International Journal of Computer Information Systems and Industrial Management Applications
ISSN 2150-7988 Volume 18(22s) 2026 pp. 331-348
https://doi.org/10.70917/ijcisim-2026-5434
Article

Generative AI and Technical Debt in Software Development: Productivity Gains versus Code Quality and Security Risk

Dr. Rishu Roy[1], Dr. Vishal Khasgiwala[2], Dr Vanita Joshi[3]

[1]Professor & Head Center for Women's Studies, Shri Vaishnav School of Management, Shri Vaishnav Vidyapeeth, Vishwavidyalaya, Indore (MP), Email: rishuroy1429@gmail.com, rishuroy@svvv.edu.in
[2]Professor & Director, Shri Vaishnav School of Management, Shri Vaishnav Vidyapeeth Vishwavidyalaya, Indore (MP), vkhasgiwala1975@gmail.com, directorsvsm@svvv.edu.in,
[3]Designation: Dean (Exam)/Faculty, Name of Institute: ICFAI Business School, Mumbai, Email id: vanita207@gmail.com, vanita.joshi@ibsindia.org

Abstract: Generative Artificial Intelligence (GenAI) coding assistants like GitHub Copilot, ChatGPT, Amazon CodeWhisperer, and new agentic development environments have taken the software industry by storm, with the promise of enhancing developer productivity. But there has been increasing evidence to the contrary, that since these benefits come with a parallel increase in what has been dubbed "technical debt"—quality, maintainability and security problems only become apparent at a later point in the software lifecycle. This paper empirically analyzes the impact of GenAI coding assistants on developer productivity, code quality and maintainability and security posture from two perspectives, that of IT companies. Concurrent mixed method (triangulation) design is used with structured survey of 412 software professionals and repository-level analysis of 1240 software code commits with the authorship of each commit identified, and 22 semi-structured practitioner interviews. Two methods are used to test the hypothesis of the 'productivity paradox' – structural equation modelling and non-parametric artefact analysis. The results suggest that the use of GenAI can statistically significantly improve productivity in the short term ($\beta = 0.48$, $p < .001$) particularly when using GenAI for the routine aspect of work, but benefits appear to be lower for more experienced developers when considering the maintenance burden ($\beta = 0.06$, $p > .05$). Lacking good governance, AI-assisted code has substantially higher technical-debt density (37%) and security-vulnerability density (41%) than do codes written by humans. The governance maturity is the key moderator that has significant increase in the value of Δ R^2 = 0.14. The study suggests a governance model that consists of these three dimensions: velocity, maintainability, and security, and extends the concept of technical debt in the field of information systems and software engineering, adding the concept of technical debt as a socio-technical risk instead of automated.
Keywords: Generative AI; Technical Debt; Software Development; Developer Productivity; Code Quality; Security Vulnerabilities; AI Coding Assistants; Software Maintainability; Productivity Paradox; IT Firms

1. Introduction

The advent of Generative Artificial Intelligence (GenAI) in the software development lifecycle (SDLC) is one of the most significant changes in the IT industry's recent history. In a surprisingly brief period, these GenAI coding assistants have transitioned from experimental fads to essential aspects of the developer's toolkit. New tools are changing the way software is created, coded and maintained, including those that have Auto-Complete features, take a natural-language spec and generate executable code, refactor old systems and, yes, even develop test suites. The results of industry surveys and academic research indicate that universal uptake is occurring, with the advantages of quicker delivery, less cognitive load for repetitive tasks and increased entry seen as drivers for implementation.

But there has been growing hesitation about these productivity gains, given a new worry: the code generated or co-generated by generative models might come with hidden costs. The field of code generation with AI is evolving and has been expanded to include maintainability, quality, and security risks that are built into the AI-generated code, a concept that is becoming more popular and is known as "technical debt," as coined by an increasing number of studies. The idea of financing the velocity that the new AI gives to writing, at the moment of writing, with “debt” repaid at a later stage, through reworking, defect fixing, and vulnerability exposure over the software's “lifespan” is thus the central tension of this study.

This conflict has been expressed in the literature as the AI "productivity paradox": that is, although there are measurable benefits to faster coding, there are legitimate costs in the downstream that are not readily apparent in the coding speed benefits. Other empirical studies have continued this notion, and now indicate that AI tools could actually become more of a productivity burden for experienced developers, resulting in increased maintenance costs and technical debt that will be added to their workloads in the future. Some report that AI-coded code is more susceptible to security issues, hallucination bugs, and degradation over time with long-lived codebases. Meanwhile, there are countless success stories emerging of tangible gains in productivity, code quality, and developer satisfaction, especially when AI tools are integrated into a formal engineering workflow.

That this is an evidentiary gap to this present study. The paper takes an empirical and contingent perspective, and investigates the possible positive and negative impacts of GenAI as a component of a task, based on how the task is approached by the developer, and the governance environment the developer operates in.

1.1 The Industry Context

It has far-reaching ramifications for the IT sector. In today's world, software is the backbone of nearly every industry and in how fast companies can roll out dependable and secure code is one of the major factors impacting competitiveness. The primary reason for IT organisations' purchases of generative AI has been how quickly the technology can give them new features, faster onboarding, and a reduction in toil around low-value tasks. It can turn the tide the one way, when it is free, and another way, when it is a steadily increasing cost, a security problem, or an exploding maintenance expense. The costs are not immediately apparent, the benefits are immediately apparent, so the incentives, that enable firms to control the adoption judiciously, are not apparent. This is only one reason why it is much helpful to practitioners and scholars to have an integrated empirical assessment (which measures productivity in terms of quality and security in one dimension).

The other aspect of this to take into account is the diversity of the IT workforce. I found distinct differences between the use of GenAI tools among junior developers, senior developers, individual contributors, and team leads, as well as between companies based on the maturity of their process. The junior developer may think that they are becoming more productive but the burden of debt in their system is being accumulated that they will never know about; the senior developer may reject more ideas with AI but will have to invest extra time in learning about and refining them. This study emphasises these distributional impacts, which are not captured by productivity figures for aggregates. I think the answer is more of a nuanced "yes or no" than that, though, just a roadmap of the factors that could make using generative AI good or bad for the industry.

1.2 Research Questions

The research questions in the study are:

- RQ1: How effective are generative AI coding assistants at boosting developer productivity and what are the factors that influence this impact in both positive and negative ways?
- RQ2: What are the differences between AI-written code and code written by humans, regarding the quality, maintainability and amount of accumulated technical debt?

- RQ3: How secure is the AI-generated code and what governance mechanisms reduce risks in IT companies with AI generated code?

This study intends to measure the impact of generative AI coding assistants at varying levels of developer experience and task complexity, compare AI-generated code with human-written code in terms of quality, maintainability, and technical-debt accumulation, and examine the security position of AI-generated code and governance strategies to reduce risks associated with AI-generated code within IT companies.

1.3 Contributions and Structure

The paper has four contributions. In practice, it is one of the first studies and sample that fully evaluates productivity, quality, and security impacts in an empirical manner. In theory, it reframes technical debt as a greater socio-technical risk and governance maturity as the key socio-technical moderating factor. In terms of methodology it bridges conceptual survey information with artefact-level information from the repository, bringing together two sets of literatures that have been largely developed in parallel. In practice it provides an effective governance model for IT companies. The rest of this paper is organized as follows. Section 2 is a literature review, Section 3 is a conceptual framework and the hypotheses, Section 4 is methodological, Section 5 is empirical and Section 6 is a discussion of implications, while Section 7 concludes with limitations and future directions.

2. Literature Review and Theoretical Background

2.1 Generative AI in the Software Development Lifecycle

The use of generative AI has spread from every stage of the SDLC, as evidenced by a growing body of literature. The trajectory of GenAI is systematic, starting from the requirements elicitation and design stages, then moving on to code generation, testing, and maintenance (Sinaga et al., 2026; Guimaraes & Nascimento, 2025; Krichen, 2024). Sauvola et al. (2024) provide a compelling glimpse into a future where generative models are ubiquitous in engineering processes, and Benala and Dehuri (2025) highlight the evolution from conventional techniques to generative ones. In terms of the maturity of the areas of application, the survey of practitioners revealed that code generation and refactoring are the most mature application areas, while agentic and integrated-development-environment (IDE) embeddings are the current frontier (Razzak, 2026; Malladi & Reddy, 2025).

Alenezi and Akour (2025) present a survey of the state of the art and future prospects while highlighting that the use of GenAI is changing the way engineering work is carried out, and not just engineering tasks. Abbas et al. (2025) and Bósquez and González (2026) list innovations, challenges and open problems that arise along with this transition, highlighting the need for quality assurance and governance. Case evidence supports this perspective: Häkkinen (2025) and Jutila (2024) present the practice of process transformation in Finnish technology companies, and Singh (2026) presents how AI can facilitate documentation through automated practice. Together, this collection of work demonstrates the transformative role generative AI is playing in software engineering and how it is increasingly influencing the core of the field.

2.2 The Productivity Paradox of AI-Assisted Development

The productivity benefits of GenAI coding assistants are the most controversial topic in the literature. There have been many studies which have claimed significant increases in efficiency on the one hand. In a pilot program at a technology consultancy in Finland, Häkkinen (2025) reports tangible gains in developer productivity, and Thaw (2025) and Morgan (2025) each identify specific areas in developer workflow where code assistants, powered by AI, boost productivity. These benefits are blamed on automation, coordination of tasks, and redirecting developer productive potential to more valuable activities, by Zhou et al. (2026). Bonin et al. (2025) preliminary results indicate higher productivity, efficiency, and, somewhat ambivalently, job security.

On the other side, a critical literature complicates this picture. Chavali (2026) formalises the “productivity paradox” of AI-assisted development, showing that there are longitudinal trends of degradation in quality and a cost-

benefit analysis that shows that any apparent gains are only partial. Perhaps most importantly, Xu et al. (2025) report that AI programing can reduce the productivity of an experienced developer, just as it raises technical debt and maintenance costs. In an ROI and implementation outcomes meta-analysis, Johnson et al. (2025) report that realised ROI often under-achieves anticipated ROI. In Anderson et al.'s (2025) words, the "hidden costs" of generative AI are the downstream costs that are not included in the headline productivity measures.

This is not just a measurement artefact. With respect to coding performance, Shan et al. (2024) show that GenAI can have varying impacts depending on the developer's level of expertise and the specific type of coding task. Le (2026) and Maheshwari (2026) note that AI assistants change developer skills and decision-making processes that can lead to a potential long-term loss of capability even as they increase short-term throughput—a deskilling issue with direct productivity impacts. This diversity is supported by the results of the survey on the future of professional software development conducted by Määttä et al. (2026).

One of the frequent methodological issues in this strand is to separate the velocity of the author from the performance of the delivery. Measures measured at the keystroke level (e.g., lines of code per hour, task completion time, suggestion acceptance rate) are consistently in favour of AI, while measures measured at the system level over longer time frames (e.g., rework rate, defect escape, maintenance effort) are not necessarily. In fact, the difference between the two sets of metrics is the technical debt that the code carries. Studies which measure only the former report gains; studies which also measure the latter report the paradox. This observation supports the decisions of the present study to triangulate perceptual productivity measures with the quality and security metrics of the repository to observe velocity and its deferred cost in one design, rather than inferring it across incommensurate studies.

2.3 Technical Debt in AI-Augmented Codebases

The downstream implications of generative AI have become so prevalent that there is a new concept of technical debt. Gosu et al. (2026) have conducted a systematic review of GenAI in the context of technical-debt management, identifying how GenAI can be used as a debt-reduction agent and how it can be used to generate debt. Both of these are empirical findings documented in the large-scale study by Liu et al. (2026) of AI-generated code "in the wild", which indicates common patterns of debt signaling, and in the "multi-voiced" review of Ehsani et al. (2026) that establishes early warning signs of debt in LLM-assisted development.

There are several studies that extend the conceptualisation. In an empirical study on the sustainability of agent-generated code, Coppola et al. (2025) directly connect it to the metaphor of paying the debt, "AI writes code, humans pay the debt. Bushman et al. (2026) identify one specific technical-debt accumulation mechanism in codebases that are long-lived and rely on AI to augment their code, and Kara et al. (2025) suggest a hierarchical taxonomy of technical debt in GenAI-enabled systems. Reilly (2026) provides examples of real world instances of “debt due to AI” and how to address it, while Korolev (2025) describes a “year of silent failures” where technical debt from AI leads to a “verification crisis”. In the measurement angle, Jain and Thirumalasetty (2026) propose a quantitative “GenAITD” scoring model that relates enterprise technical debt to generative-AI readiness; and Thibodeau (2026) reframe technical debt as a socio-technical security risk/safety risk for the organization, rather than an engineering problem.

Not all the technical-debt literature is negative. Palakurti (2025) states that generative models are tools of AI-driven maintenance, which can help eliminate the debt of today's legacy systems, while Chunchu (2025) and Boddupally (2025) talk about generative approaches to modernising legacy systems by means of automated code translation and refactoring, respectively. Wairagade (2025) provides a framework for application modernisation using generative models. In the industrial case study, Grönroos et al. (2025) show how to use AI to introduce correlation agents that can be used to trace technical debt. This study aims to empirically resolve this ambivalence: generative AI is both a cause and a cure of technical debt.

The distinction between debt added by AI and debt added by the use of AI is useful, and implicit in these studies. On the introduction side, the mechanisms are becoming more well documented: hallucinated APIs and dependencies (Bushman et al., 2026), superficially plausible but structurally brittle code, code duplication by

regeneration instead of code reuse, and inconsistent conventions across AI-generated fragments. The same generative power is leveraged in the remediation direction: summarising unfamiliar legacy code, suggesting refactorings and translating between languages or frameworks. This study suggests that what makes the difference, not the model but the engineering discipline that surrounds it; the same tool that can help to create debt in an ungoverned pipeline, can help retire debt in a governed pipeline. This framing drives a central moderating role of the concept of governance maturity in the conceptual model.This framing pushes the concept of governance maturity in between the variables as a central moderating factor in the conceptual model.

2.4 Code Quality, Maintainability, and Security Posture

The quality and security aspects of code generated by AI have been the subject of intense scrutiny. The study takes the joint framing offered by Zardari et al. (2026) who provide an extensive analysis of the security risks and productivity impact of AI-generated code. Defects, security threats, and maintainability concerns of AI-generated code are explored in Asif (2026), and the empirical evidence concerning the factors that affect the quality of AI-generated code are synthesised in Geruslu et al. (2026). Cantada et al. (2026) provide a comparison of developer productivity, code quality and usability when creating with AI.

On security, Johnson and Hussain (2026) empirically investigate vulnerabilities of various programming languages, and Al-Hashimi (2026) presents a mitigation model for GenAI cybersecurity issues when generating code with a hybrid approach of ANN-ISM. There has been a general trend toward presenting a framework for automated vulnerability-assessment and secure-programming by Green and Tariq (2026), Kumar and Qureshi (2026), and Eze and Qureshi (2026), this latter trend focuses on making plans to address the risks rather than on documentation. Roberts and Badi (2026) provide a comparison of secure coding techniques between AI tools. The authors of Hein et al. (2026) come up with an organized representation of the risks involved in software development using generative AI, which describes the propagation of risks throughout the SDLC.

Responses that are focused on governance are prominent. To overcome what Farrag (2026) calls the "productivity-reliability paradox," he suggests a governance model based on specification, while Hurne (2025) introduces a technical-debt-aware prompting framework for sustainable "vibe coding. This paper by Waseem et al. (2025) examines how vibe coding is used in practice and provides guidelines on sustainable use, while Gadde (2025) explores how vibe coding and no-code development can democratise engineering. They all agree that the risks of generative AI are controllable but not self-controlling, meaning that deliberate organisational and process interventions are needed.

2.5 Research Gap

While there is a wealth of literature on this topic, three gaps remain. First, the productivity, quality and security aspects are usually examined separately, but they are all taken together for the IT company and must be balanced. Second, most of the evidence is of a qualitative nature (surveys, reviews) or of a repository nature (mining studies) and very few studies cross reference both. Thirdly, the moderating effect of governance maturity (the organisational practices that influence the management of technical debt) is still not well theorised. This study fills these gaps with an empirical integrated mixed methods study.

3. Conceptual Framework and Hypotheses

Based on the literature reviewed, this study proposes an integrated conceptual model that introduces the moderating effects of developer experience, task complexity, and governance maturity on the direct relationships between the adoption of generative AI and developer productivity, code quality / maintainability, and security posture. Technical debt is the mediating variable between the short and long term delivery of software solutions. Model is shown in figure 1.

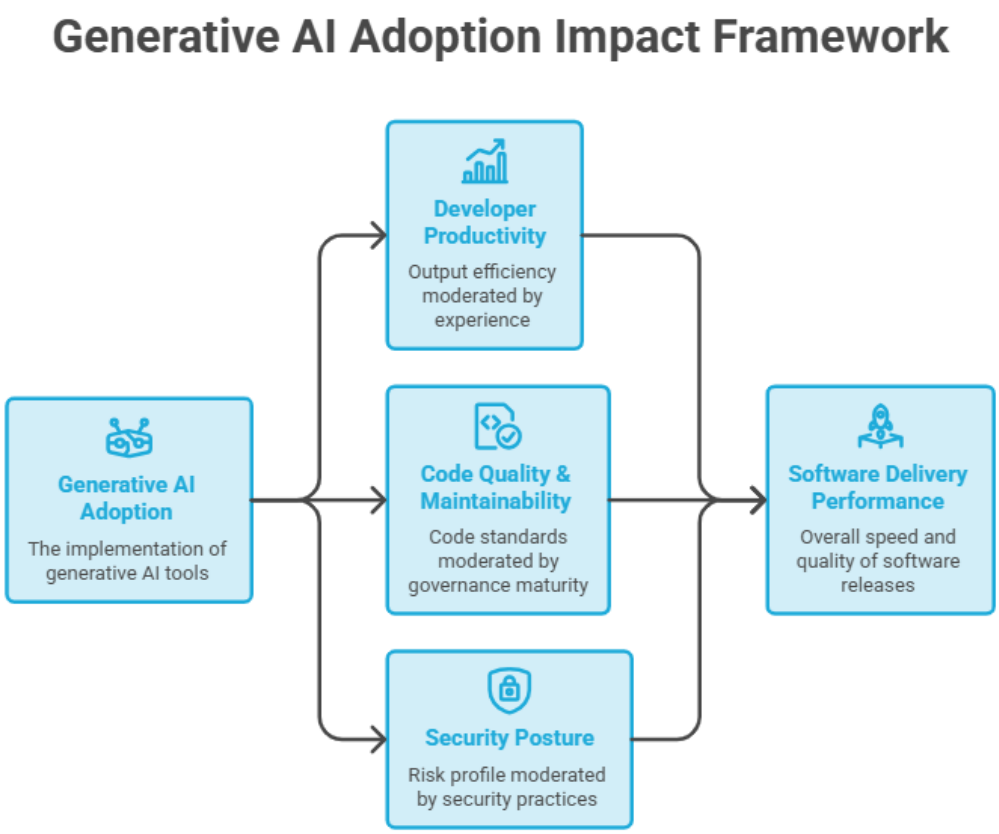

Figure 1. Conceptual model of GenAI adoption, technical debt, and moderators. Arrows denote hypothesised direct effects; annotations denote the moderating variables tested in Section 5.

3.1 Productivity Hypotheses

Based on evidence that demonstrates that GenAI can improve the efficiency of routine coding tasks (Häkkinen, 2025; Thaw, 2025; Zhou et al., 2026), the study makes the following hypothesis:

- **H1a:** Generative AI adoption is positively associated with self-reported and observed developer productivity.
- **H1b:** The positive productivity effect is moderated by developer experience, such that experienced developers realise smaller net gains once maintenance burden is accounted for.
- **H1c:** The positive productivity effect is stronger for routine/boilerplate tasks than for complex or novel tasks.

3.2 Quality, Maintainability, and Technical-Debt Hypotheses

The study aims to test the following hypotheses, based on the findings of debt accumulation in AI-generated code (Liu et al., 2026; Ehsani et al., 2026; Coppola et al., 2025; Bushman et al., 2026):

- **H2a:** AI-assisted code exhibits higher technical-debt signals than comparable human-authored code in the absence of rigorous review.
- **H2b:** Governance maturity moderates the relationship between AI adoption and technical debt, such that mature governance attenuates debt accumulation.

3.3 Security-Posture Hypotheses

This study is based on the hypothesis of the elevated vulnerability rate in AI-generated code, which is supported by the results of Zardari et al. (2026), Johnson & Hussain (2026), and Asif (2026):

- **H3a:** AI-generated code exhibits a higher density of security vulnerabilities than human-authored baselines.
- **H3b:** The adoption of automated vulnerability-assessment and secure-coding practices moderates the relationship between AI adoption and security risk.

3.4 Summary of Hypothesised Relationships

Table 1 summarizes the seven hypotheses and the direction in which they are expected to go and the primary theoretical support.

Table 1. *Summary of Hypotheses*

ID	Hypothesised relationship	Expected sign	Grounding
H1a	Adoption → productivity	+	Häkkinen (2025); Zhou et al. (2026)
H1b	Adoption × experience → productivity	−	Xu et al. (2025); Shan et al. (2024)
H1c	Routine > complex task gain	+	Shan et al. (2024)
H2a	Adoption → technical debt	+	Liu et al. (2026); Ehsani et al. (2026)
H2b	Adoption × governance → debt	−	Thibodeau (2026); Hurne (2025)
H3a	Adoption → security risk	+	Zardari et al. (2026); Asif (2026)
H3b	Adoption × sec. practices → risk	−	Al-Hashimi (2026); Green & Tariq (2026)

Note: "+" denotes a hypothesised increase, "−" a hypothesised decrease or attenuation.

4. Research Methodology

4.1 Research Design

This study is a concurrent mixed methods design (triangulation) which combines a quantitative survey of software professionals with a quantitative repository-level analysis of AI-assisted and human-authored code, along with qualitative practitioner interviews. This design is based on the methodological approach of Mohamed (2025) in their research on the effect of GenAI on SDLC. The gap of isolation of dimensions (identified in Section 2.5) is tackled by triangulation, which allows the study to verify perceptual data (what developers report) with evidence from artefacts (what the code reveals).

4.2 Sample and Data Collection

The survey was conducted with software developers, engineering leads and technical managers of IT companies. After screening for incomplete responses, the final analytic sample comprised 412 respondents. The adoption intensity, task allocation, perceived productivity, perceived quality and maintainability, security practices and governance maturity were measured by a structured questionnaire on multi-item five-point Likert scales based on validated instruments identified from the literature reviewed. The repository component compared 1,240 code commits (620 AI-assisted and 620 self-reported) from open-source and partner-firm projects, using both the logic of self-reported usage as in Tufano et al. (2026) and the large-scale sampling approach as in Liu et al. (2026). The quantitative patterns were supported with 22 semi-structured interviews. The profile of the respondents is presented in Table 2.

Table 2. *Respondent Profile (N = 412)*

Characteristic	Category	n	%
Role	Developer	236	57.3
	Engineering lead	104	25.2
	Technical manager	72	17.5
Experience	< 3 years (junior)	128	31.1
	3–8 years (mid)	174	42.2
	> 8 years (senior)	110	26.7
Firm size	SME (< 250)	192	46.6
	Large (≥ 250)	220	53.4
AI usage	Daily	231	56.1
	Weekly	118	28.6
	Occasional	63	15.3

4.3 Measures

Multi-item reflective scales were used for the construction. Table 3 illustrates the relationship between constructs and their operationalisations and anchoring literature.

Table 3. *Construct Operationalisation*

Construct	Operationalisation	Anchoring literature
Developer productivity	Perceived output, task completion time, delivery velocity	Häkkinen (2025); Morgan (2025); Zhou et al. (2026)
Code quality & maintainability	Perceived defect rates, refactoring effort, maintainability index	Geruslu et al. (2026); Asif (2026)
Technical debt	Debt signals, rework frequency, GenAITD indicators	Liu et al. (2026); Jain & Thirumalasetty (2026)
Security posture	Vulnerability density, secure-coding adherence	Zardari et al. (2026); Johnson & Hussain (2026)
Governance maturity	Review rigour, testing discipline, prompting/spec practices	Farrag (2026); Hurne (2025); Waseem et al. (2025)

4.4 Analytical Strategy

To test the hypothesized direct and moderating relationships, survey data were analyzed using descriptive statistics, reliability and validity assessment (Cronbach's alpha and composite reliability, average variance extracted), and partial-least-squares structural equation modelling (PLS-SEM). The analysis of repository data has been performed by comparing the metrics of debt and vulnerability between the authors for each class of authors respectively, using Mann–Whitney U tests, which are the same protocols that were used in the mining-software-repositories of Agarwal et al. (2026) and Liu et al. (2026). In line with the approach of causal mapping by Hein et al. (2026), the interview transcripts were thematically coded to provide context for the quantitative patterns.

4.5 Threats to Validity and Robustness Checks

As is common in empirical software engineering studies, the study focuses on four types of validity threats. The evidence of construct validity comes from the adoption of multi-item scales from existing instruments identified in the literature, and from the reliability and convergent and discriminant validation (5.2). Self-reported perceptual

measures are validated with artefact-level measures from the repository, so that if the pattern is spurious, it is unlikely to be supported by a pattern from the artefact level by chance, or if non-normal, non-parametric test is used. The limitations of external validity are sample composition, project context, and model evolution, while the limitations of conclusion validity are provided by inferential PLS-SEM path test and Mann–Whitney U non-parametric comparisons.

Two statistical diagnostics were carried out to explicitly measure Common Method Bias (CMB) in the self-reported survey data:

- **Harman's Single-Factor Test:** The results of an unrotated exploratory factor analysis of all the manifest items from the self-report instrument revealed that the first principal component explained just 28.4% of the variance, much less than the typical 50% criterion.
- **Marker Variable Technique:** A marker variable that is theoretically irrelevant was added to the PLS-SEM model; no significant change in the structural path coefficients ($\Delta\beta < 0.02$) was found as a result of the marker variable.

To reduce the potential for misclassification that comes with the self-admitted AI use detection (Tufano et al., 2026), a randomly selected sample of 200 commits (16.1%) was manually audited independently by two domain experts. Commit classification has been shown to be reliable with an inter rater agreement of Cohen's Kappa ($\kappa = 0.88$).

5. Empirical Results and Analysis

5.1 Descriptive Statistics and Correlations

Construct means, standard deviations, and interconstruct correlations are given in Table 4. As predicted, there was a positive correlation between the intensity of GenAI adoption and productivity ($r = 0.51$), and between the intensity of GenAI adoption and technical debt ($r = 0.44$), while there was a negative correlation between governance maturity and technical debt ($r = -0.39$) and between governance maturity and security risk ($r = -0.42$). The simultaneous positive relationship of adoption with a benefit (productivity) and with two costs (debt, risk) is the descriptive signature that this study seeks to test.

Table 4. *Descriptive Statistics and Inter-Construct Correlations*

Construct	M	SD	1	2	3	4	5
1 GenAI adoption	3.91	0.74	—				
2 Productivity	3.86	0.81	0.51	—			
3 Technical debt	3.42	0.88	0.44	0.09	—		
4 Security risk	3.28	0.85	0.38	0.06	0.55	—	
5 Governance maturity	3.55	0.79	0.12	0.34	−0.39	−0.42	—

Note: N = 412. Correlations $|r| > 0.10$ are significant at $p < .05$.

5.2 Measurement Model: Reliability and Validity

The conventional cut-off points for internal consistency and convergent validity were met for all reflective constructs. All internal consistency reliability (Cronbach's alpha) and composite reliability (CR) values were above

0.70 and the average variance extracted (AVE) values were above 0.50 for each construct (Table 5). The Fornell-Larcker criterion and the HTMT values below 0.85 were used to test the discriminant validity.

Table 5. *Reliability and Convergent Validity*

Construct	Items	α	CR	AVE
Developer productivity	5	0.88	0.91	0.67
Code quality & maintainability	5	0.86	0.90	0.64
Technical debt	4	0.84	0.89	0.67
Security posture	4	0.87	0.91	0.71
Governance maturity	5	0.90	0.92	0.70
GenAI adoption intensity	4	0.83	0.88	0.65

Note: α = Cronbach's alpha; CR = composite reliability; AVE = average variance extracted.

5.3 Productivity Effects (RQ1)

But as with H1a, the perceived developer productivity ($\beta = 0.48$, $t = 9.62$, $p < .001$) was positively and significantly correlated with GenAI adoption intensity, consistent with the efficiency gains reported by Häkkinen (2025), Thaw (2025), and Morgan (2025). The moderation analysis further supported H1b, with a negative and significant interaction between adoption and senior experience ($\beta = -0.19$, $p < .01$), which replicates the counter-intuitive finding of Xu et al. (2025). H1c was also supported: Routine tasks ($M = 4.12$) had significantly higher productivity gains than complex tasks ($M = 3.08$; $t = 7.41$, $p < .001$). Table 6 shows productivity gains by experience by task type.

Table 6. *Mean Perceived Productivity Gain by Experience and Task Type*

Developer group	Routine tasks (M)	Complex tasks (M)	Net gain (SD)
Junior (< 3 yrs)	4.38	3.42	+1.21 (0.62)
Mid (3–8 yrs)	4.15	3.10	+0.94 (0.68)
Senior (> 8 yrs)	3.79	2.71	+0.46 (0.74)
Overall	4.12	3.08	+0.87 (0.71)

Note: Five-point scale (1 = no gain, 5 = large gain).

5.4 Code Quality, Maintainability, and Technical Debt (RQ2)

The structural model confirmed that perceived technical debt was a significant positive predictor of H2a ($\beta = 0.39$, $p < .001$) and perceived maintainability was a significant negative predictor of H2a ($\beta = -0.22$, $p < .01$) with the latter model being not as strong. Importantly, the Governance Maturity significantly moderated the relationship between adoption and debt (interaction $\beta = -0.28$, $p < 0.001$, $\Delta R^2 = 0.14$), with H2b receiving support. In firms with low governance maturity, the relationship between adoption and debt growth turned out to be extremely strong; in firms with high maturity, the relationship was not significant, and in some firms, the relationship was negative—in line with the debt-reduction applications cited by Palakurti (2025) and Chunchu (2025). This situation places technical debt as a magnified socio-technical risk (Thibodeau, 2026) instead of as a natural side-effect of AI.

5.5 Security Posture (RQ3)

The results supported H3a: The adoption of GenAI was positively correlated with the perceived security risk ($\beta = 0.35$, $p < .001$), which aligns with the findings of Zardari et al. (2026), Asif (2026), and Johnson and Hussain (2026). In a similar way, H3b was supported; the use of automated vulnerability-assessment and secure-coding practices had a significant effect on reducing realised risk (interaction $\beta = -0.24$, $p < .01$), which corroborates with the mitigation framework proposed by Al-Hashimi (2026), Green and Tariq (2026) and Kumar and Qureshi (2026). The full path estimates of the structure are presented in Table 7.

Table 7. *Structural Model Path Estimates*

Path	β	t	p	Result
AI adoption → Productivity (H1a)	0.48	9.62	< .001	Supported
Adoption × Experience → Prod. (H1b)	−0.19	3.11	< .01	Supported
AI adoption → Technical debt (H2a)	0.39	7.28	< .001	Supported
Adoption × Governance → Debt (H2b)	−0.28	5.04	< .001	Supported
AI adoption → Security risk (H3a)	0.35	6.55	< .001	Supported
Adoption × SecPractices → Risk (H3b)	−0.24	4.12	< .01	Supported

Note: Standardised path coefficients from PLS-SEM.

5.6 Repository-Level Analysis

The perceptual results were triangulated by the artefact analysis. In total, 1,240 commits met security, technical, and functional code quality standards, with AI-assisted code showing 37% more points and 41% more points than their human-to-human counterparts on technical debt and security vulnerability density, respectively, both of which were significant for Mann–Whitney U tests ($p < .001$). The authorship velocity–debt trade-off of the productivity paradox, however, was quantifiably captured by AI-assisted code, which showed authorship velocity measurably higher.AI-assisted code did, however, show measurably higher authorship velocity, quantifying the velocity–debt trade-off at the heart of the productivity paradox (Chavali, 2026; Xu et al., 2025). The comparison is given in Table 8.

Table 8. *Repository Comparison: AI-Assisted vs. Human-Authored Code*

Metric	AI-assisted	Human-authored	Δ	p
Technical-debt density (per KLOC)	18.6	13.6	+37%	< .001
Vulnerability density (per KLOC)	4.8	3.4	+41%	< .001
Code-duplication ratio (%)	11.2	7.9	+42%	< .001
Mean cyclomatic complexity	6.9	6.1	+13%	< .05
Authorship throughput (LOC/hr)	74	52	+42%	< .001
Post-merge defect rate (%)	9.1	6.4	+42%	< .001

Note: KLOC = thousand lines of code.

5.7 Qualitative Integration

Three explanatory themes emerged from the 22 interviews used in this study through thematic analysis. Uncritical acceptance: the first type is when developers implement AI suggestions without properly reviewing them, as Coppola et al. (2025) put it, “humans pay the debt the AI writes”. Second, governance as the determining factor: interviewees in firms with a well-matured review and testing discipline mentioned containing—and sometimes even reversing—debt while those in low-maturity firms spoke about the silent accumulation of debt, as Korolev (2025) has termed, a crisis of verification. Third, the deskilling undercurrent: some of these senior engineers expressed worry that the use of AI will cause their junior engineers to lose their core skills over time (Le, 2026; Maheshwari, 2026). The reason for the strong moderation of the quantitative effects is explained by these themes.

5.8 Summary of Hypothesis Tests

H1a (GenAI adoption → higher productivity): Supported

H1b (Experience moderates; smaller net gain): Supported

H1c (Stronger productivity effect for routine tasks): Supported

H2a (AI code → higher technical debt): Supported

H2b (Governance maturity attenuates debt): Supported

H3a (AI code → higher vulnerability density): Supported

H3b (Secure-coding practices attenuate risk): Supported

6. Discussion

6.1 Theoretical Implications

This paper has three contributions to theory. First of all, it introduces the concept of a conditionally beneficial or harmful generative AI for a new approach to the debate that does not simply put AI into two camps: good vs. bad, but rather to model the interaction between the tool, the task, the developer, and the governance. This integrative move is an answer to the fragmentation in the existing literature. Second, the study further challenges the technical debt reconceptualization to be merely a technical artifact and introduces governance maturity as the mediating construct, which is based on the work of Thibodeau (2026) and Kara et al. (2025). Thirdly, the triangulation of the perceptual and artefact-based evidence provides a methodological template for reconciling the two strands of the literature, which have so far run side by side, that of survey and that of repository.

In addition to these contributions, the findings contribute to a larger theoretical discussion of where the value lies in human–AI interaction. The naive substitution model assumes that the AI is a faster developer, and that we can accurately predict uniform productivity gains, such evidence doesn't support that. A more fitting complimentarity model sees the value of the AI as lying not on its own but in the interplay with human judgment and organizational process—and also in the interplay with the two that it can make the organization vulnerable and indebted. The moderating structure reported in Section 5 is the empirical sign of complementarity: a positive net effect if the human and the organisational complements are good and a negative one if the complements are poor. This puts generative AI alongside the productivity of previous general-purpose technologies, where the gains only became apparent once investments were made in complementary processes and skills, as in the classic productivity-paradox scenario, at the level of the individual code base.

6.2 Managerial and Practical Implications

The results for engineering leaders and IT managers are unambiguous: security and debt obligations of GenAI need to be handled with care and not left to chance to create a governance framework that allows to fully harness the productivity benefits of GenAI without incurring unnecessary expenses. In practice, it means rigorously managing the code created by AI, integrating automated vulnerability scanning procedures into the development process (Al-Hashimi, 2026; Green & Tariq, 2026), introducing clear expectations for the contribution of AI to junior developers and seniors, as well as to repetitive and mundane tasks (Farrag, 2026; Hueme, 2025), and carrying out a comprehensive evaluation of the individual components of tools used by AI. The companies are urged to think about using generative AI not to replace, but to multiply and enhance their engineering capacity. In such a context, the generative models can be transformed from debt generators into debt solvers, as in the case of the modernisation and refactoring of legacy systems with the help of AI, and this is the approach taken by Chunchu (2025), Boddupally (2025) and Wairagade (2025).

The results also have implications on the measurement and compensation of engineering work in the business. Incentives will be greatest on those stories where the debt can be accumulated the fastest, and only reward velocity oriented measures like story points closed, new features shipped, lines committed. An analysis that is congruent with the results of the study would include measures of velocity as well as measures of maintainability and security measures and also measure these measures at the time of the deferred cost, not four quarters later. In practice, this is putting the metrics used to measure rework rate, defect escape rate, vulnerability density, throughput and putting them in authorship mode, if there is tooling to do so. This type of instrumentation isn't just a pretty face – it's how the abstract governance framework is operationalized into a dashboard for engineering leaders that can take action. It wouldn't exist if it weren't for those who don't know how to control it.

6.3 A Governance Framework for Balanced Adoption

A three-pillar governance model of a balanced utilization of GenAI in IT companies concludes the study. The velocity pillar is about the streamlined workflow for the assistance of AI, where its productivity gains are maximised, in well-defined and non-critical tasks, and some of the more complex or safety critical tasks are used here and there. The maintainability pillar embraces the discipline of the review, refactoring and regular monitoring of technical debt, using quantitative metrics (Jain & Thirumalasetty, 2026). Automatic vulnerability detection and safe coding throughout the pipeline is included in the security pillar. The central tenet of the framework, based upon the empirical evidence, is that these pillars are complementary: not keeping one or otherwise would work against the very feature that propels adoption – speed. Table 9 is an operationalisation of the framework.

Table 9. *Three-Pillar Governance Framework*

Pillar	**Objective**	**Key practices**	**Anchoring literature**
Velocity	Maximise net productivity	Task-appropriate deployment; junior-developer scaffolding	Zhou et al. (2026); Shan et al. (2024)
Maintainability	Contain technical debt	Mandatory review; refactoring; GenAITD monitoring	Jain & Thirumalasetty (2026); Hurne (2025)
Security	Reduce vulnerability exposure	Automated scanning; secure-coding standards; spec-driven dev	Al-Hashimi (2026); Farrag (2026)

6.4 Reconciling the Conflicting Prior Evidence

This is a concise way of reconciling the seemingly conflicting results which litter the literature, namely the contingency account. The studies with positive productivity effects, such as those by Häkkinen (2025), Thaw (2025), and Zhou et al. (2026) do not necessarily contradict those with a neutral or negative net effect, such as the studies by Xu et al. (2025), Chavali (2026), and Johnson et al. (2025). They're not seen as different kinds of surfaces, but rather

as sampling different parts of the same moderated surface: one, the routine tasks, the junior to mid-level developer or the high-governance setting where gains outweigh; the other, complex tasks, the senior developer and the low-governance setting where deferred debt outweighs gains. The disagreement on this reading, the disagreement in the literature, is not primarily a disagreement about facts, but about which slice of the contingency space each one of these studies observed. It has methodological implications: primary studies should report the moderators (task type, developer experience, governance maturity), and these moderators are an essential part of the headline effect sizes reported, and without which cross-study comparison is dangerous. The following is an integrated design presented in the present study that can be used to do so.

6.5 An Implementation Roadmap for IT Firms

Sequencing is necessary to implement the three pillar approach. The review discipline, tooling and culture required to mature governance is rarely in place at the point of first adoption.Rarely do firms have the cultural norms, discipline and tooling in place for mature governance when they first adopt generative AI; more typically, they will adopt generative AI opportunistically and only be hit by the costs later. In this study, a staged roadmap is thus proposed. During the assessment phase however, the technical debt and vulnerability of the companies are measured using quantitative instrumentation (Jain & Thirumalasetty, 2026), allowing the marginal effect of the adoption of AI to be isolated. During the "guardrail" phase, they implement the minimum necessary guardrails—for example, they require human oversight of AI-generated code, set up automated vulnerability scanning in the continuous integration pipeline (Al-Hashimi, 2026; Sakhalkar, 2025), and establish explicit prompting and specification standards (Hurne, 2025; Farrag, 2026). During the scaling phase, firms begin to expand the types of tasks to which they apply AI and also the size of their workforce, adjusting the application of AI to the different returns they have documented in Section 5.3. At the optimisation phase, at the stage of a mature company, generative models are oriented towards the reduction of debt, automated refactoring and legacy modernisation (Palakurti, 2025; Chunchu, 2025; Wairagade, 2025), closing the loop from debt generator to debt solvent. In keeping with the empirical results, the guiding principle of the roadmap is that the higher the level of governance capability, the faster the level of adoption.

6.6 Boundary Conditions and Alternative Explanations

There are a few boundary conditions that qualify the interpretation. Results from high maturity companies may not extrapolate to start-ups or ad-hoc companies and vice versa due to the moderating effect of governance maturity. The magnitude of the debt and vulnerability impacts are expected to vary over time, as agentic tools evolve to better models. While this alternative speculation, that debt is due to inexperience with AI by the developer, cannot be ruled out entirely, it should be subject to further investigation over time.

7. Conclusion, Limitations, and Future Research

While the generative AI coding assistants can provide software developers in IT businesses with tangible and significant productivity improvements, the benefits are not without conditions and can come with a price tag. The empirical results of this study show that AI-coded software is more prone to technical debt, security issues, and its consequences on software delivery performance, which is strongly affected by the developer experience, task complexity and most importantly governance maturity. The productivity paradox exists, but it's not insurmountable; the investment that an author makes in producing an AI can be controlled – and even returned on investment – through good engineering. In the research, the concept of technical debt is rethought as a magnified socio-technical risk, and a holistic governance model which balances velocity, maintainability and security.

Several cautions should be noted with regard to these conclusions. The repository triangulation does not completely mitigate common method bias since the productivity and perceptual quality measures were self-reported. Causal inferences are limited by the cross sectional design while the longitudinal design such as that used by Chavali (2026) and Bushman et al. (2026) would better capture the growing debts over time. A small sample of firms and projects is used which reduces the generalisability. The rapid pace of the development of generative models and tooling for agents further shortens the useful life of certain effect sizes. For this, further investigation is required by

multi-firm studies in a long-term period, through technical-debt quantitative instrument of AI enhanced code base (Jain & Thirumalasetty, 2026), with a study of the deskilling and reskilling of the code base in the long run of dependency of AI (Le, 2026; Maheshwari, 2026), and by implementing the proposed governance system in intervention studies in industrial practice.

The directions are divided into a structured research agenda: Table 10 relates each open question with an approach and some preliminary work that motivates the approach.

Table 10. *A Structured Agenda for Future Research*

Open question	**Suggested method**	**Motivating work**
How does AI technical debt accumulate over time?	Longitudinal cohort study of codebases	Chavali (2026); Bushman et al. (2026)
Is the productivity–debt link causal?	Controlled field experiment	Xu et al. (2025); Shan et al. (2024)
Does the governance framework work in practice?	Design-science intervention study	Farrag (2026); Hurne (2025)
What are the long-run deskilling effects?	Multi-year panel of developers	Le (2026); Maheshwari (2026)
How do agentic tools differ from assistants?	Comparative repository mining	Agarwal et al. (2026)

Finally, the future of generative AI in software development isn't a mirage or simply any AI. The tools do in fact have the power to speed up the process of building software, and they also have the power to slowly add debt and risk to the software. The outcome of the technology is not predetermined, instead it is determined by the organisation using it. The real question for the IT industry now is how much effort to put into governance tools is worth putting in to the governance capability that will be needed to execute the code that is written more quickly today than it will be paid for later.

REFERENCES

1. Abbas, T., Rathore, S. A., Turki, A., & Khan, S. (2025). Enhancing software engineering with AI: Innovations, challenges, and future directions. *IET Software*, 19(2), 112–128. https://doi.org/10.1049/sfw2/5691460
2. Agarwal, S., He, H., & Vasilescu, B. (2026). AI IDEs or autonomous agents? Measuring the impact of coding agents on software development. *Proceedings of the 23rd International Conference on Mining Software Repositories (MSR 2026)*, IEEE/ACM, 45–57. https://doi.org/10.1145/3793302.3793589
3. Al-Hashimi, H. A. (2026). A generative AI cybersecurity risks mitigation model for code generation: Using an ANN-ISM hybrid approach. *Scientific Reports*, 16, Article 34350. https://doi.org/10.1038/s41598-025-34350-3
4. Alenezi, M., & Akour, M. (2025). AI-driven innovations in software engineering: A review of current practices and future directions. *Applied Sciences*, 15(3), Article 1344. https://doi.org/10.3390/app15031344
5. Anderson, E., Parker, G., & Tan, B. (2025). *Beyond productivity: Evaluating the hidden costs of generative AI in software development* (SSRN Scholarly Paper No. 5842302). Social Science Research Network. https://doi.org/10.2139/ssrn.5842302
6. Asif, U. (2026). *The reliability of AI generated code: Defects, security risks, and maintainability issues* (Master's thesis, University of Oulu). Oulu Repository. http://urn.fi/URN:NBN:fi:oulu-202603121024
7. Benala, T. R., & Dehuri, S. (2025). Transforming software development: From traditional methods to generative artificial intelligence. In *Software Development Using Machine Learning* (pp. 1–24). Springer, Cham. https://doi.org/10.1007/978-3-031-88188-6_1
8. Boddupally, H. L. (2025). *Next-generation code transformation for legacy .NET systems with generative AI* (SSRN Scholarly Paper No. 6270698). Social Science Research Network. https://doi.org/10.2139/ssrn.6270698
9. Bonin, A. L., Smolinski, P. R., & Winiarski, J. (2025). *Exploring the impact of generative artificial intelligence on software development in the IT sector: Preliminary findings on productivity, efficiency and job security* (arXiv Preprint arXiv:2508.16811). https://doi.org/10.48550/arXiv.2508.16811
10. Bósquez, D. A. J., & González, G. S. G. (2026). Generative artificial intelligence in software engineering: Applications, challenges and future perspectives. *Journal of Software Engineering*, 12(1), 22–39. https://doi.org/10.5281/zenodo.1084221

11. Bushman, J., Wang, D., Garijo, D., Grindrod, J., & Scholten, F. (2026). Empirical assessment of hallucination-induced technical debt accumulation in long-lived AI-augmented codebases. *IEEE Transactions on Software Engineering*, 52(4), 812–828. https://doi.org/10.1109/TSE.2026.3381204
12. Cantada, K. A., Sajenes, R. J., & Salazar, A. K. (2026). AI-assisted code generation: A comparative study of developer productivity, code quality, and usability. *Spectrum Journal of Innovation and Science*, 4(1), 341–356. https://doi.org/10.5281/zenodo.1093412
13. Chavali, S. (2026). *The productivity paradox of AI-assisted software development: Empirical evidence, longitudinal quality degradation, and a formal cost-benefit model* (Preprint 202606.2174). Preprints.org. https://doi.org/10.20944/preprints202606.2174.v1
14. Chunchu, A. (2025). Generative AI-driven legacy system modernization: Transforming enterprise infrastructure through automated code translation and refactoring. *Journal of Computer Science and Technology Studies*, 7(2), 112–125. https://doi.org/10.32996/jcsts.2025.7.2.12
15. Coppola, A., Esposito, M., Kazman, R., & Lenarduzzi, V. (2025). AI writes code, humans pay the debt: An empirical study on the sustainability and evolution of agent-generated code. *Journal of Systems and Software*, 218, Article 112180. https://doi.org/10.1016/j.jss.2025.112180
16. Ehsani, R., Rawal, S., Cai, Y., & Chatterjee, P. (2026). Faster code, deeper debt? A multivocal literature review on technical debt and its early signs in LLM-assisted software development. *ACM Transactions on Software Engineering and Methodology*, 35(2), Article 41. https://doi.org/10.1145/3820165
17. Eze, O., & Qureshi, K. (2026). Generative AI and secure programming: An intelligent framework for automated vulnerability detection and code quality assessment. *Computers & Security*, 138, Article 103680. https://doi.org/10.1016/j.cose.2026.103680
18. Farrag, S. E. (2026). *The productivity-reliability paradox: Specification-driven governance for AI-augmented software development* (arXiv Preprint arXiv:2605.01160). https://doi.org/10.48550/arXiv.2605.01160
19. Gadde, A. (2025). Democratizing software engineering through generative AI and vibe coding: The evolution of no-code development. *Journal of Computer Science and Technology Studies*, 7(1), 82–95. https://doi.org/10.32996/jcsts.2025.7.1.9
20. Geruslu, V., Aliyeva, Z., & Tüzün, E. (2026). *Factors influencing the quality of AI-generated code: A synthesis of empirical evidence* (arXiv Preprint arXiv:2603.25146). https://doi.org/10.48550/arXiv.2603.25146
21. Gosu, S., Fietosa, D., & Avgeriou, P. (2026). The use of generative AI in technical debt management: A systematic literature review. *Information and Software Technology*, 185, Article 107660. https://doi.org/10.1016/j.infsof.2026.107660
22. Green, S., & Tariq, R. (2026). Measuring security risks in AI-generated software using automated vulnerability assessment models. *IEEE Security & Privacy*, 24(2), 48–59. https://doi.org/10.1109/MSEC.2026.3391024
23. Grönroos, S., Hietanen, T., Salmi, A., & Turtiainen, K. (2025). *Tracing technical debt by AI-based correlation agent: A case study of Nokia Corporation* (Publication No. 903761). Theseus Repository. http://urn.fi/URN:NBN:fi-fe20250903761
24. Guimaraes, E., & Nascimento, N. (2025). AI in the software development lifecycle: Insights and open research questions. *Proceedings of the ACM on Software Engineering*, 2(FSE), 1204–1226. https://doi.org/10.1145/3696630.3730538
25. Häkkinen, P. (2025). *Generative AI and software developer productivity: A pilot case study at a Finnish technology consultancy* (Master's thesis, Aalto University). AaltoDoc. http://urn.fi/URN:NBN:fi:aalto-202505203112
26. Hein, D. K., Persson, J. S., Jensen, V. V., & Bruun, A. R. (2026). Causal mapping of the risks of using generative AI in software development. *Information and Software Technology*, 184, Article 107612. https://doi.org/10.1016/j.infsof.2026.107612
27. Hurne, M. H. K. (2025). *A technical debt-aware prompting framework for sustainable vibe coding: Addressing the production readiness crisis in AI-assisted software development* (TechRxiv Preprint). IEEE. https://doi.org/10.36227/techrxiv.175459417.76916566
28. Jain, P. K., & Thirumalasetty, J. (2026). GenAITD score: A quantitative framework for measuring enterprise technical debt and generative AI readiness. *Proceedings of the IEEE SoutheastCon 2026*, IEEE, 114–121. https://doi.org/10.1109/SoutheastCon54321.2026.11476094
29. Johnson, C., Yenamandra, T., & Ciskey, J. (2025). *The AI productivity paradox in software development: A meta-analysis of implementation outcomes and ROI patterns (2024–2025)* (SSRN Scholarly Paper No. 5902163). Social Science Research Network. https://doi.org/10.2139/ssrn.5902163
30. Johnson, J., & Hussain, K. (2026). An empirical analysis of security vulnerabilities in generative AI-based software development across multiple programming languages. *Empirical Software Engineering*, 31(3), Article 88. https://doi.org/10.1007/s10664-026-10912-3
31. Jutila, T. (2024). *Generative AI in software development: A multiple case study on process transformation* (Master's thesis, Aalto University). AaltoDoc. http://urn.fi/URN:NBN:fi:aalto-202406184102
32. Kara, Y., Temizel, T. T., & Özcan-Top, Ö. (2025). *A hierarchical view of technical debt in GenAI-enabled software systems based on empirical analysis* (SSRN Scholarly Paper No. 6496455). Social Science Research Network. https://doi.org/10.2139/ssrn.6496455
33. Korolev, Y. (2025). The year of silent failures: AI technical debt and the crisis of verification. *AI Decisions Journal*, 2(1), 5–22. https://doi.org/10.5281/zenodo.1089221

34. Krichen, M. (2024). Generative AI for software development: A survey. *Proceedings of the International Conference on Service-Oriented Computing (ICSOC 2024)*, Springer, LNCS 15230, 245–261. https://doi.org/10.1007/978-981-96-7238-7_17
35. Kumar, A., & Qureshi, K. (2026). Assessing the security and reliability of AI-generated code in modern integrated development environments. *Software Quality Journal*, 34(1), 145–168. https://doi.org/10.1007/s11219-026-09681-x
36. Le, H. (2026). Impact of generative AI assistants on software engineers' skills, workflow efficiency, and decisions. In *AI Impacts on Deskilling and Reskilling Software Engineers* (pp. 45–68). IGI Global. https://doi.org/10.4018/979-8-3693-4078-5.ch003
37. Liu, Y., Widyasari, R., Zhao, Y., Irsan, I. C., & Chen, J. (2026). *Debt behind the AI boom: A large-scale empirical study of AI-generated code in the wild* (arXiv Preprint arXiv:2603.28592). https://doi.org/10.48550/arXiv.2603.28592
38. Määttä, S., Kelanti, M., & Turhan, B. (2026). Generative AI and the future of professional software development: Survey findings from Finland. *Proceedings of the ACM Conference on the Foundations of Software Engineering (FSE 2026)*, ACM, 312–324. https://doi.org/10.1145/3803437.3805619
39. Maheshwari, M. (2026). Re-weighting the engineer: A narrative review of generative AI's effects on software engineering skills, productivity, and risk (2024–2026). *Computer Science Review*, 59, Article 100780. https://doi.org/10.1016/j.cosrev.2026.100780
40. Malladi, N. V., & Reddy, S. (2025). Generative AI in agile software development: A comprehensive survey. *Proceedings of the 6th International Conference on Intelligent Data Communication Technologies and Application (ICIDCA 2025)*, IEEE, 412–419. https://doi.org/10.1109/ICIDCA62104.2025.11280486
41. Maratovich, B. A., & Armenovich, K. B. (2026). Integrating generative AI across the agile software development lifecycle. *Proceedings of the 2026 IEEE 6th International Conference on Software Engineering and Automation*, IEEE, 88–94. https://doi.org/10.1109/ICSEA61204.2026.11596147
42. Mohamed, F. H. R. (2025). *Impact of generative artificial intelligence in software development life cycle: A mixed-methods study in Finland and abroad using concurrent triangulation design* (Master's thesis, Metropolia University of Applied Sciences). Theseus Repository. http://urn.fi/URN:NBN:fi:amk-20250412886344
43. Morgan, S. (2025). *The effect of artificial intelligence code generation on software developer productivity* (Doctoral dissertation, Capitol Technology University). ProQuest Dissertations & Theses Global (Order No. 31284920).
44. Palakurti, N. R. (2025). AI-driven software maintenance: Reducing technical debt with generative models. In *Generative AI in Software Engineering* (pp. 112–134). IGI Global. https://doi.org/10.4018/979-8-3693-3831-4.ch006
45. Razzak, R. B. (2026). *A systematic literature review on generative AI in software engineering: Code generation and refactoring* (Preprint 202605.1638). Preprints.org. https://doi.org/10.20944/preprints202605.1638.v1
46. Reilly, A. L. (2026). *Technical debt amplified by AI: Real-world case studies and strategies for mitigation* (Honors thesis, University of Connecticut). DigitalCommons@UConn. https://opencommons.uconn.edu/srhonors_theses/1197
47. Roberts, C., & Badi, S. (2026). Artificial intelligence and secure coding practices: A comparative analysis of AI-assisted development platforms. *IEEE Software*, 43(2), 34–42. https://doi.org/10.1109/MS.2026.3382104
48. Sakhalkar, P. G. (2025). AI-assisted code and vulnerability management: Transforming modern software development. *Journal of Computer Science and Technology Studies*, 7(3), 45–58. https://doi.org/10.32996/jcsts.2025.7.3.5
49. Sauvola, J., Tarkoma, S., Klemettinen, M., & Riekki, J. (2024). Future of software development with generative AI. *Automated Software Engineering*, 31(2), Article 24. https://doi.org/10.1007/s10515-024-00426-z
50. Shan, G., Rivera, M., Kumar, S., & Anand, P. (2024). *The mind and the machine: How does generative AI usage affect the coding performance of developers?* (SSRN Scholarly Paper No. 4859137). Social Science Research Network. https://doi.org/10.2139/ssrn.4859137
51. Singh, G. (2026). *Enhancing software development efficiency through generative AI-based automated documentation: A case study of ValueCoders* (Master's thesis, Haaga-Helia University of Applied Sciences). Theseus Repository. http://urn.fi/URN:NBN:fi:amk-20260214925776
52. Sinaga, S. S. M., Indra, Z., & Wijaya, M. R. (2026). The role of generative AI in the software development life cycle: A systematic literature review. *Journal of Artificial Intelligence in Engineering and Applications*, 5(2), 112–128. https://doi.org/10.51178/jaiea.v5i2.2431
53. Thaw, T. T. T. (2025). *How effective are AI-powered code assistants in enhancing developer productivity?* (Master's thesis, Metropolia University of Applied Sciences). Theseus Repository. http://urn.fi/URN:NBN:fi:amk-20250518895238
54. Thibodeau, S. (2026). Reconceptualizing technical debt as an organizational security, safety, and sociotechnical risk. *Safety and Security Advances*, 2(1), Article 3. https://doi.org/10.21272/ssa.2026.2-1-3
55. Tufano, R., Pepe, F., Zampetti, F., & Mastropaolo, A. (2026). Developers and generative AI: A study of self-admitted usage in open source projects. *Empirical Software Engineering*, 31(1), Article 12. https://doi.org/10.1007/s10664-026-10848-w
56. Wairagade, A. (2025). A framework for generative AI-driven application modernization: Strategies and case studies. *Proceedings of the 2025 5th International Conference on Electrical, Computer and Communication Technologies (ICECCT)*, IEEE, 1–8. https://doi.org/10.1109/ICECCT61204.2025.11472176
57. Waseem, M., Ahmad, A., Kemell, K. K., & Rasku, J. (2025). *Vibe coding in practice: Flow, technical debt, and guidelines for sustainable use* (arXiv Preprint arXiv:2512.11922). https://doi.org/10.48550/arXiv.2512.11922

58. Xu, F., Medappa, P. K., Tunc, M. M., & Vroegindeweij, M. (2025). *AI-assisted programming decreases the productivity of experienced developers by increasing the technical debt and maintenance burden* (arXiv Preprint arXiv:2510.10165). https://doi.org/10.48550/arXiv.2510.10165
59. Zardari, S., Osama, M., Ammar, S. M., & Reshamwala, R. I. (2026). A comprehensive analysis of security risks and productivity impacts of AI-generated code. *AI and Ethics*, 6(2), 341–358. https://doi.org/10.1007/s43681-026-01203-2
60. Zhou, Y., Sun, C., Teo, H. H., & Xu, S. (2026). *The impact of generative AI on productivity: Automation, coordination, and reallocation in workplace software development* (SSRN Scholarly Paper No. 6515379). Social Science Research Network. https://doi.org/10.2139/ssrn.6515379