International Journal of Computer (IJC)
ISSN 2307-4523 (Print & Online)
https://ijcjournal.org/index.php/InternationalJournalOfComputer/index

Rethinking the Methodology of Pair Programming in the Context of Intelligent Agent Integration

Sergei Kuznetsov*

Lead Software Engineer ,Malaga, Spain

Abstract

This article examines the transformation of pair programming methodology in the context of integrating intelligent assistants based on large language models. The study is conducted as a structured narrative review and analytical synthesis of academic publications devoted to AI-assisted programming, human–AI collaboration, and collaborative software development. The main focus is on changes in the structure of developer interaction, the redistribution of roles, and the reconfiguration of cognitive functions between human participants and intelligent agents. Key characteristics of traditional and AI-assisted pair programming are compared, including differences in evaluation parameters, learning effects, and behavioural outcomes. It is shown that the influence of intelligent assistants is not limited to improving productivity, but is realised through changes in the organisation and perception of the development process. It is also demonstrated that isolated use of AI tools does not fundamentally alter collaborative practices unless they are integrated into a coherent interaction structure. An original model of AI-integrated pair programming is proposed, describing the distributed nature of solution generation, evaluation, and refinement within a human–AI system. The results make it possible to consider pair programming as a hybrid form of collaboration, where the effectiveness of development depends on the coordination between human and intelligent components. The article may be of interest to software engineering researchers, educators in computing disciplines, and developers of AI-supported programming tools.

Keywords: pair programming; AI-assisted programming; intelligent agents; human–AI collaboration; software development; code generation; large language models.

Received: 3/24/2026
Accepted: 5/24/2026
Published: 6/2/2026

---

* *Corresponding author.*

1. Introduction

The growing use of LLM-based intelligent agents is reshaping software development, as AI tools now generate code, detect errors, and assist in debugging, moving beyond a purely supportive role [2]. This shift is especially evident in pair programming, where interaction is no longer limited to two developers but includes an AI system involved in generating and evaluating solutions. In the classical Driver–Navigator model, responsibilities are clearly divided between coding and control; however, some of these functions are now partially handled by AI. Despite the rapid growth of research on AI in programming, most studies focus on productivity and code quality, while changes in the methodology of pair programming itself–such as role transformation and the redistribution of cognitive effort–remain less explored [5].

In the context of this study, it is important to distinguish several related but conceptually different forms of AI-supported software development. Classical pair programming refers to a collaborative programming methodology in which two human participants continuously interact while jointly solving a programming task through role distribution and real-time coordination. AI-assisted programming, by contrast, represents a broader category that includes the use of intelligent assistants during software development regardless of whether collaboration between developers is present. Within this broader category, AI-integrated pair programming is understood in the present study as a collaborative development configuration in which at least one human participant continuously interacts with an intelligent assistant during a shared programming process.

This distinction is methodologically important because not all studies devoted to AI-supported coding examine pair programming in the strict sense. Some publications focus primarily on intelligent code-generation systems, AI-assisted individual workflows, or multi-agent programming architectures. Such studies are considered in the present research only insofar as they help explain emerging interaction patterns, cognitive redistribution, and changes in collaborative development structures associated with intelligent assistant integration.

The aim of the study is to develop a conceptual model of AI-integrated pair programming. To achieve this aim, the following tasks are addressed:

– to systematise existing approaches to the organisation of pair programming;

– to analyse the impact of intelligent agents on software development processes;

– to identify changes in the roles and cognitive functions of participants when using AI assistants;

– to substantiate the need to revise the traditional methodology of pair programming.

The scientific novelty of the study lies in the development of a conceptual model of AI-integrated pair programming that reconceptualises pair programming as a distributed human–AI cognitive system rather than a purely human collaborative practice. From a theoretical perspective, the study builds upon the idea that collaborative programming can be interpreted as a distributed socio-technical process in which cognitive functions are dynamically shared between participants and technological systems. Under conditions of intelligent assistant

integration, programming increasingly involves continuous cycles of generation, interpretation, evaluation, and refinement distributed across human and AI components. Accordingly, the present research approaches pair programming not merely as a communication technique between developers, but as a hybrid interaction structure shaped by cognitive delegation, iterative feedback, and coordinated decision-making. In contrast to prior research, which predominantly evaluates AI-assisted programming in terms of productivity and code quality, this study shifts the analytical focus to the structure of interaction itself and the underlying mechanisms of collaboration.

The proposed model formalises the redistribution of core development functions—generation, evaluation, and control—between human participants and intelligent agents, thereby explicitly capturing the transformation of roles and decision-making processes within the programming workflow. It further introduces a structural distinction between human–human and human–AI interaction patterns, demonstrating that the integration of AI alters not only task execution but the cognitive architecture of collaborative development.

By positioning the intelligent assistant as an active and independent element within the interaction system, the model extends existing approaches to human–AI collaboration and provides a framework for analysing programming as a hybrid socio-technical process. This allows pair programming to be interpreted not as an augmented version of traditional collaboration, but as a qualitatively different collaborative configuration shaped by continuous human–AI coordination and iterative feedback processes.

2. Materials and Methods

The study was conducted in the form of a structured narrative review and analytical synthesis of academic publications devoted to pair programming, AI-assisted software development, and human–AI collaborative programming. The methodological approach combined systematic literature exploration with conceptual synthesis aimed at identifying how intelligent assistants influence the organisation of collaborative programming processes, the distribution of cognitive functions, and interaction patterns between developers and AI systems.

The literature search covered open-access academic publications published between 2023 and 2025. The search was performed using Google Scholar, ScienceDirect, SpringerLink, MDPI, Wiley Online Library, and arXiv. The search strategy was based on combinations of the following keywords and search expressions: “pair programming” AND “AI coding assistants”, “human–AI collaboration” AND programming, “AI-assisted programming”, “LLM code generation”, “AI pair programming”, “collaborative software development”, and “intelligent agents in programming”. Logical operators AND/OR were used to combine related concepts and broaden the search coverage across adjacent areas of AI-supported software development.

At the identification stage, 55 publications were collected. Duplicate records were removed prior to screening. During the initial screening stage, titles and abstracts were analysed in order to exclude publications unrelated to collaborative programming, software engineering practices, or intelligent assistant integration in development workflows. Studies focused exclusively on general artificial intelligence applications without relevance to programming interaction were excluded. Full-text assessment was subsequently conducted for the remaining publications. As a result, 14 studies were included in the final analytical sample.

The inclusion criteria covered publications examining at least one of the following aspects: pair programming practices, AI-assisted programming workflows, human–AI collaborative coding, intelligent code-generation systems, or cognitive and behavioural characteristics of collaborative software development. Both empirical and conceptual studies were considered if they contributed to understanding changes in interaction structures within programming activities. Publications without direct relevance to programming collaboration or developer–AI interaction were excluded from the review.

The reviewed literature included both peer-reviewed journal publications and scientific preprints indexed in arXiv. Peer-reviewed studies were used as the primary evidentiary basis for identifying recurring findings related to collaborative development, while arXiv publications were considered supplementary sources reflecting rapidly evolving research directions in AI-assisted programming. Because the field is developing faster than conventional publication cycles, the inclusion of preprints made it possible to capture emerging methodological and technological tendencies that are not yet fully represented in journal literature. At the same time, conclusions derived from non-peer-reviewed studies were interpreted with caution and were not treated as equivalent to fully validated empirical evidence.

The analytical synthesis was organised around several thematic categories identified during the review process: redistribution of programming roles, AI participation in code generation and evaluation, cognitive load transformation, trust in intelligent assistants, collaborative decision-making, and workflow restructuring in AI-supported development environments. These categories were used to compare findings across heterogeneous studies and to identify recurring interaction patterns associated with the integration of intelligent assistants into collaborative programming practices.

The proposed conceptual model of AI-integrated pair programming was developed through comparative interpretation of the reviewed literature rather than through direct empirical validation. Evidence from the analysed studies was used to identify stable tendencies related to role redistribution, iterative human–AI interaction, and the increasing involvement of intelligent assistants in intermediate stages of programming activity. At the same time, several elements of the model represent conceptual extrapolations intended to systematise emerging forms of collaborative development and to provide a structured framework for further empirical investigation.

The methodological approach adopted in the study does not claim the level of procedural formalisation associated with protocol-driven systematic reviews. Instead, the research is positioned as a structured analytical synthesis intended to conceptualise the transformation of pair programming under conditions of sustained intelligent assistant integration.

3. Results

The integration of intelligent assistants into programming processes is gradually transforming the logic of organising collaborative software development. Within the traditional model of pair programming, one participant is responsible for writing code, while the other directs the solution and monitors its correctness [8]. However, the

spread of code generation systems leads to a situation in which some functions related to navigation, error detection, and the interpretation of solutions are increasingly performed by the intelligent agent. Under these conditions, the structure of interaction shifts from the classical "developer–developer" model to a distributed problem-solving format in which AI plays an active role [1]. This shift is particularly evident in educational environments, where the use of intelligent assistants is becoming part of the learning process. A comparison of classical and AI-assisted pair programming shows that the differences between them are reflected both in the organisation of the development process itself and in the approaches used to evaluate its effectiveness [7]. The traditional model is typically assessed using a broader set of parameters, including learning outcomes and organisational costs. In contrast, studies of the "human–AI" format tend to focus primarily on productivity and code quality indicators [13]. Table 1 presents a comparison of the parameters used to evaluate the effectiveness of these two models of collaborative programming.

Table 1: Evaluation parameters of different pair programming models (Compiled by the author based on the source: [9])

Evaluation parameter	**Human–human pair programming**	**Human–AI pair programming**
Code quality	1	1
Productivity	1	1
Satisfaction	1	1
Learning effect	1	0
Costs	1	0

As follows from the data presented in the table, the traditional model of pair programming is examined through a broader range of characteristics, encompassing not only performance indicators but also educational and organisational aspects. This suggests that existing studies on AI-assisted programming are largely focused on task performance outcomes, while its educational and organisational implications remain less developed and require further analysis [4].

A separate line of research is devoted to assessing the impact of different programming formats on learning outcomes. The available evidence generally indicates a consistent effect associated with increased productivity and reduced levels of anxiety when intelligent assistants are used [11]. At the same time, classical pair programming demonstrates higher performance compared to individual work. Figure 1 presents a comparison of educational outcomes across different programming formats.

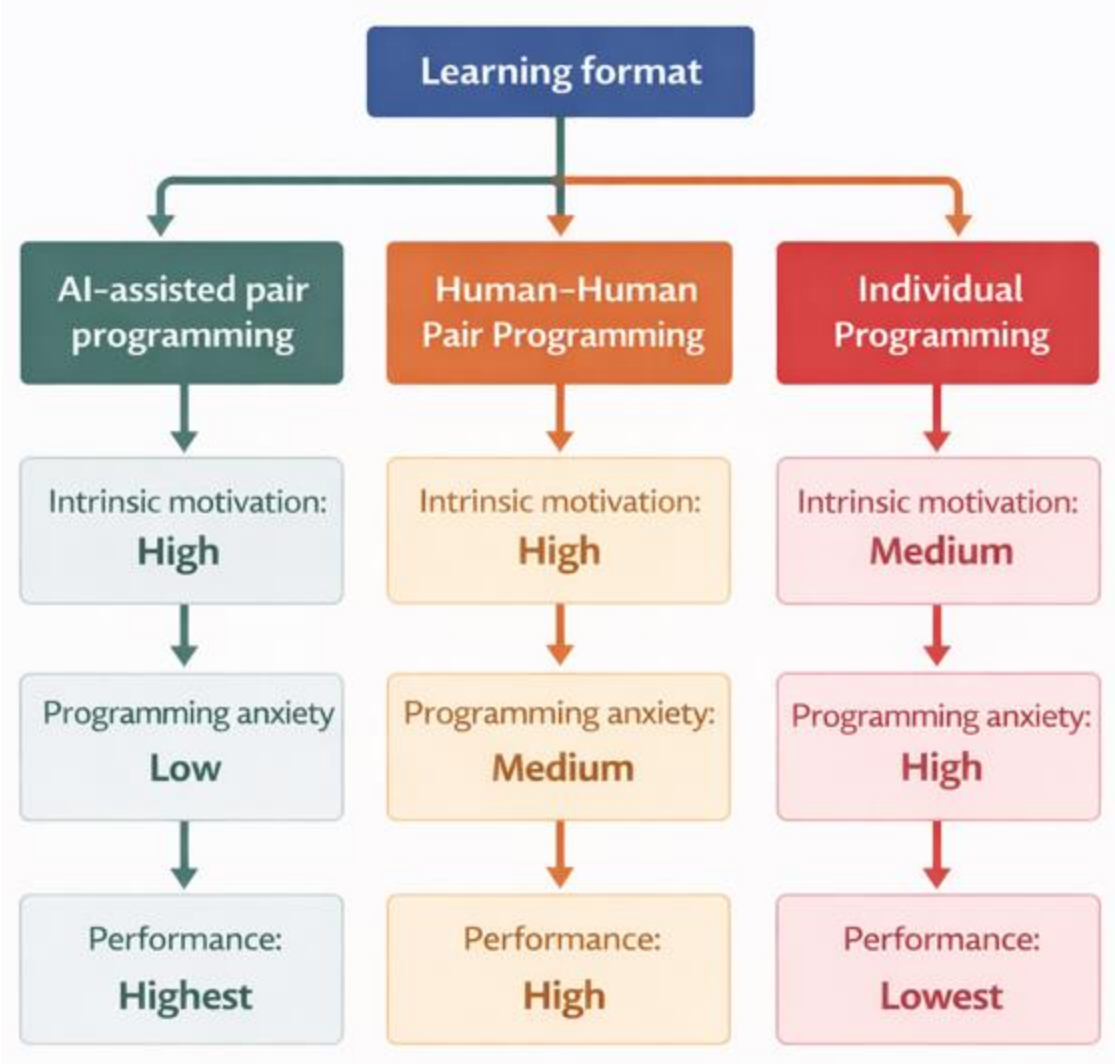

Figure 1: Comparison of educational outcomes across different programming formats (Compiled by the author based on the source: [3])

Psychological factors also play an important role in the effectiveness of collaborative development. In pair programming, the level of intrinsic motivation is higher than in individual work. Figure 2 illustrates the differences in intrinsic motivation across different programming roles.

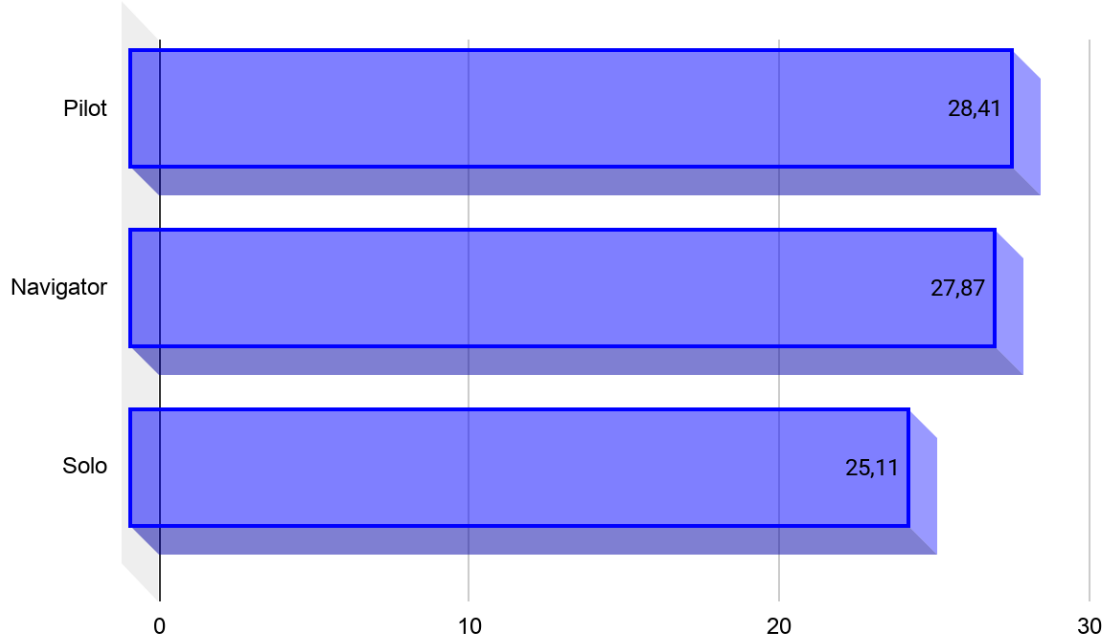

Figure 2: Level of intrinsic motivation across different programming roles (Compiled by the author based on the source: [12])

Intrinsic motivation differs across the three formats. The Pilot role shows the strongest involvement, which aligns with direct coding and continuous decision-making during development. The Navigator is slightly less engaged. The gap between the two remains small and does not change the overall pattern. Individual work stands apart. Engagement drops noticeably when tasks are performed alone, without shared responsibility or real-time feedback. Pair programming maintains higher involvement. Both roles support sustained motivation compared to solo execution.

Taken together, the results indicate that the use of intelligent assistants does not diminish the importance of pair programming but rather leads to changes in the structure of interaction within it. In AI-assisted collaborative programming environments, development processes are increasingly organised as interactions between human participants and intelligent systems, where functions related to solution generation and analysis are distributed across human and AI components.

4. Discussion

The integration of intelligent assistants into programming practice is accompanied by both an expansion of development capabilities and a change in the organisation of collaborative work itself. The findings indicate that the influence of AI is reflected in faster task execution, improved ease of programming, and a transformation of interaction structures among participants [14]. The traditional model of pair programming, based on the distribution of roles between two developers, assumes a division of responsibilities for code writing, logical control, and solution verification [10]. However, in an environment where an intelligent assistant continuously

participates in code generation, refinement, and validation, this two-role scheme no longer fully reflects the actual organisation of the process.

In this context, the issue is not merely the introduction of a new tool, but a shift in the very unit of interaction within software development. The classical model encompasses a broader range of parameters, including quality, productivity, learning outcomes, and the organisational costs of collaboration [6]. At the same time, emerging practices shaped by AI involvement exhibit different characteristics, while their methodological description remains limited and requires further refinement.

The findings related to motivation, anxiety, and productivity further support this observation. The use of intelligent assistants appears particularly effective in situations where speed of task completion, reduction of cognitive load, and in-process support are critical. At the same time, classical pair programming retains its advantages in aspects related to social interaction, joint discussion of solutions, and the sense of partner presence.

The proposed conceptual model was derived through analytical comparison of recurring interaction patterns identified across the reviewed literature. Studies devoted to AI-assisted code generation and intelligent programming support demonstrated the increasing involvement of AI systems in intermediate stages of development, particularly in solution generation, code refinement, and error detection [2, 11]. Research on collaborative programming and pair programming further indicated that human participants continue to retain evaluative, interpretative, and decision-making functions even under conditions of sustained intelligent assistant involvement [3, 9].

At the same time, several studies highlighted changes in cognitive workload distribution, interaction dynamics, and trust formation associated with AI-supported programming environments [1, 6]. These findings collectively suggested that contemporary collaborative programming increasingly operates through iterative cycles of generation, evaluation, verification, and refinement distributed across human and intelligent components. On this basis, the proposed model systematises the emerging structure of AI-integrated pair programming by combining evidence-based interaction tendencies identified in the reviewed studies with conceptual interpretation of their broader organisational and cognitive implications.

Taking these changes into account, the study proposes a model of AI-integrated pair programming aimed at describing this emerging form of collaboration. The model is relevant not only for understanding the technical dimension of development but also for capturing changes in the cognitive organisation of programming. Under these conditions, problem-solving increasingly emerges through distributed interaction structures in which the processes of generation, evaluation, discussion, and refinement of solutions are shared between developers and intelligent assistants. Figure 3 presents the proposed model of AI-integrated pair programming.

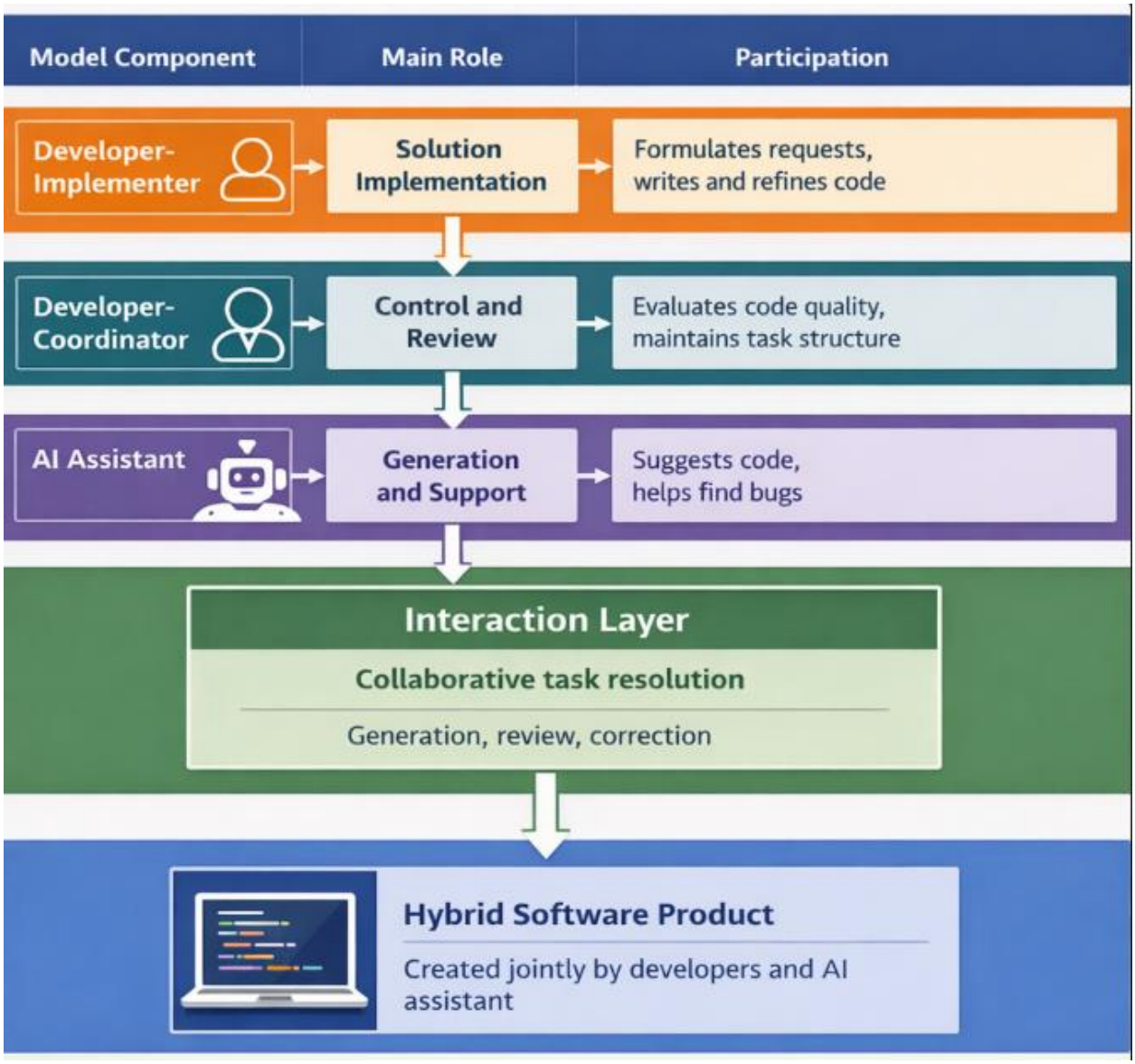

Figure 3: Hybrid model of AI-integrated pair programming (Author's development)

Several components of the proposed model are directly grounded in patterns identified across the reviewed studies. The redistribution of programming roles between developers and intelligent assistants is supported by studies describing the growing participation of AI systems in code generation, debugging, and solution refinement processes [2, 11]. Findings related to changes in cognitive load and developer behaviour are reflected in studies examining programming anxiety, motivational differences, and the increasing shift of human participants toward evaluative and interpretative functions during AI-supported development [3, 12].

The trust-related dimension of the model is informed by research demonstrating that developers' willingness to rely on intelligent assistants depends on perceived reliability, interaction coherence, and transparency of AI-generated outputs. In turn, the iterative structure of generation, evaluation, verification, and refinement incorporated into the model reflects interaction cycles described in studies devoted to collaborative AI-assisted programming workflows and feedback-driven code generation processes.

At the same time, the integration of these elements into a unified interaction structure represents a conceptual synthesis developed within the framework of the present study rather than a direct reproduction of any single reviewed model.

The proposed model formulates a research proposition according to which emerging forms of AI-integrated pair programming increasingly operate as hybrid systems of human–AI collaboration rather than exclusively human interaction practices. Under conditions of sustained involvement of an intelligent assistant, this practice is transformed into a hybrid system of joint problem-solving in which interaction occurs between developers and AI. While traditional models assumed that all core operations were performed by team members, in the current context it becomes necessary to treat the intelligent assistant as an independent element. It increasingly influences the distribution of roles, the pace of work, and intermediate stages of decision-making processes within collaborative programming environments.

Under these conditions, the nature of developers' cognitive load also changes. They act less as the sole source of solutions and increasingly perform functions related to the interpretation, evaluation, and selection of proposed options. Accordingly, programming can be viewed as a complex engineering and cognitive practice. Within this framework, the distribution of attention, the level of trust in the system, and the preservation of cognitive autonomy become particularly important.

The integration of an intelligent assistant changes not only development outcomes but also how participants behave during the process. Lower anxiety, higher motivation, and a redistribution of control suggest a shift that cannot be explained within a single disciplinary framework. Software engineering alone is not sufficient here. The emerging patterns require joint consideration of cognitive processes, interaction dynamics, and technical constraints. Practical implications follow from this shift. Intelligent assistants are most useful when they support the workflow rather than replace the developer. Their strength lies in generating intermediate options, accelerating the search for alternatives, and assisting at specific stages of coding. Control over logic, validation of results, and final decisions, however, remains with the human participant.

4.1 Limitations of the Study

At the same time, the study has several limitations that should be taken into account when interpreting the proposed model. First, the field of AI-assisted software development is evolving rapidly, and many interaction practices associated with intelligent programming assistants continue to change alongside advances in large language models and development tools. As a result, some identified tendencies may develop further as AI-supported workflows mature.

Second, the evidentiary basis of the review includes studies with different levels of methodological maturity, including both peer-reviewed publications and scientific preprints. Although the inclusion of emerging research made it possible to capture recent developments in AI-assisted programming, some findings should be interpreted cautiously due to the varying degree of empirical validation across the reviewed literature.

In addition, only a limited number of studies directly examine AI-integrated pair programming as a distinct

collaborative methodology. A considerable proportion of the reviewed literature focuses more broadly on AI-assisted programming workflows, intelligent coding assistants, or collaborative code-generation environments. Consequently, some conceptual conclusions of the study are derived through analytical interpretation of adjacent research areas rather than from a fully established body of pair programming literature.

Finally, the proposed model has a conceptual and interpretative character and has not yet undergone direct empirical validation in real-world development teams. Accordingly, the model should be understood primarily as a theoretical framework intended to systematise emerging forms of human–AI collaborative programming and to support future empirical investigation of interaction structures, cognitive dynamics, and collaborative programming practices.

5. Conclusion

The conducted analysis shows that under conditions of sustained intelligent assistant integration, pair programming increasingly extends beyond the framework of exclusively human interaction. The development process increasingly unfolds as a collaboration between a developer and AI, where solutions emerge through iterative cycles of generation, evaluation, and refinement. In this context, developer behaviour is shaped by the extent to which interaction with the system is perceived as coherent and understandable. In this regard, the key factor is not the presence of individual effective tools, but their integration into an overall workflow. Even when AI demonstrates high levels of accuracy or speed, the process may remain unstable if the interaction appears fragmented. Conversely, when tasks are clearly distributed across generation, interpretation, and control, a coherent structure of work emerges that supports trust in the system and facilitates decision-making. From a practical perspective, this implies that the organisation of collaborative development requires reconsideration. The focus increasingly shifts toward both performance and the ability to establish clear and predictable interaction structures between human developers and intelligent assistants within collaborative programming workflows. In this setting, AI serves as a means of accelerating solution search and providing in-process support, while the evaluation of correctness and final decision-making remain the responsibility of the developer.

Future research may focus on examining how human–AI interaction is formed in real development teams. Of particular interest is the analysis of how trust in intelligent assistants develops, as well as how developers' patterns of thinking evolve through prolonged interaction with such systems.

References

[1]. Alanazi, M., Soh, B., Samra, H., & Li, A. (2025). The influence of artificial intelligence tools on learning outcomes in computer programming: A systematic review and meta-analysis. Computers, 14(5), 185. https://doi.org/10.3390/computers14050185

[2]. Ali, F., Ahmed, A., Alipour, M. A., et al. (2025). Adoption of AI-coding assistants in programming education: Exploring trust and learning motivation through an extended technology acceptance model. Journal of Computer Education. https://doi.org/10.1007/s40692-025-00375-w

[3]. Fan, G., Liu, D., Zhang, R., et al. (2025). The impact of AI-assisted pair programming on student motivation, programming anxiety, collaborative learning, and programming performance: A comparative study with traditional pair programming and individual approaches. International Journal of STEM Education, 12, 16. https://doi.org/10.1186/s40594-025-00537-3

[4]. Gaitantzi, A., & Kazanidis, I. (2025). The role of artificial intelligence in computer science education: A systematic review with a focus on database instruction. Applied Sciences, 15(7), 3960. https://doi.org/10.3390/app15073960

[5]. Graßl, I., & Fraser, G. (2023). The ABC of pair programming: Gender-dependent attitude, behavior and code of young learners. arXiv. https://doi.org/10.48550/arXiv.2304.08940

[6]. Jiang, L., Yamaguchi, S., & Bin Ahmadon, M. A. (2025). SynergyAI: A human–AI pair programming tool based on dataflow. Information, 16(3), 178. https://doi.org/10.3390/info16030178

[7]. Kosar, T., Ostojić, D., Liu, Y. D., & Mernik, M. (2024). Computer science education in ChatGPT era: Experiences from an experiment in a programming course for novice programmers. Mathematics, 12(5), 629. https://doi.org/10.3390/math12050629

[8]. Kotsiantis, S., Verykios, V., & Tzagarakis, M. (2024). AI-assisted programming tasks using code embeddings and transformers. Electronics, 13(4), 767. https://doi.org/10.3390/electronics13040767

[9]. Ma, Q., Wu, T., & Koedinger, K. (2023). Is AI the better programming partner? Human–human pair programming vs. human–AI pAIr programming. arXiv. https://doi.org/10.48550/arXiv.2306.05153

[10]. Song, A., & Azman, A. (2025). Leveraging symmetry in multi-agent code generation: A cross-verification collaboration protocol for competitive programming. Symmetry, 17(10), 1660. https://doi.org/10.3390/sym17101660

[11]. Sonkin, V., & Tudose, C. (2025). Beyond snippet assistance: A workflow-centric framework for end-to-end AI-driven code generation. Computers, 14(3), 94. https://doi.org/10.3390/computers14030094

[12]. Valový, M. (2023). Psychological aspects of pair programming. arXiv. https://doi.org/10.48550/arXiv.2306.07421

[13]. Zhang, H., Cheng, W., Wu, Y., & Hu, W. (2024). A pair programming framework for code generation via multi-plan exploration and feedback-driven refinement. arXiv. https://doi.org/10.48550/arXiv.2409.05001

[14]. Zhou, X., Liang, P., Zhang, B., Li, Z., Ahmad, A., Shahin, M., & Waseem, M. (2023). Exploring the problems, their causes and solutions of AI pair programming: A study on GitHub and Stack Overflow. arXiv. https://doi.org/10.48550/arXiv.2311.01020