



Module: Advancing to Doctoral Research Stage (LCDB7012_LON)

**Evaluating the Instances of AI-Generated Code: A Governance Framework in
Understanding Socio-Technical Dimensions of Software Systems**

Submitted To:

Prof Amare Desta

Submitted By:

Anil Poudyal

Doctorate Candidate

2429341@student.uwtsd.ac.uk

Table of Contents

Abstract.....	1
Introduction.....	1
Theoretical and Practical Contributions.....	2
Business and Societal Problems	3
Objectives.....	3
Research Questions	4
Implications of the Paradigm Shift in Software Development.....	5
Visualising the New Paradigm.....	6
Objective Reality	7
Subjective Reality	7
Challenges in Implementing the New Paradigm	7
Tooling Limitations	7
Cultural Resistance	7
Quantifying the Subjective.....	7
Relevance of Kuhn’s Paradigm to the Research Context	8
Research Approach & Methodology	8
Research Approach.....	8
Phase 1: Quantitative Phase	9
Phase 2: Qualitative Phase	9
Phase 3: Synthesis and Framework Design	10
Challenges and Limitations	10
Technological Volatility and Construct Validity	10
Methodological Challenges and Reliability	10
Conclusion of Anticipated Limitations	11
Conclusion	11
Bibliography.....	13

Abstract

The rapid integration of generative AI coding tools has significantly accelerated software development, be it mobile apps, websites and other Software as a Service (SaaS) product. Yet it has introduced a critical hidden risk: AI Induced Technical Debt (AITD). The boosted velocity by these tools is often short-term and they frequently produce “illusion of correctness”. This refers to syntactically valid code that lacks deep logical context, thereby creating “epistemic debt”. This phenomenon shifts an unsuitable cognitive burden onto senior engineers who are tasked with reviewing machine-generated logic, threatening long-term software maintainability and accelerating industry burnout. This research challenges the dominant functionalist metrics of software engineering drawn from Thomas Kuhn’s theory of paradigm shifts and socio-technical systems. As the dominant paradigm ignores the hidden human cost and cognitive element which is one of the major factors in software development. Employing an explanatory sequential mixed methods design the proposed study will triangulate quantitative codebase metrics with qualitative insights into developer trust, review fatigue and cognitive burnout of senior engineers. Ultimately, this research aims to design and validate a pragmatic “Governance-as-Code” framework to recommend and guide senior engineers, engineering managers and even company directors to manage this debt. This framework will provide leaders with actionable, socio-technical metrics to systemically manage AITD.

Keywords: AI-Induced Technical Debt (AITD), Generative AI, Socio-Technical Systems (STS), Software Engineering Governance, Governance-as-Code, Epistemic Debt, Mixed-Methods Research.

Introduction

The paradigm of software engineering is undergoing a seismic shift. The tools like GitHub Copilot, Antigravity, Cursor and so on have become standard in developer’s toolchain (Peng *et al.*, 2023). They speed up delivery but introduce a growing risk: AI Induced technical debt (AITD) (Mujahid and Imran, 2026). There is a good difference between code generation and code authoring, when code is generated, it leads to

“illusion of correctness”. It may be syntactically valid yet fragile, inconsistent or difficult for teams to understand and maintain.

Studies show this pattern clearly. Software engineers often rely on AI-generated code as it boosts short-term productivity (Vaithilingam, Zhang and Glassman, 2022, pp. 1–7), but the behaviour of these models is unpredictable. Small prompt changes can produce different outputs, adding confusion and rework. Other research also shows that the productivity gained by coding faster is lost later when teams must fix or rewrite the code (Xu *et al.*, 2026). Other studies show that AI-assisted tools increase the volume of pull requests and commits from less experienced developers. This creates more review pressure on senior engineers reducing net productivity gains (Bogner, Verdecchia and Gerostathopoulos, 2021). Other works also highlight risk such as shallow understanding of the produced code (Becker *et al.*, 2022) and missed security issues leading to issues with project’s sustainability, maintainability and scalability.

Across these studies, the message is consistent: AI speeds up production but raises the long-term cost of maintaining, reviewing and securing software. Without a clear framework for managing this debt, the short-term benefits of AI-assisted development can undermine the long-term sustainability and resilience of software systems.

Theoretical and Practical Contributions

This study contributes to the Socio-Technical Systems (STS) theory by expanding the horizon of traditional definition of technical debt, with the introduction of AI in software engineering. This broadening is achieved by moving beyond the analysis of static metrics such as code complexity. It also encompasses the cognitive friction between non-deterministic AI agents and human reviewers. Practically, a “Governance-as-Code” is set to be delivered, it is designed to provide engineering teams with actionable and best practised industry standards to reduce AI-Induced Technical Debt. This in return helps not only to recued AITD but also monitors the workforce from cognitive burnout, addressing the highly complex ways developers interact with AI.

Practically, a "Governance-as-Code" framework will be delivered. This framework is designed to provide engineering leadership with actionable, socio-technical metrics to

monitor AITD and protect the workforce from cognitive burnout, addressing the highly complex ways developers interact with AI (Barke et al., 2023).

Business and Societal Problems

Without a strategic and systematic organisational governance, the unchecked adoption of AI-assisted development risks software sustainability and organisational resilience. And furthermore, cognitive and psychological burden is shifted onto senior engineers, this is due to added load of validating machine-hallucinated logic. This dynamic contributes to industry-wide burnout, compromised security and elevated operational risks (Becker *et al.*, 2022; Xu *et al.*, 2026). If this issue is unaddressed, software systems risk evolving beyond the capacity of human maintenance.

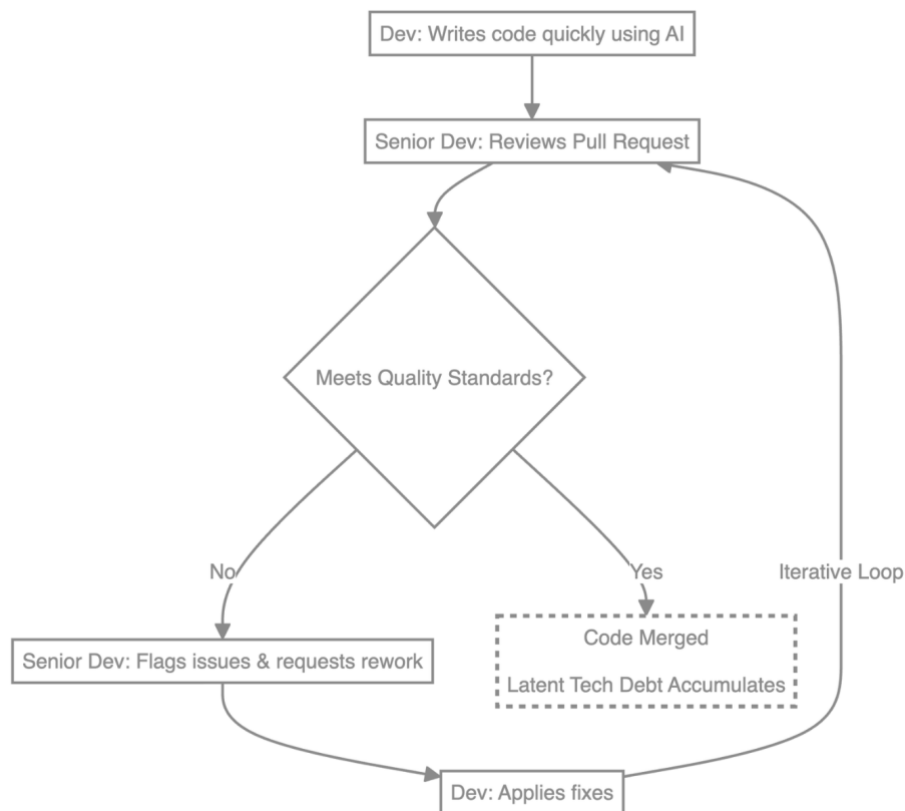


Figure 1 The AI-Induced Technical Debt (AITD) Accumulation Cycle

Objectives

- To identify and classify the structural and epistemic characteristics of AI-Induced Technical Debt (AITD) emerging within software development projects.

- To evaluate the impact of AI-generated code on software maintainability and the specific cognitive load experienced by human reviewers.
- To critically analyse the organisational practices and socio-technical team dynamics that mitigate or exacerbate the accumulation of AITD.
- To design a "Governance-as-Code" framework aimed at systematically managing AITD.

Research Questions

RQ1: What are the structural and epistemic characteristics of AI-Induced Technical Debt (AITD) within software development projects?

RQ2: How does the integration of AI-generated code impact long-term software maintainability and the cognitive load of human reviewers during the validation process?

RQ3: What organisational practices and socio-technical team dynamics serve to reduce the accumulation of AITD in active development environments?

Kuhn's Paradigm – The Kuhn Cycle in Software Engineering

The theory behind this research comes from the philosophy of science, especially the work of Thomas Kuhn. According to (Kuhn, 2012) that scientific and professional communities with a paradigm. According to him the progress is not a straight line where knowledge just builds up over time, but it follows a cycle. It starts with a normal science, where people work within the existing frameworks and over time the anomalies appear that do not fit, and this leads to a major shift called "scientific revolution" as a result we get a new paradigm.

This idea helps to explain what is happening in software engineering today, especially with the rise of generative AI. The theoretical foundation of this research is grounded in the philosophy of science.

Implications of the Paradigm Shift in Software Development

For the past two decades the software engineering landscape had seen many changes but operated under a strict dominant functionalist paradigm (Treude and Storey, 2025). Under this worldview, success is measured mathematically, objectively and deterministically. The industry operated on the established rule of test cases passed, sprint delivery and code compiles.

With the introduction of AI coding assistants, this paradigm initially appeared to be hyper-accelerated and validated. As explored in the study by (Peng *et al.*, 2023), industry reports suggested productivity increases of up to 55% for the routine tasks. The literature reinforced the functionalist belief that speed and output volume were the correct metrics for measuring software engineering success, entirely separating the objective code from the human author.

However, significant anomalies quickly emerged that the functionalist paradigm could not explain. As detailed by (Vaithilingam, Zhang and Glassman, 2022, pp. 1–7) AI models are efficient at generating code that perfectly satisfies functionalist rules; it compiles and passes automated tests perfectly but introduced severe logical flaws and vulnerabilities.

To support the above, research by (Peng *et al.*, 2023) found that developers achieved only a 19% productivity gain, falling drastically short of the expected 55%: a gap of 36%. The primary anomaly observed is the “illusion of correctness”. Developers commit code they do not fully comprehend, rapidly accumulating hidden knowledge gap. This breaks the fundamental assumption of the traditional paradigm: that the human engineers responsible for a system understand the logic comprising it.

There is a clear socio-technical fallout, because of the dominant functionalist paradigm that treats code generation as a purely mathematical transaction. An unsustainable cognitive load has been shifted onto senior engineers, as warned in the literature surrounding software maintainability, such as the arguments presented by (Becker *et al.*, 2022), the human cost of reviewing and validating machine-hallucinated logic is severe. The traditional paradigm is fundamentally broken because it relies on metrics

that show teams are highly productive, while ignoring the hidden reality of severe human burnout and degrading system maintainability.

To resolve this crisis, re-thinking is necessary, (Kuhn, 2012) originally argued that competing paradigms are mutually exclusive and cannot be reconciled. Under strict rules, a researcher must choose either a purely objective or purely subjective worldview. However, this rigid exclusivity fails in modern software systems because the machine's output cannot be isolated from the human maintaining it. Therefore, this research executes a "Pragmatic Pivot". As outlined in foundational methodology texts such as (Creswell and Plano Clark, 2018) , Pragmatism actively combines competing worldviews. It also argues that the valid knowledge required triangulation.

Visualising the New Paradigm

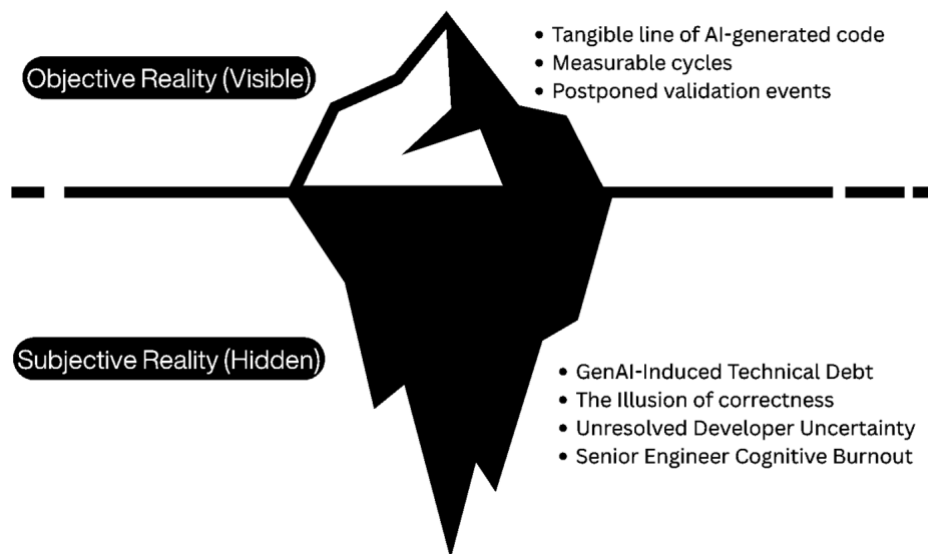


Figure 2 Iceberg Model

The iceberg model above illustrates the failure of the old paradigm and dictates the necessity of the new one:

Objective Reality

Top 10-15% of the iceberg represents the tangible lines of AI-generate code, measurable delivery cycles and postponed validation events. This is the only portion that told functionalist paradigm measures.

Subjective Reality

The bottom 85-90% represents the hidden cognitive load, illusion of correctness, unresolved developer uncertainty, and Generative AI Induced Technical Debt (GenAI-TD)

Challenges in Implementing the New Paradigm

Shifting from a purely functionalist approach to socio-technical paradigm presents profound practical challenges for the industry:

Tooling Limitations

Current tools like continuous integration/continuous deployment (CI/CD) pipelines are engineered solely to catch syntax error. They lack features to measure social aspect of engineering, like developer trust, code comprehension or cognitive load. And these tools are intentionally strict and integration of socio-technical aspects to such tools does not comprehend the idea to reduce technical debt, as this debt start to originate at much higher level.

Cultural Resistance

A significant change management is required to incorporate socio-technical maintainability over short-term coding speed in organisational culture. And often engineering management is deeply conditioned to measure success via velocity metrics.

Quantifying the Subjective

Interpreting subjective human experiences such as review fatigue, epistemic debt into actionable, measurable governance policies presents a complex methodological hurdle.

Relevance of Kuhn's Paradigm to the Research Context

For doctoral research Kuhn's framework provides a vital theoretical justification. With the interpretation of current industry landscape to Kuhn's "Crisis Stage", it has become evident that building more accurate models will not solve the underlying problem. The research must focus on frameworks. The rejection of Kuhnian "incommensurability" justifies the mixed method approach planned for this study. To understand the complete socio-technical system, quantitative repository mining (Normal Science) must be combined with qualitative human interviews.

Research Approach & Methodology

As established in the preceding theoretical analysis, investigating socio technical aspect of AI-Induced Technical Debt requires a departure from purely functionalist research models. To capture both code metrics and subjective cognitive load placed on engineering staff, a comprehensive and robust research design is required.

Research Approach

This research adopts a pragmatic research philosophy as it argues the most important determinant of the research philosophy is the research question itself. It allows for integration of multiple methods to fully understand a complex phenomenon (Saunders, Lewis and Thornhill, 2023). As AITD is fundamentally a socio-technical issue, the interaction of machine output and human comprehension, a single methodological approach is insufficient.

This study employs a Sequential Mixed-Method Research approach. It is increasingly recognised as essential in modern software engineering and management research. This gives depth to the research and helps to understand human elements of the software development (Storey *et al.*, 2024). The sequential design will first gather quantitative data to establish the breadth of the problem, and the qualitative data will give depth of the human experience.

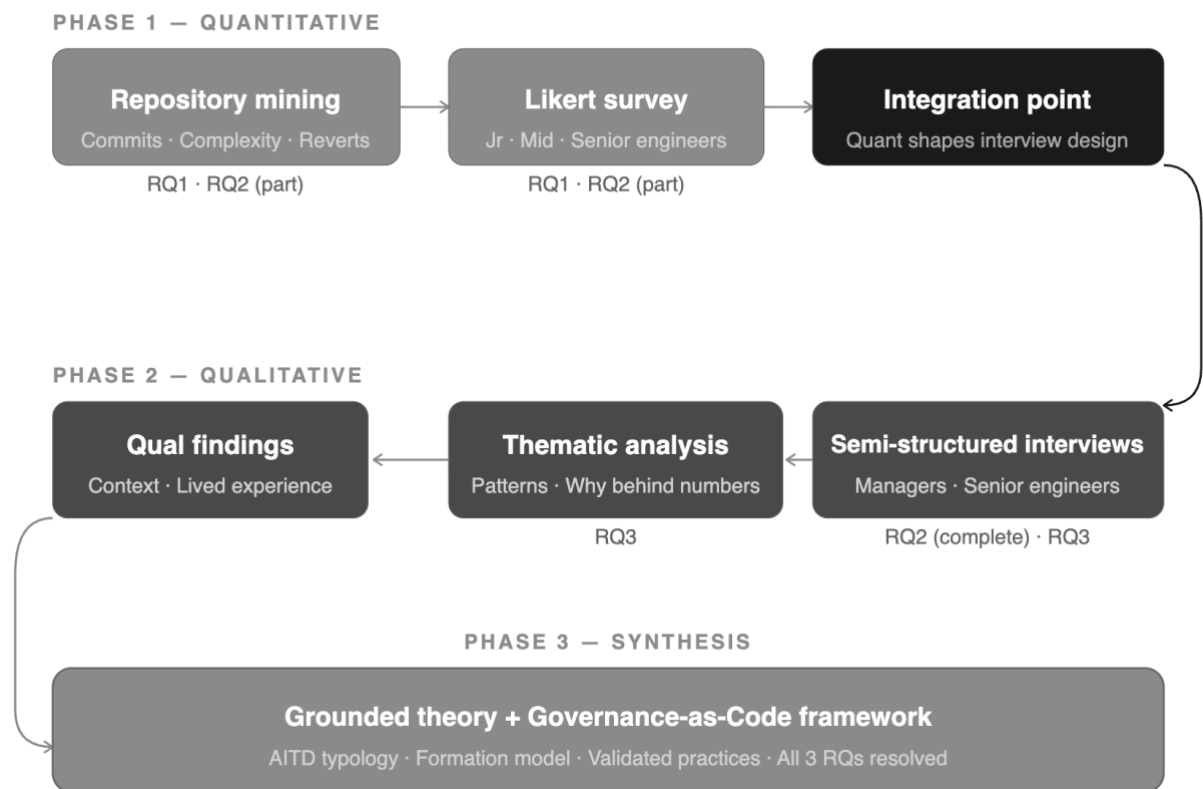


Figure 3 Research Design

Phase 1: Quantitative Phase

The first phase utilises a quantitative approach to measure structural impact of AI-generate code. This phase will involve deployment of cross-sectional Likert-Scale surveys directed at software engineers. This will map initial frequency of AI usages, volume of code generated, and self-reported time spent on validation versus active creation. This phase directly addresses RQ1 by identifying the characteristics of AITD across various development environments.

Phase 2: Qualitative Phase

The quantitative findings will directly influence the second phase to drive semi structured interviews. These will be employed to senior engineers, engineering managers and company directors. It is designed to investigate the “illusion of correctness” and the resulting “epistemic debt”. It seeks to uncover the subjective lived experiences of those reviewing machine-hallucinated logic, thereby addressing RQ2 (cognitive load) and RQ3 (organisational practices).

Phase 3: Synthesis and Framework Design

This final phase will utilise a Grounded Theory approach to synthesise the quantitative metrics and qualitative insights. The objective is to extract validated organisational practices and codify them into pragmatic “Governance-as-Code” framework required.

Challenges and Limitations

The formulation and execution of this research plan are inherently complex due to the interaction of rapidly evolving technology with human psychology. This section outlines the primary limitations and strategic measures to be implemented to ensure robust validity and reliability.

Technological Volatility and Construct Validity

The most prominent anticipated challenge for this study is the extreme volatility of the subject matter, generative AI coding tools are updated continuously. A type of syntax error or logic hallucination documented may be patched and might be obsolete before the study concludes.

To ensure long-term validity of the research validity, the research design has been deliberately abstract. Rather than measuring the specific technical accuracy of a single AI model, the study will focus on the socio-technical interaction. It specifically investigates how human engineers manage the “illusion of correctness”. While the generative AI models will evolve, and algorithms will inevitably change, the human psychological response to verifying non-deterministic systems remains a stable, measurable variable. This strategic pivot ensures the study’s future findings will remain relevant regardless of technological iterations.

Methodological Challenges and Reliability

A significant methodological limitation will lie in quantifying highly subjective human experience. And self-reported data through surveys or interviews will introduce. Inherent bias, potentially compromising the reliability of the findings.

To reduce the effects of this issue and establish strict reliability, the proposed study will employ an explanatory sequential mixed-methods design. The data gathered

during interviews will be triangulated against objective, quantitative repository data. By employing this methodology, the biasness and data validation will be stronger, and the metrics and analysis will yield highly reliable, defensible conclusions.

Conclusion of Anticipated Limitations

While the challenges of technological volatility and subjective measurement are substantial, they are actively accounted for in the proposal phase. By anchoring the study in Socio-Technical Systems theory and strict triangulation, these anticipated limitations have actively strengthened the proposed research design.

Conclusion

The paradigm of modern software engineering is undergoing a fundamental and irreversible transformation. This integration of generative AI tools like GitHub Copilot, Cursor, Claude, Codex and other language models has permanently altered the developer's toolchain. As outlined these technologies promise and often deliver, unprecedented short-term coding velocity. However, this study operated on the critical recognition that this rapid acceleration conceals a profound and escalating risk: the rapid accumulation of AITD. The core problem defining this research is the "illusion of correctness". AI models are highly proficient at generating code that is syntactically valid and capable of passing automated functional tests. Yet this code is frequently fragile, inconsistent on logics and lacks contextual depth.

When developers commit AI-generated code that they do not fully comprehend, they generate "epistemic debt". This productivity gained in initial phase is disproportionately lost during the review, maintenance and refactoring phases. And importantly this dynamic shift a massive, undocumented cognitive burden onto senior engineers. Without a formalised, socio-technical framework to manage this specific type of debt, the adoption of AI coding assistants threatens to undermine the long-term sustainability, security and resilience of software systems.

The integration of generative AI into software development represents a critical point for the industry. While the attraction of hyper-accelerated productivity is strong, the unchecked accumulation of AITD poses a threat to software sustainability and

psychological well-being of the engineering workforce. This research outlines a comprehensive and rigorous plan to address this problem in software industry.

By executing a pragmatic picot away from functionalist paradigm, expanding Socio-Technical Systems theory and also utilising a mixed methods design, this study is uniquely positioned to map both the structural and epistemic characteristics of AITD. The ultimate formulation of a set of guidelines known to be Governance-as-Code framework will provide the software engineering industry with vital and evidenced based practices necessary to harness the power of artificial intelligence with a strategy. It will help organisational leadership with the tools to navigate the illusion of correctness and protect engineering talent from burnout. Therefore, this research stands as a necessary, timely and highly practical contribution to the forefront of business, management and software engineering studies.

Bibliography

- Becker, B.A. *et al.* (2022) “Programming Is Hard – Or at Least It Used to Be: Educational Opportunities And Challenges of AI Code Generation.” Available at: <https://doi.org/10.48550/arxiv.2212.01020>.
- Bogner, J., Verdecchia, R. and Gerostathopoulos, I. (2021) *Characterizing Technical Debt and Antipatterns in AI-Based Systems: A Systematic Mapping Study*. Ithaca: Cornell University Library, arXiv.org. Available at: <https://doi.org/10.48550/arxiv.2103.09783>.
- Creswell, J.W. and Plano Clark, V.L. (2018) *Designing and conducting mixed methods research*. 3rd ed. Thousand Oaks, Calif: Sage Publications. Available at: <https://cir.nii.ac.jp/crid/1971149384823729925>.
- Kuhn, T.S. (2012) *The Structure of Scientific Revolutions*. The University of Chicago Press. Available at: <https://www.perlego.com/book/1840553/the-structure-of-scientific-revolutions-50th-anniversary-edition-pdf>.
- Mujahid, A.A. and Imran, M.M. (2026) “‘TODO: Fix the Mess Gemini Created’: Towards Understanding GenAI-Induced Self-Admitted Technical Debt.” Available at: <https://doi.org/10.48550/arxiv.2601.07786>.
- Peng, S. *et al.* (2023a) “The Impact of AI on Developer Productivity: Evidence from GitHub Copilot.” Available at: <https://doi.org/10.48550/arxiv.2302.06590>.
- Peng, S. *et al.* (2023b) “The Impact of AI on Developer Productivity: Evidence from GitHub Copilot.” Available at: <https://doi.org/10.48550/arxiv.2302.06590>.
- Saunders, M., Lewis, P. and Thornhill, A. (2023) *Research methods for business students*. Harlow: Pearson. Available at: <https://elibrary.pearson.de/book/99.150005/9781292402734>.
- Storey, M.-A. *et al.* (2024) “Guiding Principles for Using Mixed Methods Research in Software Engineering.” Available at: <https://doi.org/10.48550/arxiv.2404.06011>.

- Treude, C. and Storey, M.-A. (2025) "Generative AI and Empirical Software Engineering: A Paradigm Shift." Available at: <https://doi.org/10.48550/arxiv.2502.08108>.
- Vaithilingam, P., Zhang, T. and Glassman, E.L. (2022) "Expectation vs. Experience: Evaluating the Usability of Code Generation Tools Powered by Large Language Models." New York, NY, USA: ACM (ACM Conferences), pp. 1–7. Available at: <https://doi.org/10.1145/3491101.3519665>.
- Xu, F. *et al.* (2026) "AI-Assisted Programming Decreases the Productivity of Experienced Developers by Increasing the Technical Debt and Maintenance Burden." Available at: <https://doi.org/10.48550/arxiv.2510.10165>.