Journal of Information Systems Engineering and Management
2025, 10(62s)
e-ISSN: 2468-4376
https://www.jisem-journal.com/

Research Article

The Future of Software Development: Programmers and AI Pair Programming

Pranati Sahu

Arizona State University, USA

ARTICLE INFO

Received: 28 Sept 2025

Revised: 03 Nov 2025

Accepted: 11 Nov 2025

ABSTRACT

The integration of AI programming assistants represents a paradigm shift in software development, transforming traditional workflows into collaborative human-machine partnerships. This article traces the evolution from basic code completion tools to advanced AI pair programming systems, examining their impact on productivity, code quality, and developer well-being. The transformation extends beyond technical aspects to reshape educational approaches, career development paths, and organizational structures. While delivering clear productivity benefits, AI assistants also introduce challenges related to code quality, security, and risks of developer deskilling. Ethical considerations emerge around safety-critical applications, intellectual property, and the long-term effects of automation on developer skills. The workforce landscape is evolving with new specialized roles, restructured teams, and altered global development patterns. Looking forward, advancing AI-human collaboration in programming requires overcoming these limitations. Progress will depend on explainability, multimodal interaction, domain-specialized systems, and collaborative learning models that complement—rather than replace—human expertise.

Keywords: AI pair programming, developer productivity, software engineering education, prompt engineering, human-AI collaboration

1. Introduction: The Emergence of AI-Assisted Programming

Software development has evolved dramatically, from early line editors to today's AI-powered programming assistants. The rise of AI as a collaborative coding partner marks a paradigm shift that is transforming how software is created, maintained, and evolved.

The journey toward AI-assisted programming began with rudimentary tools designed to streamline coding workflows. Early integrated development environments (IDEs) in the 1990s introduced basic code completion, syntax highlighting, and debugging tools that reduced cognitive load and accelerated development. These early tools focused on reducing syntax errors and improving readability rather than generating substantive code. Interactions on platforms like StackOverflow show how developers relied on collective knowledge to overcome technical challenges. Modern AI tools now attempt to emulate these collaborative patterns programmatically. Research indicates that the quality of answers, response time, and community reputation systems significantly influence how developers adopt solutions, providing crucial insights into effective AI assistant design [1].

The transition from simple completion tools to sophisticated AI-powered systems occurred gradually over two decades. Traditional code completion relied on limited pattern matching and predefined templates, offering suggestions based on local context and library definitions. This approach evolved into statistical models that could predict likely code completions by analyzing vast repositories of existing code. The breakthrough came with large language models (LLMs), which infer programming intent and generate contextually appropriate code. Recent studies show AI coding assistants significantly improve task completion times, especially for routine coding. These systems accelerate boilerplate code generation, allowing developers to focus on higher-level architectural decisions and creative problem-solving. This shift transforms the developer experience across expertise levels [2, 3].

Journal of Information Systems Engineering and Management
2025, 10(62s)
e-ISSN: 2468-4376
https://www.jisem-journal.com/

Research Article

Today's landscape features a diverse ecosystem of AI programming assistants that serve as collaborative partners in the development process. These tools range from open-source plugins that enhance existing IDEs to comprehensive commercial platforms offering end-to-end assistance. Modern AI programming assistants can generate entire functions from natural language descriptions, suggest optimizations for existing code, explain complex algorithms, and identify potential security vulnerabilities. This evolution has led to "AI pair programming," where the machine acts as a collaborator, contributing expertise across languages, frameworks, and design patterns [4].

As AI assistants become integrated into workflows, key research questions emerge: How do they affect productivity across experience levels? What is their impact on code quality, maintenance, and security? How might they reshape education and professional development? This paper uses a mixed-methods approach, combining quantitative productivity metrics with qualitative analysis of developer experiences. Through surveys, experiments, and case studies, it explores the implications of this shift for individual developers, organizations, and the future of the profession.

2. Transforming Development Workflows

The integration of AI assistants into software development represents a fundamental shift in how code is conceived, written, and maintained. This section examines how AI augmentation is transforming traditional development workflows, the mechanisms enabling these changes, and evidence of successful implementations across domains.

Traditional software development workflows have historically followed a linear progression through requirements gathering, design, implementation, testing, and maintenance phases. Developers typically spent a significant portion of their time navigating documentation, writing boilerplate code, and debugging syntax errors. This created bottlenecks and cognitive overhead that limited productivity and innovation. Recent case studies show that when AI code assistants are properly integrated, development teams experience substantial workflow transformations. The research demonstrates that AI assistants significantly reduce time spent on repetitive coding tasks while allowing developers to focus on high-level problem-solving. These studies also show that AI pair programming makes development more conversational and exploratory, enabling developers to rapidly test ideas and receive immediate feedback. These workflow changes extend beyond productivity gains, influencing how developers conceptualize and approach programming challenges [3, 5].

Modern AI coding assistants employ sophisticated mechanisms to generate and suggest code, integrating at multiple points within the development lifecycle. These systems analyze context from open files, project structure, version control history, and even developer interaction patterns to produce relevant suggestions. The underlying technologies range from statistical models trained on code repositories to transformer-based architectures that infer programming intent from natural language descriptions. Integration occurs both within the editor through inline suggestions and at higher levels through dedicated interfaces for complex generation tasks. Analyses across languages and frameworks show that effectiveness varies with task complexity, domain specificity, and quality of context. The research reveals that integration approaches need to be tailored to specific development environments and team structures rather than applied uniformly. The most effective implementations create a symbiotic relationship, maintaining developer agency by providing contextually appropriate assistance that respects coding standards and architectural boundaries, enhancing developer capabilities without disrupting workflows [4].

The emergence of AI pair programming has necessitated new approaches to task allocation, with certain responsibilities shifting between human developers and their AI counterparts. Effective allocation leverages complementary strengths: humans excel at business requirements, architectural

Journal of Information Systems Engineering and Management
2025, 10(62s)
e-ISSN: 2468-4376
https://www.jisem-journal.com/

Research Article

decisions, and ethics, while AI excels at code generation, pattern recognition, and library knowledge. Studies of real-world implementations show that successful task allocation evolves through several stages as teams gain experience with AI assistants. Initially, developers tend to use AI primarily for documentation and simple code generation tasks. As trust develops, they gradually delegate more complex tasks such as refactoring, test generation, and even architectural pattern implementation. This progressive delegation creates a feedback loop where developers refine collaboration strategies based on AI performance, leading to sophisticated partnerships that maximize the strengths of both human and machine [3, 5]. The transformation of development workflows through AI assistants manifests differently across software domains, with implementation strategies tailored to specific contexts. Comprehensive cross-domain analysis of AI assistant adoption shows that success factors vary significantly between application types and organizational contexts. In web and mobile development, AI tools excel at standardized UI components and integration logic, while in systems programming their strengths are documentation generation and code review. In data-intensive applications, AI assistants demonstrate particular strength in generating data transformation pipelines and visualization code. The findings emphasize that the greatest benefits come when organizations tailor AI integration to domain constraints and team expertise rather than applying generic approaches. Tailored implementation ensures AI amplifies existing strengths and addresses pain points, producing workflows that combine human creativity with machine efficiency in ways suited to each domain [4].

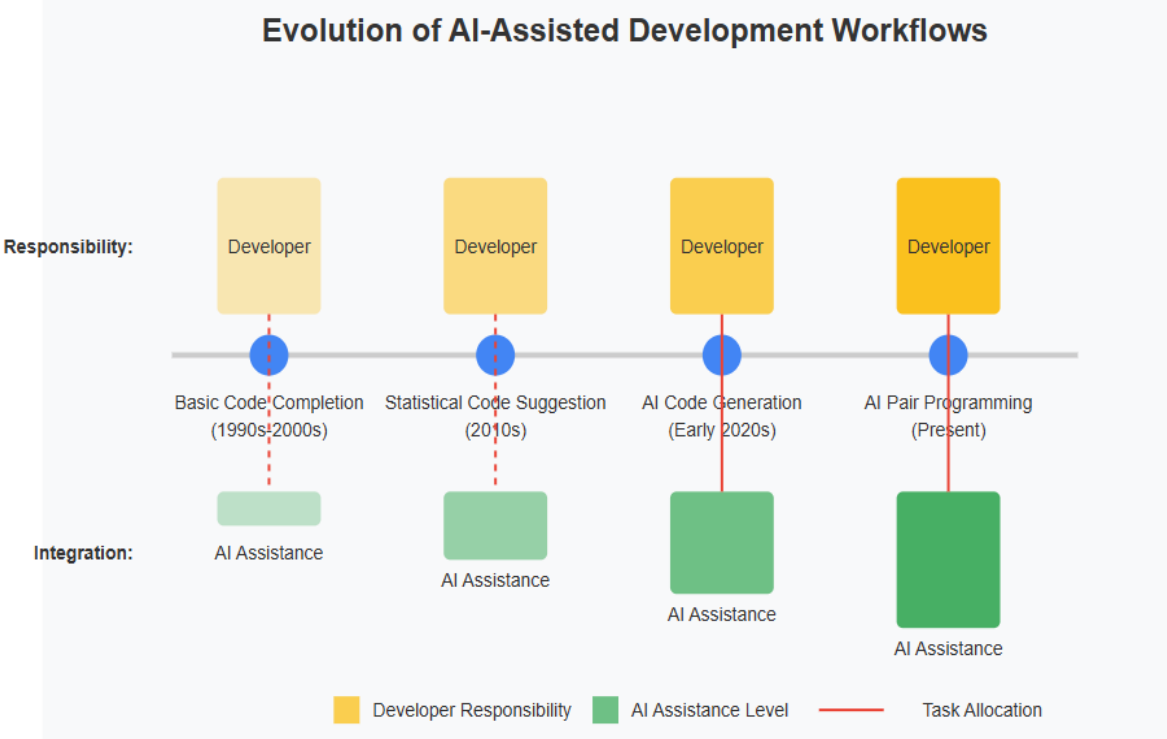

Fig. 1: Evolution of AI-Assisted Development Workflows. [3, 4]

3. Measuring Productivity and Quality Impacts

The integration of AI programming assistants into development workflows has prompted significant interest in quantifying their impact on productivity, code quality, developer experience, and organizational outcomes. This section reviews empirical evidence across these dimensions to provide a comprehensive understanding of how AI tools are transforming software development.

Journal of Information Systems Engineering and Management
2025, 10(62s)
e-ISSN: 2468-4376
https://www.jisem-journal.com/

Research Article

One of the most visible impacts of AI programming assistants is the acceleration of development velocity. Research on real-world developer interactions shows substantial changes in how programmers approach tasks when using AI code generation tools. Longitudinal studies of professional developers report meaningful increases in task completion rates across diverse programming scenarios. The research indicates that developers spend less time on repetitive coding patterns and more time on problem-solving and architectural considerations. Analyses show developers use AI most effectively for boilerplate code, API patterns, and test implementation—areas where manual work previously caused friction. Notably, the productivity impact appears particularly significant for tasks outside a developer's core area of expertise, suggesting that AI tools function as domain knowledge equalizers. Long-term measurements show developers become more effective with AI over time, building mental models of when and how to use the technology efficiently. This evolving relationship results in compound productivity gains as developers learn to collaborate more effectively with their AI counterparts. It transforms not only coding speed but also the cognitive approach to problem-solving [2, 3, 5].

Beyond accelerating development, AI programming assistants demonstrate significant potential to improve code quality across multiple dimensions. Analyses of AI-augmented development identify several quality indicators that improve when developers collaborate with AI systems. These metrics include not only traditional measures like defect density and test coverage but also more nuanced indicators such as code maintainability, documentation quality, and adherence to project-specific conventions. Research shows AI assistance excels at enhancing consistency across codebases by synthesizing patterns from large repositories. This standardization effect creates more uniform implementations that simplify maintenance and knowledge transfer within development teams. The greatest quality improvements occur when developers use AI tools interactively. This iterative process involves requesting code, critically evaluating the output, and refining it with further prompts. This collaboration produces code that combines AI efficiency with human intention and understanding. Research also shows AI-assisted development improves cross-cutting concerns such as security validation, input sanitization, and error handling—areas where humans often introduce inconsistencies [5, 6].

The psychological impact of AI programming assistants is another critical dimension of their influence on development. Research investigating developer satisfaction and well-being shows that AI code assistance tools significantly influence how developers experience their work. Surveys consistently show developers enjoy programming more with AI assistants, citing reduced frustration with boilerplate and repetition. This satisfaction stems partly from the ability to maintain a flow state. AI tools help by reducing the need for context switching to look up documentation or syntax. Research indicates developers feel accomplishment and creative partnership when collaborating with AI, often describing it as enhancing rather than diminishing their professional identity. Studies show reduced anxiety when tackling unfamiliar technologies, as AI provides scaffolding that boosts confidence and reduces impostor syndrome. This psychological benefit appears particularly significant for early-career developers and those working in isolation without immediate access to senior mentors. Data further suggests effective AI assistance creates a virtuous cycle: reduced friction leads to more experimentation, faster learning, and greater productivity [2, 5].

The productivity and quality impacts of AI programming assistants translate into meaningful economic implications for software organizations. Research on AI-augmented development identifies several mechanisms for creating organizational value. ROI studies highlight value streams beyond time savings, including faster onboarding, reduced technical debt, and greater innovation as developers shift focus from implementation to problem-solving. Research shows organizations gain the most when they pair AI adoption with process adjustments such as revised code reviews, documentation, and quality assurance practices. Economic analysis suggests that AI assistants particularly benefit organizations maintaining large, complex codebases where knowledge

Journal of Information Systems Engineering and Management
2025, 10(62s)
e-ISSN: 2468-4376
https://www.jisem-journal.com/

Research Article

fragmentation traditionally creates significant productivity barriers. Studies show successful adoption often begins with pilot programs on specific teams or projects, allowing contextual learning before scaling. Research suggests AI may alter build-versus-buy economics, as faster custom development reduces the appeal of pre-built solutions. This shift potentially enables more precise alignment between business requirements and software implementations without corresponding increases in development costs [2, 3, 6].

Dimension	Key Findings	Organizational Impact
Development Velocity	Significant task completion improvements, especially for unfamiliar domains	Accelerated time-to-market and increased development throughput
Code Quality	Improved maintainability, consistency, and adherence to best practices	Reduced technical debt and lower maintenance costs over time
Cognitive Load	Reduced mental effort for implementation details and sustained flow states	Improved developer retention and higher job satisfaction metrics
Skill Development	Accelerated learning in unfamiliar domains with reduced anxiety	Faster onboarding for new team members and broader skill distribution
Economic Value	Favorable ROI through multiple value streams beyond simple time savings	Competitive advantage through faster development cycles and increased innovation capacity

Fig. 2: Impacts of AI Programming Assistants on Software Development. [5, 6]

4. Evolving Developer Skills and Education

The rapid integration of AI programming assistants into software development practices is catalyzing fundamental changes in the skills required for professional success in the field. This transformation extends beyond individual developers, affecting educational institutions, training methods, and the broader ecosystem of software development learning. This section examines how developer competencies, educational approaches, and learning patterns are evolving in response to AI-augmented development environments.

The emergence of AI programming assistants is reshaping the skill profile of effective software developers, with certain technical competencies becoming less critical while others gain prominence. Recent research on the future of programming in the age of advanced AI reveals a substantial redefinition of software engineering roles. Studies of competitive programming show that developers using AI assistants can solve more complex problems than before, suggesting a shift in the cognitive boundaries of software development. This expansion of capability is creating a divide in the industry. Expertise is now measured less by recalling syntax and more by the ability to frame problems for effective AI collaboration. Interviews with industry leaders indicate that hiring criteria are evolving to prioritize systems thinking, problem decomposition, and contextual awareness over traditional coding proficiency metrics. Research suggests developers with a strategic perspective—understanding why approaches are chosen and how components interact—achieve better outcomes with AI than those focused only on tactical coding. This shift is especially clear in domains like machine learning

engineering and distributed systems, where navigating complexity and making architectural decisions remain the key contributions of human developers in AI-augmented environments [7, 9].

Educational institutions are responding to the changing skill landscape with significant revisions to computer science curricula and teaching approaches. Analyses of pedagogical frameworks for teaching prompt engineering and LLM interaction reveal several emerging best practices. Research identifies a three-tier curriculum: foundational courses on basic collaboration, intermediate courses on domain-specific applications, and advanced courses on meta-prompt engineering and AI system design. Studies show students explicitly taught AI collaboration strategies develop better mental models of AI capabilities and limitations, enabling more effective professional partnerships. The research highlights the importance of balancing traditional programming fundamentals with AI-specific skills. Students still need enough technical depth to evaluate and refine AI-generated code. Experiments comparing instructional approaches show studio-based learning—where students tackle complex projects with AI under faculty guidance—yields strong gains in both critical thinking and technical skills. This shift extends to assessment, focusing on students' ability to direct AI toward solving novel problems rather than producing code alone. The educational research emphasizes that preparing future developers requires cultivating both technical foundations and higher-order thinking skills that will remain distinctly human as AI capabilities continue to advance [8].

The availability of AI programming assistants is fundamentally altering how practicing developers acquire new skills and knowledge throughout their careers. Studies of professional developers adapting to AI-augmented environments reveal distinctive learning and skill acquisition patterns that differ from traditional approaches. Researchers identify "exploratory skill acquisition," where developers use AI to tackle projects beyond their expertise, with the tools scaffolding learning while building competence. Instead of studying concepts first, developers now often begin implementing with AI guidance, seeking deeper understanding only when needed to solve problems. Analyses of developer–AI interactions show this exploratory approach yields more grounded knowledge, as concepts are learned through direct application rather than in isolation. Research shows successful adopters build sophisticated mental models of AI capabilities, learning when to trust suggestions and when to scrutinize them. This metacognitive awareness represents a critical component of effective self-learning in AI-augmented environments. Importantly, studies show professional communities now emphasize collaboration strategies over purely technical solutions, reflecting these new learning patterns. This shift in self-learning has major implications for professional development and organizational knowledge management, highlighting the need for frameworks that support exploration while ensuring depth [7].

As AI becomes integral to workflows, effectively directing systems through thoughtful prompts and critical evaluation has emerged as a crucial meta-skill. Research on prompt engineering education identifies distinct competency levels that define effective AI–human collaboration in software development. Studies outline a progression from basic prompt construction to advanced system orchestration, where practitioners decompose complex problems, manage context across interactions, and evaluate outputs across quality dimensions. Analyses of educational interventions show explicit instruction in prompt engineering significantly improves developers' ability to elicit useful code, with structured frameworks outperforming ad-hoc approaches. Research also highlights critical evaluation as a complementary skill, covering both technical code quality and strategic suitability. Studies tracking AI-generated code reveal common failure modes—hallucinated functions, incorrect API usage, and subtle logic errors—that skilled developers learn to anticipate and mitigate. This research suggests effective AI collaboration requires both technical and metacognitive skills: developers must understand the domain, their AI tools' capabilities, and their own knowledge boundaries to create effective partnerships. These emerging competencies represent a major evolution in software development, reshaping how the field views expertise and professional identity in an AI-augmented landscape [6, 8].

Journal of Information Systems Engineering and Management
2025, 10(62s)
e-ISSN: 2468-4376
https://www.jisem-journal.com/

Research Article

Fig. 3: Evolving Developer Skills in AI-Augmented Environments. [7, 8]

5. Future Directions and Challenges

As AI programming assistants evolve, critical challenges and opportunities will shape their future trajectory and impact. This section examines the technical limitations of current systems, ethical considerations surrounding their use in sensitive domains, potential workforce transformations, and promising research directions for advancing human-AI collaboration in programming.

Despite their impressive capabilities, contemporary AI programming assistants exhibit significant technical limitations that constrain their utility and reliability. Empirical analyses of AI coding assistants in production reveal several persistent challenges that limit effectiveness. Studies of thousands of developer–AI interactions identify distinct failure patterns, categorized by type and severity. A major limitation is context window size, which prevents AI systems from grasping large codebases and often results in suggestions that break architectural patterns. AI systems also struggle with domain-specific logic, especially when implementation requires specialized knowledge missing from training data. Research has documented a phenomenon termed "confidence mirage," wherein AI assistants generate incorrect code with misleading certainty markers, creating potential traps for inexperienced developers. Temporal reasoning is another limitation, as AI systems struggle to anticipate how code evolves over time or interacts with concurrent processes. The hallucination problem—where systems confidently reference non-existent functions or APIs—remains prevalent despite advances in model architecture. Research also finds failure modes differ across paradigms, with functional programming posing unique challenges compared to imperative approaches. Error patterns cluster around specific types of tasks, suggesting targeted areas for improvement. These limitations force developers to devise workarounds, building intuition for when AI is reliable versus when human oversight is essential. Overall, findings emphasize that AI assistants work best as collaborative tools requiring human guidance, with effectiveness depending on developers' ability to recognize and compensate for these limitations [6].

Journal of Information Systems Engineering and Management
2025, 10(62s)
e-ISSN: 2468-4376
https://www.jisem-journal.com/

Research Article

The increasing integration of AI programming assistants into development workflows raises important ethical questions, particularly in safety-critical domains where software failures can have serious consequences. Research on balancing innovation and ethics in AI-augmented development highlights multiple tension points requiring careful consideration. Studies of regulatory compliance reveal challenges with traceability and accountability, especially in sectors such as healthcare, aviation, and financial services. Emerging frameworks for responsible AI use emphasize layered verification, with AI contributions reviewed more rigorously as risk increases. Ethical concerns also include intellectual property and attribution, since AI trained on public repositories may generate code with unclear provenance. The research documents cases where AI-generated implementations inadvertently incorporated copyrighted patterns, creating legal uncertainties that existing frameworks struggle to address. Studies of developer psychology warn that overreliance on AI may erode fundamental understanding, especially among early-career professionals. This concern is especially pronounced for security-critical implementations, where subtle vulnerabilities may escape detection by both AI systems and developers with diminished manual coding experience. Research also notes a trade-off: more explainable AI systems often show reduced capability, complicating use in high-stakes domains. Surveys of organizations show a growing consensus: human oversight remains essential for certain decisions, regardless of AI proficiency. These ethical considerations are increasingly recognized not merely as constraints but as essential design parameters for the responsible advancement of AI-augmented software development [10].

The integration of AI programming assistants signals major transformations in workforce composition, organizational structures, and industry dynamics. Empirical studies of industry adoption provide insights into emerging workforce transformations. Tracking studies show experienced developers now spend less time on implementation and more on design, review, and mentoring after adopting AI assistants. Research highlights a divide between prompt engineering as a specialty and general AI use, with organizations valuing developers skilled in sophisticated AI collaboration. Analysis of job postings across the technology sector documents the appearance of new role categories combining traditional development skills with AI expertise, including "AI-augmented developer" and "development workflow architect." These roles emphasize optimizing human–AI interaction rather than direct coding. Studies show organizations experimenting with AI-enabled teams, allowing smaller units to maintain larger and more complex systems. Research finds distributed teams report better knowledge sharing and reduced friction when shared AI assistants standardize implementation approaches. Longitudinal studies suggest developers with strong AI collaboration skills gain versatility across projects, while those who resist adapt to narrow niches where manual coding remains essential. These workforce transformations appear to be accelerating historical trends toward higher abstraction levels in programming, with AI systems increasingly handling implementation details while human developers focus on translating business requirements into technical specifications and architectural decisions [9].

Improving the effectiveness and reliability of AI–human programming collaboration requires research across multiple dimensions. Reviews of current challenges and opportunities identify high-priority research directions for advancing next-generation systems. Developer feedback highlights the need for assistants with consistent large-context understanding, pointing to retrieval-augmented generation and modular reasoning as promising approaches. Promising directions for explainability include auto-generated documentation that justifies decisions and flags weaknesses for human review. Analyses of developer–AI interaction reveal inefficiencies in text-based interfaces, suggesting research into multimodal paradigms with visual programming and natural language conversations about intent. Studies examining educational applications identify opportunities for AI systems that adapt to developer expertise levels, providing more detailed explanations for novices while offering higher-level abstractions to experienced practitioners. Research emphasizes domain-specific assistants with deep framework or industry knowledge, offering more reliable support within narrower contexts. Experiments with collaborative learning models—where AI improves through ongoing team

Journal of Information Systems Engineering and Management
2025, 10(62s)
e-ISSN: 2468-4376
https://www.jisem-journal.com/

Research Article

interaction—show promise for assistants aligned with organizational practices and standards. Finally, researchers call for evaluation methods that assess human–AI collaboration across the software lifecycle, not just code generation. These research directions aim to address current limitations, advancing toward partnership models where AI and human capabilities truly complement—rather than replace—each other [4, 10].

Fig. 4: Future Directions and Challenges in AI-Augmented Software Development

Conclusion

The emergence of AI programming assistants marks a fundamental transformation in how software is conceived, developed, and maintained. These tools have progressed from simple code completion to collaborative partners, reshaping workflows across domains and enabling productivity gains through reduced cognitive load and faster implementation. Beyond metrics, these systems profoundly influence developer satisfaction, learning, and professional identity. Educational institutions and industry organizations are adapting to this new reality by emphasizing higher-order thinking skills, architecture design capabilities, and effective AI collaboration strategies. Despite advances, challenges remain: context limitations, the need for human oversight in critical applications, intellectual property concerns, and risks of deskilling. The future of AI-augmented development depends on systems with better reasoning, domain expertise, transparency, and adaptability—enhancing human creativity while automating routine tasks. As these technologies continue to mature, the most successful developers and organizations will be those that thoughtfully integrate AI capabilities while maintaining strategic human direction and creative vision.

Journal of Information Systems Engineering and Management
2025, 10(62s)
e-ISSN: 2468-4376
https://www.jisem-journal.com/

Research Article

References

[1] Wang, S., Lo, D., & Jiang, L. “An Empirical Study on Developer Interactions in Stack Overflow.” Proceedings of the 28th ACM Symposium on Applied Computing (SAC), 2013. ACM DOI: https://dl.acm.org/doi/10.1145/2480362.2480557).

[2] Albert Ziegler, Eirini Kalliamvakou, X. Alice Li, Andrew Rice, Devon Rifkin, Shawn Simister, Ganesh Sittampalam, Edward Aftandilian, "Measuring GitHub Copilot’s Impact on Productivity," Communications of the ACM, 67(3), 2024. https://cacm.acm.org/research/measuring-github-copilots-impact-on-productivity/

[3] Cui, K. Z., Demirer, M., Jaffe, S., Musolff, L., Peng, S., & Salz, T. “The Effects of Generative AI on High-Skilled Work: Evidence from Three Field Experiments with Software Developers.” MIT/SSRN Working Paper, 2025. https://economics.mit.edu/sites/default/files/inline-files/draft_copilot_experiments.pdf

[4] Ziyin Zhang, Chaoyu Chen, Bingchang Liu, Cong Liao, Zi Gong, Hang Yu, Jianguo Li, Rui Wang, "Unifying the Perspectives of NLP and Software Engineering: A Survey on Language Models for Code," arXiv preprint arXiv:2311.07989 [cs.CL], 2024. https://arxiv.org/abs/2311.07989

[5] Paradis, E., Grey, K., Madison, Q., Nam, D., Macvean, A., Meimand, V., Zhang, N., Ferrari-Church, B., & Chandra, S. “How Much Does AI Impact Development Speed? An Enterprise-Based Randomized Controlled Trial.” arXiv:2410.12944, 2024. https://arxiv.org/abs/2410.12944

[6] Jiessie Tie, Pengyu Nie, Yifan Wu, Shurui Zhou, Danny Dig, "LLMs are Imperfect, Then What? An Empirical Study on LLM Failures in Software Engineering," arXiv preprint arXiv:2411.09916 [cs.SE], 2024. https://arxiv.org/abs/2411.09916

[7] Nguyen, S., Babe, H. M., Zi, Y., Guha, A., Anderson, C. J., & Feldman, M. Q. “How Beginning Programmers and Code LLMs (Mis)read Each Other.” arXiv:2401.15232, 2024. https://arxiv.org/abs/2401.15232

[8] Manorat, P., Wacharamanotham, W., & Anutariya, C. “Artificial intelligence in computer programming education: A systematic literature review.” Computers & Education: Artificial Intelligence, 8, 100217, 2025. https://www.sciencedirect.com/science/article/pii/S2666920X25000438

[9] World Economic Forum, Geneva, 2025. "The Future of Jobs Report 2025," World Economic Forum, 2025. https://reports.weforum.org/future-of-jobs-2025/

[10] National Institute of Standards and Technology (NIST), "Artificial Intelligence Risk Management Framework (AI RMF 1.0)," NIST AI 100-1, 2023. https://nvlpubs.nist.gov/nistpubs/ai/nist.ai.100-1.pdf