EJSIT
European Journal of Science, Innovation and Technology
ISSN: 2786-4936
www.ejsit-journal.com
Volume 6 | Number 3 | 2026

Adapting Code Review Processes to the Conditions of AI-Assisted Software Development

Bezhentsev Yurii*
Software Developer
Houston, Texas, United States

ABSTRACT

The article presents an analysis of the transformation of code review processes in the context of software development with the use of artificial intelligence. The study is conducted in the format of a systematic review and analytical synthesis of academic publications focusing on code generation, development productivity, and the organization of change validation processes. The primary focus is on the relationship between the speed of code creation, the characteristics of introduced changes, and the ability of the review process to ensure their reliable interpretation and safe integration. The key parameters determining the effectiveness of code review are examined, including temporal characteristics, the structure of changes, and cognitive load. It is established that the impact of code generation tools is indirect and manifests through the formation of asymmetry between the intensity of generation and the capacity for review. It is shown that even with improvements in the quality of automatically generated code, the risk of uncritical acceptance and the integration of hidden logical inconsistencies remains. An original adaptive code review model is proposed, reflecting a transition from single-stage review to a multi-level system that includes preliminary filtering, structured analysis, risk assessment, and a feedback loop. The results obtained allow code review to be considered as a controllable risk regulation mechanism determining the stability and predictability of the development process. The practical significance of the study lies in substantiating the need to structure code review processes, limit the scope of changes, and increase transparency in the origin of code fragments. The article will be useful for software developers, engineering managers, and specialists in software process management.

Keywords: code review, AI-assisted software development, code generation, development stability, cognitive load, risk management, validation processes

INTRODUCTION

The rapid proliferation of AI-assisted software development has fundamentally transformed the internal dynamics of software creation, shifting the primary constraint from code writing to code review and validation. Code generation and refinement systems enable changes to be produced significantly faster than teams can comprehend, align with architectural requirements, and safely integrate into the delivery pipeline [5]. At the same time, the influence of artificial intelligence is amplifying in nature. It increases individual productivity and scales existing properties of the engineering environment—enhancing efficiency in mature teams while provoking increased chaos and instability in less structured ones [6]. As a result, the ability to synchronize the speed of generation and review through the restructuring of feedback loops becomes critically important.

This contradiction is most pronounced in code review processes, which, under AI-assisted development, are transforming from a supporting quality control stage into a central mechanism for change validation [14]. In practice, accelerating generation without a

* Email: yurabezhentsev@gmail.com

corresponding acceleration in review leads to the accumulation of queues, superficial analysis of changes, increased cognitive load, and a higher probability of integrating code containing hidden defects into the main branch. A systemic shift of the bottleneck toward the review and change acceptance stage occurs, requiring its reconceptualization as the primary regulator of delivery stability [3]. Without such adaptation, increased development speed does not result in sustainable efficiency gains but is instead accompanied by growing instability and operational risks.

The aim of this study is to develop and theoretically substantiate an adaptive model of code review processes under conditions of AI-assisted software development. To achieve this aim, the following objectives are set:

– to analyze the impact of artificial intelligence on software development and code review processes;

– to identify the shift of the primary constraint within the software development lifecycle;

– to describe the key risks associated with code generated or significantly modified using artificial intelligence;

– to systematize practices for adapting code review processes.

The research hypothesis is that the use of artificial intelligence in software development without corresponding adaptation of code review processes leads to increased instability in the delivery of software changes. In contrast, accelerating feedback loops, reducing the size of reviewed changes, increasing transparency of code provenance, and structuring the review procedure reduce these risks and make the delivery flow more manageable.

The scientific novelty of the study lies in reinterpreting code review processes in the context of AI-assisted development not as a local quality control practice, but as a central element of a dynamic change flow management system in which the coherence of feedback loops plays a key role. Unlike the dominant perspective in the literature, which treats artificial intelligence primarily as a tool for enhancing individual productivity, this work shifts the analytical focus to the level of systemic delivery stability, revealing a previously underexplored phenomenon—the migration of the lifecycle bottleneck to the stage of review and change acceptance. The novelty is further manifested in the integration of engineering analysis of development processes with the logic of flow and feedback management. This approach makes it possible to conceptualize code review as a mechanism for regulating risk arising from the asymmetry between the speed of generation and the speed of validation. As a result, a foundation is established for developing models oriented not toward maximizing speed, but toward ensuring the stability and predictability of software development under intensive use of intelligent tools. Despite the growing body of research on AI-assisted development, limited attention has been paid to the systemic transformation of code review as a bottleneck and risk-regulation mechanism. In particular, the interaction between generation speed, review capacity, and delivery stability remains insufficiently conceptualized.

MATERIALS AND METHODOLOGY

The study employs methods of theoretical analysis of academic publications, categorical classification of factors influencing code review processes, and comparative analysis of relationships between the use of artificial intelligence, characteristics of code changes, and parameters of their validation processes. The analysis is aimed at identifying patterns in the shift of bottlenecks within the software development lifecycle and determining the role of code review as a mechanism for quality and risk management under conditions of accelerated change generation.

The study is conducted in the format of a systematic review of open-access academic publications for the period 2024–2026, presented in international peer-reviewed journals and

academic databases. The literature search was carried out in Google Scholar, ScienceDirect, SpringerLink, and MDPI using combinations of the following keywords: “AI-assisted software development,” “code review,” “LLM code generation,” “AI code review,” “software engineering productivity,” “code quality,” and “review processes,” applying logical operators AND/OR. The sample included English-language publications containing empirical or review-based evidence on the application of artificial intelligence in software development and code review processes. Studies focused exclusively on code generation algorithms without analysis of validation processes, as well as works that did not examine the impact of artificial intelligence on development organization, were excluded.

At the initial identification stage, 42 publications were selected. After removing duplicates and screening titles and abstracts, irrelevant studies were excluded. Full-text analysis resulted in a final sample of 16 sources that met the research objectives and ensured representative coverage of the topic.

The analysis procedure included sequential stages: source identification, duplicate removal, selection based on thematic relevance, full-text analysis, and categorical systematization of results. During the analysis, the following categories were identified: characteristics of AI-assisted code generation, parameters of code review processes, factors affecting the speed and quality of validation, and risks arising from the integration of automatically generated changes. The comparison of results was conducted by analyzing the identified factors and their impact on the stability of the development and delivery process.

The limitations of the sample are determined by the narrow focus of the study, as a significant portion of software engineering publications concentrates on code generation models, while code review processes in this context remain only partially explored.

The results obtained were used to systematize the factors influencing code review processes and to develop an original model reflecting the relationship between the speed of change generation, validation parameters, and the stability of the development process under conditions of AI-assisted software development.

RESULTS AND DISCUSSION

The integration of code generation systems has altered both the pace of software solution development and the logic of their verification. The key issue is not the fact of acceleration itself, but the emerging gap between the volume of generated changes and the ability to review them reliably [10]. As the intensity of generation increases, code review ceases to function as a neutral stage of coordination and instead becomes an overloaded process element through which an increasingly dense flow of heterogeneous changes passes. Under these conditions, not only individual metrics are transformed, but also the overall mode of engineering decision-making, as each approval of changes requires a higher degree of justification and precision [9].

The traditional organization of code review was based on the assumption that the main complexity was concentrated at the code-writing stage. In the context of generative tools, this relationship changes. The preparation of draft solutions, auxiliary procedures, and test scenarios is accelerated, yet the capacity for their correct and safe interpretation does not demonstrate comparable growth. This results in a situation where the review stage is simultaneously affected by an increased volume of incoming changes and reduced transparency of their internal structure. Within the process, this signifies a shift of the primary constraint from development to review, where the main risk zone is formed. In practical terms, the cost of superficial approval increases, as even formally correct changes may contain hidden logical inconsistencies [8].

The analysis of the temporal characteristics of the review process makes it possible to clarify the mechanism of this transformation. It is not merely a matter of identifying

acceleration, but of determining the factors that enable it and how they influence the nature of the review itself. Table 1 presents the impact of generative tools on the temporal parameters of the review process based on the DeputyDev industrial experiment.

Table 1: Impact of AI-assisted review on time metrics

Metric	Control Set 1	Control Set 2	Test Set (AI-assisted)	Improvement vs CS1, %	Improvement vs CS2, %
Avg Review Time (hours)	239,57	278,14	197,97	17,36	28,82
Avg Review Time per LOC (hours)	12,97	12,29	7,50	42,19	38,98
Median Review Time (hours)	0,76	0,78	0,41	46,36	47,52

Compiled by the author based on source: [11]

The comparison of control and experimental scenarios demonstrates a consistent reduction in review time. However, the significance of this result lies not in the acceleration itself, but in the conditions under which it is achieved. The processing of changes becomes more condensed, while the flow of code passing through review increases within a shorter time interval. The task does not become simpler; rather, it becomes more complex due to increased intensity. The growth in speed is accompanied by higher demands for the accuracy of engineering judgment and the depth of analysis.

This transformation is fundamental in nature. In traditional practice, a reduction in review time is usually associated with a decrease in the volume or complexity of the changes under consideration [2]. In the context of generative tools, a different mechanism emerges. The reduction in review duration is combined with an increase in the number of decisions made within a single cycle. The reviewer is required to more rapidly identify relevant fragments, assess risk levels, align changes with architectural constraints, and determine an appropriate depth of analysis. The observed reduction in time does not eliminate the workload on review but concentrates it within a shorter time frame [4].

Under these conditions, a key contradiction becomes evident. Code generation accelerates, while the cost of error at the stage of acceptance does not decrease. As the speed of the change flow increases, the risk of superficial review grows. Empirical evidence shows that increased development productivity is not accompanied by a corresponding reduction in instability during change integration [1]. As a result, the review stage does not disappear as a constraint but becomes a more vulnerable point in the process. Acceleration without changes to the review logic reduces development stability and increases the likelihood of accumulating hidden defects [7].

The resulting state requires compensation at the organizational level. Limiting the scope of changes, regulating the timeframes for initial review, and accelerating automated checks cease to be auxiliary measures and become necessary conditions for maintaining process controllability. In their absence, the reduction in review time begins to function not as optimization but as a displacement of defects to later stages, including testing, merging, and deployment [13].

Further analysis shifts from temporal characteristics to the properties of the changes themselves submitted for review. The use of generative tools affects not only the speed of code creation but also its internal structure. The variability of solutions increases, the boundaries between different types of changes become blurred, and the changes themselves become less transparent for rapid and reliable evaluation. Table 2 presents quantitative changes in the characteristics of such artifacts and their impact on the review process.

Table 2: Changes in AI-generated artifact characteristics and their impact on the review process

Metric / Indicator	Value	Context of application	Implication for code review adaptation
Accuracy improvement (top-1)	+80%	Fine-tuning vs ATLAS	Increased trust in AI-generated artifacts
Accuracy improvement (top-1)	+33%	Fine-tuning vs T5	Partial substitution of manual validation
Test generation success rate	36% → 99%	Prompt + static analysis	Reduced review workload
Input size reduction (tokens)	−90% (5295 → 559)	Prompt optimization	Easier comprehension during review
Success rate (Claude 3.5)	93.33%	JavaScript test generation	High applicability in review pipelines
Mutation score	89.23%	Claude 3.5	Improved defect detection capability

Compiled by the author based on source: [5]

These results demonstrate that improvements in generation quality do not linearly translate into reduced review complexity. Instead, higher plausibility of generated code increases the likelihood of implicit defects being accepted, thereby shifting the review task from error detection to semantic interpretation. Formally, an increase in the quality of automatically generated solutions should reduce the burden on code review. However, the obtained results demonstrate the opposite pattern. As the plausibility of generated code increases, so does the risk of its uncritical acceptance. External coherence begins to be perceived as a sign of correctness and gradually substitutes for actual reliability. A fundamental shift lies in the fact that errors no longer manifest in an explicit form. They may be embedded in code that appears logical, typical, and well-structured, which facilitates its rapid approval [2].

At the structural level of introduced changes, persistent shifts are observed. The volume of redundant code increases, as change sets include additional segments that are not necessary for solving the given task. These fragments emerge as a byproduct of code generation or partial refactoring. At the same time, the share of plausible but incorrect solutions increases. Such solutions are not perceived as obvious errors, as they reproduce familiar patterns but fail to account for architectural constraints, internal project rules, or edge-case scenarios. In addition, the mixing of different types of changes becomes more frequent. A single change set may simultaneously include functional modifications, cosmetic edits, and partial refactoring. The

internal structure of such sets loses clarity and becomes less suitable for rapid and reliable evaluation [15].

Under these conditions, the review stage loses the character of a simple correctness verification procedure. It begins to involve the reconstruction of the logic behind the change. First, it becomes necessary to separate meaningful components from redundant elements. Then, it is required to determine what problem the change was intended to solve and in what context it was created. Only after this can its correctness be assessed. This sequence of actions reflects a fundamental shift in the nature of the reviewer's work. The primary effort shifts from detecting errors to interpreting the meaning of introduced changes. Code generation becomes easier, while understanding the code becomes more complex.

As a consequence, cognitive load at the review stage increases. The acceleration of code creation leads to greater effort being required to validate its suitability for acceptance. A change set ceases to be a transparent object of analysis and instead becomes a heterogeneous collection of fragments requiring preliminary structuring and interpretation. As a result, the review process does not become simpler but rather more demanding, interpretation-intensive, and sensitive to internal structural inconsistencies in the changes.

The observed acceleration in the preparation of software changes is not accompanied by a corresponding increase in the capacity for their reliable review. A persistent asymmetry emerges between production and validation. The flow of changes grows faster than the review system can interpret and safely integrate them. Under such conditions, the primary constraint is no longer located at the development stage but shifts to the zone of change acceptance. This shift is supported by empirical observations showing increased productivity without a proportional improvement in delivery stability [1]. Code review accumulates an increasing volume of decisions, each requiring an assessment of correctness, contextual alignment, and potential risk.

In the classical configuration, the main complexity of development is localized on the side of code creation. Review functions as a filter that eliminates defects before integration. With the use of intelligent tools, this relationship is disrupted. Code generation becomes faster and partially automated, while its interpretation remains dependent on human analysis. As a result, a new bottleneck emerges, associated with the need to process a dense flow of heterogeneous changes. Code review ceases to serve as a secondary control mechanism and instead determines the throughput of the entire development process.

As generation speed increases, the cost of error at the stage of change acceptance rises. Reducing review time while simultaneously increasing code volume amplifies the risk of superficial approval. Under these conditions, development stability is determined not by the speed of solution creation, but by the ability of the review process to maintain the required level of reliability under constrained resources. Code review acquires the status of a central element regulating the balance between the speed and quality of changes. The restructuring of the process requires a transition from an informal review practice to a managed system. The observed adaptation mechanisms are presented in Table 3.

Table 3: Practices for adapting code review under AI conditions

Practice	Description	Effect
PR Size Limitation	Decomposition of changes into smaller units	Reduction of cognitive load and improvement of review accuracy
Review SLA	Establishment of fixed time constraints for review cycles	Stabilization of feedback loops
Structured Review	Use of standardized checklists	Enhancement of reproducibility and completeness of evaluation
Separation of Changes	Prohibition of mixing different change types	Reduction of analytical heterogeneity
AI Code Labeling	Explicit identification of AI-generated fragments	Increased focus on high-risk areas
CI Optimization	Streamlining of continuous integration processes	Reduction of technical delays while preserving review depth

Compiled by the author based on case study

The practical significance of these measures is illustrated by a typical scenario in which an automatically generated user interface is combined with application logic within a single change set. In such cases, an increase in redundant code is observed, along with the emergence of errors in edge-case scenarios and a higher load on review and testing stages. The mixing of different types of changes complicates rapid correctness assessment, as it requires simultaneous analysis of logic, structure, and side effects. This creates the need for preliminary separation and interpretation of changes prior to their substantive evaluation.

These effects demonstrate that the problem is not limited to the quality of automatically generated code. The key factor is the organization of the review process. In the absence of structured practices, accelerated development leads to the accumulation of hidden defects and the transfer of workload to later stages. When formalized practices are in place, workload redistribution is achieved and the risk of incorrect acceptance of changes is reduced. Therefore, adapting the review process becomes a necessary condition for maintaining the controllability of software development. Figure 1 presents an adaptive review model that reflects the systemic restructuring of the process under the use of intelligent tools.

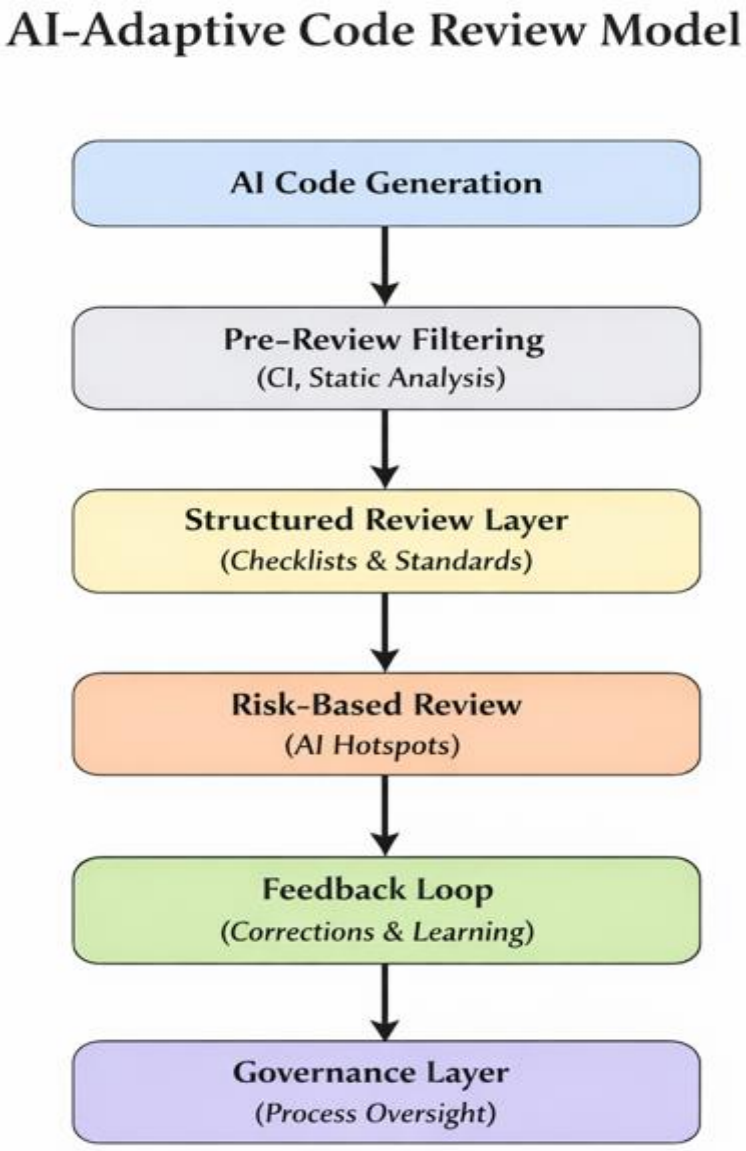

Figure 1. AI-Adaptive Code Review Model (Author's own development)

The model includes sequential layers: change generation, preliminary filtering, structured review, risk assessment, feedback loop, and process governance. Each layer performs a distinct function and reduces the load on subsequent stages. Preliminary filtering eliminates obvious defects before expert involvement. Structured review ensures consistency of analysis. Risk assessment focuses attention on areas with elevated uncertainty. The feedback loop enables the adjustment of subsequent changes. The governance layer captures process parameters and prevents degradation.

The key distinction of this model from the traditional approach lies in the redistribution of review functions. In classical practice, review is performed as a single stage after development is completed. In the proposed configuration, it is distributed across multiple layers and integrated into the flow of change formation. Review is no longer limited to confirming correctness but serves as a mechanism for risk management arising from high generation speed and code heterogeneity.

Under these conditions, development effectiveness is determined not by the maximum speed of generation, but by the alignment of the stages of change creation, review, and refinement. The absence of such alignment leads to the accumulation of defects and reduced

predictability of delivery. Conversely, a structured organization of review maintains the balance between speed and reliability under increasing development intensity. The findings indicate that AI-assisted development does not eliminate review complexity but redistributes it. The review stage evolves from a defect-filtering mechanism into a semantic interpretation layer responsible for aligning generated code with architectural and operational constraints.

CONCLUSION

The results obtained make it possible to consider code review processes in the context of AI-assisted development not as a supporting quality control stage, but as a key mechanism for ensuring the stability of the entire flow of software changes. The behavior of the development system is determined by the volume of generated code and the ability of the review stage to provide a coherent and reliable interpretation of these changes. Stability emerges to the extent that a balance is achieved between the speed of generation and the depth of validation.

The coherence of change-processing stages becomes decisive. Even with high-quality automatically generated code, the overall stability of the process remains vulnerable if review is performed in a fragmented manner and lacks sufficient structure. In conditions where changes are introduced with high intensity and exhibit internal heterogeneity, the organization of review determines whether the flow of changes remains manageable. When interconnected layers of filtering, analysis, and feedback are present, a predictable environment is formed in which the likelihood of accumulating hidden defects is reduced and the reliability of decision-making is increased.

The practical significance lies in the need to reconsider approaches to organizing development processes. Delivery stability is determined not by maximizing the speed of code creation, but by the ability to structure review as a systemic and manageable process. The formalization of procedures, limitation of change scope and structure, increased transparency of code provenance, and the implementation of multi-level analysis mechanisms are necessary conditions for maintaining quality under increasing development intensity. In this context, external code generation tools amplify existing system properties, while stability is determined by the internal organization of review processes.

Future research directions are associated with a more detailed analysis of mechanisms for interpreting changes at the review stage, as well as the development of methods for quantitatively assessing cognitive load and risk under conditions of high code flow density. Of particular interest is the study of relationships between change structure, review parameters, and delivery stability metrics, as well as the creation of tools that adapt review processes depending on the characteristics of generated code and the level of uncertainty.

REFERENCES

1. Adalsteinsson, F. S., Magnússon, B. B., Milicevic, M., Davidsson, A. N., & Cheng, C.-H. (2025). Rethinking code review workflows with LLM assistance: An empirical study. *arXiv*. https://doi.org/10.48550/arXiv.2505.16339
2. Alami, A., & Ernst, N. A. (2025). Human and machine: How software engineers perceive and engage with AI-assisted code reviews compared to their peers. *arXiv*. https://doi.org/10.48550/arXiv.2501.02092
3. Almeida, Y., Albuquerque, D., Dantas Filho, E., Muniz, F., de Farias Santos, K., Perkusich, M., Almeida, H., & Perkusich, A. (2024). AICodeReview: Advancing code quality with AI-enhanced reviews. *SoftwareX*. https://doi.org/10.1016/j.softx.2024.101677
4. Bistarelli, S., Fiore, M., Mercanti, I., et al. (2025). Usage of large language model for code generation tasks: A review. *SN Computer Science, 6,* 673. https://doi.org/10.1007/s42979-025-04241-5
5. Celik, A., & Mahmoud, Q. H. (2025). A review of large language models for automated test case generation. *Machine Learning and Knowledge Extraction, 7*(3), 97. https://doi.org/10.3390/make7030097
6. Gullstrand Heander, L., Söderberg, E., & Rydenfält, C. (2026). Code review as decision-making: Building a cognitive model from the questions asked during code review. *Empirical Software Engineering, 31*, 54. https://doi.org/10.1007/s10664-025-10791-2
7. Heander, L., Söderberg, E., & Rydenfält, C. (2025). Support, not automation: Towards AI-supported code review for code quality and beyond. In *Proceedings of the 33rd ACM International Conference on the Foundations of Software Engineering (FSE Companion '25)* (pp. 591–595). Association for Computing Machinery. https://doi.org/10.1145/3696630.3728505
8. İçöz, B., & Biricik, G. (2026). Context-aware code review automation: A retrieval-augmented approach. *Applied Sciences, 16*(4), 1875. https://doi.org/10.3390/app16041875
9. Kansab, S., Bordeleau, F., & Tizghadam, A. (2025). RevMine: An LLM-assisted tool for code review mining and analysis across Git platforms. *arXiv*. https://doi.org/10.48550/arXiv.2510.04796
10. Khade, P. S., & Sambhe, R. U. (2025). Artificial intelligence in software development: A review of code generation, testing, maintenance and security. *International Journal of Current Science Research and Review, 8*(4), 1632–1641. https://doi.org/10.47191/ijcsrr/V8-i4-08
11. Khare, V., Saini, V., Sharma, D., Kumar, A., Rana, A., & Yadav, A. (2025). DeputyDev—AI powered developer assistant: Breaking the code review logjam through contextual AI to boost developer productivity. *arXiv*. https://doi.org/10.48550/arXiv.2508.09676
12. Mahfoodh, H., Hammad, M., Alqaralleh, B. A. Y., & Zreikat, A. I. (2026). Evaluating LLMs for source code generation and summarization using machine learning classification and ranking. *Computers, 15*(2), 119. https://doi.org/10.3390/computers15020119
13. Maru, G. G., Lee, S., Ji, S., Ko, S.-K., & Im, H. (2025). Neural methods for programming: A comprehensive survey and future directions. *Applied Sciences, 15*(22), 12150. https://doi.org/10.3390/app152212150
14. Mavridou, E., Vrochidou, E., Kalampokas, T., Kanakaris, V., & Papakostas, G. A. (2025). AI-powered software development: A systematic review of recommender

systems for programmers. *Computers, 14*(4), 119. https://doi.org/10.3390/computers14040119

15. Pahl, C., Sezen, Ö. C., & Hofer, F. (2026). Artificial intelligence for infrastructure-as-code—A systematic literature review. *Electronics, 15*(4), 755. https://doi.org/10.3390/electronics15040755