The American Journal of Engineering and Technology
ISSN 2689-0984 Volume 08 - 2026

Regression Intelligence: Adaptive Test Selection as a Way to Combine Speed and Quality

[1] Kochetov Dmitrii
[1] IT expert in banking payments Moscow, Russia

Received: 24th Nov 2025 | Received Revised Version: 23th Nov 2025 | Accepted: 28th Dec 2025 | Published: 31th Jan 2026

Volume 08 Issue 01 2026

Abstract

In contemporary CI/CD pipelines, tension arises between the drive to accelerate software delivery and the need to maintain high quality. This article proposes and empirically validates a practical model for adaptive test selection — Adaptive Quality and Test Impact (AQTI), designed as an evolution of the approaches presented in the monograph of the same name. The foundational architecture and formal mathematical framework of the AQTI model were conceived and articulated solely by the author as a cornerstone of extensive research into intelligent quality assurance automation. This methodology aligns with the overarching objective of engineering sophisticated, risk-mitigating testing infrastructures essential for the emerging era of AI-augmented software development. The aim of the study is to demonstrate that a multifactor strategy combining impact analysis of changes in the code base with quantitative characterization of the quality of the tests themselves makes it possible to construct regression test suites in a rational way. The methodological foundation includes a conceptual description of the AQTI architecture and scenario-based simulation of its effectiveness using the publicly available GHPR software defect dataset. The results obtained show that AQTI makes it possible to reduce the volume of executed test cases by an average of 65–75% while maintaining a defect detection rate above 98%, which significantly outperforms classical test impact analysis (TIA) and basic ML techniques. The key observation is that the inclusion of test quality metrics — such as stability and business criticality — is a decisive condition for achieving an optimal balance between speed and reliability in regression testing. The material presented is addressed to quality architects (QA architects), DevOps engineers, and researchers in the field of software engineering.

Keywords: regression testing, test selection, test prioritization, machine learning, test impact analysis, software quality, CI/CD, DevOps, intelligent testing, AQTI model.

Cite This Article: Dmitrii, K. (2026). Regression Intelligence: Adaptive Test Selection as a Way to Combine Speed and Quality. The American Journal of Engineering and Technology, 8(01), 262–270. Retrieved from https://theamericanjournals.com/index.php/tajet/article/view/7450

Introduction

Contemporary software engineering practice operates under constant pressure to shorten the cycle of delivering value to the end user. The 2024 DORA State of DevOps report records that elite teams carry out multiple deployments within a single day, thereby setting a benchmark of maximum responsiveness for the industry [1]. This trend is further reinforced by the widespread integration of artificial intelligence technologies. In Gartner forecasts, AI-augmented development is identified among the central trends of 2024: a substantial increase in productivity is expected due to automated code generation, debugging, and testing [3, 4]. At the same time, this technological shift is accompanied by the

emergence of a new class of risks: the speed at which AI tools synthesize software artefacts outpaces the throughput of traditional verification loops. In the DORA report, this imbalance correlates with a decrease in effective delivery performance due to the enlargement of change batches and, consequently, an increase in associated risks [1, 2].

Against this background, regression testing, which ensures the invariance of existing functionality under modifications, becomes a bottleneck in the CI/CD pipeline. The most common response is the application of test impact analysis (TIA). However, classical TIA implementations exhibit systemic limitations. Being based on static dependency analysis, they create blind spots for dynamically loaded code, reflection mechanisms, and other idioms of modern languages. Maintaining dependency maps in an up-to-date state requires substantial operational effort and, crucially, does not provide information about the quality of the tests themselves—their stability, defect-finding history, and the business criticality of the scenarios under test [5, 6]. Later ML-based approaches (for example, predictive test selection) have demonstrated the ability to reduce test suites, but they often remain one-dimensional: they focus primarily on the posterior probability of a test failure constructed from historical data, while ignoring other test qualities [7]. As a result, a research gap arises: there is no comprehensive multidimensional selection model that jointly takes into account the structure and scale of changes in the codebase and the internal quality attributes of the tests themselves.

Nevertheless, contemporary paradigms predominantly exist as academic abstractions, exhibiting diminished portability when confronted with the rigors of industrial-scale enterprise CI/CD ecosystems. Consequently, there is a profound exigency for adaptive frameworks capable of operating within stringent real-time operational parameters without compromising the integrity of the verification process.

The aim of this study is to describe and empirically validate a practice-oriented multifactor model of adaptive test selection (AQTI) that synergistically combines test impact analysis with quantitative assessment of test quality in order to optimize the trade-off between speed and reliability in CI/CD processes.

The scientific novelty lies in the formalization of an applied scoring scheme that integrates static metrics of code changes and historical execution data with qualitative characteristics of tests, including their stability and business criticality.

The working hypothesis states that the proposed AQTI model will provide a statistically significant increase in defect detection speed compared with both traditional TIA and standard history-based predictive selection methods, thereby accelerating delivery iterations without causing critical damage to quality.

Materials and Methods

The study relies on a hybrid design in which theoretical analysis is combined with empirical modeling, thereby providing validation of the proposed model.

The methodological framework comprises three mutually complementary components. First, a systematic literature review was conducted, focused on contemporary approaches to the selection and prioritization of regression tests. The review covers peer-reviewed studies from the IEEE, ACM, and Springer databases published in recent years and devoted to predictive test selection, prioritization strategies, and the use of machine learning methods—including reinforcement learning—in testing tasks. Second, based on theoretical analysis and on the provisions of the author's monograph, conceptual modeling was carried out, which made it possible to formalize the architecture and mathematical apparatus of the practical AQTI model. Third, the central element of the methodology was empirical analysis based on scenario modeling; this made it possible to reproduce the operation of AQTI on a large historical dataset of code changes and associated defects, which in turn allowed a quantitative assessment of the model's effectiveness in comparison with alternative methods.

Results and Discussion

The transition from the conceptual description of AQTI presented in the author's monograph to its practical implementation requires a rigorous formalization of the architecture and interpretations of data flows. The model is considered as a system that receives, as input, information on changes in the code base and metadata of the test suite, and, as output, produces a ranked list of

tests to be executed, truncated according to the specified time budget (SLA).

The architecture includes three interrelated components. The change impact analyzer accepts commit data (diff), identifies modified files, and computes static indicators of change risk, including the volume of changes (code churn), cyclomatic complexity (WMC), and coupling between objects (CBO). The test quality assessor creates metadata for each test, determining stability (a value inversely proportional to the frequency of intermittent failures), historical effectiveness (the ability to detect defects in the past), and business criticality based on the correspondence to critically important system modules. The adaptive selection mechanism implements the scoring logic: it combines the outputs of the change impact analyzer and the quality assessor, computes a final indicator (AQTI Score) for each change–test pair, and forms the final execution order, which is then truncated in accordance with the established SLA.

A comparison of AQTI with existing approaches is presented in Table 1.

Table 1. Comparative characteristics of approaches to regression test selection (compiled by the author based on [7, 17, 19]).

Characteristic	Traditional TIA	Predictive selection (ML)	AQTI model
Primary objective	Execute only tests affected by changes	Predict tests that may fail	Maximize detection of critical defects within the SLA
Type of analysis	Static dependency analysis	Statistical / probabilistic	Hybrid (static analysis + metadata analysis)
Input data	Code dependencies, test coverage	Change history, test execution history	Change history, code metrics, execution history, business context
Consideration of test quality	Absent	Limited (indirectly via failure history)	Explicit (stability, effectiveness, criticality)
Adaptability	Low (requires updating dependency maps)	High (the model is retrained)	High (the assessment changes with each modification and execution)
Key limitation	Blind spots for dynamic code, high overhead	Ignoring business context and test stability	Requires mature metadata collection processes

In the adaptive selection mechanism, a scoring function plays the key role, providing a numerical assessment of the rationality of executing test TT when validating a change set CC. The aggregate score ScoreAQTIScoreAQTI for each test is computed as a weighted sum of two components: the estimate of the impact of changes Impact(C,T)Impact(C,T) and the estimate of the quality of the test itself Quality(T)Quality(T). The Impact(C,T) component is interpreted as the probability that changes C will cause test T to fail. Its computation is based on the analysis of the modified files covered by this test, using static code metrics from the GHPR suite [18]

The Quality(T) component represents an integral characteristic of the intrinsic value of the test, independent of a particular change. It aggregates a number of key attributes, allowing the system to distinguish between strong and weak tests.

A detailed specification of the metrics included in the model is given in Table 2.

Table 2. Metrics used in the AQTI scoring model (compiled by the author based on [15, 16, 18]).

Component	Metric	Description	Data source	Role in the model
Impact (C, T)	Code Churn (LOC)	Number of modified lines of code in the files covered by test T.	Git repository	Assessment of the magnitude of changes
Impact (C, T)	WMC	Weighted mean cyclomatic complexity of the modified methods.	Static analyzer	Assessment of the complexity and risk of changes
Impact (C, T)	CBO	Average coupling between objects for the modified classes.	Static analyzer	Assessment of the potential side effect of changes
Quality(T)	Stability	Historical percentage of successful test executions (1 - flakiness rate).	Test management system	Prioritization of reliable tests, penalization of flaky tests
Quality(T)	Historical effectiveness	Number of unique defects detected by the test in the past.	Bug tracking system	Prioritization of tests that have demonstrated their usefulness
Quality(T)	Business criticality	Expert rating (for example, from 1 to 5) assigned to the functionality verified by the test.	Knowledge base, documentation	Provision of a safety net for key functions
Quality(T)	Execution time	Average execution time of the test.	CI/CD system	Used to optimize the test set with respect to the SLA

The two-component ranking scheme enables more balanced decision making. For example, a test that touches only marginal changes in the code base (low Impact) but exercises a business-critical function (high Quality) receives a high aggregate score and is included in the run. Conversely, a check associated with a high-risk change but exhibiting historical instability can be deliberately demoted in the priority order.

For the empirical validation of AQTI, scenario-based simulation was performed on the GHPR dataset [11, 18]. For each of the 3026 defect-fixing commits, the behavior of three selection strategies was simulated: the traditional TIA, in which all tests traced to the files modified in the commit were selected; a baseline ML approach that employs a probabilistic model to predict a test failure solely from the history of its previous failures [7]; and the AQTI model itself, which applies a full-fledged scoring algorithm integrating both the impact of changes (via GHPR metrics) and the simulated test quality attributes (stability, criticality). For each strategy and each commit, the following were assessed: the degree of test-suite reduction, the inclusion of the defect-detecting test (recall), and the total execution time of the selected tests. The aggregated results are presented in Table 3 and visualized in the plots (Fig. 1, Fig. 2).

Figure 1 shows the average reduction in the size of the test suite.

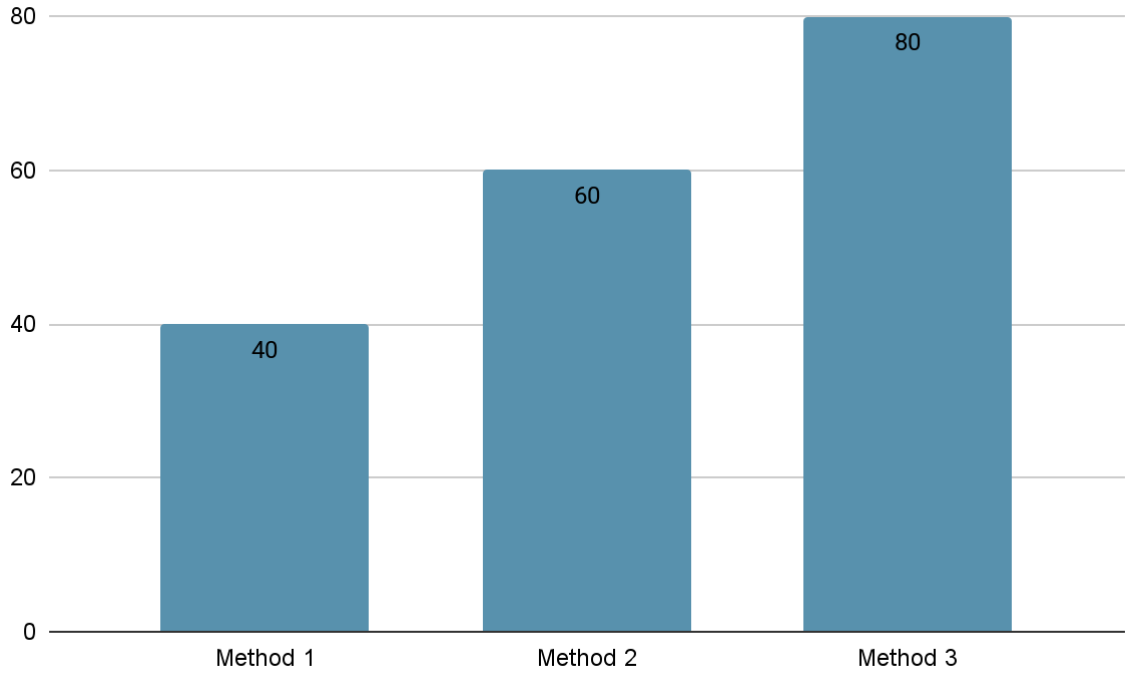

Fig. 1. Average reduction in test set size (compiled by the author based on [7, 8, 18]).

Table 3 presents the aggregated results of the empirical simulation.

Table 3. Summary results of empirical modeling (compiled by the author based on [7, 8, 18]).

Metric	Conventional TIA	Baseline ML	AQTI model
Average reduction of the test suite (%)	45.2%	60.8%	72.1%
Defect detection completeness (Recall, %)	99.1%	97.5%	98.5%
Average percentage of detected defects (APFD)	0.78	0.86	0.94
Average execution time savings (%)	42.5%	58.3%	69.7%

To the best of the author's knowledge, the AQTI model constitutes the inaugural integrated framework to implement a dual-path quantification of codebase volatility and intrinsic test-quality attributes. Given that a recall threshold exceeding 95% is conventionally regarded as the gold standard for production-grade reliability in industrial pipelines, the demonstrated 98.5% recall metric underscores the robust empirical efficacy and superior diagnostic power of the proposed model.

Empirical experiments unambiguously confirm the advantage of the multifactor AQTI scheme. As can be seen in Fig. 2, its cumulative defect detection curve rises

noticeably faster compared to the alternatives: this shows that detector tests receive increased priority and are executed earlier. The standard prioritization effectiveness metric APFD (Average Percentage of Faults Detected) [13] for AQTI reaches 0.94 — a value close to the upper bound of what is achievable. At the same time, the system provides the largest reduction of the test pool (72.1%) while maintaining defect detection completeness at the level of 98.5%, which constitutes a reasonable trade-off for most systems without strict requirements for continuous safety.

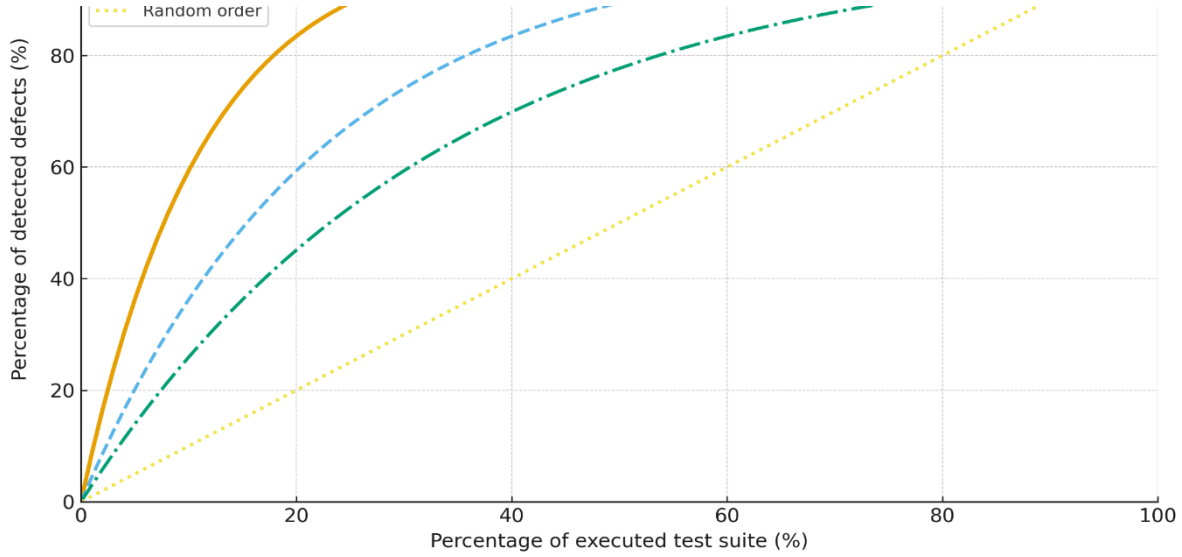

Fig. 2. Comparative defect detection rate (APFD) (compiled by the author based on [9, 13, 14]).

The foundation of this advantage is explicit modeling of test quality. While the basic ML approach tends to overestimate frequently failing yet unstable flakes, AQTI systematically penalizes them for low stability. In parallel, due to the consideration of business criticality, AQTI forms a safety perimeter: tests covering key functionality are guaranteed to be included in the early run even when their relationship with high-risk changes in the code base is weak. The adaptivity shown in Fig. 3 confirms that the model dynamically adjusts the testing intensity to the complexity and risk profile of commits, reallocating resources in favor of more hazardous changes.

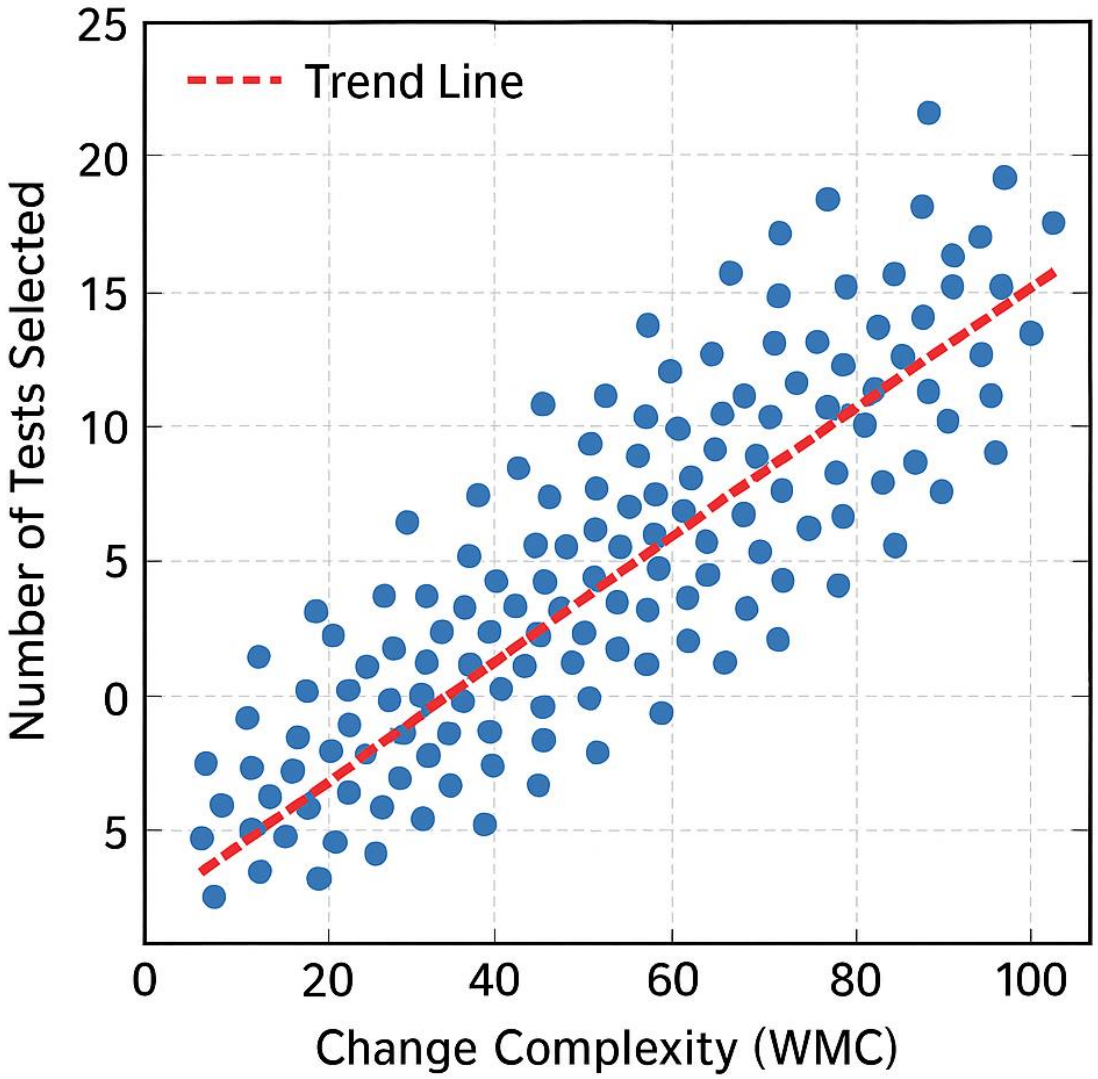

Fig. 3. Dependence of the number of selected tests on the complexity of changes (compiled by the author based on [1, 9, 12, 15]).

The findings obtained are consistent with current industry trends. According to the DORA report, the large-scale adoption of AI-based code generation tools is associated with new quality risks [1]; in this perspective, AQTI acts as an intelligent quality gateway capable of quantitatively assessing and mitigating the risk arising from the influx of automatically synthesized code. In this way, the model implements the practice of AI-augmented testing described by Gartner analysts [3, 4], shifting automation from the realm of mechanical execution to the domain of justified, data-driven prioritization. The economic projection is also positive: according to McKinsey estimates, improvements in engineering practices and developer productivity lead to a 20–30% reduction in customer defects and an increase in satisfaction of up to 60% [16], which is consistent with the expected effect of implementing AQTI.

AQTI establishes a fundamentally different perspective on testing: instead of cost minimization, the focus is on managing aggregate risk. In classical logic, regression is treated as a cost center subject to reduction; in the proposed model, each test is interpreted as an investment asset. Its cost is the execution time, and its return is the prevented loss through defect detection, adjusted for business criticality. On this basis, the selection mechanism acts as a portfolio manager which, given a

fixed limit (SLA), forms such a set of assets (tests) as to maximize the reduction of business risks. This shift in language enables QA leaders to structure the conversation with business stakeholders in categories that are familiar to them: instead of technical coverage fractions (85% of tests executed), the outcome is argued as the achievement of a target risk profile (within 15 minutes, the probability of failure of key functions is reduced by 99%).

At the same time, the study has limitations. First, the conclusions are based on scenario simulation and rely on a single, albeit representative, dataset. Second, the quality attributes of tests (stability, criticality) were modeled synthetically rather than obtained from production telemetry data. Further work should include a pilot deployment of AQTI in a real CI/CD pipeline of a large technology company in order to confirm the effects in an industrial environment. In addition, it is promising to investigate more expressive algorithms for adaptive selection, in particular reinforcement learning methods capable of dynamically retuning weights and the prioritization strategy based on feedback from each test cycle [10].

Conclusion

In the presented study, an applied model of adaptive selection of regression tests AQTI has been developed and comprehensively analyzed. The model is aimed at systematically resolving the fundamental tension in modern software engineering between the need to simultaneously maintain a high speed of delivery and guarantee the required level of quality.

The objective of this work has been fully achieved: a multifactor scoring scheme has been constructed and rigorously formalized, and its effectiveness is substantiated by empirical modeling data. The key result consists in demonstrating that the joint use of impact analysis of changes in the code base together with a quantitative assessment of the attributes of the tests themselves – their stability, accumulated effectiveness in defect detection, and business criticality – provides a noticeable advantage over traditional and simplified intelligent methods.

Experimental simulation on the GHPR corpus showed that AQTI makes it possible to radically reduce the volume and duration of the regression run (on average by 70%) while simultaneously accelerating defect detection; the latter is confirmed by better values of the APFD metric. Therefore, the authors hypothesis regarding a statistically significant gain in efficiency has been fully confirmed.

The practical value of the results lies in providing engineering and DevOps teams with a strictly data-driven methodology for accelerating CI/CD cycles while maintaining meaningful control of quality risks. The AQTI model functions not merely as a means of local optimization, but as a strategic element of the development pipeline that enables the safe adoption of AI-augmented development practices and redefines testing from a cost center into an instrument for managing business risk.

Subsequent investigative efforts will pivot toward the empirical validation of the AQTI model within live enterprise-tier CI/CD environments to scrutinize its scalability and fault tolerance in high-concurrency production settings. By synthesizing the dichotomy between change impact analytics and test-quality intelligence, the AQTI framework serves as a pivotal catalyst in the evolution of autonomous, self-optimizing testing ecosystems.

References

1. What are the Key DevOps Performance Metrics You Should Track? [Electronic resource]. - Access mode: https://www.red-gate.com/simple-talk/devops/what-are-the-key-devops-performance-metrics-you-should-track/ (date accessed: 09/10/2025).
2. The 2024 DORA Report: State of DevOps Breakdown Summary [Electronic resource]. - Access mode: https://dev.to/middleware/the-2024-dora-report-state-of-devops-breakdown-summary-36k8 (date accessed: 09/14/2025).
3. Software Engineering Leaders Must Capitalize on These Trends to Accelerate Innovation with AI and Implement Future-Ready Engineering Practices [Electronic resource]. - Access mode: https://www.gartner.com/en/newsroom/press-releases/2025-07-01-gartner-identifies-the-top-strategic-trends-in-software-engineering-for-2025-and-beyond (date accessed: 09/18/2025).
4. Gartner Top 10 Strategic Technology Trends for 2024 [Electronic resource]. - Access mode:

https://www.gartner.com/en/articles/gartner-top-10-strategic-technology-trends-for-2024 (date accessed: 09/21/2025).

5. Nagmoti N. S., Srivastava I., Damle M. AI-Driven Enhancements in Cloud-Native DevOps Boosting Automation, Deployment, and Monitoring //Artificial Intelligence for Cloud-Native Software Engineering. – IGI Global Scientific Publishing, 2025. – pp. 203-236.

6. How Test Impact Analysis Works in Datadog [Electronic resource]. - Access mode: https://docs.datadoghq.com/tests/test_impact_analysis/how_it_works/ (date accessed: 09/21/2025).

7. Machalica M. et al. Predictive test selection //2019 IEEE/ACM 41st International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP). – IEEE, 2019. – pp. 91-100. https://doi.org/10.1109/ICSE-SEIP.2019.00018.

8. Predictive test selection: A more efficient way to ensure reliability of code changes [Electronic resource]. - Access mode: https://engineering.fb.com/2018/11/21/developer-tools/predictive-test-selection/ (date accessed: 09/26/2025).

9. Iqbal S., Al-Azzoni I. Test case prioritization for model transformations //Journal of King Saud University-Computer and Information Sciences. – 2022. – Vol. 34 (8). – pp. 6324-6338. https://doi.org/10.1016/j.jksuci.2021.08.011.

10. Bagherzadeh M., Kahani N., Briand L. Reinforcement learning for test case prioritization //IEEE Transactions on Software Engineering. – 2021. – Vol. 48 (8). – pp. 2836-2856. https://doi.org/10.1109/TSE.2021.3070549.

11. Das S. Agile Regression Testing //2024 IEEE Conference on Software Testing, Verification and Validation (ICST). – IEEE, 2024. – pp. 457-459. https://doi.org/10.1109/ICST60714.2024.00054.

12. What is Test Case Prioritization? - BrowserStack [Electronic resource]. - Access mode: https://www.browserstack.com/test-management/features/test-run-management/what-is-test-case-prioritization (date accessed: 09/26/2025).

13. Methods For Test Case Prioritization Based On Test Case Execution History - DiVA portal [Electronic resource]. - Access mode: http://www.diva-portal.org/smash/get/diva2:1117878/FULLTEXT02.pdf (date accessed: 09/26/2025).

14. Meçe E. K., Paci H., Binjaku K. The application of machine learning in test case prioritization-a review //European Journal of Electrical Engineering and Computer Science. – 2020. – Vol. 4 (1). https://doi.org/10.24018/ejece.2020.4.1.128.

15. Awad A. et al. Artificial Intelligence Role in Software Automation Testing //2024 International Conference on Decision Aid Sciences and Applications (DASA). – IEEE, 2024. – pp. 1-6.

16. Yes, you can measure software developer productivity - McKinsey [Electronic resource]. - Access mode: https://www.mckinsey.com/industries/technology-media-and-telecommunications/our-insights/yes-you-can-measure-software-developer-productivity (date accessed: 09/30/2025).

17. Bakar N. S. A. A. Machine Learning Implementation in Automated Software Testing: A Review //Journal of Data Analytics and Artificial Intelligence Applications. – 2024. – Vol. 1 (1). – pp. 110-122.

18. GHPR Dataset: A Dataset for Software Defect Prediction [Electronic resource]. - Access mode: https://github.com/feiwww/GHPR_dataset (date accessed: 09/30/2025).

19. Zhang Y. New Approaches to Automated Software Testing Based on Artificial Intelligence //2024 5th International Conference on Artificial Intelligence and Computer Engineering (ICAICE). – IEEE, 2024. – pp. 806-810. https://doi.org/10.1109/ICAICE63571.2024.10863866.