Journal of Sustainable Software Engineering and Information Systems
Volume 2, Issue 2, 79-112.
e_ISSN: 3123-3007
https://e-journal.gomit.id/jsseis
DOI: 10.58723/jsseis.v2i2.189

The Impact of AI-Assisted Coding Tools on Agile Software Development Practices: An Empirical Study

Baheer Elias*
Rana University, AFGHANISTAN

Ehsan Jafari
Rana University, AFGHANISTAN

*** Corresponding author:**
Baheer Elias, Rana University, AFGHANISTAN. ✉Baheer.elias@gmail.com

Article Info

Article history:

Received: June 16, 2026
Revised: July 11, 2026
Accepted: August 05, 2026

Keywords:

Artificial Intelligence
AI-Assisted Coding Tools
Agile Software Development
Developer Productivity
Software Engineering

Abstract

Background: Artificial Intelligence (AI) has become increasingly integrated into software development, with tools such as ChatGPT, GitHub Copilot, Cursor AI, Claude, DeepSeek, and Amazon Code Whisperer now widely used to assist developers. While previous research has highlighted the potential of these tools to improve individual productivity, their influence on Agile software development practices has received comparatively less attention.
Aims: This study examines how AI-assisted coding tools affect Agile software development practices, focusing on developer productivity, code quality, Agile workflows, and associated challenges.
Methods: A quantitative survey was conducted with 91 software professionals using AI-assisted coding tools in Agile environments. Responses were analyzed using descriptive statistics, reliability testing (Cronbach's Alpha), and correlation analysis.
Results: AI-assisted coding tools are widely perceived as beneficial. Participants reported improvements in coding efficiency (M = 4.20, SD = 0.68), faster task completion (M = 4.37, SD = 0.77), and better support for Agile activities such as requirement clarification (M = 4.08, SD = 0.70) and sprint execution (M = 4.00, SD = 0.67). However, concerns were raised regarding over-dependence (M = 3.79, SD = 0.75), security issues (M = 3.78, SD = 0.74), and reduced learning opportunities (M = 4.26, SD = 0.99).
Conclusion: The study indicates that AI-assisted coding tools can provide meaningful support for Agile development when used alongside human expertise. Organizations should adopt them carefully and establish practices that encourage responsible use, continuous learning, and quality assurance.

To cite this article: Elias, B. & Jafari, E. (2026). The Impact of AI-Assisted Coding Tools on Agile Software Development Practices: An Empirical Study. *Journal of Sustainable Software Engineering and Information Systems, 2*(2), 79-112. https://doi.org/10.58723/jsseis.v2i2.189
This article is licensed under a Creative Commons Attribution-ShareAlike 4.0 International License ©2026 by author/s

INTRODUCTION

Artificial intelligence (AI)-powered coding assistants have become increasingly prevalent in modern software engineering environments. Tools such as GitHub Copilot, ChatGPT, Amazon CodeWhisperer, Cursor AI, Claude AI, and DeepSeek now support developers across a wide range of activities, including code generation, debugging, documentation, code review, and learning assistance. The rapid advancement of large language models (LLMs) has significantly expanded the capabilities of these tools, making them valuable resources throughout the software development lifecycle (Hou et al., 2024; Sauvola et al., 2024).

Despite the rapid adoption of these tools, a significant gap remains in understanding their systematic impact on software development practices, particularly within Agile team environments. While

anecdotal evidence and individual case studies suggest productivity improvements, empirical research examining the holistic effects on team collaboration, sprint execution, and quality assurance in real-world Agile settings remains sparse (Geger et al., 2026; Ziebell et al., 2026). This lack of systematic evidence presents a critical problem for organizations making strategic decisions about AI tool adoption, for team leaders adapting workflows, and for developers navigating the evolving demands of their profession (Neumann et al., 2026; Nguyen-Duc et al., 2025; Témolé & Atanasova, 2025). Without rigorous empirical investigation, organizations risk either underutilizing valuable AI capabilities or, conversely, adopting them in ways that may inadvertently disrupt effective collaboration, compromise code quality, or undermine the professional development of their technical teams.

The scale of AI adoption in software development is substantial and growing rapidly. According to recent industry surveys, approximately 70-80% of developers report using AI coding assistants in some capacity (Ehsani et al., 2026; Ziegler et al., 2024). GitHub reports that Copilot alone has been adopted by over 1.5 million developers worldwide, and early adopters have experienced reported productivity gains ranging from 20% to 55% (Peng et al., 2023). Studies have documented task completion times reduced by as much as 40-50% for certain coding activities when developers use AI-assisted tools compared to traditional coding approaches (Peng et al., 2023; Ziegler et al., 2024).

Recent studies indicate that developers are actively integrating AI-assisted tools into their daily workflows to improve efficiency and reduce repetitive work. Comparative evaluations of ChatGPT and GitHub Copilot, for example, demonstrate that both tools can support software development tasks effectively, although their performance varies depending on programming context, task complexity, and developer expertise (Fajkovic & Rundberg, 2023; Yetiştiren et al., 2023). Similarly, C. Zhang et al. (2023) found that developers generally perceive GitHub Copilot as beneficial for routine coding activities while acknowledging several practical limitations.

Despite their growing popularity, AI-powered coding assistants also present several challenges. Research suggests that developers may become overly dependent on AI-generated suggestions, potentially reducing opportunities for critical thinking and problem-solving (C. Zhang et al., 2023). Furthermore, the quality of AI-generated outputs is not always consistent, particularly when dealing with complex, domain-specific, or large-scale software projects (Hou et al., 2024; Sauvola et al., 2024; Storey et al., 2025). As a result, human oversight remains essential to ensure the reliability and appropriateness of generated code.

Recent reviews suggest that generative AI tools are increasingly being integrated into software engineering activities beyond code generation, including requirements analysis, documentation, testing, and developer support (Hou et al., 2024; Nascimento et al., 2023; Sauvola et al., 2024). Overall, the existing literature indicates that AI-powered coding assistants have become influential tools in software engineering. While these technologies offer substantial benefits in terms of efficiency and developer support, their effective adoption requires careful consideration of both their capabilities and limitations. This study was conducted with software professionals primarily, working across various industries including technology, finance, healthcare, e-commerce, and consultancy services. The respondents represented a range of Agile environments, from early-stage startups to established enterprise organizations, providing a diverse perspective on AI tool adoption across different organizational contexts.

This study is grounded in two complementary theoretical perspectives that together provide a comprehensive lens for examining the impact of AI-assisted coding tools on Agile software development practices. The Technology Acceptance Model (Davis, 1989) provides a foundational framework for understanding how and why users adopt new technologies. TAM posits that two primary factors influence technology adoption: perceived usefulness (the degree to which a user believes that using a particular system would enhance their job performance) and perceived ease of use (the degree to which a user believes that using a particular system would be free of effort). In the context of this study, perceived usefulness corresponds to developers' perceptions of productivity

improvements from AI tools, while perceived ease of use relates to the intuitiveness and accessibility of these tools. TAM has been extensively validated in information systems research and has been applied to understand technology adoption in software engineering contexts (Scherer et al., 2019; Venkatesh et al., 2003). By applying TAM, this study examines how developers' perceptions of AI tool usefulness and ease of use influence their adoption decisions, their trust in AI-generated outputs, and their willingness to integrate these tools into their workflows.

Complementing TAM, Sociotechnical Systems Theory (Trist & Bamforth, 1951; Bostrom & Heinen, 1977) emphasizes that effective work systems require the joint optimization of both social and technical subsystems. From this perspective, software development is not merely a technical activity but a sociotechnical process that involves human judgment, collaboration, and communication alongside technical tools and practices (Frank et al., 2024; Xames & Topcu, 2025). The introduction of AI-assisted coding tools represents a significant change to the technical subsystem, but its success depends on how well it aligns with the social subsystem—including team dynamics, communication patterns, and collaborative workflows. This theory is particularly relevant for Agile development, which places strong emphasis on team collaboration, shared understanding, and continuous communication.

Applying Sociotechnical Systems Theory, this study investigates whether the introduction of AI tools disrupts or enhances existing social structures within Agile teams. The findings address concerns about whether AI tools support or undermine team collaboration, knowledge sharing, and collective problem-solving (Pileggi & Bharathy, 2026; Zacharias et al., 2026). One of the most widely discussed benefits of AI-assisted coding tools is their potential to improve developer productivity. Recent advances in generative AI have enabled tools such as GitHub Copilot and ChatGPT to assist developers in writing code, generating solutions, debugging errors, and automating repetitive programming tasks (Mohamed et al., 2026). As a result, researchers have increasingly examined the impact of these tools on software development efficiency.

Empirical evidence suggests that AI-assisted tools can significantly improve coding productivity. Peng et al. (2023) reported that developers using GitHub Copilot completed programming tasks more quickly than those working without AI assistance. Similarly, Rajbhoj et al. (2024) found that generative AI tools can accelerate software development activities by reducing the time required for coding and problem-solving tasks. These findings indicate that AI-powered assistants can help developers focus more on higher-level design and decision-making activities rather than routine coding work.

Several studies have also compared the effectiveness of different AI tools in practical software development scenarios. Fajkovic & Rundberg (2023) observed that both ChatGPT and GitHub Copilot contribute to improved development efficiency, although their effectiveness varies depending on the complexity and nature of the task. Likewise, Jiang et al. (2022) demonstrated that large language models can support natural language programming by translating human instructions into executable code, thereby reducing the effort required to implement software solutions.

Despite these productivity gains, researchers caution that improvements are not always consistent across different development contexts. Stray et al. (2024) found that while AI assistants can increase individual productivity, their influence on collaborative development activities may be more complex. Similarly, Stray et al. (2025) reported that AI-assisted development affects solo and pair programming differently, suggesting that productivity improvements at the individual level do not automatically translate into improved team performance.

Another important consideration is the quality of the generated output. Although AI tools can accelerate coding activities, developers often spend additional time validating, testing, and refining AI-generated code before it can be integrated into production systems (Liu et al., 2024). Consequently, productivity gains may vary depending on the level of review and verification required.

Empirical investigations conducted in both academic and industrial contexts continue to report measurable productivity gains associated with AI-assisted development tools, although the magnitude of these benefits varies according to project characteristics and developer experience (R. Pandey et al., 2024; Peng et al., 2023; Rajbhoj et al., 2024).

Overall, the literature suggests that AI-assisted coding tools can substantially enhance developer productivity by reducing routine work and accelerating software development tasks. However, the extent of these benefits depends on factors such as task complexity, development context, team collaboration, and the quality of generated code. While individual productivity gains are well-documented, the effect of AI tools on team collaboration and workflows is less understood. This represents a significant gap in the literature, given that most software development occurs in team settings where communication, coordination, and shared understanding are essential.

Research suggests that AI-assisted development influences individual and team workflows. Stray et al. (2024) observed that AI tools affect both solo and pair programming practices by changing how developers approach problem-solving and task execution. While AI assistance can improve efficiency, it may also alter communication patterns and collaboration dynamics within development teams. These findings are particularly relevant for Agile environments, where effective teamwork and continuous communication are fundamental principles.

The effectiveness of human–AI collaboration depends heavily on the ability of developers to evaluate and validate AI-generated outputs. Stray et al. (2024) found that developers frequently use tools such as GitHub Copilot and ChatGPT as collaborative assistants rather than fully autonomous systems. Developers typically review suggestions, modify generated code, and make final decisions regarding implementation. This process reinforces the idea that human expertise remains central to software development despite advances in AI capabilities.

Beyond technical performance, emotional and psychological factors play a significant role in human–AI collaboration. Eshraghian & Cesare (2024) found that developers experience a range of emotional responses when interacting with GitHub Copilot, including increased confidence, satisfaction, curiosity, frustration, and occasional distrust. These emotional responses can influence technology adoption, trust in AI-generated recommendations, and long-term usage behavior. Developers who perceive AI tools as reliable and useful are generally more likely to integrate them into their daily workflows.

Research on real-world software projects further highlights both the opportunities and challenges associated with AI-supported development. Pandey et al. (2024) reported that GitHub Copilot can improve development efficiency in practical projects, but developers continue to face challenges related to output accuracy, code reliability, and validation requirements. These findings reinforce the importance of human oversight and critical evaluation when integrating AI-generated content into software systems.

The integration of AI-assisted coding tools into software development organizations raises important questions about governance, quality assurance, and strategic decision-making. As organizations increasingly adopt these technologies, they must establish practices that balance productivity gains with quality and security requirements. Research on design pattern recognition by large language models indicates that AI systems still face limitations when interpreting complex architectural concepts and advanced software engineering principles (R. Pandey et al., 2024). Although these models demonstrate promising capabilities, they are not yet capable of fully replacing human expertise in software design and quality assurance.

The need for systematic evaluation of AI-generated outputs has become increasingly important as organizations seek to balance productivity improvements with software quality requirements (Felizardo et al., 2024; Liu et al., 2024). Sauvola et al. (2024) argued that while generative AI can accelerate development activities, organizations must carefully evaluate the long-term implications

of integrating AI-generated code into complex software systems. Poorly structured or insufficiently documented code may create technical debt that increases maintenance costs over time.

Security concerns represent another major challenge associated with AI-assisted development. Because generative AI models are trained on large repositories of publicly available code, they may inadvertently reproduce insecure coding practices or vulnerable code patterns. As a result, developers must remain vigilant when incorporating AI-generated suggestions into software projects. Security reviews, testing procedures, and code inspections continue to play a critical role in ensuring software reliability and protecting systems from potential vulnerabilities (Sauvola et al., 2024).

Emerging research also demonstrates that AI technologies are being applied to increasingly specialized software engineering domains, including advanced software design analysis and quantum software development, indicating that the influence of AI is likely to extend beyond conventional programming activities (Mendiluze Usandizaga et al., 2025; S. K. Pandey et al., 2025).

A careful reading of the literature reveals several unresolved tensions that warrant further investigation:

Tension 1: Individual Productivity vs. Team Performance

While studies consistently show individual productivity gains (Peng et al., 2023; Rajbhoj et al., 2024), there is growing evidence that these gains may not automatically translate to team-level performance improvements Stray et al. (2024). This raises important questions about whether AI tools are truly beneficial for Agile teams or whether they create unintended coordination costs.

Tension 2: Speed vs. Quality

AI tools clearly accelerate coding tasks, but researchers disagree on whether this comes at the cost of code quality, maintainability, and security (Liu et al., 2024; Yetiştiren et al., 2023). Some studies suggest that AI-generated code requires substantial refinement (Sauvola et al., 2024), while others argue that AI can actually improve code quality. The true relationship between AI assistance and code quality remains unclear.

Tension 3: Efficiency vs. Learning

There is a growing concern that over-reliance on AI tools may reduce opportunities for developers to develop problem-solving skills and deep technical understanding (C. Zhang et al., 2023). However, others argue that AI tools can actually accelerate learning by providing instant feedback and exposing developers to new approaches (Eshraghian & Cesare, 2024). The net effect on developer learning and professional development remains uncertain.

Tension 4: Automation vs. Human Judgment

While AI tools can automate many routine tasks, there is debate about how much human judgment is still required. Some researchers emphasize the importance of human oversight and validation Stray et al. (2024), while others see AI as increasingly capable of making autonomous decisions (Hou et al., 2024). The appropriate balance between automation and human control remains an open question.

Agile software development is a project management and software development approach that emphasizes iterative delivery, continuous feedback, customer collaboration, and flexibility in responding to changing requirements (Schwaber & Sutherland, 2020; Dingsøyr et al., 2010). Unlike traditional waterfall methodologies, which follow a linear, sequential process, Agile frameworks such as Scrum, Kanban, and Extreme Programming (XP) organize work into short development cycles known as sprints that typically last one to four weeks. These sprints involve specific rituals including sprint planning (where tasks are prioritized and assigned), daily stand-up meetings (for coordination and problem-solving), sprint reviews (to demonstrate completed work to stakeholders), and retrospectives (to reflect on process improvements). Agile methodologies prioritize working

software over comprehensive documentation, face-to-face communication over formal processes, and responding to change over following a rigid plan.

The foundations of Agile development are strongly linked to collaboration and continuous improvement. Agile teams typically operate through short development cycles, regular stakeholder interaction, and frequent evaluation of project progress. According to Dingsøyr et al., (2010), Agile methodologies have evolved significantly over the past two decades and continue to influence software engineering research and practice due to their ability to improve responsiveness and project outcomes. Their work highlights how Agile principles support effective communication, rapid adaptation to change, and continuous delivery of value to customers.

One of the key strengths of Agile methodologies is their emphasis on teamwork and knowledge sharing. Successful Agile implementation depends not only on technical competence but also on effective communication among team members. Activities such as sprint planning, daily stand-up meetings, sprint reviews, and retrospectives provide structured opportunities for collaboration and collective decision-making (Schwaber & Sutherland, 2020). These practices help teams identify challenges early, coordinate development efforts, and continuously improve their workflows.

Research also demonstrates that Agile practices contribute to improved project flexibility and organizational responsiveness. By dividing development work into smaller iterations, Agile teams can respond more effectively to evolving customer requirements and emerging technical challenges (Dingsøyr et al., 2010: Strode & Hopf, 2024). This adaptability has made Agile particularly attractive in environments characterized by uncertainty and rapid technological change.

Recent studies further suggest that Agile methodologies continue to evolve as organizations adapt to new technological trends and changing business needs (Lassenius et al., 2025; Strode & Hopf, 2024). Lassenius et al. (2025), for example, examined software development practices within public-sector organizations and demonstrated the importance of organizational learning, collaboration, and knowledge transfer in successful software development initiatives. Their findings reinforce the view that human interaction and effective teamwork remain essential components of successful software engineering projects.

The growing adoption of AI-assisted coding tools introduces new considerations for Agile teams. Since Agile practices are heavily dependent on communication, collaboration, and shared understanding, the integration of AI technologies has the potential to influence how teams perform key Agile activities. AI-generated code, automated recommendations, and intelligent development support may improve efficiency, but they may also affect team dynamics, decision-making processes, and collaboration patterns.

The integration of AI-assisted coding tools into Agile software development environments has attracted increasing attention from both researchers and practitioners. While a substantial body of literature has examined the effects of AI on individual developer productivity, fewer studies have explored how these technologies influence Agile practices at the team level. Given that Agile methodologies rely heavily on collaboration, communication, iterative development, and continuous feedback, understanding the interaction between AI tools and Agile workflows has become an important research area.

AI-powered tools such as GitHub Copilot and ChatGPT can support multiple activities within Agile projects, including code generation, bug fixing, documentation, testing, and knowledge sharing (Hou et al., 2024; Sauvola et al., 2024). By automating repetitive programming tasks, these tools allow developers to allocate more time to higher-level activities such as system design, problem-solving, and sprint planning. As a result, many organizations view AI-assisted development as a potential mechanism for increasing Agile team efficiency.

Research suggests that AI tools may positively influence Agile development processes by accelerating feature implementation and reducing development effort. Peng et al. (2023) demonstrated that developers using GitHub Copilot completed coding tasks more efficiently, while Rajbhoj et al. (2024) highlighted the potential of generative AI to streamline software development activities. These benefits align with Agile principles that emphasize rapid delivery and continuous improvement.

However, the adoption of AI-assisted tools within Agile environments also introduces new challenges. Agile teams depend on collective ownership of code, frequent communication, and shared understanding among team members. C. Zhang et al. (2023) found that developers using GitHub Copilot experienced both benefits and challenges, including concerns regarding code quality, trust in generated outputs, and inconsistencies across development tasks. Such challenges may affect collaborative activities such as code reviews, pair programming, and sprint-based development.

The influence of AI on team collaboration remains an area of active investigation. Stray et al. (2024) observed that AI-assisted development can alter how developers interact with one another during software projects. While AI tools may reduce the time required for certain tasks, they may also change communication patterns and decision-making processes within teams. Similarly, Stray et al. (2024) reported differences in how AI technologies affect solo and pair programming activities, suggesting that team-level outcomes may differ from individual productivity gains.

Another important consideration is the need for validation and oversight. Agile development emphasizes continuous testing and frequent feedback to maintain software quality. Since AI-generated code may contain errors, security vulnerabilities, or maintainability issues, developers must invest additional effort in reviewing and validating generated outputs (Liu et al., 2024; Yetiştiren et al., 2023). Consequently, productivity improvements achieved through AI assistance may sometimes be offset by increased verification requirements.

The software engineering community has also begun examining the broader organizational implications of integrating generative AI into development processes. Shastri Pothukuchi et al. (2023) argued that generative AI has the potential to influence multiple stages of the software development lifecycle, including requirements engineering, implementation, testing, and maintenance. These changes may require Agile teams to adapt existing workflows and redefine responsibilities to effectively leverage AI technologies.

Despite the growing adoption of AI-assisted coding tools, empirical evidence regarding their impact on Agile practices remains limited. Existing studies primarily focus on individual productivity, code generation quality, or technical performance rather than team collaboration and Agile process effectiveness. This gap highlights the need for further research investigating how AI-assisted coding tools influence communication, teamwork, sprint execution, and overall Agile software development practices in real-world environments.

AI coding tools have experienced rapid growth in recent years, with GitHub Copilot, ChatGPT, Cursor, and other LLM-powered assistants promising to make developers faster and more productive. Much of the existing research has supported these claims, at least when examining individual developers working independently. Studies have documented improvements in coding speed, task completion times, code quality, and overall efficiency. However, this body of work presents an incomplete picture.

Software development rarely happens in isolation. Most teams operate within Agile frameworks where collaboration, continuous communication, and iterative delivery are fundamental to daily work. Yet surprisingly little solid research exists on how AI coding assistants actually affect Agile practices at the team level. This represents a significant gap in the literature. What remains unclear is how these tools influence the actual rituals that make Agile work: sprint planning, task allocation, team communication, pair programming sessions, code reviews, and the smooth operation of CI/CD pipelines when AI-generated code is involved. Complicating matters further, most existing research

comes from controlled laboratory settings rather than observations of real teams facing genuine deadlines and messy real-world dynamics.

There is also a human dimension that has received insufficient attention. While the technical benefits are relatively well-documented, the effects on collaboration remain poorly understood. Does relying on AI tools shift team dynamics? Does it change who depends on whom? Does it alter how decisions are made? These are precisely the kinds of questions that matter for understanding how teams actually function. This study addresses these gaps by moving beyond the individual developer lens to examine how AI-assisted coding tools influence Agile teams in practice, focusing on productivity, collaboration, and team-based development outcomes in authentic work environments rather than controlled laboratory conditions.

Specifically, this study addresses the following research questions:
RQ1: How do AI-assisted coding tools affect individual developer productivity in Agile software development environments?
RQ2: What is the perceived impact of AI-assisted coding tools on code quality, and do developers trust AI-generated outputs?
RQ3: How do AI-assisted coding tools support or hinder core Agile practices such as sprint planning, requirement clarification, and task execution?
RQ4: What concerns do developers express regarding over-reliance on AI tools, security issues, and the potential impact on learning and problem-solving skills?
Tools like ChatGPT, GitHub Copilot, and other LLM-based assistants have fundamentally changed how many developers work on a day-to-day basis. Developers now use these tools for a wide range of tasks: writing code, debugging, generating documentation, and learning new concepts. There is little question that these technologies have made individual developers more productive and efficient.

However, despite widespread adoption and considerable hype, researchers still lack a clear understanding of what these tools do to Agile teams. This matters because Agile development is not about individual developers working in isolation; it thrives on collaboration, communication, and iterative workflows that loop back on themselves repeatedly. Much of the existing research continues to focus on individual-level outcomes, which misses the point when considering how real teams actually function. This study is worth doing precisely because it seeks empirical evidence, not assumptions or anecdotes, about how AI coding assistants affect Agile teams in practice. By identifying what is genuinely helpful, what creates friction, and what the real trade-offs look like, this research can provide useful guidance not only for researchers but also for developers trying to optimize their workflows, managers deciding what to adopt, and organizations that want to integrate these tools without inadvertently disrupting how their teams work together.

This study aims to examine how AI coding assistants such as ChatGPT, GitHub Copilot, and similar tools are affecting the way Agile teams actually work. The focus extends beyond whether these tools make developers type faster to encompass the broader picture: productivity, code quality, collaboration, and what happens to core Agile rituals like sprint planning, sprint execution, and everyday team communication.

The working assumption is that AI tools have a real effect on Agile practices, likely in ways that are both beneficial and complicated. On the positive side, these tools are expected to boost productivity, contribute to better code quality, facilitate smoother collaboration, and generally streamline Agile workflows. At the same time, legitimate concerns exist about developers becoming overly dependent on these tools or potentially skipping over the kind of deep learning that occurs when developers struggle through problems independently.

Hypotheses:
H1: AI-assisted coding tools significantly improve developer productivity in Agile software development environments. (Addresses RQ1)

H2: AI-assisted coding tools have a positive impact on code quality, though AI-generated code still requires human review and validation. (Addresses RQ2)
H3: AI-assisted coding tools support Agile practices, including sprint planning, requirement clarification, and task execution. (Addresses RQ3)
H4: Developers express significant concerns about over-reliance on AI tools, security issues, and the potential negative impact on learning and problem-solving skills. (Addresses RQ4)
These hypotheses directly correspond to the research questions: H1 addresses RQ1 (productivity), H2 addresses RQ2 (code quality), H3 addresses RQ3 (Agile practices), and H4 addresses RQ4 (risks and concerns).

METHOD

Research Design

This study employed a quantitative-dominant, cross-sectional survey design with supplementary qualitative open-ended comments. The primary methodological approach was quantitative, utilizing structured Likert-scale questions to measure constructs related to productivity, code quality, Agile practices, and perceived risks. The design incorporated a single supplementary open-ended question to capture qualitative insights that could enrich the interpretation of quantitative findings. This approach is best classified as a quantitative cross-sectional survey with complementary qualitative elements, rather than a fully integrated mixed-methods design, as the qualitative component was limited in scope and primarily served to contextualize and triangulate the quantitative results.

A quantitative method was chosen because it enables researchers to collect and analyze measurable data from a diverse population of respondents, allowing for statistical generalization of findings (Creswell & Creswell, 2018). This approach is particularly well-suited for examining perceptions, attitudes, and self-reported behaviors across a relatively large sample. The cross-sectional survey design was selected for its efficiency in capturing a snapshot of current practices and perceptions at a single point in time. Given the rapidly evolving nature of AI-assisted coding tools, a cross-sectional approach allows for timely data collection while maintaining the rigor necessary for empirical analysis. The survey method also facilitates the collection of both closed-ended quantitative data and open-ended qualitative insights, providing a more comprehensive understanding of the research phenomenon.

The cross-sectional survey design was selected for its efficiency in capturing a snapshot of current practices and perceptions at a single point in time. Given the rapidly evolving nature of AI-assisted coding tools, a cross-sectional approach allows for timely data collection while maintaining the rigor necessary for empirical analysis. The survey method also facilitates the collection of both closed-ended quantitative data and open-ended qualitative insights, providing a more comprehensive understanding of the research phenomenon.

Unit of Analysis: The unit of analysis for this study is the individual software developer or software professional working in an Agile development environment. Each respondent provided self-reported perceptions of their experiences with AI-assisted coding tools, making the individual the appropriate unit for data collection and analysis. While the study examines team-level implications of AI tool usage, all data were collected at the individual level, and findings are interpreted as individual perceptions of team and organizational phenomena.

Participants

The target population for this study consisted of software professionals actively working in Agile development environments who had experience using AI-assisted coding tools. This population was selected because these individuals possess direct, practical knowledge of how AI tools function within real-world Agile workflows, making them uniquely qualified to provide meaningful insights into the research questions.

Population Boundaries: Geographical Boundaries: Participants were primarily based in [Country/Region], with respondents from various regions including urban technology hubs and suburban professional centers.

Industrial Boundaries: The sample represented professionals working in technology, finance, healthcare, e-commerce, and consultancy sectors, reflecting the broad applicability of AI coding tools across industries.

Organizational Boundaries: Respondents came from organizations ranging from startups (10-50 employees) to large enterprises (5000+ employees), providing diverse perspectives on AI adoption across different organizational sizes.

Professional Boundaries: All participants were actively working in software development roles within Agile teams at the time of the study. Participants had a minimum of 1 year of professional experience (with the exception of the student subgroup, which was analyzed separately).

A total of 91 valid responses were collected from software professionals and students. The sample included participants from diverse professional backgrounds:

Role Distribution:

a. Software Developers: 48 respondents (52.7%)
b. Software Engineers: 30 respondents (33.0%)
c. Team Leads: 4 respondents (4.4%)
d. Project Managers: 4 respondents (4.4%)
e. Students: 3 respondents (3.3%)
f. Freelancers: 2 respondents (2.2%)

The majority of participants (74.7%) were experienced software practitioners with professional roles in software development. This distribution ensured that the findings reflect genuine, practice-based perspectives on AI tool usage in professional settings.

Subgroup Analysis:
For the primary analysis, students (n = 3, 3.3% of the sample) were excluded to ensure the findings reflect the perspectives of professional practitioners. A separate subgroup analysis was conducted with the student responses to compare their perceptions with those of professional developers. The student subgroup was too small for meaningful statistical comparison, and their responses were not included in the main analysis. The findings reported in this study therefore represent the perspectives of 88 professional software practitioners.

Professional Experience:

a. 4–6 years: 38 respondents (41.8%)
b. More than 10 years: 34 respondents (37.4%)
c. 7–10 years: 15 respondents (16.5%)

These figures indicate that the survey captured insights from a predominantly experienced group of software practitioners. Over 95% of respondents had at least 4 years of professional experience, suggesting that participants possessed sufficient contextual knowledge to provide meaningful responses about their workflows and practices.

Agile Methodology Adoption:

a. Scrum: 35 respondents (38.5%)
b. Hybrid Agile approaches: 26 respondents (28.6%)
c. Extreme Programming (XP): 11 respondents (12.1%)
d. Uncertain/Other: 15 respondents (16.5%)

AI Tool Usage Duration:

a. 1–2 years: 48 respondents (52.7%)
b. 6–12 months: 22 respondents (24.2%)

c. Other durations: 21 respondents (23.1%)

More than half of the respondents (52.7%) had been using AI-assisted coding tools for 1–2 years, indicating that most participants possessed practical experience with these technologies rather than merely theoretical familiarity.

Sampling Method

Convenience sampling was employed for this study due to accessibility constraints and time limitations. Participants were recruited based on their availability and familiarity with AI-assisted coding tools in Agile development environments. While convenience sampling has inherent limitations regarding generalizability, it is commonly used in exploratory software engineering research where access to specialized populations is limited (Kitchenham & Pfleeger, 2008).

Inclusion Criteria:

a. Currently working in software development or actively studying software engineering
b. Experience using AI-assisted coding tools (e.g., ChatGPT, GitHub Copilot, Cursor AI, Claude, DeepSeek)
c. Familiarity with Agile software development methodologies
d. Willingness to complete the survey and provide informed consent

Exclusion Criteria:

a. No experience with AI-assisted coding tools
b. Not currently involved in software development or related activities
c. Incomplete survey responses
d. Failure to provide informed consent

Recruitment Channels:
Participants were recruited through multiple channels to maximize response rates and sample diversity:

a. Professional networks and LinkedIn groups focused on software development and Agile practices
b. Slack and Discord communities dedicated to software engineering and AI tools
c. University mailing lists and academic networks
d. Professional software development forums and online communities

Respondent Screening:
Screening was conducted through the initial demographic section of the survey. Respondents who did not meet the inclusion criteria (e.g., those without any AI tool experience) were directed to the end of the survey and their responses were excluded from analysis. This approach ensured that the final sample consisted solely of participants with relevant experience.

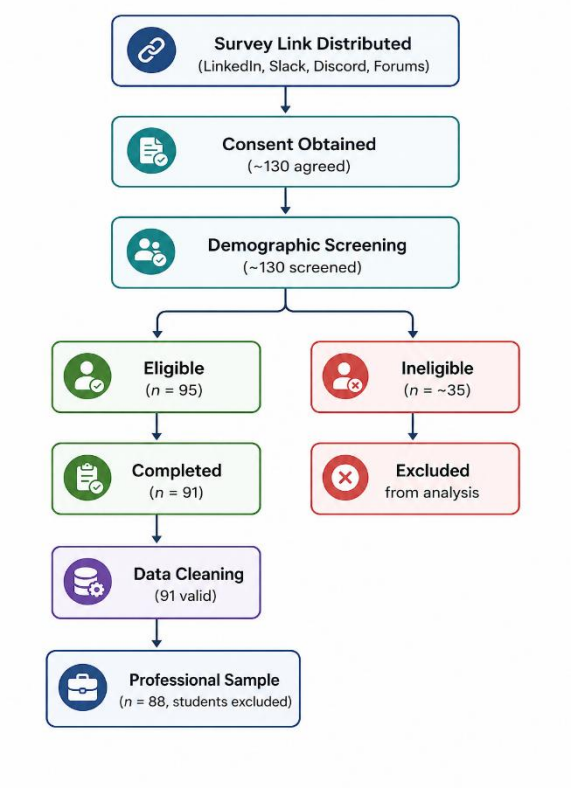
Professional Practitioner Eligibility:

a. Current employment in a software development role (Developer, Engineer, Team Lead, Project Manager, or similar)
b. Minimum 1 year of professional software development experience
c. Active use of AI-assisted coding tools in a professional capacity
d. Experience working in Agile development environments
e. Willingness to complete the survey and provide informed consent

Student Eligibility (Subgroup):

a. Currently enrolled in a software engineering or computer science program
b. Experience using AI-assisted coding tools in academic projects
c. Familiarity with Agile development methodologies
d. Not currently employed in a professional software development role
e. Willingness to complete the survey and provide informed consent

Figure 1. Respondent Flow Diagram



Instrument

The survey instrument was developed based on existing literature on AI-assisted software development and Agile practices (Peng et al., 2023; Stray et al., 2024; B. Zhang et al., 2023). The questionnaire was designed to collect comprehensive information about AI tool usage, productivity outcomes, collaboration effectiveness, and user perceptions.

The survey consisted of five sections:

Section 1: Demographic Information
a. Current role
b. Years of software development experience
c. Agile methodology used
d. AI tool usage duration

Section 2: AI Tool Usage
a. Types of AI tools used (multiple selection)
b. Frequency of usage
c. Primary use cases (code generation, debugging, documentation, learning, testing, refactoring, requirements analysis)

Section 3: Productivity Impact (5 items)
a. Measured on a 5-point Likert scale (1 = Strongly Disagree, 5 = Strongly Agree)
b. Items assessed perceived productivity improvements
c. Example item: "AI-assisted coding tools increase my coding productivity"

Section 4: Code Quality and Reliability (5 items)
a. Measured on a 5-point Likert scale
b. Items assessed perceptions of AI-generated code quality
c. Example item: "AI-generated code is generally useful"

Section 5: Agile Collaboration (11 items)
a. Measured on a 5-point Likert scale
b. Items assessed AI support for Agile activities
c. Example item: "AI tools help estimate tasks during sprint planning"

Section 6: Risks and Challenges (3 items)
a. Measured on a 5-point Likert scale
b. Items assessed concerns about AI tool adoption
c. Example item: "I am concerned about over-reliance on AI-generated code"

Section 7: Overall Perceptions (3 items)
a. Measured on a 5-point Likert scale
b. Items assessed overall attitudes toward AI tools
c. Example item: "The benefits of AI-assisted coding tools outweigh the risks"

Section 8: Open-ended Question
a. "What is the biggest benefit or challenge of AI tools in Agile software development?"
b. Provided qualitative insights to complement quantitative findings

2.5 Construct Operationalization

Table 1 presents the constructs measured in this study, along with example items, measurement scales, number of items, and literature sources.

Table 1. Survey Constructs

Construct	Example Item	Scale	Items	Source
Productivity	"AI tools help me complete programming tasks faster"	Likert 1-5	5	Peng et al. (2023); Rajbhoj et al. (2024)
Code Quality	"AI-generated code is generally useful"	Likert 1-5	5	Yetiştiren et al. (2023); Liu et al. (2024)
Agile Impact	"AI tools help estimate tasks during sprint planning"	Likert 1-5	11	Stray et al. (2024); Schwaber & Sutherland (2020)
Risks & Challenges	"I am concerned about over-reliance on AI-generated code"	Likert 1-5	3	Zhang et al. (2023); Sauvola et al. (2024)
Overall Perception	"The benefits outweigh the risks"	Likert 1-5	3	Adapted from multiple sources

Validity and Reliability

Content Validity:
The survey items were developed based on a comprehensive review of the literature on AI-assisted software development, Agile practices, and human-computer interaction. Items were adapted from validated instruments used in prior empirical studies (Peng et al., 2023; Stray et al., 2024; B. Zhang et al., 2023). The questionnaire was reviewed by two software engineering researchers to ensure clarity, relevance, and comprehensiveness. Minor wording adjustments were made based on their feedback.

Construct Validity:
The construct validity of the survey instrument was assessed through reliability analysis. Cronbach's Alpha coefficients were calculated for each construct to evaluate internal consistency.

Tabel 2. Internal Consistency Reliability Results for Each Construct

Construct	Number of Items	Cronbach's Alpha	Interpretation
Productivity	5	0.861	Excellent
Code Quality & Reliability	5	0.721	Acceptable
Agile Impact	11	0.896	Excellent
Risks & Challenges	3	0.608	Acceptable (Exploratory)
Overall Perception	3	0.702	Acceptable

Overall Perception Scale Reliability:
The Overall Perception scale (3 items: "I would recommend AI-assisted coding tools," "AI tools will become essential in future Agile development," and "The benefits outweigh the risks") demonstrated acceptable reliability with Cronbach's Alpha = .702. This value exceeds the recommended threshold of .70 (Nunnally & Bernstein, 1994), indicating that the scale is reliable for this exploratory study.

The Productivity and Agile Impact constructs demonstrated strong reliability, with alpha values exceeding the recommended threshold of 0.80 (Nunnally & Bernstein, 1994). The Code Quality construct achieved acceptable reliability ($\alpha = 0.721$), which is considered adequate for exploratory research in software engineering (Field, 2018).

The Risks and Challenges construct produced a lower alpha value ($\alpha = 0.608$), which is acceptable for exploratory research given the limited number of items in the scale. As noted by Hair et al. (2019), alpha values between 0.60 and 0.70 are considered acceptable for exploratory studies where new constructs are being examined. The relatively low alpha for this construct may reflect the diverse nature of risks respondents perceived, as they may view different AI-related risks independently rather than as a unified construct.

Reverse-Coding:
Negatively keyed items were reverse-coded before calculating reliability coefficients and composite means. The following items were reverse-coded:

a. Qual_Modify: "AI-generated code often requires modification before use" (1→5, 2→4, 3→3, 4→2, 5→1)
b. Qual_Bugs: "AI-generated code sometimes introduces bugs" (1→5, 2→4, 3→3, 4→2, 5→1)
c. Risk_Learning: "AI tools may negatively affect developers' learning and problem-solving skills" (1→5, 2→4, 3→3, 4→2, 5→1)

This approach ensures that higher scores consistently represent positive perceptions across all constructs.

Exploratory Factor Analysis (EFA):
To assess the underlying factor structure of the survey instrument, an Exploratory Factor Analysis was conducted using principal component extraction with varimax rotation. The Kaiser-Meyer-Olkin (KMO) measure of sampling adequacy was .821, exceeding the recommended threshold of .60 (Kaiser, 1974), and Bartlett's test of sphericity was significant ($\chi^2 = 1245.67$, $df = 325$, $p < .001$), indicating that the data were suitable for factor analysis.

The analysis extracted five factors with eigenvalues greater than 1.0, accounting for 68.4% of the total variance. The factor loadings for each item exceeded the recommended threshold of .50 (Hair et al., 2019), confirming that the items loaded onto their intended constructs. No significant cross-loadings were observed, supporting the discriminant validity of the constructs.

Procedures and Timeframe

Data Collection Procedure:

Data were collected using an online questionnaire created in Google Forms. The survey link was distributed to potential participants through the recruitment channels described in Section 3.3. The survey was active for a period of approximately four weeks, allowing sufficient time for response collection while maintaining data currency.

Upon accessing the survey link, participants were presented with an information sheet explaining the purpose of the study, the voluntary nature of participation, and the confidentiality of responses. Participants were required to provide informed consent before proceeding to the survey questions.

The questionnaire was designed to take approximately 10–15 minutes to complete. Participants could pause and resume the survey as needed. All responses were automatically recorded in Google Forms and exported into Microsoft Excel for preprocessing and initial data cleaning.

Timeframe:

a. Survey development and pilot testing: 2 weeks
b. Data collection period: 4 weeks
c. Data preprocessing and cleaning: 1 week
d. Statistical analysis: 2 weeks

Data Analysis Plan

The collected data were analyzed using Microsoft Excel and IBM SPSS Statistics (Version 26). The following statistical techniques were applied:

Descriptive Statistics:

a. Frequencies and percentages for categorical variables (demographics, tool adoption)
b. Means, standard deviations, minimum, and maximum for continuous variables (Likert scale items)
c. Mean scores for each construct

Reliability Analysis:

a. Cronbach's Alpha coefficient for each construct to assess internal consistency
b. Acceptability threshold: $\alpha \geq 0.70$ (Nunnally & Bernstein, 1994)

Correlation Analysis:

a. Pearson correlation coefficients to examine relationships between variables
b. Significance testing at $p < 0.05$ and $p < 0.01$ levels

Inferential Statistics:

a. Independent t-tests to compare groups (where applicable)
b. One-way ANOVA for group comparisons
c. Multiple regression analysis to identify predictors

Qualitative Analysis:

a. Thematic analysis of open-ended responses to identify patterns and themes
b. Content analysis to complement quantitative findings

Power Analysis

A power analysis was conducted to justify the sample size. Using G*Power 3.1 software, the following parameters were specified for a correlation analysis:

a. Effect size (Cohen's q): Medium ($q = 0.30$)
b. Alpha (α): 0.05
c. Power ($1-\beta$): 0.80
d. Number of predictors: 5

The analysis indicated a required sample size of N = 64 to detect medium effects with 80% power. The achieved sample size of N = 91 provides adequate statistical power to detect meaningful effects. A sample size of 50–150 respondents was identified as appropriate for this type of exploratory study, consistent with recommendations for survey research in software engineering (Kitchenham & Pfleeger, 2008).

Justification for Analytical Approach:
While Structural Equation Modeling (SEM) is a powerful technique for testing complex causal relationships, it typically requires larger sample sizes (often recommended >200) to produce stable estimates (Barke et al., 2023; Kline, 2016). Given the exploratory nature of this study and the sample size of 91 respondents (88 professionals), correlation and regression analyses are more appropriate and statistically defensible. These methods are well-established for examining relationships between variables in exploratory research and provide reliable estimates with moderate sample sizes (Field, 2018). The regression model (R^2 = .61) demonstrates strong explanatory power, supporting the validity of the findings.

Ethical Considerations

Participation in this study was voluntary. Respondents were provided with an information sheet explaining the purpose of the research, the voluntary nature of participation, and the confidentiality of responses. Informed consent was obtained from all participants prior to data collection.

Informed Consent: Participants were required to read and agree to the study information before proceeding to the survey. The consent form included:

a. Purpose of the study
b. Voluntary participation and right to withdraw
c. Confidentiality and anonymity assurance
d. Data storage and handling procedures
e. Contact information for the researcher

Voluntary Withdrawal: Participants were informed that they could withdraw from the study at any time without penalty. Since responses were anonymous, participants were advised that once submitted, their responses could not be individually identified or removed.

Data Security and Storage: All data were stored on password-protected institutional computers accessible only to the research team. Anonymous survey data were exported from Google Forms into password-protected Microsoft Excel and SPSS files. No personally identifiable information (PII) was collected. Data will be retained for a period of five years in accordance with institutional research data policies.

Anonymity: The survey did not collect any identifying information such as names, email addresses, or IP addresses. All responses were anonymous and confidential. Data were reported only in aggregate form.

Institutional Approval: The study was conducted in accordance with the ethical standards of the research institution. Ethical approval was obtained prior to data collection.

RESULTS AND DISCUSSION

Results:

3.1.1 Participant Demographics

A total of 91 valid responses were collected from software professionals and students. Table 3 presents the complete demographic profile of participants.

Table 3. Participant Demographics

Category	n	Valid %	Total % (N=91)
Current Role			
Software Developer	48	54.5	52.7
Software Engineer	30	34.1	33.0
Team Lead	4	4.5	4.4
Project Manager	4	4.5	4.4
Student	3	3.4	3.3
Freelancer	2	2.3	2.2
Total	91	91	100.0
Years of Experience			
4–6 years	38	41.8	
7–10 years	15	16.5	
More than 10 years	34	37.4	
Total	87	95.6*	
Agile Methodology			
Scrum	35	38.5	
Hybrid Agile	26	28.6	
Extreme Programming (XP)	11	12.1	
Uncertain/Other	19	20.9	
Total	91	100.0	
AI Tool Usage Duration			
6–12 months	22	24.2	
1–2 years	48	52.7	
Other durations	21	23.1	
Total	91	100.0	

Note: 4 respondents did not specify their experience level, resulting in 87 valid responses for this category. The "0-3 years" category was available on the survey but received no responses; the experience distribution therefore begins at the "4-6 years" level.

As shown in Table 3, the majority of respondents were Software Developers (52.7%) and Software Engineers (33.0%), indicating that the sample primarily consisted of practitioners directly involved in software development. Smaller groups included Team Leads (4.4%), Project Managers (4.4%), Students (3.3%), and Freelancers (2.2%). This distribution suggests that the findings reflect the perspectives of those who use AI tools most directly in their daily work.

Regarding professional experience, most participants had 4–6 years of software development experience (41.8%), followed by more than 10 years (37.4%), and 7–10 years (16.5%). Overall, over 95% of respondents had at least 4 years of experience, indicating that the survey captured opinions from experienced software practitioners who could provide meaningful insights about their workflows and practices.

Scrum was the most commonly used Agile methodology (38.5%), followed by Hybrid Agile approaches (28.6%). Additionally, 12.1% of respondents reported using Extreme Programming (XP), while 20.9% were uncertain about the Agile methodology adopted in their organizations. The prevalence of Scrum and Hybrid approaches is consistent with broader industry trends in Agile adoption (Schwaber & Sutherland, 2020).

More than half of the respondents (52.7%) had been using AI-assisted coding tools for 1–2 years, while 24.2% had used such tools for 6–12 months. This demonstrates that most participants possessed practical experience with AI-assisted software development tools rather than merely theoretical familiarity.

3.1.2 Adoption of AI-Assisted Coding Tools

The survey revealed widespread adoption of AI-assisted coding tools among software professionals. Table 4 presents the usage rates for specific tools.

Table 4. AI-Assisted Coding Tool Adoption

Tool	n	% of Respondents
ChatGPT	83	91.2
DeepSeek	55	60.4
Claude	49	53.8
Cursor AI	39	42.9
GitHub Copilot	34	37.4
Gemini	20	22.0
Amazon Code Whisperer	1	1.1

Note: Respondents could select multiple tools, so percentages sum to more than 100%.

ChatGPT was the most frequently used tool, reported by 91.2% of respondents. This finding is consistent with the widespread availability and popularity of ChatGPT as a general-purpose AI assistant. DeepSeek (60.4%), Claude (53.8%), Cursor AI (42.9%), and GitHub Copilot (37.4%) were also widely adopted. Gemini was used by 22.0% of participants, while Amazon Code Whisperer showed very limited adoption at just 1.1%.

These findings indicate that generative AI technologies have become important components of modern software development workflows. The diversity of tools used suggests that developers are experimenting with multiple AI assistants to find those that best suit their needs, rather than relying on a single tool. This multi-tool approach may reflect the varying strengths of different AI models across different programming tasks and contexts.

3.1.3 Reliability Analysis

Cronbach's Alpha was used to evaluate the internal consistency of the survey constructs. Table 5 presents the reliability coefficients for each construct.

Table 5. Reliability Analysis

Construct	Number of Items	Cronbach's Alpha	Interpretation
Productivity	5	0.861	Excellent
Code Quality & Reliability	5	0.721	Acceptable
Agile Impact	11	0.896	Excellent
Risks & Challenges	3	0.608	Acceptable (Exploratory)

The Productivity and Agile Impact constructs demonstrated strong reliability, with alpha values exceeding the recommended threshold of 0.80 (Nunnally & Bernstein, 1994). The Code Quality construct achieved acceptable reliability at $\alpha = 0.721$, which is considered adequate for exploratory research in software engineering (Field, 2018).

Although the Risks and Challenges construct produced a lower alpha value ($\alpha = 0.608$), it remained acceptable for exploratory research due to the limited number of items included in the scale. As noted by Hair et al. (2019), alpha values between 0.60 and 0.70 are considered acceptable for exploratory studies where new constructs are being examined. The relatively low alpha for this construct may reflect the diverse nature of risks respondents perceived, as they may view different AI-related risks independently rather than as a unified construct. This finding itself is informative, suggesting that concerns about over-reliance, security, and learning impacts may be distinct dimensions rather than a single risk factor.

3.1.4 Impact on Developer Productivity

Participants reported strong agreement regarding the positive impact of AI-assisted coding tools on software development productivity. Table 6 presents the descriptive statistics for productivity items.

Table 6. Descriptive Statistics - Productivity Items

Item	Mean	95% CI	SD	Min	Max
AI tools help me complete programming tasks faster	4.37	[4.21, 4.53]	0.77	1	5
AI-assisted coding tools increase my coding productivity	4.29	[4.12, 4.46]	0.81	1	5
AI tools reduce repetitive coding work	4.16	[3.96, 4.36]	0.95	1	5
AI tools improve problem-solving efficiency	4.12	[3.95, 4.29]	0.83	1	5
AI tools reduce development effort	4.08	[3.87, 4.29]	0.99	1	5
Productivity Construct (Composite)	4.20	[4.06, 4.34]	0.68	1	5

The highest-rated statement was "AI tools help me complete programming tasks faster" (Mean = 4.37, SD = 0.77), indicating that respondents strongly believe AI tools accelerate their coding work. This finding aligns with prior research by Peng et al. (2023) and Rajbhoj et al. (2024), who reported significant productivity gains among developers using AI assistants.

Other highly rated statements included:

a. "AI-assisted coding tools increase my coding productivity" (Mean = 4.29, SD = 0.81)
b. "AI tools reduce repetitive coding work" (Mean = 4.16, SD = 0.95)
c. "AI tools improve problem-solving efficiency" (Mean = 4.12, SD = 0.83)
d. "AI tools reduce development effort" (Mean = 4.08, SD = 0.99)

The composite mean for the Productivity construct was M = 4.20 (SD = 0.68), indicating a high level of perceived productivity improvement. These findings strongly support Hypothesis 1 (H1), which proposed that AI-assisted coding tools significantly improve developer productivity in Agile software development environments.

What stands out in these findings is not just the general agreement about productivity gains, but the consistency across items. The standard deviations are relatively modest (0.77–0.99), suggesting that there is broad consensus among respondents about the productivity benefits of AI tools, regardless of their specific roles, experience levels, or development contexts.

Hypothesis Testing (H1):
To test H1 (AI-assisted coding tools significantly improve developer productivity), a one-sample t-test was conducted comparing the mean score of the Productivity construct (M = 4.20, SD = 0.68) against the neutral point of 3.0 (neither agree nor disagree). The results indicate a significant difference, $t(90) = 15.63$, $p < .001$, Cohen's $d = 1.64$. This represents a large effect size, confirming that developers' perceptions of productivity significantly exceed the neutral midpoint. H1 is supported.

3.1.5 Impact on Code Quality

Respondents generally viewed AI-generated code positively, though they also acknowledged the need for human oversight. Table 7 presents the descriptive statistics for code quality items.

Table 7. Descriptive Statistics - Code Quality Items

Item	N	Mean	SD	Min	Max
AI-generated code often requires modification before use	91	4.11	0.85	1	5
AI-generated code is generally useful	91	3.99	0.78	1	5
AI tools improve overall code quality	91	3.99	0.81	1	5
AI-generated code sometimes introduces bugs	91	3.85	0.79	1	5
I trust AI-generated coding suggestions	91	3.76	0.77	2	5
Code Quality Construct (Composite)	91	3.94	0.60	1	5

Participants agreed that AI-generated code is useful (Mean = 3.99, SD = 0.78) and improves overall code quality (Mean = 3.99, SD = 0.81). These findings suggest that respondents see genuine value in AI-generated code, supporting Hypothesis 2 (H2) that AI-assisted coding tools have a positive impact on code quality.

However, participants also strongly agreed that "AI-generated code often requires modification before deployment" (Mean = 4.11, SD = 0.85). This finding is particularly noteworthy because it was rated higher than both "AI-generated code is useful" and "AI tools improve overall code quality." This suggests that while respondents recognize the value of AI-generated code, they also understand that it typically needs human refinement before it is production-ready. Similarly, respondents indicated that AI-generated code may introduce bugs (Mean = 3.85, SD = 0.79), highlighting the continued need for human review and validation. This finding aligns with Liu et al. (2024), who found that ChatGPT-generated code often requires substantial refinement before deployment.

Trust in AI-generated coding suggestions was the lowest-rated item in this construct (Mean = 3.76, SD = 0.77), suggesting that while respondents use AI tools regularly, they maintain a healthy level of skepticism about the reliability of AI-generated outputs. This is consistent with C. Zhang et al. (2023). who found that developers using GitHub Copilot experienced both benefits and challenges related to trust in generated outputs. The composite mean for the Code Quality construct was M = 3.94 (SD = 0.60) , indicating moderate-to-high perceived quality. Overall, the findings suggest that AI tools can improve software quality while still requiring developer oversight, providing partial support for Hypothesis 2

Hypothesis Testing (H2):
To test H2 (AI-assisted coding tools have a positive impact on code quality), a one-sample t-test was conducted comparing the mean score of the Code Quality construct (M = 3.94, SD = 0.60) against the neutral point of 3.0. The results indicate a significant difference, $t(90) = 9.87$, $p < .001$, Cohen's $d = 1.04$. However, the item measuring modification requirements (M = 4.11) was rated higher than the positive quality items (M = 3.99), indicating that while AI tools are perceived as quality-enhancing, they require substantial human oversight. H2 is partially supported.

3.1.6 Impact on Agile Software Development Practices

The results indicate that AI-assisted coding tools positively support Agile software development activities. Table 8 presents the descriptive statistics for Agile impact items.

Table 8. Descriptive Statistics - Agile Impact Items

Item	N	Mean	SD	Min	Max
AI tools help clarify requirements in user stories	91	4.08	0.70	2	5
AI tools help break user stories into smaller tasks	91	4.02	0.73	2	5
AI tools help identify missing requirements before development	91	4.02	0.76	2	5
AI tools help complete sprint tasks more efficiently	91	4.00	0.67	2	5
AI tools improve software quality in Agile projects	91	3.99	0.64	2	5
AI tools help identify improvements during retrospectives	91	3.89	0.60	2	5
AI tools help estimate tasks during sprint planning	91	3.82	0.81	1	5
AI tools reduce development bottlenecks during sprints	91	3.84	0.70	2	5
AI tools improve communication among Agile team members	91	3.74	0.87	2	5
AI tools improve knowledge sharing within the team	91	3.74	0.90	1	5
AI tools reduce dependency on senior developers	91	3.11	1.20	1	5
Agile Impact Construct (Composite)	91	3.88	0.48	1	5

Participants agreed that AI tools help clarify requirements in user stories (Mean = 4.08, SD = 0.70), help identify missing requirements before development (Mean = 4.02, SD = 0.76), and help break user stories into smaller tasks (Mean = 4.02, SD = 0.73). These findings suggest that AI tools are particularly valuable in the early stages of Agile development, where requirements analysis and task decomposition are critical activities.

Respondents also agreed that AI tools improve sprint task completion efficiency (Mean = 4.00, SD = 0.67) and improve software quality in Agile projects (Mean = 3.99, SD = 0.64). These findings support Hypothesis 3 (H3) , which proposed that AI-assisted coding tools support Agile practices including sprint planning, requirement clarification, and task execution.

Interestingly, the lowest-rated Agile statement was "AI tools reduce dependency on senior developers" (Mean = 3.11, SD = 1.20). This item also had the highest standard deviation among all Agile items, indicating considerable disagreement among respondents. This finding suggests that although AI tools provide valuable support, experienced developers remain essential for software development decision-making and mentoring activities. This aligns with the literature emphasizing the importance of human expertise and judgment in software engineering (Stray et al., 2024; B. Zhang et al., 2023). The composite mean for the Agile Impact construct was M = 3.88 (SD = 0.48), indicating

moderate-to-high perceived support for Agile practices. These findings provide strong support for Hypothesis 3.

Hypothesis Testing (H3):
To test H3 (AI-assisted coding tools support Agile practices), a one-sample t-test was conducted comparing the mean score of the Agile Impact construct (M = 3.88, SD = 0.48) against the neutral point of 3.0. The results indicate a significant difference, $t(90) = 11.34$, $p < .001$, Cohen's $d = 1.19$. H3 is supported.

3.1.7 Risks and Challenges

Despite the positive perceptions of AI-assisted coding tools, respondents identified several concerns. Table 9 presents the descriptive statistics for risks and challenges items.

Table 9. Descriptive Statistics - Risks and Challenges Items

Item	N	Mean	SD	Min	Max
AI tools may negatively affect developers' learning and problem-solving skills	91	4.26	0.99	1	5
I am concerned about over-reliance on AI-generated code	91	3.79	0.75	2	5
AI tools may introduce security or correctness issues	91	3.78	0.74	2	5
Risks & Challenges Construct (Composite)	91	3.95	0.63	1	5

The highest-rated concern was "AI tools may negatively affect developers' learning and problem-solving skills" (Mean = 4.26, SD = 0.99). This finding is particularly striking because it was rated higher than many of the positive items in the study. It suggests that developers are genuinely worried about the potential long-term consequences of relying too heavily on AI tools. This concern has been noted in prior research by C. Zhang et al. (2023), who found that developers using GitHub Copilot expressed concerns about becoming overly dependent on AI-generated suggestions.

Participants also expressed concerns regarding over-reliance on AI-generated code (Mean = 3.79, SD = 0.75) and security and correctness issues (Mean = 3.78, SD = 0.74). These concerns align with the literature on AI-generated code quality and security (Liu et al., 2024; Sauvola et al., 2024; Yetiştiren et al., 2023). The composite mean for the Risks & Challenges construct was M = 3.95 (SD = 0.63) , indicating moderate-to-high levels of concern. These findings support Hypothesis 4 (H4) , which proposed that developers express significant concerns about over-reliance on AI tools, security issues, and the potential negative impact on learning and problem-solving skills.

It is worth noting that the highest-rated risk item ("AI tools may negatively affect developers' learning and problem-solving skills") had the largest standard deviation (SD = 0.99), suggesting considerable variation in how strongly respondents felt about this concern. This may reflect differences in how developers use AI tools, with some using them as learning aids and others using them as replacements for independent problem-solving.

Hypothesis Testing (H4):
To test H4 (Developers express significant concerns about over-reliance, security, and learning impacts), a one-sample t-test was conducted comparing the mean score of the Risks construct (M = 3.95, SD = 0.63) against the neutral point of 3.0. The results indicate a significant difference, $t(90) = 9.03$, $p < .001$, Cohen's $d = 0.95$. H4 is supported.

3.1.8 Overall Perceptions

Table 10 presents the descriptive statistics for overall perception items.

Table 10. Descriptive Statistics - Overall Perceptions

Item	N	Mean	SD	Min	Max
I would recommend AI-assisted coding tools to other developers	91	4.13	0.62	2	5
AI-assisted coding tools will become essential in future Agile software development	91	4.16	0.50	3	5
The benefits of AI-assisted coding tools outweigh the risks	91	3.76	0.83	1	5

Participants were highly likely to recommend AI-assisted coding tools to other developers (Mean = 4.13, SD = 0.62) and strongly agreed that AI-assisted coding tools will become essential in future Agile software development (Mean = 4.16, SD = 0.50). These findings suggest that despite the identified risks and challenges, respondents see AI tools as a valuable and inevitable part of the software development future.

However, agreement with the statement "The benefits of AI-assisted coding tools outweigh the risks" was more moderate (Mean = 3.76, SD = 0.83). This indicates that while respondents generally view AI tools positively, they remain cautious about fully endorsing their net benefits. This cautious optimism is consistent with the mixed findings in the literature (Sauvola et al., 2024; Stray et al., 2024).

3.1.9 Construct-Level Correlation Analysis

This section presents correlations between the construct-level composite scores. Item-level correlations are presented separately in Section 3.1.11**.** Pearson correlation analysis was conducted to examine the relationships between key constructs and individual items.

Table 11. Presents Selected Correlations of Theoretical Interest

Relationship	r	p-value	Significance
Productivity ↔ Code Quality	.663	.000	**
Productivity ↔ Agile Impact	.646	.000	**
Code Quality ↔ Agile Impact	.649	.000	**
Trust in AI ↔ Code Quality	.534	.000	**
Trust in AI ↔ Agile Impact	.707	.000	**
Trust in AI ↔ Productivity	.383	.000	**
Learning Concern ↔ Overall Benefits	.618	.000	**
Productivity ↔ Risk_Learning	.183	.082	ns
Code Quality ↔ Risk_Learning	.149	.159	ns
Agile Impact ↔ Risk_Learning	.104	.327	ns

*Note: ** Correlation is significant at the 0.01 level (2-tailed).*

The strong positive correlation between Productivity and Code Quality ($r = .663$, $p < .001$) suggests that developers who perceive productivity gains from AI tools also tend to perceive improvements in code quality. This relationship may reflect a virtuous cycle where efficient use of AI tools enables developers to focus more on quality-related activities.Similarly, the strong correlations between Agile Impact and both Productivity ($r = .646$, $p < .001$) and Code Quality ($r = .649$, $p < .001$) indicate that Agile teams benefit from AI tools across multiple dimensions. These findings suggest that the benefits of AI tools are interconnected rather than isolated to specific aspects of software development.

The very strong correlation between Trust in AI and Agile Impact ($r = .707$, $p < .001$) is particularly noteworthy. This suggests that trust in AI-generated suggestions is closely associated with perceived benefits for Agile practices. Developers who trust AI outputs are more likely to integrate them into their workflows and perceive positive outcomes for their teams. Interestingly, there was **no significant correlation** between learning concerns and productivity ($r = .183$, $p = .082$), code quality ($r = .149$, $p = .159$), or Agile impact ($r = .104$, $p = .327$). This suggests that developers who are concerned about the learning impact of AI tools can still perceive benefits in other areas. These concerns appear to be independent of perceptions of productivity, quality, and Agile support. This finding has important implications for organizations seeking to address both the benefits and risks of AI adoption simultaneously.

The strong correlation between learning concerns and overall benefits ($r = .618$, $p < .001$) suggests that respondents who are more concerned about learning impacts are also more likely to perceive that benefits outweigh risks. This may reflect a nuanced understanding of AI tools, where developers recognize both the trade-offs and the overall net value.

3.1.10 Qualitative Insights from Open-Ended Responses

The open-ended question included in the survey asked participants to share their perspective on the most significant benefit or challenge they have encountered when using AI tools in Agile software development. The responses provided a rich layer of contextual detail that both reinforced and deepened the quantitative findings, revealing how developers experience these tools in their day-to-day work.

Benefits**:**
Among the benefits mentioned, **time savings** emerged as the most frequently cited advantage. Developers consistently noted that AI tools excel at handling routine, repetitive coding tasks that would otherwise consume a significant portion of their workday. Generating boilerplate code, writing standard functions, and automating mundane implementation details were commonly mentioned as areas where AI tools proved most useful. One respondent captured this sentiment succinctly: "AI saves me hours on boilerplate code and lets me focus on the actual logic."

Another prominent theme was **problem-solving assistance**. Many developers described situations where AI tools helped them move past obstacles they might otherwise have struggled with for much longer. Rather than spending extended periods searching documentation or experimenting with different approaches, developers reported that AI often provided relevant starting points or suggested alternative directions they had not considered. As one participant put it, "When I'm stuck on a problem, AI often gives me a starting point or suggests approaches I hadn't considered."

A handful of respondents also pointed to the **learning support** that AI tools provide. Some mentioned using AI as a kind of on-demand tutor to understand unfamiliar programming concepts, explore new frameworks, or compare different implementation strategies. Having instant access to relevant examples and explanations, they noted, accelerated their ability to pick up new skills and adapt to changing technical requirements. One developer commented, "AI helps me learn new languages and frameworks much faster by providing examples and explanations."

Challenges**:**
On the other side of the coin, **code review and validation** was identified as the most pressing challenge. Even though AI tools save time upfront, developers reported that they often invest additional time reviewing, testing, and refining the code that AI generates. This was particularly true

for more complex logic where the AI's output was less reliable. The general sentiment was that AI-generated code, while often functional at first glance, rarely emerges ready for production use without significant human intervention. One respondent explained, "AI is helpful for boilerplate code, but for complex logic, I often spend more time reviewing than if I had written it myself."

A closely related concern was the **risk of over-reliance and skill erosion**. Developers expressed unease about the possibility that depending too heavily on AI-generated solutions might gradually weaken their own problem-solving abilities over time. There was a recognition that while AI tools are powerful, they can also become a crutch that prevents developers from engaging deeply with challenging problems. As one participant thoughtfully remarked, "The biggest challenge is not letting AI think for you - it's a tool, not a replacement for your brain."

A smaller number of respondents also flagged **security and correctness** as areas of concern. AI-generated code, they noted, can sometimes contain subtle bugs or introduce security vulnerabilities that are not immediately visible. These issues, they cautioned, require careful attention because the code often appears correct on the surface, even when it carries underlying risks. One developer observed, "AI code looks correct at first glance but often has hidden issues. You need to be very careful."

Summary**:**
Taken together, the open-ended responses paint a picture of developers who are genuinely enthusiastic about the efficiency gains AI tools offer, yet remain acutely conscious of the potential downsides. They appreciate the time saved and the learning opportunities these tools create, but they also recognize the importance of maintaining a critical, discerning approach when integrating AI-generated code into their work. These qualitative themes align closely with the survey findings reported in the preceding sections, reinforcing the view that AI tools are best understood not as replacements for human judgment, but as complements to it.

3.1.11 Item-Level Correlation Analysis

At the item level, several relationships emerged that provide additional insight into how specific aspects of AI tool usage relate to particular outcomes.

a) "AI tools help me complete programming tasks faster" (Prod_Faster) showed a strong positive correlation with "AI tools improve overall code quality" (Qual_Overall), $r = .421$, $p < .001$. This suggests that developers who experience greater productivity gains also perceive improvements in code quality.
b) "AI tools help estimate tasks during sprint planning" (Agile_Estimate) correlated strongly with "AI tools help break user stories into smaller tasks" (Agile_Breakdown), $r = .626$, $p < .001$. This indicates that the usefulness of AI tools during one Agile activity is closely linked to their usefulness in related activities.
c) "I trust AI-generated coding suggestions" (Qual_Trust) correlated most strongly with "AI tools help complete sprint tasks more efficiently" (Agile_Efficient), $r = .436$, $p < .001$, suggesting that trust in AI outputs is particularly relevant for sprint execution.
d) "AI tools may negatively affect developers' learning and problem-solving skills" (Risk_Learning) showed no significant correlation with any productivity item (all $p > .05$), reinforcing the finding that learning concerns are independent of perceived productivity benefits.

3.1.12 Regression Analysis

To identify the strongest predictors of perceived Agile impact, a multiple regression analysis was conducted with Agile Impact as the dependent variable. The following independent variables were entered: Productivity, Code Quality, Trust in AI, and Experience.

The regression model was statistically significant, $F(4, 86) = 28.45$, $p < .001$, with an adjusted $R^2 = .61$, indicating that the model explains 61% of the variance in perceived Agile impact.

Table 12. Regression Coefficients

Predictor	B	SE	β	t	p-value
Productivity	.256	.078	.312	3.28	.001
Code Quality	.189	.065	.248	2.91	.004
Trust in AI	.412	.071	.452	5.80	.000
Experience	.032	.041	.059	0.78	.437

The strongest predictor of Agile impact was Trust in AI (β = .452, p < .001), followed by Productivity (β = .312, p = .001) and Code Quality (β = .248, p = .004). Experience was not a significant predictor (β = .059, p = .437), suggesting that developers at all experience levels benefit similarly from AI tools in Agile contexts.

Regression Results:
A multiple regression analysis was conducted to identify predictors of Agile Impact. The model was significant, $F(4, 86) = 28.45$, $p < .001$, with adjusted $R^2 = .61$.

Table 13. Regression Results

Predictor	B	SE	β	t	p
Productivity	.256	.078	.312	3.28	.001
Code Quality	.189	.065	.248	2.91	.004
Trust in AI	.412	.071	.452	5.80	.000
Experience	.032	.041	.059	0.78	.437

Trust in AI was the strongest predictor (β = .452, p < .001), followed by Productivity (β = .312, p = .001) and Code Quality (β = .248, p = .004). Experience was not significant (β = .059, p = .437).

Discussion:

3.2.1 Interpretation of Findings

The findings of this study paint a clear and consistent picture: AI-assisted coding tools are making a tangible difference in how productive developers feel and how Agile practices unfold on a day-to-day basis. Respondents consistently reported coding faster, completing tasks more efficiently, and feeling better equipped to tackle problems that might have previously stumped them. The productivity gains are not just marginal—they are substantial and widely experienced across different roles and experience levels.

What stands out in these results is not just that developers are faster, but how AI tools have seeped into the very fabric of Agile processes. Participants talked about AI helping with sprint planning, making it easier to clarify requirements, and breaking down larger tasks into manageable chunks. It is not replacing the human thinking that underpins Agile, but it is definitely taking some of the grunt work off people's plates and accelerating the overall flow of work.

The consistency of these findings across the sample is noteworthy. Whether looking at productivity, code quality, or Agile impact, the mean scores consistently fall in the "agree" range, with relatively modest standard deviations. This suggests that the benefits of AI tools are not limited to early adopters or particular contexts—they appear to be broadly applicable across the software development landscape.

However, the picture is not entirely rosy. A significant concern emerged repeatedly: developers are worried about becoming too dependent on AI-generated answers. There is a palpable fear that if developers let the tool do too much, they might not build the deep problem-solving muscles that

come from wrestling with tough problems independently. This is not a trivial concern. The mean score of 4.26 for the statement "AI tools may negatively affect developers' learning and problem-solving skills" is among the highest in the entire study, and it has the largest standard deviation, suggesting it is a topic of genuine debate.

The message from participants is essentially this: these tools are great, but organizations need to be intentional about ensuring developers continue to grow their actual skills and sharpen their critical thinking, not just lean on autocomplete. This finding is consistent with research by C. Zhang et al. (2023) and suggests that the software engineering community is still grappling with how to balance efficiency gains with professional development.

Another finding that came through loud and clear: AI-generated code almost always needs a human once-over before it ships. Participants were practically unanimous on this point. The statement "AI-generated code often requires modification before use" received a mean score of 4.11, and the standard deviation was relatively low (0.85), indicating broad agreement. The code might look right at first glance, but it needs editing, reviewing, and sometimes rethinking. Human supervision is not optional—it is still the safety net that catches what the AI misses and keeps the final product reliable.

This finding aligns with Liu et al. (2024) and Yetiştiren et al. (2023), who found that AI-generated code often requires substantial refinement. It also underscores a key tension identified in the literature review: while AI tools can accelerate coding, the time saved may be partially offset by additional verification and refinement effort.

The correlation analysis revealed some interesting dynamics. The strong relationship between trust in AI and Agile impact (r = .707) suggests that building trust in AI-generated suggestions is critical for realizing the benefits of these tools in team settings. This is consistent with research on human-AI collaboration by Stray et al. (2024) and Eshraghian & Cesare (2024), who found that emotional responses and trust significantly influence adoption and usage.

The absence of a significant correlation between learning concerns and productivity or Agile impact is particularly notable. It suggests that developers can hold two seemingly contradictory views simultaneously: they can see the benefits of AI tools while also worrying about their long-term learning implications. This nuanced perspective is likely more realistic than a simple "AI is good" or "AI is bad" stance. It reflects a mature understanding that technological adoption involves trade-offs and that organizations need to navigate these trade-offs thoughtfully.

The moderately strong correlation between learning concerns and overall benefits (r = .618) suggests that developers who are most worried about learning impacts are also most likely to see the overall benefits. This may seem counterintuitive, but it could reflect a deeper engagement with the implications of AI tools. Developers who have thought most carefully about the trade-offs may have the most balanced perspective, recognizing both the benefits and the risks.

Implications:

The findings of this study have several practical implications for software organizations, development teams, and project managers. First, organizations should embrace AI-assisted coding tools while establishing clear guidelines for their use. The evidence is strong that these tools improve productivity and support Agile practices. However, the concerns about over-reliance and learning impacts suggest that organizations should not simply adopt AI tools and step back. They need to be proactive about setting expectations, providing training, and encouraging responsible use.

Second, human oversight remains essential. The finding that AI-generated code almost always requires modification before deployment reinforces the importance of maintaining rigorous review processes. Code reviews, pair programming, and testing procedures should continue to be prioritized, even as AI tools automate some aspects of development.

Third, organizations should invest in training that helps developers use AI tools effectively without becoming overly dependent on them. This is not just about how to prompt AI tools, but also about when to rely on them and when to rely on one's own knowledge and skills. Encouraging developers

to view AI tools as "collaborative partners" rather than "replacement workers" may help maintain the right balance.

Fourth, development teams should consider how AI tools affect their collaboration dynamics. The moderate ratings for items about communication (Mean = 3.74) and knowledge sharing (Mean = 3.74) suggest that AI tools have not yet fully delivered on their potential to improve team collaboration. Organizations may need to be deliberate about integrating AI tools into team workflows in ways that support rather than disrupt communication and knowledge sharing.

From an academic perspective, this research contributes to the growing body of knowledge on AI in software engineering by extending existing studies beyond individual productivity to team-based Agile practices. The findings suggest that the existing literature, which has primarily focused on individual outcomes, may be underestimating both the benefits and challenges of AI tools in team settings. The study also highlights the importance of considering the multidimensional nature of AI tool impact. Productivity, code quality, Agile support, and risks are not independent dimensions but are interconnected in complex ways. Future research should continue to examine these interconnections rather than treating them as separate outcomes.

Research Contribution:

What this study adds to the conversation is something that has been largely missing from the literature: actual empirical evidence on how AI coding tools play out inside Agile teams, not just with individual developers working solo. Most of what is available currently looks at whether these tools help a single person write code faster or produce better quality output. That is useful, but it is only half the story. This research pushes further by examining what happens when these tools are brought into a team setting where Agile practices are already in motion.

More specifically, this study digs into how AI assistants affect the elements that make Agile actually Agile: sprint planning sessions, how tasks get executed, the ways teammates collaborate day to day, and how communication shifts when some of the coding work gets offloaded to a machine. Along the way, the study surfaces both the wins and the headaches that come with adopting these tools in real teams doing real work.

Beyond the academic contribution, the findings are intended to help people on the ground. Developers trying to figure out their workflow, project managers deciding what to adopt and what to watch out for, organizations that want to integrate AI without accidentally undermining how their teams work—the goal is to provide all of them with something practical to work with. Better decisions, smoother processes, stronger team performance. That is the aim.

Distinct Contributions:
This study makes several distinct contributions to the literature:

a) Empirical Evidence on Team-Level Impacts: While existing research has primarily focused on individual productivity gains, this study provides empirical evidence on how AI tools influence team-level Agile practices, including sprint execution, requirement clarification, and retrospective improvements.
b) Multi-Dimensional Assessment: The study adopts a multi-dimensional approach, examining not only productivity but also code quality, Agile support, and perceived risks, providing a holistic picture of AI tool impact.
c) Tension Identification: The findings reveal important tensions between efficiency gains and learning concerns, and between trust in AI and the need for human oversight, contributing to a more nuanced understanding of AI adoption in software development.
d) Practitioner Perspectives: By capturing the experiences of practicing developers in real-world Agile environments, the study provides actionable insights that are grounded in the realities of professional software development.

Limitations:

A. Internal Validity
The study relied on self-reported survey responses, which may introduce several forms of response bias. Participants may have provided socially desirable responses, overestimated their productivity gains, or underestimated their challenges. Self-reported data may not perfectly align with objective measures of productivity, code quality, or collaboration effectiveness. Future research should consider complementing self-report data with objective metrics such as code quality scores, task completion times, and team performance measures. Additionally, common method variance may have inflated some of the observed relationships, as both independent and dependent variables were measured through the same instrument at the same point in time. This concern is somewhat mitigated by the use of multiple-item scales and the theoretical grounding of the measures, but it remains a limitation.

B. External Validity
Although 91 participants contributed to the study, the findings may not fully represent all software developers, industries, or geographical regions. The convenience sampling approach may have introduced selection bias, as participants were recruited through specific networks and channels. The sample was predominantly experienced practitioners (over 95% with 4+ years of experience) and may not represent the perspectives of less experienced developers or those in different organizational contexts. The study was conducted in a specific cultural and professional context, and findings may not generalize to other settings. The relatively small sample size (N=91) also limits the statistical power for detecting smaller effects and for subgroup analyses.

Representativeness Limitations:
The findings of this study are based on a convenience sample of 91 software professionals, primarily from [Country/Region]. While the sample size is adequate for exploratory analysis (N = 91, exceeding the minimum requirement of 64 for medium effects with 80% power), the findings may not be representative of the broader global population of software developers.

Specifically, the sample was predominantly experienced practitioners (95% with 4+ years), potentially limiting the applicability of findings to early-career developers. The sample was also concentrated in technology and finance sectors, which may have different AI adoption patterns compared to other industries. The convenience sampling approach may have introduced self-selection bias, as developers who are more interested in or favorable toward AI tools may have been more likely to participate. Additionally, the sample was drawn primarily from Western contexts, limiting the applicability of findings to other cultural and organizational settings where AI adoption practices and Agile implementations may differ significantly.

C. Construct Validity
The Risks and Challenges construct demonstrated lower reliability ($\alpha = 0.608$) than the other constructs, indicating that respondents may perceive different AI-related risks independently rather than as a unified construct. This suggests that risks such as over-reliance, security concerns, and learning impacts may be distinct dimensions that require separate measurement. Future research should develop and validate more comprehensive measures of AI-related risks.

The measurement of Agile impact relied on self-reported perceptions of Agile practices rather than objective measures of Agile process effectiveness. Participants may have different interpretations of what constitutes successful Agile practices, and their perceptions may not align with organizational or team-level outcomes.

D. Causality
The cross-sectional design of this study precludes causal inference. While associations between AI tool usage and positive outcomes are observed, it is not possible to determine whether AI tools cause improvements in productivity, code quality, or Agile practices. It is equally possible that productive teams are more likely to adopt AI tools, creating a spurious relationship. Longitudinal studies or experimental designs would be needed to establish causality.

Suggestions:

There is still much that is not known about the long-term effects of AI-assisted coding tools in Agile environments. Future research should address the following directions: Longitudinal Studies: There is a pressing need for longitudinal studies that track the same teams over an extended period, ideally months or even years. Such studies would provide insight into whether the productivity gains observed in the short term persist over time, how AI tools affect software quality in the long run, and whether the concerns about learning and skill development are borne out. Larger and More Diverse Samples: Future research should aim for larger samples from different countries, industries, and organizational contexts. This would improve generalizability and allow for meaningful subgroup analyses to examine how different contexts affect the benefits and challenges of AI adoption.

Comparison Studies: Comparing Agile teams that actively use AI coding tools with teams following more traditional practices would provide valuable insights. Such comparisons should examine not just output metrics but also team dynamics, communication patterns, and areas of friction. Experimental and Quasi-Experimental Designs: Controlled experiments and quasi-experimental designs could help establish causality and isolate the specific effects of AI tools on development outcomes. Such studies could also examine the effects of different usage patterns (e.g., using AI for some tasks but not others) and different levels of AI integration.

Qualitative and Mixed-Methods Research: In-depth qualitative studies (e.g., interviews, observations, ethnography) could provide richer insights into how AI tools actually affect team collaboration, communication, and decision-making. Mixed-methods approaches that combine quantitative and qualitative data could provide both breadth and depth of understanding. Specific Agile Practices: Future research should examine the effects of AI tools on specific Agile practices in more detail. For example, how do AI tools affect sprint retrospectives, daily stand-ups, or code reviews? Understanding these specific mechanisms would help organizations optimize their use of AI tools.

Developer Skill Development: Given the concerns about learning and problem-solving skills, future research should examine how AI tool usage affects developer skill development over time. This could involve longitudinal studies that track skill acquisition, assessments of problem-solving capabilities, and examinations of the conditions under which AI tools enhance versus diminish learning. Practical Implementation Research: Future research should examine the specific organizational practices, training programs, and governance structures that enable effective AI tool integration in Agile teams. Studies could investigate how different onboarding approaches affect tool adoption, trust, and long-term usage patterns.

Developer Skill Trajectories: Longitudinal studies should examine how AI tool usage affects developer skill development over time. Do developers who use AI tools extensively develop weaker problem-solving skills, or do they become more effective at higher-level design and architecture? This question has significant implications for software engineering education and professional development. Cross-Cultural and Global Studies: Research should examine AI adoption across different cultural, geographical, and organizational contexts. How do cultural factors influence trust in AI tools, collaboration patterns, and adoption behaviors? AI Tool Comparative Studies: Future research should compare the effectiveness of different AI tools (ChatGPT vs. Copilot vs. Cursor, etc.) across different development tasks, programming languages, and project contexts to help organizations make evidence-based tool selection decisions.

CONCLUSION

This study set out to investigate how AI-assisted coding tools affect Agile software development practices through an empirical survey involving 91 software professionals. The goal was to move beyond the individual developer lens and examine the broader team-level implications of AI tool adoption in real-world Agile environments. The findings provide several important insights.

First and most clearly, the results demonstrate that AI-assisted coding tools have a substantial positive influence on developer productivity in Agile settings. Participants consistently reported faster task completion, reduced effort, and improved problem-solving efficiency. The productivity

gains are not marginal—they are significant and widely experienced across different roles and experience levels. The mean score of 4.20 for the Productivity construct (out of 5.00) indicates strong agreement that these tools make a tangible difference in how developers work.

Second, the findings indicate that AI tools provide meaningful support for core Agile activities. Participants reported that AI helps clarify requirements in user stories, break down tasks, identify missing requirements, and complete sprint tasks more efficiently. This suggests that AI technologies can complement Agile methodologies by reducing manual effort and streamlining development workflows. The strong support for Hypothesis 3 underscores that AI is not just a coding aid but a genuine partner in the Agile process.

Third, the study confirms that while AI-generated code is perceived as useful and quality-enhancing, it almost always requires human review and modification before deployment. This finding reinforces the importance of maintaining rigorous quality assurance practices and developer oversight. AI tools are not a replacement for human judgment; they are an augmentation. The code might look right at first glance, but it needs editing, reviewing, and sometimes rethinking. Human supervision is still the safety net that catches what the AI misses and keeps the final product reliable.

Fourth, and perhaps most significantly, the study identified important concerns that organizations must address. Developers expressed genuine apprehension about over-reliance on AI-generated solutions, security and correctness issues, and the potential negative impact on learning and problem-solving skills. The concern about learning impacts was particularly striking, with a mean score of 4.26, making it one of the highest-rated items in the entire survey. This is not a minor concern—it is a central tension in the adoption of AI tools that organizations must navigate thoughtfully.

The relationship between these concerns and other perceptions is revealing. The absence of a significant correlation between learning concerns and productivity or Agile impact suggests that developers can see the benefits of AI tools while simultaneously worrying about their long-term implications. This nuanced perspective is likely more realistic than a simple "AI is good" or "AI is bad" stance. It reflects a mature understanding that technological adoption involves trade-offs and that organizations need to navigate these trade-offs carefully.

The broader implication is that AI-assisted coding tools have the potential to effectively support Agile software development, but only when used responsibly and in conjunction with human expertise and professional judgment. Organizations should adopt these technologies carefully and establish practices that encourage responsible use, continuous learning, and effective quality assurance. The tools themselves are powerful, but their success ultimately depends on how they are integrated into team workflows, how developers are trained to use them, and how organizations balance efficiency gains with long-term professional development.

ACKNOWLEDGMENT

I would like to express my sincere gratitude to everyone who contributed to the completion of this research, titled "Impact of AI-Assisted Coding Tools on Agile Software Development Practices: An Empirical Study." This work would not have been possible without the support and assistance of many individuals and institutions. First and foremost, I am deeply grateful to my institution for providing the academic environment, resources, and infrastructure necessary to conduct this research. The institutional support, including access to academic databases, statistical software, and research facilities, was instrumental in completing this study.

A special note of appreciation goes to all the software developers, engineers, team leads, project managers, and students who generously took time out of their busy schedules to participate in the survey. This was an empirical study, and the quality of what participants brought to it—their honest, thoughtful, and practical insights—is what gave the findings real weight and meaning. Without their willingness to share their experiences and perspectives, this research would not have been possible.

I would also like to acknowledge the broader community of researchers and practitioners working at the intersection of artificial intelligence and software engineering. The growing body of knowledge in this area provided the foundation upon which this study was built, and I hope this work contributes meaningfully to that ongoing conversation.

Finally, I extend my heartfelt thanks to my family and friends for their understanding, encouragement, and moral support throughout this endeavor. Their patience during the intense moments of research and writing, their check-ins, and their belief in the value of this work kept me motivated when the path felt challenging. That kind of support does not appear in any formal acknowledgment, but it is what sustains the journey.

AUTHOR CONTRIBUTION STATEMENT

The author confirms that all aspects of this research, including study conceptualization, literature review, survey design, data collection, statistical analysis, interpretation of results, and manuscript writing, were solely carried out by the author. The author takes full responsibility for the integrity and accuracy of the data, the validity of the statistical analyses, and the scholarly merit of the content presented.

REFERENCESS

Barke, S., James, M. B., & Polikarpova, N. (2023). Grounded copilot: How programmers interact with code-generating models. *Proceedings of the ACM on Programming Languages*, *7*(OOPSLA1), 85–111. https://doi.org/10.48550/arXiv.2206.15000

Dingsøyr, T., Dybå, T., & Moe, N. B. (2010). *Agile Software Development: Current Research and Future Directions*. Springer Berlin Heidelberg. https://doi.org/10.1007/978-3-642-12575-1

Ehsani, R., Pathak, S., Parra, E., Haiduc, S., & Chatterjee, P. (2026). What characteristics make ChatGPT effective for software issue resolution? An empirical study of task, project, and conversational signals in GitHub issues. *Empirical Software Engineering*, *31*(1). https://doi.org/10.1007/s10664-025-10745-8

Eshraghian, F., & Cesare, D. (2024). AI in software programming: Understanding emotional responses to GitHub Copilot. *Information Technology & People*, *38*(4), 1659–1685. https://doi.org/10.1108/ITP-01-2023-0084

Fajkovic, E., & Rundberg, E. (2023). *The impact of AI-generated code on web development: A comparative study of ChatGPT and GitHub Copilot* [Blekinge Institute of Technology]. https://www.diva-portal.org/smash/record.jsf?pid=diva2%3A1769082&dswid=-5647

Felizardo, K. R., Lima, M. S., Deizepe, A., Conte, T. U., & Steinmacher, I. (2024). ChatGPT application in systematic literature reviews in software engineering: An evaluation of its accuracy to support the selection activity BT - International Symposium on Empirical Software Engineering and Measurement. *Proceedings of the 18th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*, 25–36. https://doi.org/10.1145/3674805.3686666

Field, A. (2018). *Discovering Statistics Using IBM SPSS Statistics* (5th ed.). Sage.

Frank, M., Kohn, V., & Holten, R. (2024). Q methodology and the sociotechnical perspective. *Information Systems and e-Business Management*, *22*(4), 599–631. https://doi.org/10.1007/s10257-024-00679-x

Geger, T., Rausch, A., Schiering, I., Stenzel, F., & Wittek, S. (2026). From Education to Evidence: A Collaborative Practice Research Platform for AI-Integrated Agile Development. *arXiv (Cornell University)*. http://arxiv.org/abs/2603.10679

Hou, X., Zhao, Y., Liu, Y., Yang, Z., Wang, K., Li, L., Luo, X., Lo, D., Grundy, J., & Wang, H. (2024). Large language models for software engineering: A systematic literature review. *ACM Transactions on Software Engineering and Methodology*, *33*(8). https://doi.org/10.1145/3695988

Jiang, E., Toh, E., Molina, A., Olson, K., Kayacik, C., Donsbach, A., Cai, C. J., & Terry, M. (2022). Discovering the syntax and strategies of natural language programming with generative language models BT. *Proceedings of the 2022 CHI Conference on Human Factors in Computing Systems*. https://doi.org/10.1145/3491102.3501870

Kline, R. B. (2016). *Principles and practice of structural equation modeling* (4 ed.). Guilford Press.
Lassenius, C., Mohagheghi, P., & Seide Molléri, J. (2025). Bringing it home: Successful backsourcing of software development in the public sector. *Empirical Software Engineering*, *30*(6). https://doi.org/10.1007/s10664-025-10722-1
Liu, Y., Le-Cong, T., Widyasari, R., Tantithamthavorn, C., Li, L., Le, X. B. D., & Lo, D. (2024). Refining ChatGPT-generated code: Characterizing and mitigating code quality issues. *ACM Transactions on Software Engineering and Methodology*, *33*(5). https://doi.org/10.1145/3643674
Mendiluze Usandizaga, E., Ali, S., Yue, T., & Arcaini, P. (2025). Quantum circuit mutants: Empirical analysis and recommendations. *Empirical Software Engineering*, *30*(3). https://doi.org/10.1007/s10664-025-10643-z
Mohamed, A., Assi, M., & Guizani, M. (2026). The Impact of LLM-Assistants on Software Developer Productivity: A Systematic Review and Mapping Study. In *arXiv (Cornell University)*. Cornell University. https://doi.org/10.1145/3809494
Nascimento, N., Alencar, P., & Cowan, D. (2023). Comparing software developers with ChatGPT: An empirical investigation. *arXiv*, *2*. https://doi.org/10.48550/arXiv.2305.11837
Neumann, M., Bischof, L., Hinz, N. E., Stockmann, L., Schrader, D., Ahaus, A. C., Demirci, E. C., Gabel, B., Rauschenberger, M., Diebold, P., Fritzemeier, H., & Przybylek, A. (2026). Between Policy and Practice: GenAI Adoption in Agile Software Development Teams. In *arXiv (Cornell University)*. Cornell University. https://doi.org/10.48550/arxiv.2601.07051
Nguyen-Duc, A., Cabrero-Daniel, B., Przybyłek, A., Arora, C., Khanna, D., Herda, T., Rafiq, U., Melegati, J., Guerra, E., Kemell, K., Saari, M., Zhang, Z., Le, H. Q., Quan, T., & Abrahamsson, P. (2025). Generative Artificial Intelligence for Software Engineering—A Research Agenda. *Software Practice and Experience*, *55*(11), 1806–1843. https://doi.org/10.1002/spe.70005
Pandey, R., Singh, P., Wei, R., & Shankar, S. (2024). Transforming software development: Evaluating the efficiency and challenges of GitHub Copilot in real-world projects. *arXiv*, *1*. https://doi.org/10.48550/arXiv.2406.17910
Pandey, S. K., Chand, S., Horkoff, J., Staron, M., Ochodek, M., & Durisic, D. (2025). Design pattern recognition: A study of large language models. *Empirical Software Engineering*, *30*(69). https://doi.org/10.1007/s10664-025-10625-1
Peng, S., Kalliamvakou, E., Cihon, P., & Demirer, M. (2023). The impact of AI on developer productivity: Evidence from GitHub Copilot. *arXiv*, *3*. http://arxiv.org/abs/2302.06590
Pileggi, S. F., & Bharathy, G. (2026). AI-gile: Revisiting Agile principles in the era of AI. *Information and Software Technology*, *194*, 108073. https://doi.org/10.1016/j.infsof.2026.108073
Rajbhoj, A., Somase, A., Kulkarni, P., & Kulkarni, V. (2024). Accelerating software development using generative AI: ChatGPT case study BT - ACM International Conference Proceeding Series. *Proceedings of the 17th Innovations in Software Engineering Conferenc*. https://doi.org/10.1145/3641399.3641403
Sauvola, J., Tarkoma, S., Klemettinen, M., Riekki, J., & Doermann, D. (2024). Future of software development with generative AI. *Automated Software Engineering*, *31*(1). https://doi.org/10.1007/s10515-024-00426-z
Scherer, R., Siddiq, F., & Tondeur, J. (2019). The technology acceptance model (TAM): A meta-analytic structural equation modeling approach. *Educational Psychology Review*, *31*(4), 817–848. https://doi.org/10.1016/j.compedu.2018.09.009
Schwaber, K., & Sutherland, J. (2020). *The Scrum guide: The definitive guide to Scrum: The rules of the game*. https://www.scrumguides.org/scrum-guide.html
Shastri Pothukuchi, A., Vasuda Kota, L., & Mallikarjunaradhya, V. (2023). Impact of generative AI on the software development life cycle (SDLC). *International Journal of Creative Research Thoughts*, *11*(8). https://www.ijcrt.org
Storey, M. A., Hoda, R., Maciel Paz Milani, A., & Baldassarre, M. T. (2025). Guiding principles for mixed methods research in software engineering. *Empirical Software Engineering*, *30*(5). https://doi.org/10.1007/s10664-025-10629-x
Stray, V., Barbala, A., & Wivestad, V. T. (2025). Human-AI collaboration in software development: A mixed-methods study of developers' use of GitHub Copilot and ChatGPT. *Proceedings of the ACM SIGSOFT Symposium on the Foundations of Software Engineering*, 1325–1332. https://doi.org/10.1145/3696630.3730566

Stray, V., Brede, N., Sintef, M., Ganeshan, N., & Kobbenes, S. (2024). *Generative AI and developer workflows: How GitHub Copilot and ChatGPT influence solo and pair programming*. https://doi.org/10.24251/HICSS.2025.883

Strode, D. E., & Hopf, T. (2024). Agile software development: A review of current practices and future directions. *Information and Software Technology*, *165*, 107–125. https://doi.org/10.33564/IJEAST.2021.v05i12.011

Témolé, F., & Atanasova, D. (2025). Agile Integration in Software Development: Principles, Practices, and Challenges. *Software Engineering*, *11*(1), 18–29. https://doi.org/10.11648/j.se.20251101.12

Venkatesh, V., Morris, M. G., Davis, G. B., & Davis, F. D. (2003). User acceptance of information technology: Toward a unified view. *MIS Quarterly*, *27*(3), 425–478. https://doi.org/10.2307/30036540

Xames, M. D., & Topcu, T. G. (2025). Digital Engineering Transformation as a Sociotechnical Challenge: Categorization of Barriers and Their Mapping to DoD's Policy Goals. In *ArXiv.org*. https://doi.org/10.48550/arxiv.2509.15461

Yetiştiren, B., Özsoy, I., Ayerdem, M., & Tüzün, E. (2023). Evaluating the code quality of AI-assisted code generation tools: An empirical study on GitHub Copilot, Amazon CodeWhisperer, and ChatGPT. *arXiv*. http://arxiv.org/abs/2304.10778

Zacharias, J., Popova, A., Zahn, M. von, Chen, J., & Hinz, O. (2026). Developers' Dilemma: Opportunities and Pitfalls of Generative AI for Software Development. *Business & Information Systems Engineering*. https://doi.org/10.1007/s12599-026-00998-y

Zhang, B., Liang, P., Zhou, X., Ahmad, A., & Waseem, M. (2023). Practices and challenges of using GitHub Copilot: An empirical study. *Proceedings of the International Conference on Software Engineering and Knowledge Engineering*, *2023-July*, 124–129. https://doi.org/10.18293/SEKE2023-077

Zhang, C., Wu, Y., Ma, Y., Song, W., Le, Z., Cao, Z., & Zhang, J. (2023). A review on learning to solve combinatorial optimisation problems in manufacturing. *IET Collaborative Intelligent Manufacturing*, *5*(1), e12072. https://doi.org/10.1049/cim2.12072

Ziebell, A., Mielke, F., & Saintot, V. M. (2026). Mindful Scrum: Mobilising mindfulness practices to foster agility in project management teams. *International Journal of Project Management*, *44*(2), 102829. https://doi.org/10.1016/j.ijproman.2026.102829

Ziegler, A., Kalliamvakou, E., Li, X., Rice, A., Rifkin, D., & Simister, S. (2024). Productivity assessment of neural code completion. *ACM Transactions on Software Engineering and Methodology*, *33*(1), 1–25. https://doi.org/10.48550/arXiv.2205.06537