



# AI-Driven Automation of Code Review Processes: Enhancing Software Quality and Reducing Human Error

Evgenii Lvov

Head of Engineering, Senior Full Stack Architect, Batumi, Georgia.

## Abstract

*In contemporary software engineering, expert code review practices have entered a phase of profound reconsideration under the influence of generative artificial intelligence technologies. In 2024–2025, a qualitatively new, exponential stage of integrating large language models (LLM) into the software development life cycle (SDLC) is being observed, which radically changes the balance between development speed, quality assurance, and the security of software systems. The aim of the study is to provide a comprehensive assessment of the effectiveness of using AI to automate Code Review processes, to analyze how such technological interventions modify software quality metrics, and to identify latent risks conditioned by the human factor. The focal point is the phenomenon of the productivity paradox: the acceleration of code writing with AI assistants leads to the review and deployment stages becoming the bottleneck, where the throughput of the team in fact decreases. Based on quantitative indicators, it is demonstrated that the introduction of AI correlates with a 7,2% decrease in delivery stability and an increase in architectural technical debt, while developers themselves subjectively interpret what is happening as an increase in their own productivity. Particular emphasis is placed on a comparative analysis of traditional static application security testing (SAST) tools and LLM agents, on identifying specific vulnerabilities induced by neural network models (including the impact of politically charged triggers on code security), as well as on examining the cognitive effects of AI use for experienced software engineers. It is shown that experts may lose up to 19% of their working time when involving AI in solving complex tasks due to the need for additional verification and correction of contextual model hallucinations. The article proposes a scientifically grounded typology of errors generated by AI and formulates recommendations for transitioning to agentic workflows in which AI functions not only as a generator of code fragments, but also as an interactive verifier of developer intentions, operating in a mode of close human–machine synergy.*

**Keywords:** Digital Transformation, Organizational Management, Information Technologies, Management Efficiency, Digital Platforms, Business Processes, Data Analysis, Innovative Development.

## INTRODUCTION

Historically, code review has been one of the most resource-intensive—and at the same time one of the most critical—stages of the software development life cycle. Classical approaches based on manual inspection and asynchronous reviews via pull requests (PRs) served as the primary gate preventing defects from reaching production. The rapid diffusion of generative AI assistants in software engineering has changed the economics of code production and shifted attention from “writing code” to “validating and integrating code.” In practice, as AI tools are increasingly embedded into development workflows, the bottleneck is less the ability to generate code and more the human capacity to read, interpret, and verify model-produced changes under real project constraints [1].

As a result, a paradigm shift is observed: emphasis moves from the act of coding to reviewing, meaningful integration, and orchestration of artifacts produced by both developers and models. This shift measurably increases the load on senior engineers and team leads, because the overall flow of changes becomes larger and requires more intensive verification. In the 2024 Stack Overflow Developer Survey, 76% of respondents reported that they use or plan to use AI tools in their development process (61.8% already use them, and 13.8% plan to soon), indicating that AI-assisted development has moved from experimentation to mainstream practice [4]. Under these conditions, automation of code review is no longer an optional improvement but a structural requirement for maintaining throughput and engineering reliability.

**Citation:** Evgenii Lvov, “AI-Driven Automation of Code Review Processes: Enhancing Software Quality and Reducing Human Error”, Universal Library of Innovative Research and Studies, 2025; 2(2): 39-45. DOI: <https://doi.org/10.70315/uloap.ulirs.2025.0202006>.

The relevance of the problem is determined both by the maturity of generative models and by the depth of their penetration into industrial development processes. According to the Stack Overflow Developer Survey 2024, 72% of respondents reported a favorable or very favorable attitude toward AI tools for development [4]. However, by 2025, the same survey series records a clear shift toward pragmatic skepticism: positive sentiment falls to ~60%, even as AI usage continues to grow [5]. This combination—wider adoption alongside weakening enthusiasm—suggests that teams increasingly experience the practical costs of AI integration (verification overhead, error correction, and governance requirements) rather than viewing AI purely as a productivity accelerator.

Survey data also highlight a “productivity–trust” tension. In 2024, professional developers most frequently described the expected benefit of AI tools as increasing productivity (82.7%) and speeding up learning (60.8%) [4]. At the same time, trust in AI outputs remains limited: developers are split on accuracy, with 43% reporting that they “feel good” about AI accuracy while a substantial share remains skeptical [4]. Moreover, almost half (45%) of professional developers rate AI tools as bad or very bad at handling complex tasks—precisely the category of work where mistakes are costly and review demands are highest [4]. At the organizational level, DORA’s 2024 findings reinforce this nuanced picture: while a 25% increase in AI adoption is associated with improvements such as +3.1% code review speed and modest gains in code and documentation quality, the same increase is linked to an estimated –1.5% decrease in delivery throughput and –7.2% reduction in delivery stability [7]. In other words, perceived local acceleration does not automatically translate into improved system-level delivery outcomes.

Against this background, a specific crisis of trust emerges that directly intensifies the burden on code review. In the 2024 Stack Overflow survey, the most frequently cited team-level challenges to using AI code assistants include “don’t trust the output or answers” (66.2%) and lack of codebase context (63.3%) [4]. This creates a form of cognitive dissonance: AI tools are adopted to save time, yet their outputs require heightened scrutiny, contextual checking, and corrective iterations—partially neutralizing the expected productivity gains and further overloading the code review stage.

The aim of the study is to identify and substantiate organizational-managerial and technological solutions that ensure an increase in the effectiveness of organizational management on the basis of the introduction and development of digital technologies.

The scientific novelty of the study lies in the development of an integrated model of the digital transformation of an organization’s management system, which includes:

- a refinement of the conceptual apparatus of digital transformation of management and digital maturity of an organization;
- a justification of the criteria and indicators for evaluating the effectiveness of managerial decisions in the context of digitalization;

The author’s hypothesis is based on the assumption that the implementation of an integrated model of digital transformation of an organization’s management system, grounded in the reengineering of key business processes and the use of data analytics, leads to a statistically significant increase in the effectiveness of managerial decisions and an improvement in the main socio-economic performance indicators of the organization.

### MATERIALS AND METHODS

In preparing the study, the method of systematic literature review was used, supplemented by a meta-analysis of industrial data. Such a combined design made it possible to combine academic evidentiality with large-scale industrial empirics. The chronological frame of the study covers the period from 2023 to the first quarter of 2025, which ensures a focus on contemporary state-of-the-art models, including GPT-4o, Claude 3.5 Sonnet, Gemini 1.5 Pro, and DeepSeek-V3, and makes it possible to evaluate them under conditions of current practical use.

The empirical base was formed from three complementary classes of sources, providing a balance between scientific rigor and practical relevance. First, academic publications (IEEE/ACM/arXiv) were used: peer-reviewed articles and preprints were analyzed that examine the empirical properties of LLMs in tasks of code generation and code review. Special emphasis was placed on works studying interactive workflows (for example TICODER), comparative studies of code quality in Human vs LLM settings, as well as psychological and cognitive aspects of developer interaction with AI. Recourse to journals such as IEEE Transactions on Software Engineering and to materials of leading conferences such as ICSE and FSE serves as a guarantee of the validity and representativeness of the results used. Second, industrial analytics and reports were used: data from global surveys and telemetry from technology companies and consulting agencies were analyzed, including DORA 2024 (Google Cloud) reports, GitHub Octoverse 2024, the McKinsey study The State of AI, as well as security reports from CrowdStrike and Apiiro. These materials provide a macro-level view of the role of AI in production processes and reflect the behavior of systems in the wild (in-the-wild). Third, technical benchmarks and leaderboards were used: for an objective assessment of model accuracy, results from the Hallucination Leaderboard were analyzed, as well as performance on the HumanEval and MBPP datasets and on the specialized SWR-Bench (Software Review Benchmark).

## RESULTS AND DISCUSSION

One of the central research questions is the analysis of the extent to which LLMs are able to compete with classical static analysis tools (SAST) in solving code review tasks. Traditional static analysis based on strictly predefined rules (SAST) is characterized by complete determinism of results, but at the same time demonstrates limited sensitivity to the broader system context and interrelations between components. In contrast, LLMs possess a pronounced ability for semantic interpretation of program code, yet are inevitably associated with the risks of hallucination generation and the stochastic nature of outputs.

An empirical study based on the Quarkus (Java) project

revealed substantial discrepancies in code quality assessments produced by SonarQube and by models of the GPT family. In particular, SonarQube classified 95,49% of classes into the category with a maintainability rating A (the maximum level), whereas the GPT-4o model systematically exhibited optimistic bias, assigning high scores to code fragments in which latent structural defects persisted. At the same time, more recent data on GPT-4.0 show a noticeable increase in model precision (Precision = 0.79) compared to the open model DeepSeek-V3 (Precision = 0.42) in detecting specific code smells such as Change Preventers. Table 1 presents a comparative analysis of the effectiveness of different approaches to code review.

**Table 1.** Comparative characteristics of the effectiveness of code review methods (compiled by the author based on [7; 15-18]).

Characteristic / Tool	SonarQube (Static Analysis)	GPT-3.5 Turbo / Llama 3	GPT-4o / Claude 3.5 Sonnet	Agent-based systems (Graphite, CodeRabbit)
Methodology	Deterministic rules, AST parsing	Probabilistic generation, limited context	Multimodal analysis, deep semantics	RAG, multi-step reasoning, integration with Git
Level of analysis	Syntax, style, known vulnerabilities	Code snippets, local logic	Classes, modules, cross-file dependencies	Context of the entire PR, commit history
False positive rate (False Positives)	Low (< 1.1% in an enterprise environment)	High (tendency to hallucinations)	Medium/Low (with appropriate prompting)	Optimized (5-15%)
Detection of logical errors	Practically absent	Limited by context window	High (understanding of business logic)	High (alignment with requirements)
Detection of complex vulnerabilities	Limited by signature database	Medium (skipping architectural issues)	High (data flow analysis)	Very high (semantic security analysis)
Impact on review speed	Instant feedback	Requires time for generation and verification	Depends on API latency	Asynchronous review, saving human effort

From the data presented in Table 1, it follows that agent architectures deployed on top of high-performance LLMs effectively attempt to synthesize the strengths of classical static analyzers and generative models. As a result, they succeed in reducing the share of false positives to a range of about 5-15%, which can already be considered a practically acceptable level for operation in industrial environments and for use as full-fledged engineering assistants.

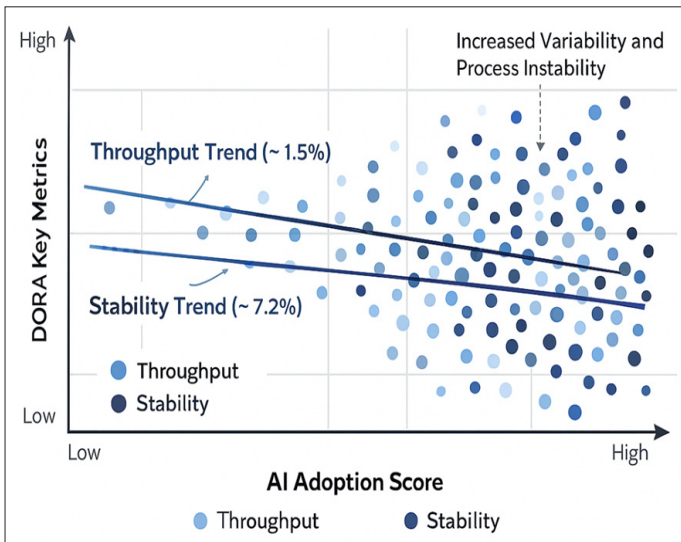
Although an intuitive belief has taken root in the professional community that AI tools unambiguously accelerate development, the accumulated empirical data indicate a substantially more complex, nonlinear picture. According to the DORA 2024 report, an increase in the degree of AI adoption in an organization is statistically associated with a reduction in team throughput. This seemingly paradoxical dynamic is conveniently interpreted through the concept of bottleneck shift: the limiting resource is no longer artifact generation, but the subsequent stages of the lifecycle [7, 8].

When developers begin actively using Copilot-class tools,

the rate of code production rises sharply. Telemetry data record a 98% increase in the number of Pull Requests, while the average time to review them increases by 91%.<sup>8</sup> Human cognitive capacities involved in analyzing, reviewing, and reconciling changes do not scale as easily as computational resources for generation. As a result, teams become overloaded with a rapidly growing volume of code that must be inspected, understood, and safely integrated.

Additionally, a GitClear study covering 150 million lines of code recorded an alarming shift: an increase in the Code Churn metric and a simultaneous decline in code reuse metrics (Code Reuse) [3]. In other words, AI tools push developers toward continual generation of new fragments instead of systematic refactoring and consolidation of the existing codebase. This leads to violation of the DRY (Don't Repeat Yourself) principle, fragmentation of logic, and an overall increase in the entropy of the software system.

Below, Figure 1 presents the dynamics of software delivery metrics as a function of the level of AI adoption



**Fig. 1.** Dynamics of software delivery indicators depending on the level of AI implementation (compiled by the author based on [3, 18]).

Automation of code review processes using AI gives rise to a new class of risks related both to the quality of the generated artifacts and to the security of the models themselves. The 2025 Apiiro report records an explosive growth in the number of vulnerabilities in code created with AI involvement: over a six-month period, the volume of new security issues increased by a factor of 10. The use of AI radically transforms the structure of typical software defects. Whereas humans are predominantly prone to syntactic errors, typos, and local logical mistakes, AI models as a rule generate code that is formally impeccable from the standpoint of syntax, yet may contain deep architectural vulnerabilities. Empirical studies show a reduction in the share of syntactic errors by approximately 76%, but at the same time record an increase in the number of architectural defects and potential privilege escalation points by 322%. Thus, the error shifts from the level of visible syntactic flaws to the level of system design

and threat models, where it is much more difficult to detect using standard linters and tests [9, 10].

LLM hallucinations, that is, the confident generation of factually incorrect or nonexistent information, remain a fundamental problem. In the context of Code Review, this manifests itself, in particular, in recommendations to use methods or packages that do not exist in the target ecosystem. Such suggestions create a new attack vector: an attacker can anticipate the development of the situation and register a package with the name invented by the model, thereby implementing a classical Typosquatting scheme in a disguised form. According to McKinsey, the hallucination rate for leading models is estimated as follows: GPT-4.1 — 5.6%; DeepSeek-V3 — 5.3–5.5%; Claude 3.5 Sonnet — on the order of 3–4% (estimate based on similar architectures); Google Gemini-2.5-flash-lite — 3.3%.<sup>21</sup> Even an apparently low rate on the order of 3% represents an unacceptable risk for safety-critical systems in the absence of strict human oversight and formalized verification procedures.

A separate, less intuitive vulnerability channel was identified in a CrowdStrike study: the influence of politically charged context on the quality and security of generated code. In experiments with the DeepSeek-R1 model, it was shown that including topics that are politically sensitive for the model developers (for example, mention of Tibet) increases the probability of generating vulnerable code by almost 50%, from 19% to 27.2%. This effect demonstrates the fragility of safety alignment mechanisms: the behavior of the model can be significantly shifted through manipulation of the request context, which opens the possibility of purposefully bypassing safety constraints without directly modifying the model parameters.

Below, Table 2 presents a comparative analysis of types of errors made by humans and by AI.

**Table 2.** Comparative analysis of types of errors made by humans and AI (compiled by the author based on [3, 18]).

Error Type	Human-Written Code (Relative Level)	AI-Generated / Verified Code (Relative Level)	Change vs. Human	Interpretation
Syntactic Errors	High	Very Low	-76%	AI tools drastically reduce basic syntax mistakes.
Logical Errors	Medium	Medium	-60%	Logic bugs decrease but still remain a noticeable portion of defects.
Architectural Vulnerabilities	Low	Very High	+153%	AI shifts errors toward deeper, harder-to-detect design and security issues.
Secrets Leakage / Misconfiguration	Medium	High	Increase (no exact % given)	AI code more often contains exposed secrets or risky configuration patterns.

One of the most paradoxical findings of the studies conducted in 2024–2025 was the discovery of a specific effect of AI tools on the work of highly qualified engineers. The METR

experiment demonstrated that, when solving realistic applied tasks with the support of advanced programming assistants (Cursor Pro, Claude 3.5), experienced open-

source developers on average performed their work 19% more slowly compared to the control group working without AI involvement [14].

The key mechanism behind this slowdown is associated with a sharp increase in cognitive load arising during the verification of generated code. In essence, the developer finds themselves in the position of a reviewer: they have to parse and interpret someone else's code (and code produced by an AI is functionally equivalent to code written by another person), which on average requires more effort than constructing one's own solution. Constant context switching between the original task, model suggestions, and the final implementation, as well as the need for continuous factual checking and detection of subtle defects, lead to the accumulation of mental fatigue and a decrease in work pace. At the same time, the subjective picture is radically distorted: participants felt that the use of AI had accelerated them by approximately 20%, whereas objective measurements showed a statistically significant slowdown.

On the other hand, the results of the TICODER study indicate that a well-designed organization of the workflow can substantially reduce the cognitive load on developers. In particular, the use of an interactive test generation approach redirects engineers' attention away from direct analysis of

the source code toward the evaluation of the observable behavior of the software artifact through a test system, which, in turn, leads to a significant increase in the Pass@1 Accuracy metric—on average by 45.97% over five iterations [12].

To overcome the fundamental limitations of single LLM instances, the industry is gradually shifting toward multi-agent architectures. The SWR-Bench (Software Review Benchmark), which includes a corpus of 1,000 manually verified pull requests, shows that modern single models are insufficiently effective for full-fledged code review, systematically losing important design and architectural context. At the same time, applying a strategy of aggregating multiple reviews makes it possible to compensate for this drawback, providing an increase in F1-score of 43.67%. Systems such as Graphite Agent and CodeRabbit implement precisely this approach: they take over the stage of preliminary analysis of changes, their condensed summarization, and the filtering out of trivial or low-significance comments, leaving to the human expert only the consideration of high-level architectural decisions.

Table 3 below will present the impact of the introduction of AI agents on Code Review metrics.

**Table 3.** Impact of the introduction of AI agents on Code Review metrics (compiled by the author based on [12, 14]).

Metric	Value / Change	Interpretation
Share of human replies to the agent's comments	56%	High engagement; developers read and respond
Share of code changes following the agent's review	18%	Only every fifth remark results in an actual modification
Developer sentiment (positive)	36%	A significant proportion perceives benefits
Developer sentiment (negative)	8%	Low level of rejection; the majority are neutral (56%)
Impact on PR closure time	Increase (in the short term)	Bottleneck effect due to the increased volume of reviews
Documentation quality	Increase of ~7.5%	Agents effectively enforce commenting on and describing PRs

Taken together, the results presented demonstrate that LLMs and multi-agent systems built on top of them are already capable of complementing, but not replacing, classical SAST tools in code review tasks, while altering both the distribution of defects and the organizational dynamics of development. On the one hand, modern models (of the GPT-4 class) and agentic solutions (Graphite, CodeRabbit) exhibit high semantic sensitivity, reliably identify complex logical and architectural vulnerabilities, reduce the share of false positives to an industrially acceptable 5–15%, and significantly outperform both rule-based analyzers in understanding business logic and open models in the accuracy of detecting specific code smells. On the other hand, large-scale adoption of AI tools leads to a shift of the bottleneck from artifact generation to the review and integration stages: an increase is observed in the number of PRs, the time required to process them, code churn, and a decrease in code reuse, which undermines the DRY principle

and degrades software delivery stability metrics. At the same time, AI radically transforms the error profile—from syntactic issues to deep architectural and configuration vulnerabilities exacerbated by hallucinations and context-induced failures of safety alignment mechanisms, thereby creating new attack vectors (hallucinated packages, politically charged prompts) [11, 13]. Finally, the impact of AI on the productivity of skilled engineers turns out to be ambivalent: the direct use of assistants without revisiting the process increases cognitive load and may slow down work, whereas process and methodological adaptations (for example, interactive test generation and multi-agent aggregation of reviews) make it possible to partially compensate for these effects by shifting expert attention from manual code analysis to the verification of behavior and high-level architectural decisions. Consequently, AI-based code review tools should be regarded not as an autonomous replacement for human expertise and static analysis, but as a layer of intelligent

automation whose effectiveness and safety directly depend on the quality of their integration into engineering practices and systemic verification mechanisms.

### CONCLUSION

The analysis carried out allows us to conclude that in 2024–2025 AI-based Code Review automation has moved beyond experimental deployments and entered a phase of critical rethinking and purposeful structural integration into engineering processes. The technology has demonstrated the ability to profoundly transform the software development landscape; however, the shift accompanying this transformation has exposed a number of serious systemic challenges.

First, the phenomenon of the productivity paradox has been empirically confirmed: unrestricted and weakly regulated use of code generators leads to overload of verification loops, reduces the actual throughput of teams, and increases the time to recover from failures and regression incidents. AI serves as an effective acceleration tool for junior engineers, raising their productivity and the quality of their solutions to the level of a notional average specialist, but at the same time it can turn into an inertial factor for experts, especially when solving atypical, weakly formalizable tasks.

Second, the qualitative profile of the software being created is itself changing. The decrease in the number of trivial, syntactic, and mechanical errors is accompanied by a relative increase in the share of complex semantic, architectural, and system-integration defects, as well as the emergence of new classes of security risks caused by model hallucinations and vulnerability to adversarial influences on them.

Third, the trajectory of further automation development is associated not with the complete displacement of humans from the loop, but with the formation of hybrid agentic ecosystems. Such systems should internalize routine operations — verification and coordination of documentation, generation and updating of tests, initial analysis of vulnerabilities and compliance with security policies — while leaving engineers the functions of decision-making in the areas of architecture, design trade-offs, and business logic. A key condition for sustainable adoption is the transition from the Human-in-the-Loop paradigm to Human-on-the-Loop. This, in turn, implies an increase in trust in the tools by reducing the level of false positives and increasing the transparency of algorithmic decisions, including the possibility of their interpretation and audit.

### REFERENCES

1. The state of AI: How organizations are rewiring to capture value. Retrieved from: <https://www.mckinsey.com/capabilities/quantumblack/our-insights/the-state-of-ai> (date accessed: March 20, 2025).
2. How an AI-enabled software product development life cycle will fuel innovation. Retrieved from: [www.mckinsey.com/industries/technology-media-and-telecommunications/our-insights/how-an-ai-enabled-software-product-development-life-cycle-will-fuel-innovation](https://www.mckinsey.com/industries/technology-media-and-telecommunications/our-insights/how-an-ai-enabled-software-product-development-life-cycle-will-fuel-innovation) (date accessed: March 5, 2025).
3. Yetiştiren, B., Özsoy, İ., Ayerdem, M., & Tüzün, E. (2022, May). Evaluating the code quality of AI-assisted code generation tools: An empirical study on GitHub Copilot, Amazon CodeWhisperer, and ChatGPT. In Proceedings of the 19th International Conference on Mining Software Repositories (MSR '22) (pp. 658–669). Association for Computing Machinery. <https://doi.org/10.1145/3524842.3528454>.
4. AI | 2024 Stack Overflow Developer Survey. Retrieved from: <https://survey.stackoverflow.co/2024/ai> (date accessed: February 10, 2025).
5. Octoverse: AI leads Python to top language as the number of global developers surges. Retrieved from: <https://github.blog/news-insights/octoverse/octoverse-2024/> (date accessed: February 6, 2025).
6. Hamza, M., Siemon, D., Akbar, M. A., & Rahman, T. (2024, April). Human-AI collaboration in software engineering: Lessons learned from a hands-on workshop. In Proceedings of the 7th ACM/IEEE International Workshop on Software-intensive Business (IWSiB '24). <https://doi.org/10.1145/3643690.3648236>.
7. Announcing the 2024 DORA report. Retrieved from: <https://cloud.google.com/blog/products/devops-sre/announcing-the-2024-dora-report> (date accessed: January 22, 2025).
8. Almeida, Y., Albuquerque, D., Dantas Filho, E., Muniz, F., & Perkusich, A. (2024). AI CodeReview: Advancing code quality with AI-enhanced reviews. *SoftwareX*, 26, 101677. <https://doi.org/10.1016/j.softx.2024.101677>.
9. Peng, S., Kalliamvakou, E., Cihon, P., & Demirel, M. (2023). The impact of AI on developer productivity: Evidence from GitHub Copilot (arXiv:2302.06590). <https://doi.org/10.48550/arXiv.2302.06590>.
10. Simões, I. R. D. S., & Venson, E. (2024, November). Evaluating source code quality with large language models: A comparative study. In Proceedings of the XXIII Brazilian Symposium on Software Quality (pp. 103–113). <https://doi.org/10.1145/3701625.3701650>.
11. Tosi, D. (2024). Studying the quality of source code generated by different AI generative engines: An empirical evaluation. *Future Internet*, 16(6), 188. <https://doi.org/10.3390/fi16060188>.
12. Fakhoury, S., Naik, A., Sakkas, G., Chakraborty, S., & Lahiri, S. K. (2024). LLM-based test-driven interactive code generation: User study and empirical evaluation. *IEEE Transactions on Software Engineering*. <https://doi.org/10.1109/TSE.2024.3428972>.

13. Liu, Y., Le-Cong, T., Widyasari, R., Tantithamthavorn, C., Li, L., Le, X. B. D., & Lo, D. (2024). On the reliability and explainability of language models for program generation. *ACM Transactions on Software Engineering and Methodology*. <https://doi.org/10.1145/3641540>.
14. Zhou, Y., Liu, Y., Yang, X., & Zhang, L. (2024). An empirical study on problems, causes and solutions of AI pair programming. *Journal of Systems and Software*, 212, 112204. <https://doi.org/10.1016/j.jss.2024.112204>.
15. Ozdenizci Kose, B. (2024). Mobilizing DevOps: Exploration of DevOps adoption in mobile software development. *Kybernetes*. <https://doi.org/10.1108/K-04-2024-0989>.
16. Bogner, J., & Merkel, M. (2022, May). To type or not to type? A systematic comparison of the software quality of JavaScript and TypeScript applications on GitHub. In *Proceedings of the 19th International Conference on Mining Software Repositories (MSR '22)* (pp. 658–669). Association for Computing Machinery. <https://doi.org/10.1145/3524842.3528454>.
17. March 2025 TIOBE Index: Legacy “dinosaur” languages continue to surge. Retrieved from: <https://www.techrepublic.com/article/tiobe-index-march-2025-legacy-programming-languages/> (date accessed: March 18, 2025).
18. The state of AI in early 2024: Gen AI adoption spikes and starts to generate value. Retrieved from: <https://www.mckinsey.com/capabilities/quantumblack/our-insights/the-state-of-ai-2024> (date accessed: January 30, 2025).