для сучасних програмних систем (особливо вебсервісів і мікросервісів) продуктивність є не лише «нефункціональною вимогою», а й фактором, що визначає масштабованість, стабільність під піковим навантаженням і економічність експлуатації [1–4].

У моделі ISO/IEC 25010 характеристика «performance efficiency» деталізується як ступінь, з яким продукт виконує функції в межах заданих часових і пропускних (throughput) параметрів та з урахуванням використання ресурсів і місткості (capacity) [1]. Це означає, що навіть за коректної функціональності програмна система може ставати неприйнятною для користувача через перевищення часу відповіді, істотну варіативність «хвоста» затримок або неефективність використання CPU / пам'яті / мережі.

Традиційна організація performance-перевірок часто має низку обмежень: запуск «перед релізом», ручне налаштування стендів, нерепрезентативне навантаження, складність відтворення результатів та відсутність автоматичного рішення «pass/fail» у конвеєрі постачання [5]. На цьому тлі інтеграція продуктивнісних перевірок у CI/CD розглядається як спосіб раннього виявлення деградацій і формування базових «еталонів» (baseline) для порівняння змін між збірками [6].

Аналіз наукових і практичних підходів. У стандартизованому підході до вимірювань важливо, щоб метрики були пов'язані з інформаційними потребами та приводили до дії, а не накопичення даних «заради даних». Це відповідає тезі ISO/IEC/IEEE 15939 про те, що процес вимірювання має бути цілеспрямованим і підтримувати визначення/застосування/удосконалення набору вимірювань у межах проєкту або організації [3]. Для метрик саме якості продукту стандарт ISO/IEC 25023 визначає формат документування та приклади застосування «quality measures» для кількісної оцінки характеристик (у т.ч. часових параметрів) і порівняння з вимогами або трендами [2].

З боку тестової практики термінологія ISTQB описує навантажувальне тестування (load testing) як різновид тестування продуктивності, що оцінює поведінку системи за різних навантажень (від низького до типового та пікового), а також вводить поняття «load profile» як документування кількості віртуальних користувачів, транзакцій та інтервалу часу, що відображає очікувану експлуатацію [4]. Це важливо для автоматизації: без формалізованого профілю неможливі ні відтворюваність, ні коректне порівняння результатів між збірками.

Сучасні open-source інструменти для навантажувального тестування підтримують сценарний підхід і запускаються у headless/CLI режимі, що спрощує інтеграцію в CI. Наприклад, Apache JMeter позиціонується як open-source продукт для навантажувального тестування та вимірювання продуктивності; інструмент має детальну специфікацію елементів тест-плану та підтримує віддалене (розподілене) виконання для збільшення генерованого навантаження [7]. З іншого боку, Grafana k6 орієнтований на розробницькі команди: сценарії описуються кодом (JavaScript), а механізм thresholds надає нативний «pass/fail» на основі заданих очікувань продуктивності [8].

Автоматизація продуктивнісного тестування в CI/CD зазвичай потребує ще одного класу технологій – спостережуваності та вимірювання стану системи під тестом. OpenTelemetry визначається як vendor-neutral, open-source підхід до збору та експорту telemetry-сигналів (traces, metrics, logs), що дозволяє корелювати поведінку запитів із ресурсними показниками та подіями [9]. Prometheus, як open-source система моніторингу, забезпечує збір та запити до часових

рядів (time series), включно з pull-моделлю збору [10]. Візуалізація та аналітика на практиці часто реалізуються через Grafana-дашборди як набір панелей, що відображають дані з джерел та дають «at-a-glance» картину стану системи [11].

Окремий сучасний напрям – автоматичне виявлення регресій продуктивності (performance regression testing). В оглядових роботах останніх років підкреслюється, що це активна область досліджень із багатьма викликами: варіативність середовищ, статистична надійність, вибір метрик та критеріїв «значущої» деградації [5]. Практичні рекомендації індустріальних документів для k6 підкреслюють роль baseline-порівнянь як «ядра» методології автоматизованого performance-тестування [12].

Постановка задачі дослідження. Метою статті є розробка та обґрунтування концепції автоматизованої системи тестування продуктивності програмних систем, яка:

1) формалізує вимоги до продуктивності через характеристику performance efficiency ISO/IEC 25010 (time behaviour/resource utilization/capacity) [1];

2) застосовує стандартизований підхід до вимірювань (measurement process) і вибору метрик [3];

3) інтегрується в CI/CD як quality gate із автоматичним рішенням pass/fail за порогами [8];

4) забезпечує відтворюваність тестового середовища та спостережуваність (correlation traces/metrics/logs) для пояснюваного аналізу причин деградації [9].

Для досягнення мети потрібно вирішити такі задачі:

1) визначити функціональні модулі системи та їх взаємодію в межах тестового циклу;

2) сформувати мінімально достатній набір метрик (SLI) та правила агрегування (наприклад, середнє/перцентилі) для часових характеристик і ресурсних параметрів;

3) описати механізм «baseline → порівняння → рішення» для виявлення регресій;

4) задати правила інтеграції у CI/CD (тригери запуску, клас тестів за вагою: smoke/peak/soak, планування ресурсів).

Результати дослідження

1. Архітектура та функціональні компоненти системи. Концептуальна структура автоматизованої системи тестування продуктивності складається з модулів оркестрації в CI/CD, генерації навантаження, профілів тестів, збору телеметрії, аналізу/quality gate та зберігання артефактів і baseline (рис. 1).

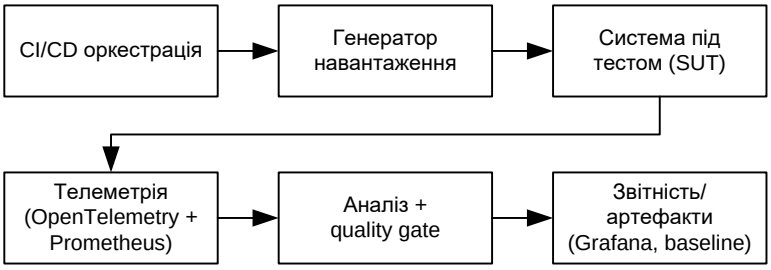

Рис. 1. Узагальнена архітектура автоматизованої системи тестування продуктивності

Модуль оркестрації (CI/CD-інтеграція). Точка входу – джоба конвеєра CI/CD, яка викликає:

1) розгортання тестового стенда;

2) запуск навантажувального тесту;

3) збір метрик/логів/трейсів;

4) оцінку порогів;

5) публікацію артефактів (звіт, дашборд, порівняння з baseline).

Потреба включати performance-тести до CI/CD як частину «глибини автоматизованого тестування» прямо відзначається у практичних рекомендаціях щодо CI/CD-best practices [6].

Модуль генерації навантаження. Як реалізаційні варіанти доцільні:

– Apache JMeter як інструмент, призначений для load-testing і вимірювання продуктивності; його модель test plan та наявність remote/distributed режимів дозволяє масштабувати генерацію віртуальних користувачів;

– Grafana k6 як інструмент, орієнтований на автоматизацію, з можливістю задавати thresholds як правила pass/fail та запускати сценарії в CI.

Вибір конкретного генератора навантаження можна трактувати як параметр системи: важливішою є стандартизація контрактів (вхідні параметри, артефакти, метрики) та можливість запуску з командного рядка.

Модуль профілів навантаження (test profiles). На основі ISTQB-понять «load profile» і «load management» профіль навантаження має бути артефактом, який:

а) формалізує кількість віртуальних користувачів/транзакцій;

б) визначає період/динаміку (ramp-up/steady-state/ramp-down);

в) зв’язує сценарії з очікуваними умовами експлуатації.

Для покриття різних ризиків доцільно підтримати щонайменше «peak/spike/soak» класи тестів (як поширені типи в load-testing практиці).

Модуль спостережуваності та збору телеметрії. Система повинна використовувати узгоджений канал збору:

– OpenTelemetry як стандартний, vendor-neutral спосіб отримувати traces/metrics/logs та корелювати їх у межах одного контексту запиту;

– Prometheus як механізм зберігання та запитів до метрик у часових рядах (інструментуван-

ня/скрейпінг/PromQL), що потрібні для аналітики під час тесту;

– Grafana як шар візуалізації та аналітики (дашборди як набір панелей).

Модуль відтворюваності середовища. Для зменшення шуму вимірювань середовище бажано описувати як інфраструктуру-як-код (IaC). Terraform як IaC-інструмент дозволяє описувати бажаний стан інфраструктури декларативно та керувати життєвим циклом ресурсів, що підтримує повторюваність стендів для тестів [13].

Модуль зберігання артефактів і baseline. Практика порівняння з baseline як «ядро» автоматизованого performance-тестування прямо підкреслюється у документації Grafana Cloud k6: baseline повинен бути репрезентативним, але не надмірно «важким», щоб залишатися стабільним референтом. Артефактами зберігання є: конфігурація тесту, raw-метрики, агрегати (перцентилі p50/p95/p99), звіти й рішення pass/fail.

Сукупно запропонована архітектура забезпечує:

1) автоматичне виконання;

2) вимірюваність;

3) відтворюваність;

4) пояснюваність результату через спостережуваність.

2. Методика вимірювання, метрики та критерій приймання. ISO/IEC/IEEE 15939 описує процес вимірювання як набір дій для визначення та застосування вимірювань, що закривають конкретні інформаційні потреби. Для продуктивності доцільно вважати інформаційними потребами: «чи витримує система очікуване навантаження», «який запас місткості», «чи з'явилася деградація після зміни», «де вузьке місце».

Модель ISO/IEC 25010 задає три підхарактеристики performance efficiency:

– time behaviour (час відповіді, час обробки, throughput);

– resource utilization (використання ресурсів);

– capacity (граничні можливості за параметрами). Тому система вимірювань має включати метрики часу/пропускної здатності та метрики ресурсів на рівні хоста/контейнера/процесу.

3. Базові метрики тестування продуктивності. Спираючись на ISO/IEC 25023, який задає «quality measures» та приклади інтерпретації (conformance і time series аналіз), пропонується мінімальний набір метрик для автоматизованої системи:

1) час відповіді та його розподіл. Середній час відповіді може бути визначений як

$$\bar{t} = \frac{1}{n}\sum_{i=1}^{n} t_i,$$

але для користувацького досвіду важливі також перцентилі (p95/p99), оскільки середнє може приховувати повільні запити в «хвості» розподілу. У CI/CD-контексті практичною є стратегія контролю p95/p99 як критеріїв SLO/thresholds;

2) пропускна здатність (throughput) і інтенсивність. У продуктивнісній моделі ISO/IEC 25010 time behavior включає throughput як частину вимог до часової поведінки системи. Для сервісів це може бути RPS (requests per second), TPS (transactions per second) або обсяг оброблених повідомлень за інтервал;

3) похибки та помилки (error rate). У межах автоматизованих порогів k6 thresholds прямо підтримує умови за error rate та latency-показниками, що дозволяє перетворити нефункціональні вимоги на формальні критерії «fail build»;

4) використання ресурсів. Prometheus визначає збір і запит до метрик як основу для dashboarding/alerting; для ресурсів це, як правило, CPU utilization, memory usage, network I/O, disk I/O;

5) місткість (capacity) і точки насичення. Capacity у ISO/IEC 25010 описує ступінь, з яким максимальні межі параметра системи задовольняють вимоги. На практиці це виражається як «максимальна кількість паралельних користувачів/запитів» при збереженні заданих SLO по latency/error rate.

4. Автоматичні критерії pass/fail та baseline-порівняння. Критерій k6 визначає thresholds як критерії pass/fail для тестових метрик: якщо умови не виконані, тест завершується зі статусом failure, що природно вписується в CI/CD як quality gate.

Для автоматизованого виявлення регресій пропонується комбінований підхід:

1) абсолютні пороги (SLO-пороги). Наприклад, p95 latency ≤ X мс, error rate ≤ Y%, throughput ≥ Z RPS (самі числа мають походити з вимог/очікувань або з емпіричного baseline);

2) відносні пороги до baseline. Порівняння «поточна збірка» vs «еталонна збірка» як методологічно ключовий елемент automated performance testing; documentation Grafana Cloud прямо описує baseline як контрольний тест для пошуку відмінностей;

3) трендовий контроль (time series). ISO/IEC 25023 наводить ідею time series порівнянь для того, щоб бачити зміни метрик у часі (наприклад, як змінюється mean response time протягом дня).

Критерії виявлення регресій при порівнянні з baseline наведено в табл. 1.

Таблиця 1 – **Критерії виявлення регресій при порівнянні з baseline**

Метрика	Критерій регресії	Примітка
Latency p95	Δ > +10%	Медіана із 3-х прогонів
Latency p99	Δ > +15%	Чутлива до «хвоста»
Throughput	Δ < –5%	Падіння пропускної здатності
Error rate	new > base + 0,2% або SLO	relative+hard
CPU/Memory	Δ > +10% (p95/max)	Неефективність / витік

Оскільки performance regression testing є чутливим до шуму й варіативності середовища, у сучасних оглядах підкреслюється необхідність уважного вибору методів порівняння та критеріїв значущості регресії.

Саме тому модуль відтворюваності середовища (IaC) та стандартизований збір метрик (Prometheus/OpenTelemetry) є не «додатком», а центральною вимогою системи.

Пороги для quality gate наведено в табл. 2.

Таблиця 2 – **Приклад порогів для quality gate (калібруються під систему та стенд)**

Метрика	Поріг/правило	Коментар
Latency p95	p95 ≤ 300 мс	для критичних endpoint-ів
Latency p99	p99 ≤ 800 мс	контроль «tail latency»
Error rate	Errors ≤ 0,5%	включно з timeout
Throughput	RPS ≥ 200 або Δ ≥ –5%	hard+relative
CPU avg	CPU ≤ 70% (warn)	gate після калібрування
Memory max	RAM ≤ 75% (warn)	запас для піків

5. Практичний сценарій виконання тестового циклу

Спрощену схему виконання циклу тестування продуктивності в CI/CD наведено на рис. 2.

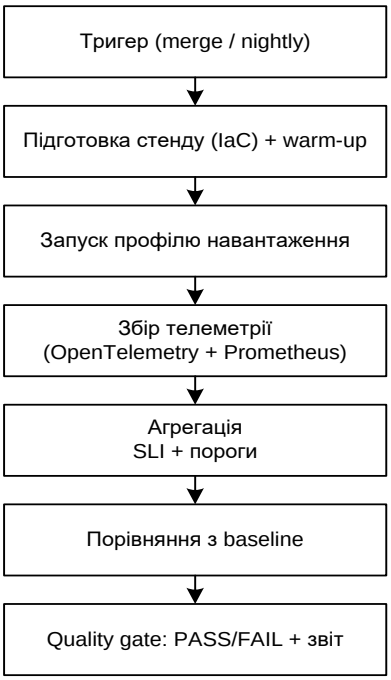

Рис. 2. Спрощена схема виконання циклу тестування продуктивності в CI/CD

Узагальнений цикл автоматизованого тесту продуктивності в CI/CD виглядає так:

1) розгортання тестового середовища (IaC) і прогрів (warm-up) для стабілізації кешів/компіляції;

2) запуск навантажувального профілю (load profile) відповідно до ISTQB-визначення;

3) генерація навантаження та паралельний збір телеметрії;

4) агрегація метрик та оцінка порогів thresholds (k6) або зовнішній «gating-модуль»;

5) порівняння з baseline (якщо увімкнено режим performance regression), оновлення сховища baseline за контрольованим правилом (наприклад, лише після релізного тегу);

6) публікація результатів: артефакти CI (лог, summary, JSON), посилання на дашборд Grafana, короткий висновок «pass/fail» і (за потреби) діагностичний зріз «який компонент/endpoint погіршився».

Артефакти та вихідні результати автоматизованого тесту продуктивності наведено в табл. 3.

Висновки

У статті обґрунтовано актуальність інтеграції тестування продуктивності в CI/CD як механізму раннього виявлення деградацій і формування керованих (threshold-based) quality gates. Показано, що стандартизоване розуміння «продуктивності» доцільно будувати на ISO/IEC 25010 (performance efficiency: time behavior/resource utilization/capacity), а підхід до вимірювань – узгоджувати з ISO/IEC/IEEE 15939.

Таблиця 3 – **Артефакти та вихідні результати автоматизованого тесту продуктивності**

Артефакт	Формат/приклад	Призначення
Конфігурація тесту	test plan, профіль	Відтворення запуску
Raw-результати	JSON/JTL + логи	Повторний аналіз
Агрегований звіт	HTML/Markdown	Висновок та ключові SLI
Telemetry snapshot	Grafana link/PNG	Діагностика причин
Baseline dataset	Версіонований набір	Порівняння збірок
Verdict у CI	PASS/FAIL + причина	Блокування регресій

Запропоновано модульну архітектуру автоматизованої системи, де навантажувальні інструменти (JMeter або k6) поєднуються з observability-стеком (OpenTelemetry + Prometheus + Grafana), що підвищує пояснюваність результатів і практичну керованість деградаціями.

Перспективи подальших досліджень у цьому напрямку включають:

1) статистично обґрунтовані методи виявлення регресій (з урахуванням варіативності середовища);

2) автоматичне формування «реалістичних» профілів навантаження на основі production-телеметрії;

3) розширення метрик до рівня бізнес-транзакцій і user-journeys;

4) оцінювання ефективності системи на реальних кейсах та формування практичних рекомендацій для команд розробки.

Конфлікт інтересів. Автори декларують, що не мають конфлікту інтересів стосовно даного дослідження, в тому числі фінансового, особистісного характеру, авторства чи іншого характеру, що міг би вплинути на дослідження та його результати, представлені в даній статті.

Використання засобів штучного інтелекту. Автори підтверджують, що не використовували технології штучного інтелекту при створенні представленої роботи.

Список літератури

1. ISO/IEC 25010:2011. Systems and software engineering. Systems and software Quality Requirements and Evaluation (SQuaRE). DOI: https://doi.org/10.3403/30215101. URL: https://www.iso.org/standard/35733.html.
2. ISO/IEC 25023:2016. Systems and software engineering. Systems and software Quality Requirements and Evaluation (SQuaRE) URL: https://www.iso.org/standard/35747.html.
3. ISO/IEC/IEEE 15939:2017. Systems and software engineering URL: https://www.iso.org/standard/71197.html.
4. ISTQB. Standard Glossary of Terms Used in Software Testing (Version 4.0). URL: https://glossary.istqb.org/.
5. Dos Santos L.B.R., Trubiani C., Pinciroli C., et al. Performance regression testing initiatives: a systematic mapping study. Information and Software Technology. 2024. DOI: https://doi.org/10.1016/j.infsof.2024.107641.
6. Gatling. Integrate performance testing into your CI/CD pipeline. URL: https://gatling.io/blog/performance-testing-ci-cd.
7. Matam S., Jain J. Pro Apache JMeter: Web Application Performance Testing. Apress, 2017. DOI: https://doi.org/10.1007/978-1-4842-2961-3.
8. Grafana Labs. k6 Documentation (running in CI; thresholds). URL: https://grafana.com/docs/k6/latest/get-started/running-k6/; https://grafana.com/docs/k6/latest/using-k6/thresholds/.
9. Blanco D. G. Practical OpenTelemetry: Adopting Open Observability Standards Across Your Organization. Apress, 2023. DOI: https://doi.org/10.1007/978-1-4842-9075-0.
10. Prometheus Authors. Prometheus Documentation. URL: https://prometheus.io/docs/introduction/overview/.
11. Sanches J., Pereira P. R. Network and Systems Monitoring with Prometheus and Grafana. CISTI 2025, Lecture Notes in Networks and Systems, vol. 1716. Springer, 2026. DOI: https://doi.org/10.1007/978-3-032-10929-3_32.
12. Grafana Labs. Grafana Cloud k6: Compare tests (test comparison). URL: https://grafana.com/docs/grafana-cloud/testing/k6/analyze-results/test-comparison/.
13. Kasarla N. K. Implementing Infrastructure as Code (IaC) with Terraform for Scalable Cloud Deployments. Journal of Information Systems Engineering and Management. 2025. DOI: https://doi.org/10.52783/jisem.v10i60s.13257.
14. Humble J., Farley D. Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation. Addison-Wesley, 2010. DOI: https://doi.org/10.5555/1869904.

Received (Надійшла) 19.01.2026
Accepted for publication (Прийнята до друку) 29.04.2026
Publication date (Дата публікації) 22.05.2026

Відомості про авторів / About the Authors

Запорожець Олег Васильович – кандидат технічних наук, доцент, доцент кафедри електронних обчислювальних машин, Харківський національний університет радіоелектроніки, Харків, Україна;
Oleg Zaporozhets – PhD, Associate Professor, Associate Professor of Department of Electronic Computers, Kharkiv National University of Radio Electronics, Kharkiv, Ukraine;
e-mail: oleg.zaporozhets@nure.ua; ORCID Author ID: http://orcid.org/0000-0002-7831-8479;
Scopus Author ID: https://www.scopus.com/authid/detail.uri?authorId=15728942500

Калашников Павло Андрійович – студент кафедри електронних обчислювальних машин, Харківський національний університет радіоелектроніки, Харків, Україна;
Pavlo Kalashnykov – student, Department of Electronic Computers, Kharkiv National University of Radio Electronics, Ukraine;
e-mail: pavlo.kalashnykov@nure.ua; ORCID Author ID: https://orcid.org/0009-0000-2054-0037.

Automated performance testing system for software applications: architecture, metrics, and ci/cd integration

O. Zaporozhets, P. Kalashnykov

Abstract. Relevance. Performance is one of the key characteristics of software system quality, significantly affecting user experience, service reliability, and infrastructure costs. Considering the ISO/IEC 25010 model (performance efficiency: time behavior, resource utilization, capacity) and DevOps CI/CD practices, it is important to automate regular and reproducible load tests with a formal pass/fail decision and explainable regression diagnostics. **Object of research**. An automated system for testing the performance of software systems in the CI/CD pipeline, combining load tests and observability (metrics/logs/traces). **Purpose of research.** To develop and justify the concept of such a system, define its architecture, modules, test cycle execution order, and a minimally sufficient set of metrics and quality gate threshold criteria based on engineering SLOs. **Research results.** A modular architecture is proposed, the test cycle execution procedure and the approach to forming metrics and baseline comparisons (ISO/IEC 25023, ISTQB terminology) for automatic detection of performance regressions are described. **Conclusions.** Integrating performance testing into CI/CD enables early detection of degradations and reduces the risk of them entering the production environment, while combining load with observability increases the explainability of the causes of degradation.

Keywords: performance, performance testing, load testing, CI/CD, metrics, observability, thresholds, performance regression.