International Journal of Computer (IJC)

ISSN 2307-4523 (Print & Online)

https://ijcjournal.org/index.php/InternationalJournalOfComputer/index

AI Approaches to Software Quality Assessment: From Defect Prediction to Test Coverage Optimization

Taras Buriak*

Software Development Engineer in Test,Austin, Texas

Email:tarasburyak@gmail.com

Abstract

The article addresses the problem of growing inefficiency in traditional approaches to software quality assurance (QA) under accelerated cycles of continuous integration and delivery (CI/CD). The aim of the study is to present and evaluate a new integrated platform that leverages large language models (LLMs) to automate and optimize key QA processes. The methodology is based on an industrial case study of a system utilizing Claude 4 and Amazon Q CLI for semantic log analysis and predictive test prioritization. The paper presents key quantitative results, including an increase in nightly build stability from ~70% to over 90%, a one-third reduction in regression testing time, and a halving of pull request verification time. Additionally, it is emphasized that the adoption of such solutions enhances testing transparency, improves collaboration between development and QA teams, and accelerates release cycles while ensuring higher product quality. The main contribution of the article is the provision of empirical evidence from a large corporate environment, confirming that modern AI-driven approaches can significantly improve the efficiency, accuracy, and strategic value of software testing. The article will be useful for researchers, quality engineers, CI/CD architects, and DevOps practitioners interested in applying LLMs to optimize testing processes.

Keywords : AI; large language models; quality assurance; test automation; predictive prioritization; semantic log analysis; CI/CD; empirical study.

Received: 9/19/2025
Accepted: 11/19/2025
Published: 11/29/2025

----

* *Corresponding author.*

1. Introduction

Modern software development operates under a dual pressure. On the one hand, the complexity of systems is constantly increasing, including microservice architectures, APIs, and distributed environments. On the other, business requirements dictate the need to accelerate product release cycles. Agile and DevOps methodologies, which have become the industry standard, demand faster and more reliable testing cycles that traditional approaches can no longer provide [1]. Manual testing and outdated script-based automation methods are becoming significant bottlenecks in the development process, characterized by long execution times and high maintenance overheadA clear example of this problem is the experience where full regression test runs took 10–12 hours, critically slowing down the feedback loop [8].Artificial intelligence (AI) and machine learning (ML) represent a paradigm shift in the field of quality assurance. Key areas of AI application include automatic test case generation, defect prediction, and intelligent test prioritization [3]. Recent breakthroughs, such as generative AI and large language models (LLMs), are opening up unprecedented opportunities for automating complex intellectual tasks in QA [4].

Despite extensive academic research, there is a noticeable lack of detailed empirical studies based on real-world industrial implementations. Many scientific works are experimental in nature and do not account for the practical complexities associated with scaling and integrating such systems into existing production processes. According to industry surveys, interest in AI-driven testing is high, yet the actual adoption rate remains low [11]. This gap between theoretical potential and documented industrial application is the central problem this article aims to address.

The main thesis of this article is as follows: this work describes the design, implementation, and empirical evaluation of a novel, LLM-based platform for software quality assessment in a large-scale industrial environment. By integrating semantic failure analysis and predictive test prioritization, this system demonstrates a significant increase in testing efficiency and product stability, thereby offering a validated model to bridge the gap between academic research and industrial practice.

The objective of this study is to analyze the possibilities of applying large language models in software quality assurance processes, to develop an integrated platform for semantic failure analysis and predictive test prioritization, and to conduct its empirical evaluation in a large-scale industrial environment to identify the practical advantages and limitations of such solutions.

2. Participants, Materials, and Methods

This research is an industrial case study conducted within the large corporate environment of a major airline company. The software ecosystem is characterized by high complexity and includes native desktop applications, mobile applications, and web platforms [7]. Before the implementation of the described system, the quality assurance process faced several critical problems that served as serious obstacles to accelerating development cycles.

The main pain points included long regression testing times, low build stability, and high manual labor costs for failure analysis [4]. Nightly regression test suites, covering the entire system's functionality, ran for 10–12 hours, leading to significant delays in receiving feedback. The stability of nightly builds was at a problematic level of around 70%, which meant frequent failures not due to actual defects but because of test instability or environmental issues. Furthermore, quality assurance engineers had to spend a significant amount of time manually analyzing hundreds of failure reports, many of which were duplicates, diverting resources from more important tasks [7].

To address the aforementioned problems, a centralized, event-driven system was designed and implemented. The central element of the architecture is the MCP server, which acts as an orchestrator. Its main functions include receiving requests from the CI/CD pipeline, distributing analytical tasks to the appropriate AI services, and managing the caching of results. The system's technology stack includes the MCP server, the Cypress automated testing framework, the Allure reporting tool, the Amazon Q CLI for interacting with AWS services, and the Claude 4 large language model for semantic analysis [2].

The first key component of the system is aimed at automating the process of analyzing and classifying failures. After a test cycle is completed, the MCP server automatically collects the logs of all failed tests and passes them for processing to a combination of Amazon Q CLI and Claude 4 tools. The Claude 4 model conducts a semantic analysis of each failure to determine its root cause. Based on this analysis, the system performs an intelligent grouping of failures by semantic similarity rather than by syntactic match. The result is a report in which, instead of hundreds of individual errors, a QA engineer sees 3-4 clearly defined groups with a description of the common problem [13].

The second key component is aimed at accelerating the verification of pull requests (PRs). When a developer opens a new PR, the MCP server analyzes the PR's content, including the changed files and commit messages. The system queries a database with the history of all previous failures and compares it with the current changes. The Claude 4 model analyzes the natural language of the commit messages to understand the developer's intent. Based on this analysis, the system generates a reduced but highly relevant set of tests that is run to provide fast feedback.

This study has several limitations that should be taken into account. The findings are based on a single industrial case within the studied airline's ecosystem, which restricts the generalizability of the results. The platform relies on proprietary LLM services, creating reproducibility and vendor-dependency constraints. The performance of the system depends on the quality and completeness of internal failure logs and commit data, which may limit the applicability of the approach in environments with insufficient or noisy historical datasets. In addition, the evaluation was conducted in a real production setting rather than under controlled experimental conditions, so long-term causal effects require further validation in broader empirical studies.

3.Results

The implementation of the described LLM platform led to significant and measurable improvements in the quality assurance process. The quantitative results obtained during the system's operation demonstrate a comprehensive positive impact on the entire software development lifecycle. The main performance metrics before and after the system's implementation are summarized in Table 1.

Table 1: System Performance Metrics Before and After Implementation (Compiled by the author based on sources: [3,7,10])

Metric	**Before Implementation (Traditional Approach)**	**After Implementation (LLM-Based Platform)**	**Improvement**
Stability of nightly builds	Success rate ~70%	Success rate >90%	Increase of more than 20 percentage points
Time for full regression testing	X hours (e.g., 12 hours)	~67% of X (e.g., 8 hours)	Reduction by one third
Pull request verification time	Y minutes	~50% of Y	Reduction by approximately half
Number of defects in production	Baseline level	Significantly less	Qualitative improvement
Release frequency	Standard	Significantly fewer	Qualitative improvement
QA team effort distribution	High costs for manual log analysis	Focus on designing complex tests and improving the system	Shift from reactive to proactive tasks

The data from the table show that the proposed platform not only solved the initial problems but also achieved qualitative improvements in areas such as release frequency and the strategic allocation of the QA team's efforts. For a more detailed understanding of the transformation of operational processes, a comparative analysis of key work aspects before and after the system's implementation was conducted, the results of which are presented in Table 2.

Table 2: Comparative Analysis of QA Workflows (Compiled by the author based on sources: [5, 7, 9])

Process Aspect	**Traditional Approach**	**LLM-Based Approach**
Failure analysis (Triage)	Manual review of hundreds of individual logs; grouping by exact error match.	Automatic semantic analysis; intelligent grouping by root cause; report consolidated into 3–4 clusters.
Test prioritization for PR	Execution of the full regression test suite (10–12 hours) or manual test selection.	Predictive analysis of changes and commits; generation of a targeted test set; execution in ~50% of the time.
Role of QA engineer	Reactive failure analysis, maintenance of unstable tests, routine work with reports.	Proactive system improvement, design of complex tests, management of the AI orchestrator.
Feedback cycle	Delay until the next working day (for nightly builds).	Rapid feedback on PRs; prompt identification of systemic failure causes.

In addition to the presented tables, the implementation results demonstrate a qualitative transformation of the entire testing ecosystem. The reduction in regression run times and PR verification times led to faster release cycles, allowing business units to bring updates to market more quickly. The increased stability of nightly builds reduced the level of false positives, and the decrease in the number of defects in production enhanced end-user trust in the system [13]. The reallocation of QA engineers' efforts toward proactive tasks strengthened the quality team's role as a strategic partner in development, rather than just an executor of routine operations. Collectively, this confirms that the use of an LLM platform has a systemic, not just a localized, impact on the performance and stability of development processes.

4.Other Recommendations

The results demonstrate a direct link between the system's architectural decisions and the improvements achieved. The reduction in pull request verification time by approximately 50% is a direct consequence of the predictive test prioritization component. The algorithm, which uses an LLM to analyze the semantics of commits, makes it possible to forgo a full regression run in favor of executing a small but highly relevant set of tests. In turn, the increase in the stability of nightly builds from ~70% to over 90% is directly related to the automatic semantic grouping of failures component [7]. By providing QA engineers with a concise and meaningful report on the causes of failures, the system allows for faster identification and elimination of unstable tests. The cumulative effect of these improvements is expressed in accelerated release cycles, higher product quality, and more efficient use of engineering resources [12].

The results obtained indicate a fundamental transformation of the role of the Software Development Engineer in Test (SDET). The work described in the case study does not involve creating a separate AI model, but rather integrating and orchestrating several AI services through a centralized platform [1]. This requires a skill set that goes beyond traditional script writing and includes designing distributed systems, API integration, and prompt engineering. This evolution of the role aligns with the forecasts of analysts such as Gartner, who predict the emergence of autonomous "AI agents" capable of independently performing tasks throughout the entire development lifecycle [9]. To ensure a balanced perspective, it is necessary to critically evaluate the limitations of the study and the general challenges in the field of applying AI to QA. Table 3 summarizes the key challenges discussed further below.

Table 3: Key Challenges and Limitations in Implementing AI in QA (Compiled by the author based on sources: [6, 13])

Challenge Category	**Description**	**Consequences and Mitigation Strategies**
Vendor dependency	The use of proprietary models (Claude 4, Amazon Q) creates risks of vendor lock-in and dependence on pricing policies.	Exploration and transition to open-source models to increase control and reduce costs.
Data security	Transferring logs and code to external servers introduces risks of confidential information leakage.	Implementation of strict data security policies; consideration of local (on-premise) model deployment.
Interpretability (“Black box”)	Difficulty in understanding the decision-making logic of LLMs, which complicates debugging and ensuring predictability.	Introduction of human-in-the-loop validation mechanisms; development of metrics for monitoring model behavior.
Data quality	Model effectiveness directly depends on the quality of historical failure and commit data (“garbage in – garbage out”).	Establishing processes to ensure data cleanliness and completeness; regular reassessment and cleansing of training datasets.

The main limitation of the presented system is its dependence on proprietary, closed-source commercial models, which creates a trade-off between capabilities and control [6]. This approach is associated with data security and confidentiality risks, economic risks, and the "black box" problem. Furthermore, this study represents a single case, and its results may not be fully generalizable to other organizations.

In addition to the specific limitations of the project, there are also broader challenges. First and foremost is data quality and dependency. The effectiveness of AI models directly depends on the completeness and accuracy of historical data [5]. Furthermore, integration and the skills gap represent a complex technical and organizational challenge. AI systems also create new security risks, such as prompt injection. Finally, the bias and reliability of models require attention, as AI can inherit biases from training data, and LLMs are prone to "hallucinations" [3].

5.Conclusion

The main contribution of this article lies in the successful design, implementation, and validation of an integrated, LLM-based platform for quality assurance in a real industrial environment. The work represents a rare example of a detailed description of the industrial application of advanced AI technologies, thereby bridging the gap between academic research and practice. Significant, quantitatively measurable improvements in key development process metrics were demonstrated: the stability of nightly builds increased by more than 20 percentage points, regression testing time was reduced by a third, and pull request verification time was halved. The presented study serves as convincing empirical evidence that modern large language models can be effectively used to transition from traditional test automation to a more intelligent, predictive, and efficient quality engineering paradigm.

Based on the conducted research and the identified limitations, several promising directions for future work can be outlined. One of the most important steps is the investigation of open-source models, which would solve the "proprietary black box" problem. A logical development of the platform is a move toward self-healing automation, where an LLM will analyze failures and attempt to automatically correct test scripts. Furthermore, the proposed architectural pattern can be applied in related areas, such as performance or security testing. Finally, to increase the generalizability of the conclusions, broader empirical studies in other organizations are necessary.

References

[1]. Ding, Y. (2024). Artificial Intelligence in Software Testing for Emerging Fields: A Review of Technical Applications and Developments. Applied and Computational Engineering, 112(1), 161–175. https://doi.org/10.54254/2755-2721/2025.18116

[2]. Narwal. (2024). Revolutionizing Software Testing: The Role of AI Powered Quality Engineering in Driving Innovation. Narwal. https://narwal.ai/revolutionizing-software-testing-the-role-of-ai-powered-quality-engineering-in-driving-innovation/

[3]. Trudova, A., Dolezel, M., & Buchalcevova, A. (2020). Artificial Intelligence in Software Test Automation: A Systematic Literature Review. In Proceedings of the 15th International Conference on Evaluation of Novel Approaches to Software Engineering (ENASE 2020), Volume 1 (pp. 181–192). SCITEPRESS. https://doi.org/10.5220/0009417801810192

[4]. IEEE Computer Society. (2025). Next-Generation Software Testing: AI-Powered Test Automation. IEEE Software, 42(4), 25–33. https://www.computer.org/csdl/magazine/so/2025/04/11024091/27gSQcKD6jC

[5]. Ideas2IT. (n.d.). AI's Role in Enhancing Quality Assurance in Software Testing. Retrieved September 10, 2025, from https://www.ideas2it.com/blogs/ai-redefining-qa-software-testing

[6]. Battina, D. S. (2019). Artificial Intelligence in Software Test Automation: A Systematic Literature

Review. International Journal of Emerging Technologies and Innovative Research, 6(12), 1329–1332. http://www.jetir.org/papers/JETIR1912176.pdf

[7]. Baqar, M., & Khanda, R. (2024). The Future of Software Testing: AI-Powered Test Case Generation and Validation. arXiv. https://doi.org/10.48550/arXiv.2409.05808

[8]. Islam, M., Khan, F., Alam, S., & Hasan, M. (2023, September). Artificial Intelligence in Software Testing: A Systematic Review. In Proceedings of the IEEE TENCON 2023 Conference (pp. 1–6). IEEE. Retrieved from https://www.researchgate.net/publication/374263724_Artificial_Intelligence_in_Software_Testing_A_Systematic_Review

[9]. Karhu, K., Kasurinen, J., & Smolander, K. (2025). Expectations vs Reality -- A Secondary Study on AI Adoption in Software Testing. arXiv. https://doi.org/10.48550/arXiv.2504.04921

[10]. Khalid, A., Badshah, G., Ayub, N., Shiraz, M., & Ghouse, M. (2023). Software Defect Prediction Analysis Using Machine Learning Techniques. Sustainability, 15(6), 5517. https://doi.org/10.3390/su15065517

[11]. Ponnala, R., & Reddy, C. R. K. (2021). Software Defect Prediction using Machine Learning Algorithms: Current State of the Art. Solid State Technology, 64(2), 6541–6556. Retrieved from https://www.researchgate.net/publication/351746329_Software_Defect_Prediction_using_Machine_Learning_Algorithms_Current_State_of_the_Art

[12]. Albattah, W., & Alzahrani, M. (2024). Software Defect Prediction Based on Machine Learning and Deep Learning Techniques: An Empirical Approach. AI, 5(4), 1743–1758. https://doi.org/10.3390/ai5040086

[13]. Harsh, H., Santoshi, H., & Singh, P. (2025). Comparative Study of Machine Learning Based Defect Prediction Models for Python Software. In Proceedings of the 2025 6th International Conference on Inventive Research in Computing Applications (ICIRCA). IEEE. https://doi.org/10.1109/ICIRCA65293.2025.11089647