

From Code Author to Code Reviewer: A Systematic Review of Practitioner Experience with Generative AI in Software Engineering

Bassey Isong¹, Tshipuke Vahangwele², Nosipho Dladlu³

^{1,2,3}North-West University, Mahikeng Campus, South Africa

Received:

May 7, 2026

Revised:

July 21, 2026

Accepted:

September 2, 2026

Published:

October 24, 2026

Corresponding Author:

Author Name*:

Bassey Isong

Email*:

bassey.isong@nwu.ac.za

DOI:

10.63158/journalisi.v8i5.1826

© 2026 Journal of Information Systems and Informatics. This open access article is distributed under a (CC-BY License)



Abstract. Practitioner use of generative AI in software engineering has progressed faster than the evidence assessing its effects. This review examined 58 empirical studies published from 2024 to early 2026 on GitHub Copilot, ChatGPT, Claude, and Gemini. Using PRISMA-guided selection from five digital libraries, the studies addressed productivity, role shifts, code quality, adoption behaviour, and evidence gaps. Findings were synthesised into the AI Integration Consequence Model, a conceptual four-layer framework linking adoption conditions, role shifts, quality exposure, and organisational consequences. Evidence indicates that developers increasingly move from code production to code review, as more than half of AI-generated code is discarded after evaluation. Generative AI improves efficiency for boilerplate code, documentation, and unit-test scaffolding, but provides limited benefits for architectural decisions and enterprise debugging. A meta-analysis of 47 studies reported a 45.6-point gap between self-reported and objectively measured productivity. AI-generated code also showed exploitable vulnerabilities at estimated rates of 40–50%, varying by task, tool version, programming language, and measurement method. Social influence strongly predicted adoption intent, while concerns about peer judgement encouraged undisclosed use. Experienced engineers generally received an AI Boost, whereas junior engineers faced an AI Drag. Structured mentorship reduced this gap in one 14-month cohort study. Key evidence gaps include cross-cultural adoption, longitudinal skill development, and unreplicated gender disparities.

Keywords: Generative AI; Software Engineering; Developer Productivity; Code Quality; Human–AI Collaboration

1. INTRODUCTION

In recent years, the integration of generative AI (GenAI) and large language models (LLMs) into software engineering (SE) ecosystems has moved faster than the research community's ability to assess it [1], [2]. However, practitioner reports on GenAI tools and empirical research findings have not yet converged. This gap widens as tools such as GitHub Copilot, ChatGPT, and Claude move from early adoption into daily use at rates comparable to Agile and DevOps [1], [3], [4]. This pace has exceeded empirical understanding as productivity gains of up to 50% appear on bounded tasks such as boilerplate generation, unit test scaffolding, and rapid prototyping [4], [5-9]. Many also report improved flow states, as AI reduces the context-breaking interruptions created by Stack Overflow searches, a daily friction point for most developers [4], [6], [8]. However, these gains come with a redistribution of effort, as the time spent writing code is now directed toward verification [1], [10], [11], because AI-generated code contains bugs or security vulnerabilities at rates estimated between 40% and 50%, a range that shifts with task type, tool version, and measurement method [1], [4], [8], [12]. This pattern points to a redistribution of engineering effort rather than a straightforward increase in output.

In the same vein, current models to reason across full architecture or business domains set the ceiling within which those gains operate. This is a limitation practitioners like Felder et al. [1] called the context wall. The PINTURA case study, which is an 18-month integration observation of a mid-size development team, illustrates the tension in one documented case. Lead time dropped by 600% in this instance, but the same integration also led to a productivity-quality contradiction as the speed was lost in discard rates and validation expenses [13], [14]. Both results are real, showing that speed and quality do not move together and the tension will be uneven across experience levels. For example, junior engineers regard AI output as reference material, reducing early skill acquisition. Senior engineers hold back because a long-term reliance on AI would damage the core skills needed to solve complicated problems and make codebases debt-laden [1], [6], [14]. This is a sound decision to minimise risk, and this disjointed approach to mentorship and team learning is what adoption statistics do not reflect [6], [15]. At the organisational level, integration results are not a matter of tooling. Most organisations lack training and data-handling policies, which makes it difficult for engineers to control risks such as inadvertently accessing proprietary code without institutional support [14-16]. Where

prompting practices move across teams, adoption stabilises, and where they do not, the results vary enough to make firm-level comparisons unstable [15], [17], [18]. Consistent with this, organisations operating the same tools under different governance conditions have reported different outcomes, indicating that tool capability alone does not determine adoption success.

Based on the above background, this paper reviews empirical results from case studies, interviews with practitioners, and larger-scale surveys from 2024 to early 2026. This review examines how AI is reshaping Agile roles, particularly the agent architect [1], who controls and evaluates machine-generated output rather than writing code from scratch. Although prior research has examined adoption rates and task-level performance, less attention has been given to practitioner experience, skill development, and team-level dynamics within team culture. We therefore focus on engineers' experiences with these tools, since practitioner evidence captures adoption, trust, and identity effects that benchmark studies of model output cannot measure, making the two evidence bases complementary. The review focuses on professional software engineers, although three included studies from Miranda et al., Melegati et al., and Welter et al., used student samples. These are treated as supplementary evidence and are flagged where they inform a finding, not as direct evidence of professional practice. Most of the underlying evidence still comes from North America, Europe, East Asia, and large technology organisations, so claims about global adoption patterns are scoped accordingly [15], [17].

In this paper, five research questions (RQs) guided this review. RQ1: How do practitioners describe changes to their productivity, and how does this compare with objectively measured performance? RQ2: How does generative AI use reshape developer roles and mentorship or team-learning practices? RQ3: What code quality and security risks do practitioners and researchers report for AI-generated code? RQ4: What organisational and social factors shape adoption behaviour, including disclosure and governance? RQ5: What evidence gaps limit generalisation of current findings across settings and over time? Also, several constructs recur throughout this review and are defined here before first use. Role shift is the redistribution of developer effort from writing code to evaluating, correcting, and directing AI-generated output [1]. According to Felder et al. [1] and Liu et al., the agent architect is the resulting position: directing and rectifying machine output rather than authoring code from scratch. AI Boost is the productivity

gain experienced by engineers reporting from AI use. Gudimetla et al. referred to AI Drag as the skill-acquisition cost junior engineers risk when AI output substitutes for independent practice. Shadow AI is undisclosed AI tool use, often driven by fear of peer judgement or unclear policy [15]. The productivity-validation tension describes the pattern in which time saved on code generation is offset by time spent evaluating and discarding output. Epistemic debt is the gap between apparent AI-assisted knowledge and what a developer can reproduce unaided.

Section 2.2 situates this review against prior surveys of generative AI in SE. Those surveys organise the literature by technical framework, SDLC phase, or process model. By contrast, this review organises it by practitioner-reported experience, the gap those frameworks leave open. Thus, there are five contributions made by this review. In contrast to the previous literature, which focuses on technical frameworks, this review uses practitioner accounts, that is, what engineers describe about AI in their day-to-day work, rather than theoretical tools. It separates vulnerability rates from bug reduction, two phenomena that operate across various time periods and measurement tools but are often confused in the literature. Adoption intent is more reliably predicted by social impact, peer judgement, and organisational culture than by tool performance. This conclusion has significant implications for rollout design. Across the reviewed studies, the adoption data more often point toward social factors than tool capacity. The lack of cross-cultural adoption research outside of North America, Europe, and East Asia, except South Africa by Isong and Vhahangwele [19], and the lack of longitudinal data on junior skill trajectories beyond 24 months are three evidence gaps that limit generalisation. Furthermore, Besser et al. [20] found no industrial replication of the gender gap. Lastly, we propose the AI integration consequence model (AICM), a conceptual synthesis model that is, to our knowledge, the first in this literature to organise AI adoption as a chain of consequences rather than a capacity threshold. AICM arranges adoption circumstances, role shifts, social dynamics, and long-term practitioner outcomes into four sequentially ordered layers; it is proposed from this synthesis and requires future empirical validation, not a confirmed causal model.

The remainder of the paper is structured as follows. In Section 2, relevant literature is reviewed to place our study in the context of earlier research. In Section 3, the process

is described. The findings, their discussion and the research gaps are covered in Section 4. Implications and potential study directions are discussed in the conclusion of Section 5.

2. RELATED WORKS

2.1. AI in software development

Today, how engineers interact with AI agents is actually pulling software development in directions that professional norms have not yet settled around [1]. This interaction occurs in two modes: acceleration, where developers use AI to pursue known goals faster, and exploration, where they test options under uncertainty [1], [6]. Across both modes, the main strategy is “Accept, but Modify,” in which developers reshape machine-generated logic to meet specific requirements [7], [21]. However, two barriers limit the strategy that this holds. Felder et al. [1] identified spatial context blindness, the failure of models to track codebase interdependencies, and temporal context decay, where training cutoffs produce suggestions tied to deprecated code. Researchers further proposed context engineering, which is the practice of supplying models with file trees and database schemas beyond prompts, trims the retrieval errors that both barriers share at their root [1], [6], [22]. At the interface level, Meta's CodeCompose shows that triggering multi-line suggestions only at the end of defined code scopes can double keystroke savings while reducing what they termed the jarring effect of displaced code [1], [23].

The role of AI in the software development life cycle (SDLC) has extended past isolated completion tasks into full lifecycle coordination. This is because current work, like Autonomous Multi-Agent Systems MAS frameworks, including ALMAS and CogniSim, deploy specialised agents simulating Agile roles such as Product Owner, Developer, and QA, covering development from requirements through maintenance [24–26]. AI systems now handle some decisions that previously required human judgement through planning, memory, and tool-invocation components [24], [26]. This shift is also changing Agile work: product owners are taking on more requirements strategy and rapid design to constrain AI output, while engineers increasingly evaluate generated code and assume architectural responsibilities [13], [16]. AgileGen illustrates this change by linking user choices to AI-generated code through Gherkin specifications and defining acceptance criteria where requirements are incomplete [16], [27]. However, tool capability alone does not determine

adoption; workflow fit is more important [15], [28]. Where formal training is absent, shadow AI cultures can emerge as engineers conceal AI use to avoid peer review. This limits integration and can distort the adoption statistics reported by organisations [15].

This also applies to team dynamics, which AI technologies change in ways that are both quantifiable and counterproductive. According to Ferino et al. [6], AI technologies allow junior developers to ask enquiries without being scrutinised by their peers, a pattern of growing autonomy they refer to as LLM entrepreneurship [6]. That freedom, though, comes at a structural cost. According to their findings, 40-50% of users say that using GenAI technologies has reduced their reliance on coworkers, and the frequency of mentorship is correlated with peer contact [6], [29]. Similarly, social influence and peer pressure have a greater impact on adoption decisions than technical capacity, since engineers look at their colleagues' signals regarding the usage of AI before assessing the tools' actual capabilities. AI hence reduces team member productivity disparity by 40%. However, senior engineers are nonetheless worried that junior reliance on AI output may replace the development of fundamental skills that come from solving challenging challenges. AI-driven self-reliance weakens cohesiveness and structure, which are the best indicators of team learning in Agile environments [17].

Subjective and objective metrics differ in the productivity view, and this difference is not arbitrary. For example, controlled trials demonstrate that AI-pair programming can cut work completion time by up to 55.8%, with lower-skilled developers experiencing the biggest gains [30], [31]. Self-reported gains average 31.4%, although controlled trials show a mean decline of 14.2% [3], indicating that self-report outpaced trial data. The divergence can be replicated. Because checking inaccurate ideas devoured generation speed before any net advantage could appear, experienced developers using AI aid took 19% longer on complicated tasks [1], [4], [32]. A P-A-E Divergence was found in the longitudinal study conducted by Tomaz et al. [10] using the SPACE framework, which covers Satisfaction, Performance, Activity, Communication, and Efficiency. In this instance, teams produced 59.1% more value, but the volume of committed code remained constant, indicating that AI raises the value density of development activity rather than its volume [8], [10], and [33].

Studies have shown that software quality outcomes differ depending on the measurement method applied. Practitioner surveys report an 85% reduction in bugs [5], whereas repository analyses using Halstead metrics and SonarQube find exploitable security vulnerabilities in 40-50% of AI-generated code [3], [4], [12]. The use of AI-generated code can introduce code smells and maintainability problems that accumulate as technical debt [10], [31], [34]. Johnson et al. [3] captured this trade-off with a credit card metaphor: the immediate velocity gained from AI can create a maintenance burden that reduces future velocity by 12-18% [1], [3]. Review tools such as SonarQube can detect up to 60% of security vulnerabilities that human reviewers miss. These findings support mandatory security scanning and alignment with quality standards such as ISO 25000 [4], [5]. This shows that without deliberate governance, the risks of social loafing, where the availability of AI reduces critical human thinking, remain unresolved. Human-in-the-Loop (HITL) configurations and seamful design have been proposed as mechanisms for preserving human agency in hybrid teams [30], [35], [36].

2.2. Existing reviews

The most thorough coverage has been given to recent reviews and surveys focusing on how AI is changing SE, concentrating on technical frameworks and integration procedures, whereas practitioners' reports about using these tools receive very little attention. In order to create an organised process model for AI-enabled development, Erdogan et al. [37] mapped tasks to SWEBOK knowledge domains. Similar to this, Malladi and Reddy [28] categorised the literature by SDLC phases, Agile practices, and model types, while Diebold [16] put up ten hypotheses on changing Agile roles under generative AI. Both approaches addressed the issue as one of process structure rather than practitioner experience. Panyam and Gujar [30] also explored Jevons' Paradox in AI-driven development, while Liu et al. [24] reviewed 124 papers on LLM-based agent architectures, a corpus spanning end-to-end automation potential, reliability concerns, cost, and ethics. Inman et al. [38] looked into the tension between productivity and creativity in tool design, Yalçiner et al. [14] mapped generative tools to lifecycle stages, and Alamir et al. [25] looked at agent paradigms and interactive simulacra for automated code updates, all of which increased technical coverage while leaving a practitioner gap.

Table 1. Summary of Related Studies

Author(s)	Focus	Contribution
[14]	Framework for GenAI integration in SE	Methodological framework: mapping tasks to generative tools across lifecycle stages
[16]	Generative AI's impact on Agile role evolution	Position paper: proposal of 10 hypotheses on shifting Agile roles
[30]	Transformation of SE and product development	Practitioner overview: analysis of Jevon's Paradox and implications of AI-driven development
[24]	LLM-based agents for SE	Survey: review of 124 papers on agent architectures for SE tasks
[25]	Frameworks for automated code updates	Technical overview: analysis of generative agent paradigms and interactive simulacra
[31]	GenAI applications in Agile SDLC	Survey: classification of literature by SDLC stages, Agile practices, and model types
[37]	Process models for AI-enabled software development	Systematic literature review: synthesis of 82 studies to create a process model integrated with SWEBOK
[38]	Seamful AI for creative SE	Position article: discussion of productivity vs. creativity in tool design

As Table 1 shows, none of these studies fully document what engineers report about AI, including their opinions, trust behaviours, adoption trends, and concerns about their professional identities. Technical frameworks and practitioner narratives address separate questions, and addressing one necessarily leaves the other unsolved; hence, the omission is intentional. This review focuses on the practitioner side and offers evidence of how engineers face AI in their daily work to complement the methodological literature [14-16], [24], [25], [28], [30], [37], and [38].

3. METHODS

In this work, we followed established guidelines for evidence-based SE [39] and PRISMA [40] reporting rules. Five research questions form the basis of this review. We categorised their productivity results and how well they matched measured performance; role

changes and mentoring strategies; code quality and security risks; organisational and societal factors impacting adoption; and evidence gaps restricting long-term understanding of generative AI in practice. The search focused on peer-reviewed journals and conference proceedings from IEEE Xplore, ScienceDirect, ACM, Springer, and Google Scholar that were released between 2024 and the beginning of 2026. This differentiation was deliberate because the availability of practitioner skills for large-scale research began in 2024 with the widespread adoption of technologies like GitHub Copilot, ChatGPT, and Claude [1], [28]. Previous research primarily reflects opinions rather than practices. Productivity, roles, quality, adoption, and evidence gaps were all search phrases that were tailored to each RQ.

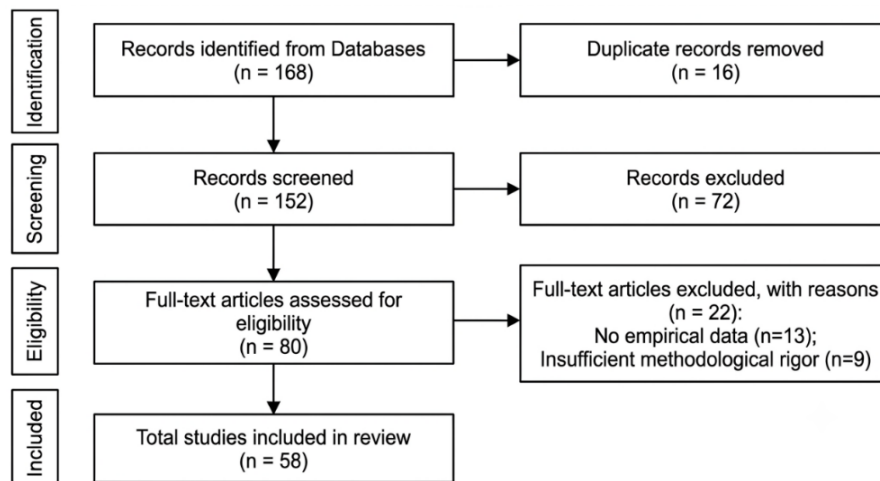


Figure 1. Study selection workflow

Research on engineers' attitudes, actions, or experiences with AI-assisted development that offered empirical data was taken into consideration. Exclusion criteria eliminated works focused primarily on team dynamics without AI application, technical studies of model architectures without practitioner data, and works where no engineer experience was described. Peer-reviewed journal articles and conference papers formed the core corpus. A small number of arXiv and SSRN preprints were retained where they reported original empirical data (survey, interview, or repository-based) directly relevant to an RQ, with no peer-reviewed equivalent available at the time of screening. This shows how quickly practitioner-facing GenAI research moves ahead of the peer-review cycle in this area. Retained preprints are identified as such in the reference list. Similarly, a multi-

stage protocol was used for screening. Out of the 168 records that were found, 16 duplicates were removed, 72 were rejected at the title and abstract stage, and 22 were rejected at the full-text stage, leaving 58 papers in the final synthesis corpus as shown in Figure 1. Of the 22 full-text exclusions, thirteen lacked empirical data, and nine did not meet rigour requirements.

Only studies meeting three explicit criteria were retained for quality evaluation. The quality assessment considered three aspects of methodological quality: accuracy, sample adequacy, and reporting clarity. Accuracy was judged by whether the reported method matched the claimed study design, while sample adequacy concerned whether sample size and composition supported the stated claims. Reporting clarity focused on whether the procedures and results contained enough detail to assess rigour. These assessments informed the Strong, Moderate, and Weak evidence-strength labels in Table 9, which also account for sample size, objective versus self-reported measurement, and independent replication across studies. Full study-level detail for the 58 included studies appears directly in Tables 2 through 8, not a separate appendix. Appendix A maps each finding to its research question, while Appendix B classifies every included study by design type and peer-review status. Each entry includes method, sample, and evidence type, traceable to its research question and quality rating. Contextual elements, claimed advantages, reported difficulties, social and cognitive aspects, and methodological traits were the main focus of data extraction. Inductive codes that emerged from the data, constructions, and trust behaviours coded under adoption dynamics, technological debt, and vulnerabilities under risk factors were mixed with deductive categories from previous protocols. By combining quantitative results with qualitative information on mentorship concerns and autonomy perceptions, thematic analysis revealed recurrent themes. Instead of being smoothed, contradictions were noted. Disparities between perceived efficiency and measured production were addressed by returning to the source data. This was achieved by two independent reviewers who reported a high level of agreement in study selection and evaluation. A formal statistic such as Cohen's Kappa was not calculated given the corpus size; disagreements or discrepancies were resolved through discussion and consensus, in accordance with PRISMA recommendations [40]. However, the review is unrestricted. The variety of study formats, from small-scale interviews to large-scale trials, hindered comparability. The majority of studies concentrated on Western and East Asian enterprise contexts, with limited involvement from the Global

South or open-source organisations. The speed at which tools were produced at the time of the review, big changes in capability are expected within the review window, which might mean some of the results are out of date. As a result, the synthesis is more concerned with identifying patterns and not making broad generalisations.

4. RESULTS AND DISCUSSION

The review's findings about code quality, role shift, productivity, team dynamics, knowledge transfer, security, technical debt, and related topics are presented in this part. Tables 2 through 8 present full study summaries.

4.1. Code quality

Research on the effects of using AI technologies on code quality is examined. Using Halstead software metrics, Clark et al. [12] compared versions 3.4 and 3.5 of 625 Python samples from the DevGPT dataset, a corpus of ChatGPT-generated code derived from public developer chats. Although the code was clever and structurally dense, both versions included recurrent problems that were comparable to those created by human developers. Therefore, before incorporation into production systems, human evaluation was required. By creating an open benchmark of 8,132 annotated code suggestions covering 17 failure categories across more than 2,000 LeetCode issues in C, Java, JavaScript, and Python, Mo et al. [22] also expanded that perspective. Regardless of language, an accuracy of 89% was attained on simple problems but fell to 43% on challenging ones, a difference of 46 percentage points. Java was the most accurate, while C was the least accurate. Binary tree problems demonstrated the best performance, while graph and heap structures showed the worst.

Tosi [41] evaluated GPT-4 and Google Bard on university exam assignments excluded from their training data to assess generalisation. They reported that GPT-4 passed 29 of 32 test situations, compared with 22 for Bard, while quality metrics showed modest complexity, frequent code smells, and a few security issues. Muñoz et al. [34] similarly compared eleven AI engines using SonarQube static analysis on CRUD activities. Bing and Gemini produced fewer suggestions, and performance varied across languages and tools, with no engine consistently generating error-free output. Evidence from real-world projects points to limited direct reuse of AI-generated code. Jin et al. [42] analysed 11,751

code snippets from GitHub projects and found that only 16.8% of AI-generated code used in production precisely matched the original proposal. The rest was altered, repurposed, or used as a conceptual reference, while another 32.8% served as additional data or motivation. Discussions tied to specific commits were more effective than those based on generic source files, but reuse remained the exception.

Kuhail et al. [43] examined ChatGPT 3.5 on 180 LeetCode problems across three difficulty levels and surveyed 99 professional developers about their perceptions of AI tools. ChatGPT performed well on easy and medium problems but struggled with hard problems, consistent with the correctness-complexity pattern reported by Mo et al. [22]. The practitioner responses showed a different concern: although most developers felt secure in their current roles because human expertise remained necessary, they remained concerned about long-term job security and AI's implications for computer science education. Boilerplate generation and code explanation were the main reported uses.

Table 2. Code quality studies

Authors	Focus	Methodology	Sample Size	Findings	Limitations
Clark et al. [12]	ChatGPT Python code quality	Quantitative Halstead metric analysis	625 code snippets	Good quality code produced; recurring bugs comparable to human developers; human review necessary.	Halstead metrics may not capture all complexity aspects.
Mo et al. [22]	GitHub Copilot correctness	Empirical analysis of 2,033 problems	8,132 code suggestions	89% accuracy on easy tasks, 43% on hard tasks; Java highest accuracy, C lowest.	Non- determinism in Copilot output.

Authors	Focus	Methodology	Sample Size	Findings	Limitations
Muñoz et al. [34]	AI tool comparison across 11 engines	SonarQube static analysis	11 AI engines, CRUD tasks	Bing and Gemini produced fewer flagged issues; no engine consistently delivered stable code.	Lack of stable behaviour across all quality tests.
Tosi [41]	LLM functional correctness	Test suite execution and SonarCloud metrics	3 LLM engines, 3 complex problems	GPT-4 passed 29/32 test cases, Bard passed 22/32; security vulnerabilities rare; expert supervision required.	Academic problems focus on logic, not enterprise practice.
Jin et al. [42]	ChatGPT use in production	Empirical evaluation via DevGPT dataset	11,751 code snippets	16.8% exact match in production; 32.8% used as supplementary information.	Manual labelling risk of bias.
Kuhail et al. [43]	ChatGPT 3.5 effectiveness and developer perceptions	Quantitative code evaluation and survey	180 LeetCode problems; 99 professionals	Effective on easy/medium problems; unreliable on hard ones; developers value boilerplate use but fear long-	Limited to ChatGPT 3.5; LeetCode problems do not represent full daily development tasks.

Authors	Focus	Methodology	Sample Size	Findings	Limitations
				term job displacement.	

4.2. Development role shift

Several studies have shown that AI tool adoption has brought about an immense role shift in development practice. Felder et al. [1] documented a shift from code creator to reviewer of AI-generated logic through a mixed-methods study of 109 participants in the German SE industry. They invented the context wall concept, which is the tool's failure to capture project-specific architectures, and the experience paradox, junior developers depending heavily on AI while senior engineers remain sceptical of its reasoning, together naming the two fault lines that role shift opens. Welter et al. [29] compared knowledge transfer in traditional pair programming with human-AI collaboration using GitHub Copilot across 19 participants. Sessions with AI assistance ran more code and had less dialogue. As developers started to rely on AI-generated reasoning without completely comprehending it, the gap between task achievement and actual understanding widened. More focus was given to the work. A reduction in rationale. AI warned programmers of neglected logic, such as database commits, but spontaneous explanation events that frequently aid in team learning decreased.

Ferino et al. [6] used 22 semi-structured interviews with cross-continental practitioners to describe LLM entrepreneurship as a pattern of increasing practitioner autonomy and decreasing peer dependence. In this case, LLMs served as personal advisors and became more independent as mentorship possibilities decreased. While manual labour decreased, productivity rose. On the other hand, a change in role boundaries was indicated by a shift in technical skill depth and information transfer between senior and younger engineers. Mendes et al. [7] performed 14 semi-structured interviews with developers in parallel. The results showed that while developers maintained control using "Accept but Modify" tactics, AI assistants helped sustain momentum by assigning boilerplate jobs. Here, the emphasis turned to high-level design and confusing API navigation; intrusive interruptions and poor accuracy remained unresolved. Additionally, Tabarsi et al. [13] conducted interviews with sixteen early adopters in professional SE settings and mapped the results across four dimensions: people, process, product, and society. By outsourcing repetitive operations, developers were able to save time and devote the same amount of

work to review and verification in order to identify insecure code and hallucinations. LLMs were most helpful while debugging and implementing, and least helpful when obtaining requirements.

Gudimetla et al. [44] created a Preceptorship Framework to address what they refer to as seniority-biased technical change. They detected two asymmetric impacts using quasi-experimental cohort research with three cohorts of 102 developers. Experienced developers reported productivity multipliers ranging from 2.7x to 3.4x, indicating an AI Boost. Those with less than three years of experience, on the other hand, experienced an AI Drag because, in comparison to their mentored colleagues, their unstructured AI use actively hindered the development of core skills. Over 14 months, the Preceptor-Learner-Agent triad reduced time-to-competency by 41% by having senior mentors use Socratic AI settings that ask juniors for methods before giving code. The epistemic debt problem's underlying cause is addressed by the framework. Junior engineers who accept AI output without understanding the reasons develop a lack of basic judgement with every careless decision.

In a similar vein, Seo et al. [45] investigated how AI autonomy alters psychological ownership by monitoring ownership perception as AI contribution varied in a within-subjects experiment with thirty developers. Developers' sense of ownership and emotional connection to the code sharply decreased as the percentage of AI-generated code rose. Accountability remained even if ownership was lost. The agent-architect transition is directly impacted by this disconnect. This is because developers who take full responsibility for mistakes but feel less ownership over their program may lessen the critical input that authorship often encourages. Through thirty in-depth industry interviews, Zacharias et al. [46] saw a similar pattern, characterising it as a transition from coder to supervisor and system architect. This rewording highlights the Dunning-Kruger risk, which occurs when developers overestimate the quality of AI output, hence limiting verification work. According to the mentorship-reduction pattern described by Ferino et al. [6] and Welter et al. [29], team cohesion in their sample declined along with peer communication.

Table 3. Role shift studies

Authors	Focus	Methodology	Sample Size	Findings	Limitations
Felder et al. [1]	AI adoption in the German SE industry	Mixed-methods (interviews and survey)	109 submissions	Context Wall identified as a major adoption barrier, role shift from code creation to validation, "Experience Paradox" between junior and senior engineers.	Cross-sectional design needs longitudinal tracking.
Ferino et al. [6]	Benefits and risks of LLM adoption	Socio-technical grounded theory	22 semi-structured interviews	LLM entrepreneurship increases independence but reduces mentorship; skill degradation and reduced knowledge transfer are key risks.	IP and demographic discrepancies in recruitment.
Mendes et al. [7]	Daily developer experience with AI	Semi-structured interviews	14 software developers	AI maintains momentum on boilerplate; Accept but Modify is the dominant strategy.	Potential for researcher bias.
Tabarsi et al. [13]	LLM impact on SE practice	Semi-structured interviews	16 professional engineers	Productivity-quality paradox confirmed; LLMs	Geographic limitation (primarily US).

Authors	Focus	Methodology	Sample Size	Findings	Limitations
				most useful in implementation and debugging.	
Welter et al. [29]	Knowledge transfer in pair programming	Qualitative voice/screen recordings	19 participants	AI reduces mentorship and explanation episodes; trust behaviour reduces knowledge exchange.	Small sample limits generalisation.
Gudimetla et al. [44]	Preceptorship Framework for AI-era SE	Quasi-experimental cohort comparison	102 developers, 3 cohorts	AI Boost for seniors (2.7x–3.4x gains); AI Drag for juniors; Preceptorship reduced time-to-competency by 41% over 14 months.	Cohort assignment by onboarding date, not random; single-site study.
Seo et al. [45]	AI autonomy and psychological ownership	Within-subjects experiment	30 developers	Ownership and affective investment decline as AI autonomy increases; professional accountability remains stable regardless of AI contribution.	Student population with co-op experience; single language (Python).
Zacharias et al. [46]	Sociotechnical impact of GenAI	Qualitative expert interviews	30 industry practitioners	Role shift from coder to supervisor;	Subjective impressions; no causal

Authors	Focus	Methodology	Sample Size	Findings	Limitations
	on developers			Dunning-Kruger risk from overestimating AI output quality; team cohesion weakens as peer communication declines.	mechanisms established.

4.3. Productivity

Based on controlled research, a number of studies have documented how the deployment of AI affects productivity. In research by Paradis et al. [33], for example, 96 engineers from active production teams participated in a randomised controlled trial within Google's infrastructure, and those who used AI features finished jobs roughly 21% quicker than the control group. Senior developers and those who coded for longer periods of time had the biggest benefits, while novice developers made less progress. Kumar et al. [32] monitored 300 engineers using DeputyDev over the course of a year in a related study. In this case, the pull request review cycle time decreased by 31.8%, and the volume of code sent to production increased by 61% due to high engagement. Productivity gains of up to 77% were observed by junior developers. In both cases, the senior-junior disparity persisted. Additionally, Meta's CodeCompose was scaled from single-line to multi-line completions for thousands of developers by Dunay et al. [23]. Results showed that flash attention and continuous batching reduced latency by 60% and that semantic-context algorithms only generated recommendations at logical scope borders. Keystroke savings rose from 9% to 17% in comprehensive A/B testing, while multi-line suggestions accounted for half of acceptable characters.

In a parallel study, Tomaz et al. [10] monitored the SPACE framework: Satisfaction, Performance, Activity, Communication, and Efficiency, across three agile teams for 13 months using GitHub Copilot and internal GPT systems. Team performance and efficiency improved while committed code volume stayed constant, which is consistent with the P-A-E Divergence documented in the related work. Skill withering was recognised as a real

long-term issue, and task-dependent productivity benefits were noted. However, this trend continued beyond Google. Hasan et al. [47] also observed a 35% increase in task productivity in a mixed-method study integrating AI into coding, testing, and bug-fixing workflows. Similar to this, Li et al.'s study [48] contrasted a control group that used conventional techniques with an experimental group that used deep learning tools. Although the experimental group's project completion time decreased from 24 to 16 weeks, the improvements were constrained by computational demands and data dependency.

Additional research revealed significant discrepancies between measurement and perception. For example, in a thorough assessment of 47 studies, Johnson et al. [3] found a 45.6-point difference between felt and measured productivity. Developers felt more productive in this situation even though controlled testing revealed more vulnerabilities and slower completion times. Similar to this, Sarkar [4] discovered that React engineers earned more money from AI technologies than Vue.js or Angular developers. He ascribed this discrepancy to the density of training data. De Souza et al. collected developer stories that show the variance that aggregate figures condense, ranging from "scary reliability" to "crazy good accuracy" [49]. This phenomenon, which Stray et al. [50] named "Fence-Sitters," happens when seasoned engineers with over 20 years of experience wait to use a technology until their peers have confirmed its usefulness. In this case, AI made deep work possible and decreased repetitive activities, yet 45% of users said they relied less on peers for advice.

The greatest controlled productivity study in this corpus was carried out by Cui et al. [51] in three randomised field tests involving 4,867 developers from Microsoft, Accenture, and a major manufacturing company. In this case, pull requests showed a 26% increase in weekly completed jobs as a result of AI coding assistant access. Review approval rates remained constant or increased, and there was no discernible drop in code quality. In this instance, junior and less experienced developers had the biggest increases in productivity, which closed rather than widened the gap with senior professionals. In a quasi-experiment with 1,219 programmers at Ant Group, Gambacorta et al. [52] discovered the same seniority pattern. All developers were more productive when direct LLM code generation was used, and junior developers gained greater code comprehension than older coders.

Afroz et al. [53] used the SPACE framework, a multifaceted tool covering Satisfaction, Performance, Activity, Communication, and Efficiency, to gather data from 415 practitioners across 56 organisations. Activity measures such as code volume and commit frequency increased when AI was used. Later on, however, these advantages were offset by more stringent code review regulations. This constraint redistribution is referred to by Afroz et al. as “spurious productivity,” and it happens when effort saved during drafting returns as verification load. Effort changed but did not decrease in this case. In a mixed-methods case study of a 36-person fintech unit employing GitHub Copilot and Windsurf, Winckler et al. [54] reached the same conclusion: while cognitive load increased on output verification, a flow state improved on everyday operations. They refer to this as the production-validation conflict. These threads were integrated into a multivocal literature analysis by Farrag [55], which included 67 sources and a pilot poll of 14 engineers. Here, individual throughput gains of 20-56% were noted; however, the system-level gains were countered by a 7.2% decrease in delivery stability and a 91% increase in review time. The system slowed as people accelerated. This is known as the Productivity-Reliability Paradox. Adoption of AI accelerates individual task performance at the expense of systemic reliability. Developers in controlled trials on complex work observed a 19% slowdown, which can be explained by senior developers taking up to 4.3 minutes per recommendation on mature codebases to confirm architectural alignment.

Table 4. Productivity studies

Authors	Focus	Methodology	Sample Size	Findings	Limitations
Johnson et al. [3]	AI Productivity Paradox	Systematic review (47 studies)	77,000 organisations, 211M lines of code	45.6-point gap between perceived and measured productivity; technical debt accumulates at 12-18% future velocity cost.	Definition variability of “success” across studies.
Sarkar [4]	AI in frontend engineering	Empirical analysis of stack preferences	Enterprise deployment analysis	React benefits most from AI training density; 40–	Declining trust levels despite high adoption.

				50% security vulnerability rate; declining trust despite high adoption.	
				P-A-E	
				Divergence: performance increased while code volume stayed flat; skill atrophy identified as a long-term concern.	
Tomaz et al. [10]	Agile team productivity with AI	Longitudinal multi-case study (13 months)	3 agile teams		Skill atrophy requires further longitudinal investigation.
				Keystroke savings doubled from 9% to 17%; latency reduced by 60%; opt-out rates minimal.	
Dunay et al. [23]	Multi-line code suggestion scaling	Industrial-scale A/B rollout	Thousands of developers		Context confined to monolithic repository.
				31.8% reduction in PR review cycle time; junior developers gained up to 77% productivity improvement.	
Kumar et al. [32]	DeputyDev impact at scale	Quasi-experimental longitudinal	300 engineers		Single-organisation study limits generalizability.
Paradis et al. [33]	AI impact on development speed	Randomised controlled trial	96 participants	Engineers using AI completed	Laboratory context, single point in time.

				tasks 21% faster; benefits most pronounced among senior developers.	
Hasan et al. [47]	Human-AI co-creation framework	Mixed methods (case studies and interviews)	Organisation- level metrics	35% increase in task efficiency; developer trust and transparency identified as psychological barriers.	Adaptation and training time are major barriers.
Li et al. [48]	Deep learning tools vs. traditional methods	Randomised experimental and control study	Grouped academic participants	Experimental group completion time decreased from 24 to 16 weeks; computational demands identified as constraints.	Needs research into wider environments.
Stray et al. [50]	AI adoption in Norwegian public sector	Mixed- methods (interviews and survey)	13 interviews, 114 surveys	Fence-Sitters delay adoption until peers validate tools; 45% rely less on colleagues; skill retention concerns raised.	2025 Findings may not generalise to future states.
De Souza et al. [49]	Developer narratives	Narrative analysis	Diverse narratives	Perceptions span from	Twitter-based narratives may

	on AI assistants			accuracy to reliability concerns; incorrect code valued as a starting point.	differ from Stack Overflow.
Cui et al. [51]	Effect of AI coding assistant on developer productivity	Randomised controlled field experiments	4,867 developers across 3 firms	26% increase in completed tasks; no quality decline; junior developers narrowed productivity gap with seniors.	High weekly variation in outcomes; early experiments had low initial take-up.
Afroz et al. [53]	Perceived productivity across the SPACE framework	Large-scale survey	415 practitioners, 56 organizations	Activity gains offset by verification tax downstream; spurious productivity identified; AI supports individual tasks not team coordination.	Self-reported perception; usage frequency used as proxy for adoption depth.
Farrag [55]	Productivity-Reliability Paradox in AI-augmented development	Multivocal literature review and pilot study	67 sources; 14 engineers (pilot)	Individual gains of 20–56% offset by 91% increase in review time and 7.2% delivery stability decrease at system level.	Review dominated by GitHub Copilot; pilot statistical power limited; excluded junior developers.

Gambacorta et al. [52]	LLM-generated code impact on productivity	Quasi-experiment (CodeFuse introduction)	1,219 programmers	Direct LLM code generation significantly boosts productivity; junior developers benefit more in code comprehension than seniors.	Specific corporate context; potential internal organisational bias.
				Flow state improved on routine tasks; cognitive load increased due to output verification burden; productivity-validation tension identified.	
Winckler et al. [54]	Developer experience with Copilot and Windsurf	Mixed-methods case study	36-person technical unit		Single organisation; limited telemetry period.

4.4. Team dynamics and knowledge transfer

Adoption of AI has a substantial impact on team dynamics and knowledge exchange in development. At TiMi Studio, a large-scale Chinese game development company, Chen [31] examined ten paired and ten non-paired programming projects. Lower-skilled developers benefited the most from a 40% fall in productivity inequality and a 39.3% reduction in time. Most of that gain was driven by less skilled developers. Technical debt and code smells were also said to have decreased. However, the same subject was addressed at the organisational level by Ahmad et al. [17]. They used covariance-based structural equation modelling (CB-SEM) and confirmatory factor analysis using survey data from 159 experts in five international software companies. Cohesion and reflexivity were enhanced by team structure, which, when paired with active knowledge sharing, resulted

in the team learning necessary for long-term growth. Similarly, Miranda et al. [56] presented a multimodal dataset that included 76 individuals from 19 simulated agile teams and recorded both spoken and nonverbal interactions. Substantial variability in gaze patterns across teams was observed. Behavioural synchrony correlated with team gender composition and coordination techniques, including Planning Poker.

On longitudinal data mining, Pal et al. [57] conducted a study of 1,599 open-source GitHub projects comprising 120,243 observations. An inverted U-curve relationship between team interaction and performance was identified: communication improved issue closure rates up to a threshold, after which additional interaction reduced performance. Project manager participation flattened the curve. Restrepo-Tamayo et al. [18] further suggested a dual-phase methodology combining a systematic mapping study of 40 studies with an industry survey to quantify social and human factors. Teamwork quality promoted productivity through leadership. In another study, Li et al. [15] investigated motives and challenges for AI tool adoption through 26 interviews and 395 surveys. The culture of sharing increased adoption rates, while fear of skill loss hindered it. Peer judgment was identified as a social barrier to open AI use.

In a recent study by Isong and Vhahangwele [19], which drew on 40 South African software development professionals, developers, testers, DevOps engineers, and team leads, it is the only industrial study in this corpus outside North American, European, and East Asian settings. They reported that interpersonal trust outran AI capability as a predictor of development speed ($\beta = 0.485$), exceeding all AI-specific variables in the regression model. Culture predicted speed, yet AI capability did not. GitHub Copilot was the most popular tool, reported by 75% of respondents. Early adoption initially slowed development, creating a productivity paradox in which workflow disruption preceded the benefits associated with sustained use. Felder et al. [1] and Zacharias et al. [46] described the agent architect pattern, with 95% of respondents reporting that their roles had expanded to include validation and oversight, with positive effects on team productivity. However, self-reported measures introduce response bias, while the cross-sectional design and small sample limit generalisability and causal inference. These constraints make the finding inconclusive, but it provides a signal for further investigation.

Table 5. Team dynamics studies

Authors	Focus	Methodology	Sample Size	Findings	Limitations
Li et al. [15]	AI adoption motives and challenges	Grounded theory (interviews and survey)	26 interviews, 395 surveys	Culture of sharing increases adoption; fear of skill loss and peer judgment hinder it.	Convenience sampling limits results.
Miranda et al. [56]	Multimodal agile team dynamics	Multimodal analytics (video and audio)	19 groups (76 participants)	Significant variability in gaze patterns; behavioural synchrony correlates with gender composition.	Student sample, limited ecological validity.
Ahmad et al. [17]	Team learning in large-scale agile	CFA and CB-SEM analysis	159 professional respondents	Team structure improves cohesion and reflexivity; active knowledge sharing produces team learning.	Self-reported data subject to recall bias.
Restrepo-Tamayo et al. [18]	Qualitative productivity variables	Systematic mapping and survey	40 studies	Teamwork quality promotes productivity through leadership; soft variables	Small number of studies for some variables.

Authors	Focus	Methodology	Sample Size	Findings	Limitations
				can be quantified.	
Pal et al. [57]	Communication and OSS performance	Longitudinal data mining	1,599 projects (120,243 observations)	Inverted U-curve between team interaction and performance; project manager participation flattens the curve.	Lateral shift hypothesis inconclusive.
Chen [31]	AI impact on pair programming and equity	Case study (repositories, reviews, survey)	10 paired, 10 non-paired projects	39.3%-time reduction; 40% reduction in productivity inequality; lower-skilled developers benefit most.	Specificity to mobile gaming sector.
Isong and Vhahangwele [19]	AI tool impact on team roles and delivery in South African SE teams	Mixed-methods (MLR, Spearman correlation, thematic analysis)	40 professionals	Interpersonal trust ($\beta = 0.485$) outpredicted all AI variables; 95% reported role shift toward validation; AI initially slowed, then improved	Small sample; cross-sectional design; self-report bias.

Authors	Focus	Methodology	Sample Size	Findings	Limitations
				speed with frequency.	

4.5. Adoption behaviour, security, and technical debt

Adoption behaviour, security, and technical debt in relation to the use of AI tools in software development have not received much attention in research. In contrast, Shao and Ishengoma [2] used the Unified Theory of Acceptance and Use of Technology (UTAUT) paradigm with 305 University of Dodoma participants. For both professional and student groups, social influence was a more powerful predictor of adoption intention than technical performance expectations. While experts focused on workflow integration and methodological vulnerability modification, students stressed effort expectancy and accessibility. In a multiple-case study conducted across three software development organisations, Jensen et al. [58] interviewed fifteen participants, including late-majority engineers and early adopters. Thirteen different expectations were listed. The enthusiasm was fleeting. Before the study ended, certain expectations decreased due to privacy policies and creative choices, as well as a shift in initial enthusiasm during experiential learning.

In a pilot case study including 13 seasoned professionals, Coutinho et al. [8] found that knowledge acquisition was more advantageous than direct code development. Although reliability issues and data security restrictions limited adoption, practitioners valued these tools for accelerating documentation and expanding domain expertise. This pattern was consistent with Jensen's observation that expectations deteriorate with continued use. In a different study, Besser et al. [20] used a mixed-methods survey of 251 academics and research staff from several disciplines to monitor GenAI use in research software development at a large university. Findings revealed nearly one-third already used AI tools for rapid code generation and debugging. A gender disparity cut through the adoption data: 41% of men recorded using GenAI for software development, compared with 15% of women. However, no other study in this corpus measures it. The authors flag the risk that AI tools could deepen existing equity gaps in computing if adoption is not deliberately managed.

In all of the analysed research, technical debt and security were shown to be the most significant hazards. Zhou et al. [21] classified deliberate design flaws by looking at 1,321 data items from Stack Overflow and GitHub flaws, two sources that record difficulties engineers raise in real-world situations rather than in controlled environments. Compatibility and operational issues were more common than logical or syntactic mistakes. The main causes were internal server malfunctions and network connectivity issues. Muratovic et al. [35] integrated 25 interviews with a confidential Deloitte survey of 2,200 CEOs in a related study. Their results demonstrate that while generative AI sped up code creation and reduced repetitive chores, a human-in-the-loop strategy was needed to control technical debt, standardise design procedures, and allay accuracy worries. It demonstrates how repetitive chores were reduced, but technical debt grew. Prompt engineering centres of excellence, workforce upskilling, and new performance metrics, such as defect density and API leakage rates, that are currently not tracked by assessment systems were among the suggested remedies.

Xiao et al. [59] examined self-reported GenAI usage in over 200,000 GitHub repositories. They discovered tracking code churn in 151 of the 1,292 explicit self-admissions. They discovered no overall increase in turnover across their sample, despite industry telemetry indicating a doubling of churn following AI deployment. Through project-level protocols including clear contribution criteria, code review gates, and disclosure standards, developers controlled the usage of AI. The outcome was driven by governance rather than AI deployment, and churn outcomes reflect the development of governance. Only a lower-bound estimate of the actual number is represented by self-reported admission figures. The 1,292 admissions underestimate the true GenAI penetration across the sample since the usage of shadow AI leaves no traceable disclosure in repository metadata.

Table 6. Behaviour, security and technical debt studies

Authors	Focus	Methodology	Sample Size	Findings	Limitations
Shao and Ishengoma [2]	Adoption intention drivers	UTAUT model survey and regression	305 participants	Social influence is the strongest predictor of adoption, outweighing technical	Needs instruments for distinct educational and

Authors	Focus	Methodology	Sample Size	Findings	Limitations
				performance expectations.	professional needs.
Jensen et al. [58]	Organisational AI expectation management	Multiple-case study and action research	15 participants	Thirteen distinct expectations identified; initial enthusiasm shifted during experiential learning.	Nascent tools shape current results.
Coutinho et al. [8]	Pilot case study on AI tool adoption	Pilot case study (questionnaire and Slack)	13 experienced professionals	Knowledge acquisition considered a higher benefit than direct code generation; reliability and security constraints limit adoption.	Small participant count from a single company.
Zhou et al. [21]	GitHub Copilot issue taxonomy	Open coding and constant comparison	1,321 data items	Operational and compatibility issues most frequent; most resolved through official fixes.	Rapid releases may resolve reported issues quickly.
Muratovic et al. [35]	GenAI impact on SDLC quality	Interviews and Deloitte proprietary survey	25 interviews, 2,200 executives	HITL configurations address technical debt; prompt engineering centres of excellence recommended.	Reliance on proprietary data sources.

Authors	Focus	Methodology	Sample Size	Findings	Limitations
Besser et al. [20]	GenAI adoption in research software development	Online survey (mixed methods)	251 faculty and research staff	41% of men vs 15% of women reported AI tool use; approximately one-third already using AI; gender disparity signals equity risk.	Single institution; potential self-selection bias.
Xiao et al. [59]	Self-admitted GenAI usage and code churn in OSS	Repository mining and regression discontinuity	200,000+ repos; 1,292 self-admissions; 151 repos (churn analysis)	No general increase in code churn found; developers manage AI use through project-level governance policies; self-admissions are a lower bound.	Findings represent a lower bound; many developers do not self-admit to AI usage.

4.6. Agile roles and organisational change

AI tools adoption has also exerted a great impact on Agile roles and organisational change, as reported across studies. Voutama [60] tracked the PINTURA project, a real-world Agile deployment across 54 respondents using GPT-4, Gemini, and Blackbox AI, where tools automated backlog refinement, sprint planning, and code generation. Development time accelerated by 600%; a projected six-month timeline was completed in under two months. Cinkusz et al. [61] pushed further by developing CogniSim, integrating cognitive agents powered by large language models into the SAFe framework across six simulation IDs. While agents provided inconsistent results in the open-ended and innovation phases, GPT-4 did well in code generation. In contrast, Zhang et al. [27] created AgileGen, which was assessed in 40 projects involving 20 participants. Through human-AI cooperation, the framework employed Gherkin to convert unfinished user needs into useful software. Semantic consistency was maintained via a dedicated

memory pool and iterative feedback loops; however, cascading errors in multi-agent hallucination, a failure scenario in which compounded model flaws spread throughout pipeline stages, emerged as a challenging limitation.

In a different study, 22 software engineers participated in a workshop led by Hamza et al. [36], which blended structured activities with unstructured problem-solving to reveal role boundaries. Code generation, documentation, and optimisation were all areas in which generative models excelled. Complex reasoning and security flaws required human supervision. Clear role allocation was proposed as the structural solution once social loafing emerged as a risk. Gupta [62] expanded the image by integrating 25 interviews and 400 surveys from the software, legal, and medical industries. ChatGPT and GitHub Copilot are examples of tools that function as cognitive extensions in real-time decision-making. There was a structural shift in reasoning. GitHub Copilot, Vercel's v0, and Meta AI were tested in several proof-of-concept applications by Udoidiok et al. [9]. Copilot provides general support, Meta AI provides instructive explanations, and v0 works well for frontend component development. Backup Behaviour, where AI systems take over particular team duties to reduce coordination overhead at a large software company, was studied by Hoffmann et al. [63]. Interactive AI features reduce interruptions and improve code quality. The requirement for AI to work as a cooperative team member rather than a passive tool was found to be human-like communication.

Table 7. Agile roles studies

Authors	Focus	Methodology	Sample Size	Findings	Limitations
Udoidiok et al. [9]	Copilot, v0, and Meta AI efficacy	Systematic approach and hands-on evaluation	Multiple PoC applications	v0 effective for frontend generation; Copilot for general assistance; data quality and workflow integration are constraints.	Challenges with data quality and workflow integration.

Authors	Focus	Methodology	Sample Size	Findings	Limitations
Zhang et al. [27]	AgileGen human-AI development	Experimental evaluation and HCI survey	40 projects, 20 participants	AgileGen improves code executability and user satisfaction; cascading errors in multi-agent hallucination identified.	Cascading errors in multi-agent hallucination.
Hamza et al. [36]	Human-AI engineering collaboration	Thematic analysis (workshop)	22 software engineers	Generative models effective for code generation and optimisation; social loafing identified as a risk.	Small Foundation for broad generalisation.
Voutama [60]	AI task coordination in PINTURA	Survey and case study	54 respondents	600% acceleration in development time; six-month timeline completed in under two months.	Small survey sample size
Cinkusz et al. [61]	Cognitive agents in SAFE framework	CogniSim framework simulation	6 simulation IDs	GPT-4 excels in code generation; agents struggle in open-ended phases.	Mixed results in adaptability and innovation phases.
Hoffmann et al. [63]	Chat-based AI and productivity	SPACE framework Likert-scale	Organizational study	Interactive AI improved code quality and	Future focus needed on implementation

Authors	Focus	Methodology	Sample Size	Findings	Limitations
				reduced interruptions; human-like communication identified as a condition.	decision-making.
Gupta [62]	AI copilots in knowledge work	Mixed-methods design	400 surveys, 25 interviews	Structural shift in reasoning recorded; tools act as cognitive extensions; deskilling and bias identified as risks.	Lack of investigation into long-term impacts.

4.7. Other development aspects

Equity, prejudice, and autonomy in AI adoption were examined in a number of studies. Koyanagi et al. [64] benchmarked 1,624 programming tasks from the AtCoder platform, a competitive programming archive with a variety of languages and complexity levels ranging from easy to challenging. Japanese prompts generated the highest correctness, but accuracy fell precipitously as task difficulty increased across all languages; Chinese prompts produced a 25% lower median accuracy for tough tasks compared to English. Language had an impact. Challenges were more important. Melegati et al. [65] took a direct approach to the autonomy dilemma by conducting a qualitative study in which 38 students worked through programming difficulties with varied degrees of AI access. Concerns about dependency and decreased autonomy surfaced, but confidence and productivity rose. Worker displacement, changes in professional identity, and depersonalised management were identified as possible long-term societal repercussions that would not be evident from productivity data alone.

Al-Ahmad et al. [66] carried out a scenario-based case study on a single e-commerce web project from the viewpoints of debugging, environmental restrictions, and quality requirements. The purpose of this constrained deployment was to separate debugging

and prototyping behaviour under controlled circumstances. LLMs produced documentation, generated code, and fixed problems to speed up prototyping. However, access to environmental logs became the main barrier to corporate debugging, and the lack of local environment and CI/CD pipeline integration limited performance. The ceiling was access to the local environment. In a similar vein, Souza [5] used Gioia analysis and K-Means clustering on 101 practitioners in Brazil's public and private sectors, two groups with structurally distinct priorities. While private sector teams prioritised speed, public sector developers focused on stability and compliance. The sectors had distinct desires. In other situations, generative AI in line with ISO 25000 quality requirements accelerated delivery by 65% and decreased bug density by up to 85%. A multi-year longitudinal study focusing on how developers' reading order shifts when examining AI-generated versus human-written code was also proposed by Măănă [11]. The results suggest that hallucinations and overreliance are risk failure modes that accumulate over time rather than appearing in short-term trials. Although AI technologies can increase productivity, prolonged use may weaken critical thinking and problem-solving abilities.

Gillenson et al. [67] evaluated AI-generated test cases against human-led and combinatorial testing using a design science research approach. Their evaluation covered two Python systems: a payroll program and a shipping logistics platform. AI-driven testing matched or exceeded conventional techniques in fault discovery, but performance depended strongly on prompt specificity, with ambiguous prompts producing noticeably lower test coverage. The authors therefore developed prescriptive prompt-engineering concepts based on Cognitive Load Theory and treated prompt design as an essential QA competency.

Evidence from broader software-quality research shows similar practical constraints. Gheventer et al. [68] used a Rapid Multivocal Literature Review covering academic prototypes and industrial deployments to evaluate 24 GenAI solutions across SDLC phases and ISO quality criteria. Most technologies performed well in test automation and vulnerability identification, but academic prototypes often lacked the production support and dependability required for industrial deployment. Proprietary platform lock-in and the high infrastructure costs of open-source models further created a strategic adoption conundrum. Zhong et al. [69] examined 25 primary studies on quality and problems in GenAI-enabled systems, identifying unreliability, maintainability, and performance

efficiency as the main quality dimensions. The most common issue mentioned in 76% of the included research was the unreliability of GenAI content. According to the results, the most reported quality issues were maintainability and performance efficiency. Academic research continuously lags behind industry reality due to GenAI's practice-driven pace. The practical relevance of high-quality findings for practitioners working with current tool generations is diminished by the research's inability to keep up.

Table 8. Other software development aspects

Authors	Focus	Methodology	Sample Size	Findings	Limitations
Al-Ahmad et al. [66]	ChatGPT for web development	Scenario-based analysis	Single case project	Accelerates prototyping, lacks environmental log access and CI/CD pipeline integration.	Integrate real-time context and CI/CD pipelines.
Koyanagi et al. [64]	Natural language bias in Copilot	Repetitive automated execution	1,624 questions	Chinese prompts produced 25% lower median accuracy on hard tasks; accuracy declines with task difficulty.	Translation precision issues.
Määttä [11]	Cognitive trust in AI code review	Scoping review	Planned multi-year study	Overreliance and hallucinations identified as risks; AI increases productivity but may erode critical thinking.	Tool choices depend on future participant usage.
Melegati et al. [65]	Human and social	Science fiction	38 participants	Reduced autonomy	Sample limited to students in

Authors	Focus	Methodology	Sample Size	Findings	Limitations
	implications of AI tools	exercise and problem task		alongside increased productivity; worker displacement identified as a potential longer-term effect.	one environment.
Souza [5]	AI influence on ISO 25000 quality	Mixed-methods (K-Means and Gioia analysis)	101 practitioners	85% bug reduction and 65% faster delivery in certain contexts; 78% cited enhanced collaboration.	Sample predominantly Brazilian, lacks longitudinal data.
Gillenson et al. [67]	AI-generated test cases vs traditional methods	Design Science Research experiment	2 Python test applications	AI testing matched or exceeded traditional methods; performance was highly sensitive to prompt specificity; prompt engineering principles were proposed.	Seeded defects may not represent real enterprise faults.
Gheventer et al. [68]	Industrial readiness of GenAI quality tools	Rapid Multivocal Literature Review	24 AI tools evaluated	Last-mile problem: most tools are academic prototypes lacking	Relies on reported evidence, not hands-on tool execution.

Authors	Focus	Methodology	Sample Size	Findings	Limitations
				production readiness; strategic dilemma between open- source costs and proprietary lock-in.	
Zhong et al. [69]	Quality attributes for GenAI- enabled systems	Systematic literature review	25 primary studies	Unreliability most reported challenge (76% of studies); maintainability and performance efficiency are primary quality concerns.	Academic studies lag behind industrial reality in a fast-moving field.

4.8. Discussion

The results of the review are discussed in this section. In particular, we concentrate on three questions: what 58 empirical studies reveal about GenAI in SE practice, where the data is conflicting, and where it is still lacking. Figure 2, which shows the thematic associations within themes, and Figure 3, which shows the suggested AICM and organises the results into four sequentially ordered layers: adoption circumstances, role and workflow change, quality and risk profile, and organisational consequences, both support this.

To bring these shifts into a single theoretical contribution, we propose the AICM, shown in Figure 3. It is a conceptual synthesis model, not a tested one, that arranges the reviewed findings into four sequentially ordered layers rather than reading them as independent themes. The layers describe an observed pattern of association across the reviewed studies, not a tested causal chain. Governance choices at Layer 1 are associated with what is reported at Layer 4, and each layer builds on the one before it. This ordering

is a synthesis proposal drawn from the reviewed evidence. It requires future empirical validation, for example through longitudinal or experimental designs, before it can be treated as causal.

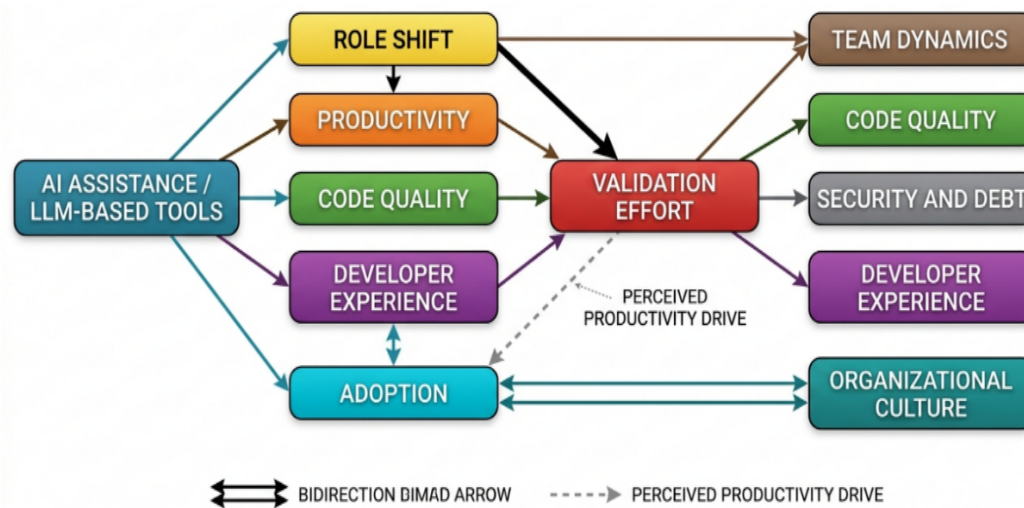


Figure 2. Thematic map of the observed relationships in AI integration across SE teams and systems

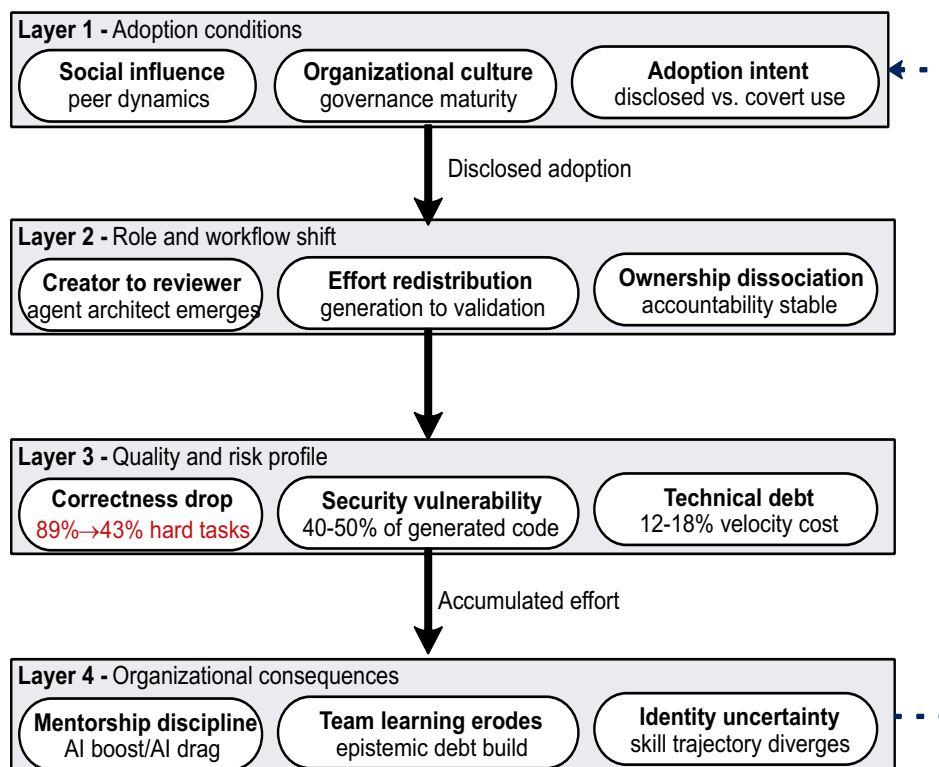


Figure 3. The AI integration consequence framework

Figure 3 presents the AICM as a sequential chain linking adoption conditions to role changes and, ultimately, software quality and risk. At the first layer, adoption circumstances influence the speed of integration and whether AI use is disclosed or remains covert, as reported by Shao and Ishengoma [2], Li et al. [15], and Stray et al. [50]. The recurring risk of shadow AI also places governance and disclosure at this stage, including stated tool-use policies and channels for reporting AI-assisted work. These mechanisms affect whether risks emerging at Layer 3 become visible to the organisation. The second layer concerns changes in roles and workflows. Developers shift effort from code generation towards validation and from writing to evaluating code [1], [13], [10], [45], marking the emergence of the architect role. Seo et al. [45] extend this view by showing that authorship and ownership may separate while accountability remains unchanged, a distinction that function-shift data alone cannot capture.

These role changes lead to the quality and risk conditions represented at Layer 3 [3], [4], [12], [22]. Technical debt, complexity-correctness trade-offs, and vulnerability rates of 40-50% form part of this profile. The reviewed studies point to several quality gates at this stage, including mandatory security scanning, code-review policies for AI-assisted contributions, disclosure standards, and prompt-governance practices. The effects at Layer 4 vary by seniority [6], [17], [29], [44]. Information transmission slows, team learning frameworks weaken, and mentorship declines. Gudimetla et al. [44] characterise this imbalance as an AI Boost for seasoned engineers and an AI Drag for novice engineers. The reviewed evidence supports this as an emerging pattern rather than an established law. It requires replication across organisations, cultures, and tool generations beyond the single cohort study in which it was observed. The reviewed studies report this junior-senior contrast consistently but rarely disaggregate a mid-level tier, which we flag as a gap in the current evidence rather than fill with an unsupported claim. However, rather than improving supervisory judgement, unstructured use raises the risk of accumulating epistemic debt.

The four layers are not independent, but sequential in nature. Layer 1 governance shortcomings encourage shadow AI cultures, which create risk exposure at Layer 3 without Layer 4 mechanisms to absorb it. There are some exceptions. The only tested intervention that ends the loop is the Preceptorship Framework [44] developed by Gudimetla et al. The feedback arrow that connects Layer 4 to Layer 1 in Figure 3 illustrates

how structured Socratic mentorship, which was implemented as policy at Layer 1, reduced the Layer 4 AI Drag impact by 41% in time-to-competency. Thus, deliberate governance design can mitigate organisational harm downstream. The AICM illustrates how the layers connect, while Figure 2 depicts the associations observed within layers.

4.8.1. What evidence establishes

Table 9 presents the evidence strength matrix. Seven findings in this corpus meet the strength criteria set out in Table 9, and several findings are strong enough to treat as reliable. These include the complexity-correctness inverse relationship, the 45.6-point productivity perception gap, the inverted U-curve in team communication, team structure as a prerequisite for collective learning, social influence as the dominant adoption predictor, the divergence between productivity perception and story-point delivery without a rise in committed code volume, and the 40-50% security vulnerability rate. The 40-50% vulnerability rate is reported across languages, model generations, and measurement instruments. However, evidence from single case studies and simulation runs, including the PINTURA 600% figure and CogniSim's Agile role replication, does not support claims of broader generalisation. The social-influence finding is based primarily on survey samples, and effect sizes from adoption studies that combine students and professionals should therefore be interpreted as indicative of a dominant pattern rather than fully representative of industry.

Two trade-offs recur in the reviewed evidence. The first concerns productivity and validation effort: time saved during code generation is accompanied by time spent evaluating and discarding output [3], [13], meaning that headline productivity gains may overstate net gains unless validation time is accounted for. The second concerns bug reduction and vulnerability risk. AI assistance can reduce some routine bugs while also introducing exploitable vulnerabilities [4], [12], [22]. Given the 40–50% vulnerability rate, quality and security outcomes may move in different directions and should therefore be reported separately. The reviewed evidence does not resolve either trade-off; instead, it indicates that aggregate productivity and quality claims should be presented with their underlying trade-offs.

Interpretation is also constrained by likely publication bias. Studies reporting large productivity gains or dramatic outcomes, such as PINTURA's 600% figure, are more visible

and citable than null results. Without a registered protocol, this review cannot fully address that bias. Tool-version instability compounds the problem. GitHub Copilot, ChatGPT, Claude, and Gemini changed materially in capability between 2024 and early 2026, so version-tied findings may not hold for the current release. Cross-study comparisons therefore implicitly compare different tool generations even when the tool name is the same. The reviewed studies also raise ethical questions that AICM's governance layer only partly addresses. Code ownership is one example because AI-assisted output blends developer and model contributions in ways existing attribution norms do not anticipate. Accountability for AI-generated defects is another. Reviewed studies describe the shift toward evaluation work but do not establish who is responsible when an evaluated, approved defect ships. Bias and accessibility concerns follow from the same training-data and interface constraints that affect other AI systems. The single documented gender-adoption gap in this corpus [20] has not been examined for either dimension. These are evidence gaps, not settled findings.

However, three methodological patterns limit what the current evidence base can establish. The most important is cross-sectional dominance: as most research lacks longitudinal designs with pre- and post-adoption measurement, Ahmad et al. [17] on team learning and Li et al. [15] on adoption establish correlations rather than causality. The second limitation is the student-as-professional proxy: Miranda et al. [56], Melegati et al. [65], and Welter et al. [29] used student samples and observed limited ecological validity; controlled academic tasks do not replicate the organisational culture, codebase complexity, and legal constraints of enterprise environments [18], [37], and [60]. The third is self-report bias: the 45.6-point productivity difference [3] shows that behaviour and perception differ sufficiently to regard repository telemetry and self-report as separate variables rather than noisy copies of the same one. Causation necessitates pre-registered controlled trials in business environments with outcome metrics unrelated to developer self-evaluation.

Table 9. Evidence strength matrix

Claim	Studies	Evidence type	Strength
Complexity-correctness inverse:			
AI correctness falls from 89% on easy tasks to 43% on hard ones	Mo et al. [22], Tosi [41]	Empirical benchmark analysis	Strong

Claim	Studies	Evidence type	Strength
Productivity perception gap: 45.6-point divergence between self-reported gains (31.4%) and measured improvement (14.2%)	Johnson et al. [3]	Meta-analysis of 47 studies	Strong
Inverted U-curve of team interaction: communication improves performance up to a threshold, then reverses through overload	Pal et al. [57]	Longitudinal data mining, 1,599 projects over 104 weeks	Strong
Team structure predicts learning: structure produces cohesion and reflexivity, which produce collective learning.	Ahmad et al. [17]	SEM and CFA, 159 professionals across 5 companies	Strong
Social influence predicts adoption ($\beta = 0.945$): outweighs technical performance expectations.	Shao and Ishengoma [2]	UTAUT survey, 305 participants	Strong
P-A-E Divergence: 59.1% increase in story point delivery with no corresponding rise in committed code volume	Tomaz et al. [10]	Longitudinal multi- case study, 13 months	Strong
Security vulnerability rate: 40– 50% of AI-generated code contains exploitable vulnerabilities	Sarkar [4], Johnson et al. [3], Clark et al. [12]	Repository analysis across multiple independent studies	Strong
Productivity-quality paradox: time saved on generation is offset by effort spent evaluating and discarding output	Tabarsi et al. [13]	Thematic analysis, 16 semi-structured interviews	Moderate

Claim	Studies	Evidence type	Strength
Inequality reduction: AI disproportionately benefits lower-skilled developers, reducing productivity inequality by 40%	Chen [31]	Case study, 20 projects at one studio	Moderate
Culture of sharing increases adoption; fear of peer judgment produces shadow AI	Li et al. [15], Stray et al. [50]	Grounded theory, 26 interviews and 395 surveys	Moderate
AI use produces individual independence while reducing mentorship and knowledge transfer.	Ferino et al. [6], Welter et al. [29]	Interview-based case studies, 22 and 19 participants	Moderate
600% development acceleration in the PINTURA project	Voutama [60]	Single case study, 54 survey respondents	Weak
Agile role replication by cognitive agents in the CogniSim framework	Cinkusz et al. [61]	Simulation study, 6 simulation runs	Weak

The evidence most consistently supports the role shift finding. The major role of the developer has shifted from producing code to evaluating it, as demonstrated by Tabarsi et al. [13], Felder et al. [1], and Ferino et al. [6], who independently document it across various national settings and techniques. After review, about half of the code produced by AI is rejected [13]. This is a shift of cognitive effort toward a skill set that most developers were not educated for, but it is not an increase in productivity. Instead of creating names from scratch, the agent architect builds [1], a position characterised by the capacity to direct and rectify machine output. The first controlled proof of its psychological link is provided by Seo et al. [45], who demonstrate that ownership of code separates from authorship while accountability for results remains unchanged. The second well-supported finding from all the studies is the productivity redistribution trend. The same structural conclusion is reached by three separate studies using various methodologies, industries, and nations. The P-A-E Divergence, where almost 59.1% more narrative points were supplied without an increase in committed code, is described by

Tomaz et al. [10]. In a similar vein, Afroz et al. [53] referred to the process as constraint redistribution, in which writing time savings materialise as review load later on. Similarly, a single financial unit exhibits the same pattern, which Winckler et al. [54] refer to as the productivity-validation tension. The evidence suggests that effort is increasing rather than decreasing. Companies that continue to measure output by lines of code may therefore misinterpret this trend.

Security vulnerabilities emerged as a key concern across the reviewed evidence. Clark et al. [12], Johnson et al. [3], and Sarkar [4] each reported exploitable flaws in 40-50% of AI-generated code, with the rate remaining consistent across languages, model generations, and measurement tools. This concern extends beyond security alone: Zhong et al. [69] found that 76% of 25 major studies identified reliability as the most common issue affecting AI-generated material. The technical debt consequence is the same. Johnson et al. [3], for example, report a fourfold increase in code cloning and a decrease in refactoring from 25% to less than 10% of repository activity, which translates to a future velocity cost of 12-18%. The fourth result, which has the greatest immediate consequences for rollout design, is that adoption in the reviewed studies was more often driven by social influence than by tool performance. This is demonstrated numerically ($\beta = 0.945$) by Shao and Ishengoma [2] and qualitatively by Stray et al. [50] using the Fence-Sitter pattern. Fence-Sitters are seasoned developers who hold off on using a tool until their colleagues have verified it, according to [50]. In an industrial setting, interpersonal trust, rather than AI capability, was the strongest predictor of development speed ($\beta = 0.485$) among 40 South African practitioners, according to Isong and Vhahangwele [19]. This finding suggests that the social variable driving adoption also determines how quickly teams extract value from the tools they adopt. Finally, Li et al. [15] add the organisational layer, where sharing prompting practices is culturally normalised, adoption rises, and where peer judgment is feared, shadow AI forms instead. Rollouts that treat deployment as a technical problem without addressing the social conditions surrounding it will produce uneven and covert adoption.

Team learning is the subject of the fifth finding. Using structural equation modelling of 159 professionals, Ahmad et al. [17] found that team structure leads to cohesiveness and reflexivity, which lead to team learning. Pal et al. [57] discovered the inverted U-curve in 1,599 open-source projects. According to their findings, communication enhances

performance up to a certain point before being reversed by overload. AI tool selection is not as reliable in predicting results as managing communication density. Once more, Welter et al. [29] and Ferino et al. [6] document how AI alters that structure. They found that 45% of users express less dependence on coworkers, a decline in spontaneous explanation events, and a deterioration in mentorship. AI indeed creates a safe area where junior devs can ask questions without fear of being judged by their peers. It also serves as a way to circumvent senior contextual coaching. In the corpus, the seniority asymmetry is the sixth finding. This is exactly what Gudimetla et al. [44] described: an AI Drag for juniors, whose unstructured AI use actively hindered the development of basic skills during a 14-month cohort, and an AI Boost for experienced engineers, where productivity multipliers of 2.7x to 3.4x were identified. This seems to be contradicted by Cui et al. [51], who reported that junior developers benefited more from their RCT involving 4,867 developers. However, the conflict is resolved by task type. Additionally, Cui et al.'s [51] field locations included well-scoped greenfield jobs, where the AI Drag effect manifests itself on intricate, structurally challenging work where output validation requires judgement. Deliberate governance at AICM Layer 1 (Figure 3) can counteract the Drag effect, as demonstrated by the fact that structured Socratic mentorship via the Preceptor-Learner-Agent triad reduces time-to-competency by 41%.

The practitioner's psychological division is the seventh discovery. According to Li et al. [15], although 91.4% of practitioners value AI for learning, 74.7% are worried about losing their knowledge and becoming overly dependent. Kuhail et al. [43] add the temporal dimension: although most developers feel comfortable in the present and acknowledge the continuous need for human abilities, they are worried about long-term job security and the next generation of graduates. An individual can be both secure in the present and concerned about the future. This explains why practitioners support XAI in several independent studies [47], [62]. It is a logical explanation of a continuous but imperfect transformation. Without explainability, judgement functions without the necessary knowledge.

4.8.2. What evidence contradicts

There were four obvious inconsistencies in this corpus. Three resolve once the measurement level is examined, and one does not. The first is the contradiction of bug reduction. Clark et al. [12], Johnson et al. [3], and Sarkar [4] claim a 40-50% vulnerability

rate in AI-generated code, whereas Souza [5] finds an 85% decrease in defects. Measurement level rather than empirical conflict is the source. Perception data gathered at task completion accounts for 85% of the figure. Repository analysis using SonarQube and Halstead metrics yields the 40-50% result. Counting fewer exploitable vulnerabilities and experiencing fewer problems are two distinct phenomena that are measured at separate time frames. Practitioners will underinvest in code review if they see the 85% statistic as proof of lower security risk. The production paradox is the second. In controlled research, Paradis et al. [33] found a 21% increase; the PINTURA project found a 600% acceleration [60]; and Cui et al. [51] found a 26% increase among 4,867 developers with no quality cost. In contrast, seasoned engineers using well-established codebases used AI 19% more slowly [55]. In their respective contexts, each of the four estimates is reasonable. 54 survey responses are used to calculate the 600% number for a single greenfield project. Specifically, the 19% delay happens when generation savings are outweighed by verification costs in mature, structurally dense codebases. The moderating factor is task abstraction level rather than AI proficiency. No single productivity metric is appropriate for all task types and codebase maturity levels.

The third difference is in the precision of the benchmark. Production code performance falls short of 89% accuracy on simple tasks and high accuracy on controlled benchmarks [22]. According to Mo et al. [22], accuracy falls to 43% when faced with challenging single-function tasks. Similarly, Tosi [38] discovers that for issues not covered by training data, models require expert supervision. The industrial impact is also documented by Gheventer et al. [68]: the majority of AI quality tools are academic prototypes that do not adapt to production conditions. Within-tool variance is important; as Gillenson et al. [67] showed, a single tool can produce good or poor test coverage depending on prompt specificity. Benchmark numbers consistently exaggerate what practitioners deal with daily. The code churn paradox is the one that does not resolve as smoothly. Here, industry telemetry predicting a doubling of turnover after GenAI deployment is cited by Farrag [55]. After analysing 200,000 repositories, Xiao et al. [59] discovered no overall rise. Because repository telemetry is used in both investigations, the measurement level is unable to account for the divergence. Variation in governance is the most likely explanation. The developers of Xiao et al. controlled the usage of AI using project-level regulations, such as disclosure standards, code review gates, and contribution guidelines that are not mentioned in the higher-churn environments described by Farrag [55]. The

AICM's positioning of Layer 1 conditions as upstream of Layer 3 outcomes is directly supported if churn measures governance maturity instead of AI use volume. However, shadow AI increases the uncertainty because, according to Xiao et al. [59], their self-admission count is a lower bound, meaning that neither study measures undisclosed-use repositories.

Team dynamics constitute a fifth, more subtle paradox. Zacharias et al. [46] describe a Dunning-Kruger dynamic in which developers tend to overestimate the quality of AI output when they reduce peer engagement. The process counts. When learning from a senior colleague, mistakes are revealed through conversation; when learning from AI, reasonable but incorrect outputs are not socially challenged before they are included in the codebase. The reduced-reliance findings from Stray et al. [50] and Ferino et al. [6], which record the identical behaviour but interpret it as increasing independence, conflict with this. Degradation of quality is not incompatible with independence. As of right now, the reviewed studies measure them independently.

4.8.3. What the evidence cannot yet establish

Regardless of how much research supports a particular conclusion, three methodological patterns throughout the investigations restrict what can be inferred from this corpus. The most important of these is cross-sectional dominance. This is because adoption findings from Li et al. [15] and team learning findings from Ahmad et al. [17] establish correlations rather than causality. Cross-sectional data makes it impossible to determine whether the adoption of AI leads to a fall in mentorship or whether organisations that are already lacking in mentorship are more prone to use AI without governance. The majority of this research lacks longitudinal designs with pre- and post-adoption measurements. The closest study, a 14-month observational cohort by Gudimetla et al. [44], is limited in what it can verify due to non-random assignment.

The second one is the student-as-professional proxy. Poor ecological validity was documented by Melegati et al. [65], Welter et al. [29], Miranda et al. [56], and Seo et al. [45], all of whom used student populations. Academic assignments are not analogous to enterprise environments, where corporate culture, team interdependencies, codebase complexity, and regulatory constraints create pressures that cannot be replicated in a lab. Seo et al.'s ownership-dissociation discovery [45] is both theoretically important and

methodologically sound. It is unclear if a fifty-person engineering team can ship to production clients. The third is bias based on self-report. The 45.6-point productivity gap [3] is proof in and of itself that there is a consistent difference between perception and behaviour. Research that relied solely on self-report for primary productivity or quality claims looked at a distinct phenomenon from research that used repository telemetry. However, they are presented side by side in the literature as though they answer the same question. The last-mile discovery made by Gheventer et al. [68] is applicable here as well: studies with positive results frequently employ specially designed tools in controlled environments that are very different from what practitioners experience in production. Pre-registered longitudinal designs in corporate contexts, random assignments when possible, and outcome measurements independent of developer self-evaluation will all be necessary to confirm causation. Even the study by Cui et al. [51], which was limited to particular task kinds and organisational contexts, came closest to that benchmark.

4.8.4. Possible research directions

Based on the review conducted, eight gaps constrain what this review can establish or recommend. The only proven defence against AI Drag in this corpus is the Preceptorship intervention by Gudimetla et al. [44], which reduces junior developer time-to-competency by 41% over 14 months. What that finding can verify is limited by a single site and non-random group assignment. A multi-site RCT with a no-mentorship control arm and a 36-month follow-up is necessary to determine whether the effect survives random assignment and lasts longer than 14 months. Additionally, role identification confusion has not been examined as a key research goal, although appearing frequently across the literature [1], [13], and [62]. Seo et al. [45] provided initial evidence that psychological ownership dissociates from authorship under high AI autonomy, in a student sample, on a single task. Therefore, a longitudinal study replicating their design with full-time mid-career practitioners across multiple sectors would establish whether the effect is durable and career consequential.

Reduced reliance on colleagues is already reported by about 45% of developers [50]. Nevertheless, no research examines whether a loss in task-level mentoring leads to a reduction in interpersonal skills throughout job transitions. This cannot be measured by self-report. Therefore, the approach that would catch it is behavioural evaluation at role

transitions, comparing AI-heavy and AI-light groups on standardised activities with independent rates across three years. Similarly, single-function problems under 60 words are covered by existing benchmarks [24]. Across easy-to-hard single-function tasks, accuracy falls from 89% to 43% [22]; enterprise codebases have domain-specific logic and cross-file dependencies that benchmarks cannot match. Extending evaluation to production-quality repositories across at least three sectors is the prerequisite for cross-study comparisons to hold.

Across the reviewed studies, adoption was more often driven by social influence ($\beta = 0.945$) [2], which means that uptake is shaped by peer dynamics and institutional trust, both of which are culturally specific. Present data are clustered in East Asian, European, and North American contexts [13], [15], [62]. With 40 South African practitioners and a mixed-methods design, Isong and Vhahangwele [19] offer the first industrial data point outside of those regions. However, a single study with a limited sample size is a signal rather than a pattern. Therefore, before governance proposals claim broader application, UTAUT replication studies with at least 200 individuals per region are required in Sub-Saharan Africa, South Asia, and Latin America. Big IT companies provide the majority of industrial data [14], [37]. Prompt engineering centres of excellence [35] and other governance measures demand organisational size, which SMEs do not have. This gap would be filled by a multi-case study with 8-12 businesses with fewer than 100 developers in the public sector, manufacturing, healthcare, and financial services, with governance policy serving as the main variable at the 12-month follow-up.

No study across the reviewed studies measures the energy or carbon cost of multi-agent AI workflows [28]. As ALMAS [25] and CogniSim [61] move toward production deployment, this becomes a regulatory and sustainability question that survey data cannot answer. Infrastructure-level telemetry measuring power draw per agent-hour, comparing multi-agent and single-agent configurations at equivalent output quality, is the only approach that produces actionable data. Also, Besser et al. [20] report a gender disparity in AI tool adoption: 41% of men versus 15% of women in a single university sample. No other study in this corpus measures it at scale. Peer dynamics at AICM Layer 1 may function asymmetrically across gender lines if the pattern holds industrially, exacerbating already-existing representation discrepancies. To ascertain whether this is local or structural, a

multi-organisation survey that replicates their instrument in addition to the social-influence variables from Shao and Ishengoma [2] is the bare minimum required.

5. CONCLUSION

Techniques for assessing generative AI in SE have not kept pace with practitioner experience. This review synthesises findings from 58 empirical studies on AI integration in development, drawing on peer-reviewed sources together with a small number of justified preprints. Findings show that practitioners described AI as a collaborator that changes the requirements of the task. The most consistent finding is the role change, where engineers spend more time assessing, fixing, and contextualising machine-generated output and less time writing code. After evaluation, almost half of AI-generated code is discarded. Productivity gains are real but activity-dependent, holding for documentation and unit test scaffolding more than for debugging and architectural decisions. The gap between perceived and measured productivity is structural rather than coincidental. Across the reviewed studies, adoption was more often predicted by social influence than by tool capability, with fear of peer judgment producing shadow AI. Cultures of sharing prompt practices increase uptake instead. Mentorship is declining, and the developmental content it provided has not been replaced. These findings should be read against several limitations. The search covered five databases and is not exhaustive. Google Scholar results in particular are not fully reproducible by later searchers. The corpus mixes study designs, from controlled trials to single case studies, and a minority of studies drew on student rather than professional samples. A small number of included preprints have not yet completed peer review. GenAI tool capability changed materially within the review window, so some findings may already be dated by the time of publication. This review cannot establish long-term causal effects on developer skill, mentorship, or team learning. The cross-sectional designs that dominate this corpus support correlation, not causation. Cross-cultural adoption evidence and the documented gender-adoption gap both remain preliminary, each resting on a small number of studies.

Across the studies, several research directions were identified, and one gap is partially addressed. In a single 14-month cohort study, structured mentorship cut junior developers' time-to-competency by 41%, yet replication across sites has not occurred.

The AI integration consequence model proposed here is a synthesis of these patterns, not a validated model. It requires testing in future empirical work, ideally longitudinal or experimental designs capable of supporting causal claims. Priority directions are longitudinal tracking of practitioner skill and identity over multi-year AI use, and industrial replication of the AI Boost/AI Drag pattern beyond a single cohort. Two further directions matter. Cross-cultural validation of adoption findings outside North America, Europe, and East Asia, and controlled mentorship interventions that test whether the 41% time-to-competency effect generalises beyond the original study, are also needed.

REFERENCES

- [1] Felder et al., "Adoption of generative artificial intelligence in the German software engineering industry: An empirical study," arXiv:2601.16700, 2026.
- [2] D. Shao and F. Ishengoma, "Empirical analysis of generative AI tool adoption in software development," *Inf. Softw. Technol.*, 2026, Art. no. 108036.
- [3] C. Johnson et al., "The AI productivity paradox in software development: A meta-analysis of implementation outcomes and ROI patterns (2024–2025)," SSRN 5902163, 2025.
- [4] S. Sarkar, "The effect of AI tools on modern software development for frontend engineering: An empirical analysis," SSRN 5442494, 2025.
- [5] M. E. Souza and E. C. S. Murillo, "How generative artificial intelligence tools can improve the quality of software produced by software development teams," 2025.
- [6] S. Ferino, R. Hoda, J. Grundy, and C. Treude, "Walking the tightrope of LLMs for software development: A practitioners' perspective," arXiv:2511.06428, 2025.
- [7] W. Mendes, S. Souza, and C. de Souza, "'You're on a bicycle with a little motor': Benefits and challenges of using AI code assistants," in *Proc. 2024 IEEE/ACM 17th Int. Conf. Cooperative and Human Aspects of Software Engineering*, 2024.
- [8] M. Coutinho et al., "The role of generative AI in software development productivity: A pilot case study," in *Proc. 1st ACM Int. Conf. AI-Powered Software*, 2024.
- [9] I. Udoidiok, H. Reza, and J. Zhang, "Exploring AI integration in software development: Case studies and insights," in *Proc. NAECON 2024 IEEE National Aerospace and Electronics Conf.*, 2024.
- [10] R. Tomaz et al., "Impacts of generative AI on agile teams' productivity: A multi-case longitudinal study," arXiv:2602.13766, 2026.

- [11] S. Määttä, "How do programmers evaluate AI-generated code?" in *Proc. 2025 ACM/IEEE Int. Symp. Empirical Software Engineering and Measurement*, 2025, pp. 522–527, doi: 10.1109/ESEM64174.2025.00057.
- [12] A. Clark, D. Igbokwe, S. Ross, and M. F. Zibran, "A quantitative analysis of quality and consistency in AI-generated code," in *Proc. ICoSSE 2024*, Paris, France, 2024, pp. 37–41, doi: 10.1109/ICoSSE62619.2024.00014.
- [13] B. Tabarsi et al., "LLMs' reshaping of people, processes, products, and society in software development," arXiv:2503.05012, 2025.
- [14] A. Yalçiner, E. Gökalp, and A. Dikici, "Transforming software engineering processes through generative AI: A framework for integration and implementation," *Computer*, vol. 58, no. 7, pp. 53–65, Jul. 2025, doi: 10.1109/MC.2025.3539347.
- [15] Z. S. Li et al., "AI tool use and adoption in software development by individuals and organisations: A grounded theory study," arXiv:2406.17325, 2024.
- [16] P. Diebold, "From backlogs to bots: Generative AI's impact on agile role evolution," *J. Softw. Evol. Process*, vol. 37, no. 1, Art. no. e2740, 2025.
- [17] M. O. Ahmad, H. Ghanbari, T. Gustavsson, and B. R. Upreti, "It all starts with structure: Investigating learning dynamics in large-scale agile software development," *J. Syst. Softw.*, vol. 230, 2025.
- [18] L. M. Restrepo-Tamayo, G. P. Gasca-Hurtado, and J. Valencia-Calvo, "Characterising social and human factors in software development team productivity," *IEEE Access*, vol. 12, pp. 59739–59755, 2024.
- [19] B. Isong and T. Vhahangwele, "AI-assisted development tools and team dynamics in South African software engineering teams," *J. Inf. Syst. Inform.*, vol. 8, no. 2, pp. 1–29, Jun. 2026, doi: 10.51519/journalisi.v8i2.1615.
- [20] S. A. Besser, E. A. Jensen, and D. S. Katz, "How generative AI is shaping research software development and maintenance at a research-intensive university," *Open Res. Eur.*, vol. 6, p. 56, 2026, doi: 10.12688/openreseurope.22009.1.
- [21] Zhou et al., "Exploring the problems, their causes and solutions of AI pair programming," *J. Syst. Softw.*, vol. 219, Art. no. 112204, 2025.
- [22] R. Mo, D. Wang, W. Zhan et al., "Assessing and analyzing the correctness of GitHub Copilot's code suggestions," *ACM Trans. Softw. Eng. Methodol.*, vol. 34, no. 7, Art. no. 194, 2025, doi: 10.1145/3715108.
- [23] O. Dunay, D. Cheng, A. Tait et al., "Multi-line AI-assisted code authoring," in *Proc. FSE 2024 Companion*, 2024, doi: 10.1145/3663529.3663836.

- [24] J. Liu et al., "Large language model-based agents for software engineering: A survey," arXiv:2409.02977, 2024.
- [25] V. Tawosi, K. Ramani, S. Alamir, and X. Liu, "ALMAS: An autonomous LLM-based multi-agent software engineering framework," in *Proc. ASEW 2025*, 2025, doi: 10.1109/ASEW67777.2025.00059.
- [26] K. Cinkusz and J. A. Chudziak, "Towards LLM-augmented multi-agent systems for agile software engineering," in *Proc. ASE '24*, 2024, doi: 10.1145/3691620.3695336.
- [27] S. Zhang, Z. Xing, R. Guo et al., "Empowering agile-based generative software development through human-AI teamwork," *ACM Trans. Softw. Eng. Methodol.*, vol. 34, no. 6, Art. no. 156, 2025, doi: 10.1145/3702987.
- [28] N. V. Malladi and S. R. K., "Generative AI in agile software development: A comprehensive survey," in *Proc. ICIDCA 2025*, 2025, pp. 1199–1208.
- [29] A. Welter et al., "An empirical study of knowledge transfer in AI pair programming," in *Proc. ASE 2025*, IEEE, 2025.
- [30] S. Panyam and P. Gujar, "How AI agents are transforming software engineering and the future of product development," *Computer*, vol. 58, no. 5, pp. 71–77, 2025.
- [31] T. Chen, "The impact of AI-pair programmers on code quality and developer satisfaction: Evidence from Timi Studio," in *Proc. 2024 Int. Conf. Generative AI and Information Security*, 2024.
- [32] A. Kumar et al., "Intuition to evidence: Measuring AI's true impact on developer productivity," arXiv:2509.19708, 2025.
- [33] E. Paradis et al., "How much does AI impact development speed? An enterprise-based randomised controlled trial," in *Proc. ICSE-SEIP 2025*, IEEE, 2025.
- [34] Muñoz et al., "Comparative study of AI code generation tools: Quality assessment and performance analysis," *LatIA*, vol. 2, p. 104, 2024.
- [35] F. Muratovic, D. Kearns-Manolatos, and A. Alibage, "Generative AI in software development: Challenges, opportunities, and new paradigms for quality assurance," *Computer*, vol. 58, no. 7, pp. 31–39, Jul. 2025.
- [36] M. Hamza, D. Siemon, M. A. Akbar, and T. Rahman, "Human-AI collaboration in software engineering: Lessons learned from a hands-on workshop," in *Proc. ACM/IEEE Workshop on Software-Intensive Business*, 2024.
- [37] T. G. Erdoğan, H. Altunel, and A. Tarhan, "A process model for AI-enabled software development: A synthesis from validation studies in white literature," *J. Softw. Evol. Process*, vol. 37, no. 1, 2025, doi: 10.1002/smr.2743.

- [38] S. Inman et al., "Seamful AI for creative software engineering," *IEEE Softw.*, vol. 42, no. 3, pp. 59–66, May–Jun. 2025.
- [39] B. Kitchenham et al., "Systematic literature reviews in software engineering: A tertiary study," *Inf. Softw. Technol.*, vol. 52, no. 8, pp. 792–805, 2010.
- [40] M. J. Page et al., "The PRISMA 2020 statement: An updated guideline for reporting systematic reviews," *BMJ*, vol. 372, 2021.
- [41] D. Tosi, "Studying the quality of source code generated by different AI generative engines," *Future Internet*, vol. 16, no. 6, p. 188, 2024.
- [42] K. Jin et al., "Can ChatGPT support developers? An empirical evaluation of large language models for code generation," in *Proc. MSR 2024*, 2024.
- [43] M. A. Kuhail, S. S. Mathew, A. Khalil, J. Berengueres, and S. J. H. Shah, "'Will I be replaced?' Assessing ChatGPT's effect on software development and programmer perceptions of AI tools," *Sci. Comput. Program.*, vol. 235, p. 103111, 2024, doi: 10.1016/j.scico.2024.103111.
- [44] S. Gudimetla, S. Gangula, R. Adinarayana, C. Challa, S. Vunnam, and S. Buhyavarapu, "From AI drag to AI boost: A preceptorship framework for developing software engineering expertise in the agentic era," in *Proc. 2026 Int. Conf. Artificial Intelligence, Systems, and Emerging Technologies*, Cairo, Egypt, 2026, pp. 1–12, doi: 10.1109/ICAISSET66439.2026.11542057.
- [45] J. Seo, E. Deldari, and H. M. Mentis, "Whose code is it? How AI autonomy reshapes ownership, responsibility, and disclosure in AI-assisted programming," in *Proc. 31st Int. Conf. Intelligent User Interfaces*, 2026.
- [46] J. Zacharias, A. Popova, M. von Zahn et al., "Developers' dilemma: Opportunities and pitfalls of generative AI for software development," *Bus. Inf. Syst. Eng.*, 2026, doi: 10.1007/s12599-026-00998-y.
- [47] Hasan et al., "Human-AI collaboration in software design: A framework for efficient co-creation," *Adv. Int. J. Multidiscip. Res.*, 2025.
- [48] K. Li et al., "Utilising deep learning to optimise software development processes," arXiv:2404.13630, 2024.
- [49] C. R. B. de Souza et al., "The fine balance between helping with your job and taking it," *IEEE Softw.*, vol. 41, no. 6, pp. 111–118, 2024.
- [50] V. Stray, A. Barbala, and V. T. Wivestad, "Human-AI collaboration in software development: A mixed-methods study of developers' use of GitHub Copilot and ChatGPT," in *Proc. FSE Companion '25*, 2025, doi: 10.1145/3696630.3730566.

- [51] K. Z. Cui et al., "The effects of generative AI on high-skilled work: Evidence from three field experiments with software developers," *Manage. Sci.*, 2026.
- [52] L. Gambacorta, H. Qiu, S. Shan, and D. M. Rees, "Generative AI and labour productivity: A quasi-experiment on coding," *J. Financ. Stab.*, vol. 84, p. 101543, 2026, doi: 10.1016/j.jfs.2026.101543.
- [53] S. Afroz et al., "The fast and spurious: Developer productivity with GenAI," in *Proc. ACM Int. Conf. Foundations of Software Engineering*, 2026.
- [54] S. C. Winckler et al., "AI-assisted collaboration: Exploring developer experience with GitHub Copilot and Windsurf," *IEEE Softw.*, 2026.
- [55] S. E. Farrag, "The productivity-reliability paradox: Specification-driven governance for AI-augmented software development," arXiv:2605.01160, 2026.
- [56] D. Miranda et al., "A multimodal experimental dataset on agile software development team interactions," *Data Brief*, vol. 61, p. 111828, 2025.
- [57] S. Pal, A. Nair, and Z. Zuo, "Collaborative dynamics in open source software development," *J. Oper. Manag.*, vol. 70, no. 7, pp. 1076–1099, 2024.
- [58] V. V. Jensen et al., "Managing expectations towards AI tools for software development: A multiple-case study," *Inf. Syst. e-Bus. Manag.*, vol. 23, pp. 869–901, 2025.
- [59] T. Xiao, Y. Fan, F. Calefato et al., "Self-admitted GenAI usage in open-source software," *IEEE Trans. Softw. Eng.*, vol. 52, no. 6, pp. 1891–1910, Jun. 2026, doi: 10.1109/TSE.2026.3681886.
- [60] A. V. Voutama, "Generative artificial intelligence in agile product management," *Indones. J. Comput. Sci.*, vol. 14, no. 3, 2025.
- [61] K. Cinkusz, J. A. Chudziak, and E. Niewiadomska-Szynkiewicz, "Cognitive agents powered by large language models for agile software project management," *Electronics*, vol. 14, no. 1, p. 87, 2025.
- [62] N. Gupta, "The rise of AI copilots: Redefining human-machine collaboration in knowledge work," *Int. J. Humanit. Inf. Technol.*, vol. 7, no. 3, 2025.
- [63] P. Hoffmann, D. Zercher, and T. C. Schimmer, "How AI communication capabilities affect developer productivity in AI pair programming," 2025.
- [64] K. Koyanagi et al., "Exploring the effect of multiple natural languages on code suggestion using GitHub Copilot," in *Proc. MSR 2024*, 2024.
- [65] J. Melegati et al., "Exploring potential implications of intelligent tools for human aspects of software engineering," in *Proc. CHASE 2024*, 2024.

- [66] A. Al-Ahmad, H. Kahtan, L. Tahat, and T. Tahat, "Enhancing software engineering with AI: Key insights from ChatGPT," in *Proc. DASA 2024*, 2024.
- [67] M. L. Gillenson et al., "Benchmarking software test cases produced by generative AI against traditional methods: A design science approach," *JDM*, vol. 37, no. 1, pp. 1–32, 2026.
- [68] A. Gheventer et al., "Generative AI solutions for software quality: Assessing industrial readiness," *Softw. Qual. J.*, vol. 34, no. 2, p. 24, 2026.
- [69] C. Zhong, H. Huang, J. Lyv, and S. Li, "Quality attributes, challenges, and solutions for designing GenAI-enabled systems," *Inf. Softw. Technol.*, vol. 198, p. 108207, 2026, doi: 10.1016/j.infsof.2026.108207.

APPENDIX A: RQ-BY-RQ SYNTHESIS

This is compiled from findings already reported in Sections 3-4 (Tables 2-9), organised by research question for direct traceability.

RQ	Question	Where answered	Summary Finding
RQ1	How do practitioners describe changes to their productivity, and how does this compare with objectively measured performance?	Table 4; Discussion – “What evidence establishes” and the four-inconsistency analysis	Perceived and measured productivity diverge by 45.6 points on average (meta-analysis, 47 studies) [3]. Gains are consistent for boilerplate code, documentation, and unit test scaffolding, but not for architectural decisions or enterprise debugging.
RQ2	How does generative AI use reshape developer roles and mentorship or team-learning practices?	Tables 3 and 5; AICM Layers 2 and 4 (Discussion)	Effort shifts from writing to evaluating code (the “agent architect” role). Mentorship and team learning decline as peer interaction falls; a single 14-month cohort study found structured mentorship reduced this effect by 41% [44].
RQ3	What code quality and security risks do practitioners and researchers report for AI-generated code?	Table 2; AICM Layer 3 (Discussion)	AI-generated code contains exploitable vulnerabilities at an estimated 40-50% rate, depending on task type, tool version, language, and measurement method [3], [4], [12], [22]. Some routine bug classes fall while security risk rises – the two do not move together.
RQ4	What organisational and social factors shape adoption behaviour, including disclosure and governance?	Table 6; AICM Layer 1 (Discussion)	Social influence more often predicts adoption than tool capability ($\beta = 0.945$) [2]. Fear of peer judgement drives undisclosed (“shadow AI”) use; governance and disclosure mechanisms are proposed as the Layer 1 lever to address this.
RQ5	What evidence gaps limit generalisation of current findings across contexts and over time?	Conclusion; Methods limitations	Three gaps recur: cross-cultural adoption evidence (concentrated in North America, Europe, East Asia, and one South African study [19]); longitudinal skill-trajectory data; and a documented but unreplicated gender-adoption gap [20].

APPENDIX B: Study Design Classification And Peer-Review Status

Design classification and peer-review status for all 58 included studies, derived from the Methodology column of Tables 2-8 and cross-referenced against the reference list. Ten studies are preprints (arXiv/SSRN), retained per the justification in Methods; all others are peer-reviewed. Source-database attribution (which of the five search databases returned each record) is not recorded per study in the original screening records and is not included here; rather than estimated.

Study	Design/evidence type	Peer-review status
Clark et al. [12]	Repository/artifact analysis	Peer-reviewed
Mo et al. [22]	Repository/artifact analysis	Peer-reviewed
Muñoz et al. [34]	Repository/artifact analysis	Peer-reviewed
Tosi [41]	Repository/artifact analysis	Peer-reviewed
Jin et al. [42]	Repository mining	Peer-reviewed
Kuhail et al. [43]	Survey + artefact analysis (mixed)	Peer-reviewed
Felder et al. [1]	Mixed-methods (interview/survey)	Preprint (arXiv)
Ferino et al. [6]	Interview (grounded theory)	Preprint (arXiv)
Mendes et al. [7]	Interview	Peer-reviewed
Tabarsi et al. [13]	Interview	Preprint (arXiv)
Welter et al. [29]	Case study (qualitative) – student sample	Peer-reviewed
Gudimetla et al. [44]	Quasi-experiment	Peer-reviewed
Seo et al. [45]	Controlled experiment	Peer-reviewed
Zacharias et al. [46]	Interview	Peer-reviewed
Johnson et al. [3]	Systematic review/meta-analysis	Preprint (SSRN)
Sarkar [4]	Repository/survey analysis	Preprint (SSRN)
Tomaz et al. [10]	Longitudinal case study	Preprint (arXiv)
Dunay et al. [23]	RCT / A-B experiment	Peer-reviewed
Kumar et al. [32]	Quasi-experiment	Preprint (arXiv)
Paradis et al. [33]	RCT	Peer-reviewed
Hasan et al. [47]	Mixed-methods (case study/interview)	Peer-reviewed
Li et al. [48]	RCT	Preprint (arXiv)
Stray et al. [50]	Mixed-methods (interview/survey)	Peer-reviewed
De Souza et al. [49]	Interview (narrative)	Peer-reviewed
Cui et al. [51]	RCT (field experiment)	Peer-reviewed
Afroz et al. [53]	Survey	Peer-reviewed
Farrag [55]	Multivocal review	Preprint (arXiv)
Gambacorta et al. [52]	Quasi-experiment	Peer-reviewed

Study	Design/evidence type	Peer-review status
Winckler et al. [54]	Case study (mixed-methods)	Peer-reviewed
Li et al. [15]	Mixed-methods (interview/survey)	Preprint (arXiv)
Miranda et al. [56]	Observational/repository (multimodal) – student sample	Peer-reviewed
Ahmad et al. [17]	Survey (structural equation modelling)	Peer-reviewed
Restrepo-Tamayo et al. [18]	Systematic review + survey	Peer-reviewed
Pal et al. [57]	Repository mining (longitudinal)	Peer-reviewed
Chen [31]	Case study (mixed-methods)	Peer-reviewed
Isong and Vhahangwele [19]	Mixed-methods (survey/interview)	Peer-reviewed
Shao and Ishengoma [2]	Survey	Peer-reviewed
Jensen et al. [58]	Case study (action research)	Peer-reviewed
Coutinho et al. [8]	Case study (pilot)	Peer-reviewed
Zhou et al. [21]	Repository mining (issue analysis)	Peer-reviewed
Muratovic et al. [35]	Mixed-methods (interview/survey)	Peer-reviewed
Besser et al. [20]	Survey	Peer-reviewed
Xiao et al. [59]	Repository mining	Peer-reviewed
Udoidiok et al. [9]	Controlled evaluation	Peer-reviewed
Zhang et al. [27]	Controlled experiment + survey	Peer-reviewed
Hamza et al. [36]	Workshop / qualitative	Peer-reviewed
Voutama [60]	Case study + survey	Peer-reviewed
Cinkusz et al. [61]	Simulation	Peer-reviewed
Hoffmann et al. [63]	Survey	Peer-reviewed
Gupta [62]	Mixed-methods (interview/survey)	Peer-reviewed
Al-Ahmad et al. [66]	Scenario-based evaluation	Peer-reviewed
Koyanagi et al. [64]	Controlled/automated evaluation	Peer-reviewed
Määttä [11]	Scoping review	Peer-reviewed
Melegati et al. [65]	Design fiction / qualitative task – student sample	Peer-reviewed
Souza [5]	Mixed-methods (repository + qualitative)	Peer-reviewed
Gillenson et al. [67]	Design science/experiment	Peer-reviewed
Gheventer et al. [68]	Multivocal review	Peer-reviewed
Zhong et al. [69]	Systematic review	Peer-reviewed