



Quantitative Research Methods (LCDB7011_LON)

**Measuring AI-Induced Technical Debt: An Empirical Analysis of Code Churn
and Cognitive Load**

Submitted To:

Dr. Akshay Narayanan

Submitted By:

Anil Poudyal

Doctorate Candidate

2429341@student.uwtsd.ac.uk

Table of Contents

Abstract.....	1
Introduction.....	1
Background to the Study	1
The Business and Societal Problem	1
Research Rationale and Gap	2
Research Objectives	2
Research Questions	3
Literature Review.....	3
The Evolution of Technical Debt in the Era of AI	3
The "Illusion of Correctness" and Epistemic Debt	3
Operational Impacts: PR Volume, PR Size, and Code Churn	4
The Productivity Paradox and Cognitive Load	4
Hypotheses Formulation	5
Research Methodology	5
Sample Population and Sampling Strategy	5
Data Collection Instrument	5
Operationalization of Variables	6
Data Analysis and Findings	6
Descriptive Statistics and Sample Demographics	6
Exploratory Data Analysis (EDA) of Core Variables	7
Influx of AI Code	8
Quality Deficit	8
Burden Shift	8
Operational Impact	8
Cognitive Burden	9
Productivity Paradox Consensus	10
Discussion.....	11
Conclusion	12
References	13
Appendices	15
Survey Questions	15
IBM SPSS Generated Reports	21

Script Log	21
Frequency Tables	22
Diagrams.....	27
 Figure 1 Experience Bar Chart.....	7
Figure 2 Company Size Bar Chart.....	7
Figure 3 PR Volume After AI	8
Figure 4 Regression Model Summary	9
Figure 5 ANOVA Table for Regression.....	9
Figure 6 Ranks Table for Kruskal Wallis Test	9
Figure 7 Kruskal Wallis Test Statistics Table	10
Figure 8 AI Induced Flaws Table	11
Figure 9 AI Induced Trade Offs Table	11

Abstract

With adoption of generative AI in software engineering the development has accelerated highly. This comes with severe long-term operational costs. This quantitative study investigates the phenomenon of AI-Induced Technical Debt (AITD). This research explores whether the initial speed gained from AI coding tools is ultimately lost to increased code flaws. This research utilises a cross-sectional survey of 51 software engineering professionals. Data was analysed using IBM SPSS to test operational and psychological impacts of AI adoption. The empirical findings reveal a highly significant predictive relationship between volume of AI-generated pull requests. Furthermore, analysis demonstrates that the cognitive burden of verifying machine hallucinated logic falls on senior engineers. This ultimately creates an unsustainable “epistemic debt”. The study concludes that without proper safeguards, generative AI operates as a technical debt generator. To reduce this problem organisations and engineering teams must transition from volume-based metrics to socio-technical frameworks that prioritises human review capacity and sustainable software maintenance.

Introduction

Background to the Study

The domain of software industry is seeing a massive pivot from traditional software engineering practises due to the introduction of Generative Artificial Intelligence (GenAI). Tools such as GitHub Copilot, Cursor, Claud Code, Codex, Antigravity and so on have been added to developer’s toolchain. And in recent times these tools have rapidly moved from experimental novelties to standard components. These tools operate by suggesting code snippets, automating repetitive tasks, providing boilerplate for head start and assisting developers in solving complex syntax problems. Initial industry reports and many academic studies have heavily praised these tools for their ability to increase coding speed and ultimately promising to improve productivity gains. And one of the studies by (Peng *et al.*, 2023) showed over 50% improvements in productivity. As a result, software development teams across the globe are adopting AI to deliver features faster and reducing the time to get to market. These solutions also have helped engineers and business personnel to get their prototype and minimum viable product (MVP) to market quickly.

The Business and Societal Problem

However, this extreme acceleration and promise of productivity gain in code generation comes with a severe, often undocumented cost: the accumulation of AI-Induced Technical Debt (AITD). While AI tools can generate code quickly with often correct syntax,

they do not inherently understand the deep architectural and structural context of the specific business logic of the systems they are creating code or modifying the code of. This whole phenomenon is coined to be “illusion of correctness” by (Vaithilingam, Zhang and Glassman, 2022, pp. 1–7). The code generated by AI tool often complies successfully and passes basic automated tests, but it frequently contains subtle logical flaws to it. These flaws are but not limited to security vulnerabilities, code churn, inefficient code, and unscalable architecture.

Because the code looks professional and structurally sound on the surface, developers often approve the code for production use-case without deep comprehension and quality audit. This creates “epistemic debt”, a situation where the team no longer fully understands the code in codebase that they are supposed to maintain. And adding on to it, this dynamic shift a massive psychological and operational burden onto senior engineers. The phenomenon has introduced more activities within the cycle where junior engineers submit more pull requests (PRs) at a faster rate and senior engineers must expend significantly more mental effort to review, verify and fix this machine hallucinated logic. The productivity gained during the writing phase by junior engineers is ultimately lost during the review and maintenance phases. This ultimately threatens long-term software sustainability and increasing the risk of senior engineer burnout due to cognitive load (Becker *et al.*, 2022).

Research Rationale and Gap

Most of the current academic discussion surrounding AI in software engineering and the industry focuses either on benchmarking the technical accuracy of the AI models. These benchmarking are being done in isolated environments or exploring qualitative developer sentiments. There is lack of quantitative and novel research measuring the exact impact of AI on the daily operations of software teams. This study addresses this particular gap in the domain. By utilising primary quantitative data collected from active software engineering professionals across various organisational sizes and regions. This research moves beyond theory to gather and evaluate the findings. It mathematically measures the “productivity paradox” by quantifying review fatigue, code churn and developer trust.

Research Objectives

The primary aim of this project to evaluate the impact of AI-generated code on the review workload, maintainability and accumulation of technical debt within software development teams. To achieve this the following are the set out specific objectives:

1. To measure the statistical change in Pull Request (PR) volume and code churn following the adoption of AI coding tools.
2. To quantify the mental effort and "trust gap" experienced by engineers when reviewing AI-assisted code compared to human-written code.

3. To determine if there is a significant correlation between the speed of AI code generation and the long-term loss of productivity due to review and maintenance burdens.
4. To analyse how the burden of AI-Induced Technical Debt (AITD) is distributed across different levels of engineering seniority.

Research Questions

To guide the quantitative analysis, this study proposes the following research questions:

1. **RQ1:** To what extent does the integration of AI coding tools increase PR volume and subsequent code churn in software projects?
2. **RQ2:** How does the "illusion of correctness" statistically impact the mental effort and trust levels of engineers conducting code reviews?
3. **RQ3:** Is there a measurable "productivity paradox" where the speed gained from AI is offset by the time spent fixing and maintaining the generated code?

Literature Review

The Evolution of Technical Debt in the Era of AI

The concept of “Technical Debt” was originally coined by Ward Cunningham to describe long-term cost of choosing an easy, short-term software architecture solution over a better, more difficult one (Letouzey and Whelan, 2016). Traditionally, technical debt was a human-driven metric, developers consciously took “loans” against future maintainability to meet immediate business deadlines (Dargo, 2019).

However, the introduction of Large Language Model (LLMs) in software engineering has fundamentally altered the landscape and is in verge of major paradigm shift. Recent literature has identified a new subset of this phenomenon called AI-Induced Technical Debt (AITD) (Ehsani *et al.*, 2026). It is different from the traditional technical debt, AITD is rarely accumulated consciously. It gets accumulated overtime also with integration of epistemic debt in many cases. This happens because generative AI tools such as GitHub Copilot or Cursor or Codex act as probabilistic generators rather than deterministic logic engines. They frequently generate code that is syntactically flawless but architecturally fragile (Kashif *et al.*, 2026). The literature overwhelmingly suggests that the sheer volume of code generated by AI is rapidly overshadowing the industry’s capacity to govern and monitor it. It is turning technical debt from a deliberate management strategy into an uncontrollable byproduct of speed.

The "Illusion of Correctness" and Epistemic Debt

The major critical driver of AITD identified in current peer reviewed research is the “illusion of correctness”. When junior developers or AI-assisted tools generate code, the output often successfully passes automated unit tests and syntax checks on continuous

integration (CI) (Vaithilingam, Zhang and Glassman, 2022, pp. 1–7). Because the code appears highly professional and functional, developers inherently lower their guard.

This leads to a serious amount of accumulation of “epistemic debt” as termed by researchers. Epistemic debt occurs when a team commits and deploys code that they fully do not understand (Storey, 2026). (Becker *et al.*, 2022) observed that developers frequently enter a bimodal state where they act either authors or editors. When they are in editing state of code, their cognitive engagement drops, leading to blind approvals of logic they have not deeply verified or understood. This creates a massive gap in trustworthiness within the engineering teams. Some engineers are forced to decide whether to verify every line of code that is generated by a generative AI model or to blindly trust the AI, the latter of which inevitably introduces systemic vulnerabilities.

Operational Impacts: PR Volume, PR Size, and Code Churn

The current scholarship demonstrates that AI tools alter the physical operations of software development pipelines with introduction of delay in final product shipment. The primary operational impact is the explosion of Pull Requests (PRs) volume. Due to reduced barriers to generating lines of code, developers submit PRs more frequently and with significantly larger code footprints over less time (Yetiştiren *et al.*, 2023).

However, the major myth is that high volume of PR cycles and large amount code is directly related to first-pass quality. Studies show that AI-assisted code often suffers from code churn. It is a metric which denotes that the written/generated code is often must go through rewrites or deletion. This is due to introduction of bugs, out of context features or wrong business logics (Rahman and Shihab, 2026). Also, (Paradis *et al.*, 2024) also highlights similar pattern, while developers write code faster, the resulting code churn creates severe bottleneck at the senior review stage.

The Productivity Paradox and Cognitive Load

AI have been heavily praised for its raw speed in early stage but there is an unseen friction, and this friction leads to the central theme of this research. The friction is coined to be “Productivity Paradox”. The longitudinal peer-reviewed studies reveal a much darker reality regarding long-term software maintainability (Peng *et al.*, 2023).

The earlier gained productivity by the individual developer using AI tools is disproportionately lost by the team during the review and further maintenance phase (Xu *et al.*, 2026). I one hand it seems that the productivity is boosted but due to bugs, maintenance issues and review load the productivity is lost. This paradox is driven by cognitive load. Reviewing human-written code is relatively predictable, as humans make logical, narrative errors. In the other hand reviewing AI-written code requires significantly higher mental effort as AI models are known for their hallucinated logic in unpredictable, non-human patterns (Kalai *et al.*, 2025). Consequently, senior engineers absorb a

massive, undocumented cognitive burden, transforming AI tools from a net-productivity gain into a long-term maintenance liability.

Hypotheses Formulation

Following the literature review, here are the hypotheses that have been derived for this research.

- **Hypothesis 1 (H1):** There is a significant positive correlation between AI-driven Pull Request Volume and increase in code churn.
- **Hypothesis 2 (H2):** The level of "Mental Effort" required to review AI-generated code differs significantly based on the engineer's Years of Experience.

Research Methodology

This study adopts a Positivist research philosophy; it relies on quantifiable observations and statistical analysis to uncover the objective realities. The study employs a cross-sectional design of survey as it is highly appropriate for this research as it allows to collect data a single point in time across a diverse range of software engineers.

Sample Population and Sampling Strategy

The target population for this study comprises software engineering professionals. It ranges from junior developers to engineering managers who actively use generative AI coding tools withing their workflows.

Purposive and convenience sampling strategy was utilised, and survey was distributed via professional engineering networks. After cleaning the dataset to remove incomplete responses the final sample size yielded N=51 valid responses. While this sample size is relatively small, it provided a highly focused, expert-level snapshot of the industry.

Data Collection Instrument

Primary data was collected using an online, structured questionnaire through Qualtrics. Using an online survey tool allowed for standardized data collection, ensuring that all respondents answered identical questions under the same conditions. It consists of 15 questions divided into three distinct sections:

1. **Informed Consent (Q1):** Establishing the academic purpose and ensuring anonymity.
2. **Demographic and Control Variables (Q2 - Q5):** Capturing the respondent's experience level, primary role, organization size, and geographic region.
3. **Core AITD Variables (Q6 - Q15):** Utilizing a mix of 4-point and 5-point Likert scales, multiple-choice, and categorical questions to measure the operational and psychological impacts of reviewing AI-generated code.

Operationalization of Variables

Following variables are identified to perform statistical analysis in SPSS. These relate to the concepts discussed in the literature review.

1. Independent Variables:

- 1.1. PR Volume (Q6) and PR Size (Q7): Measured on a 4-point ordinal scale to quantify the raw increase in code generation.

2. Mediating Variables:

- 2.1. Mental Effort (Q8): Measured on a 4-point ordinal scale to capture the cognitive strain of verifying machine-hallucinated logic.
- 2.2. The Trust Gap (Q9): Measured on a 5-point Likert scale to quantify epistemic uncertainty.

3. Dependent Variables:

- 3.1. Code Churn (Q10): Measured on a 4-point frequency scale to track how often AI code requires subsequent fixes.
- 3.2. The Trade-off (Q13): Measured on a standard 5-point Likert scale. This serves as the primary metric for the "Productivity Paradox,".

4. Contextual Variables:

- 4.1. Burden Shift (Q12), Nature of Flaws (Q11), and Current Safeguards (Q14) are used as categorical variables to understand how different teams attempt to manage this new form of technical debt.

Data Analysis and Findings

Descriptive Statistics and Sample Demographics

The raw dataset was imported into IBM SPSS Statistics. The valid sample size of N=51 of software engineering professional was found. To ensure the reliability of the inferential findings, an exploratory analysis of the demographic and contextual variables was conducted.

The sample represents a highly mature technical demographic. Most survey respondents possess 6-10 years of professional experience which accounts for 43.1%, followed by early career engineers with 3-5 years 29.4%, and highly experienced veterans with 11-15 years of experience 27.5%. Regarding the role distribution, the dataset captures perspectives across the entire software development lifecycle. It features 19 Software Engineers, 13 Executive Leaders and 11 Engineering Managers.

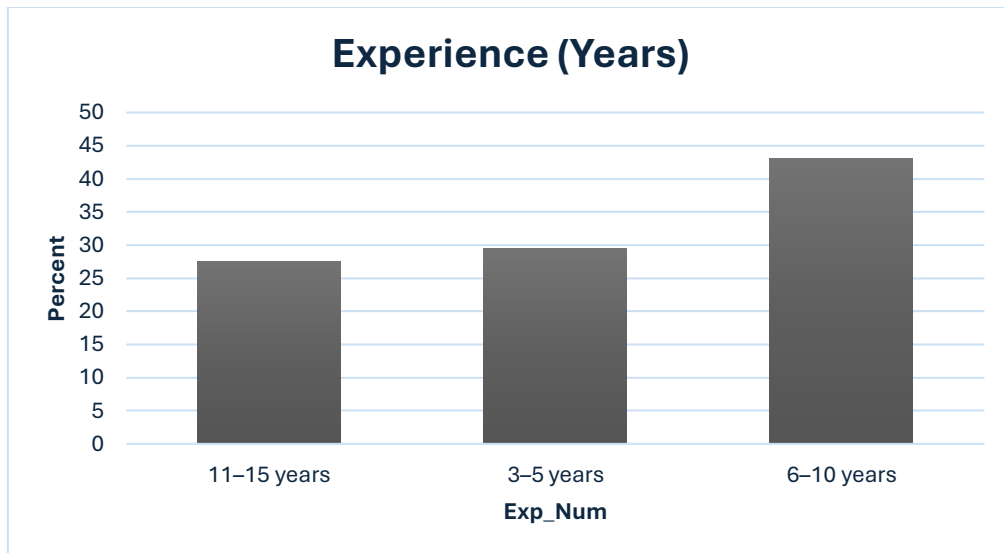


Figure 1 Experience Bar Chart

With regards to geographic location the data represents a diverse cross-section of global tech industry. The largest segment of respondents operates within United Kingdom & Europe (37.3%), followed by South Asia (25.5%) and North America (19.6%). And among these majority of respondents represent Small to Medium software firms with over 55%.

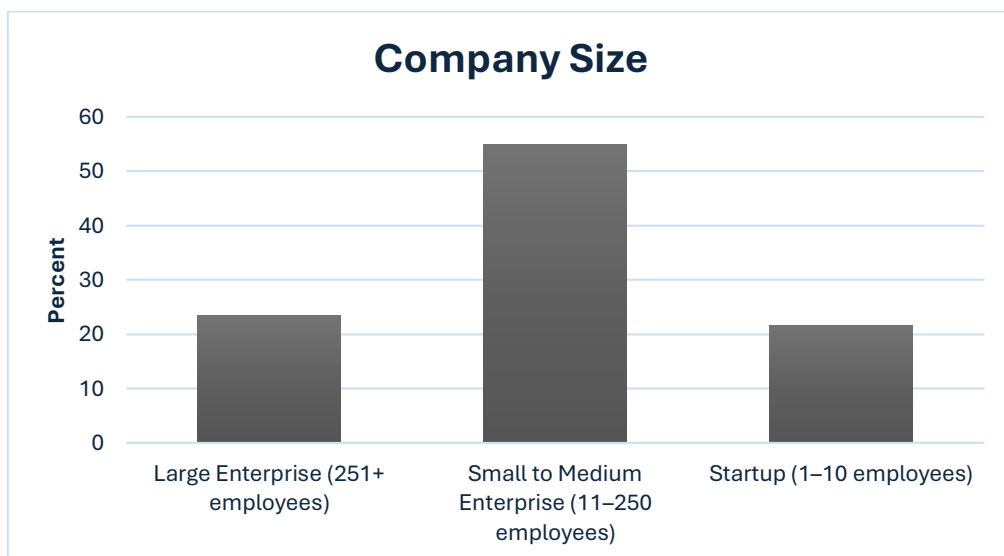


Figure 2 Company Size Bar Chart

Exploratory Data Analysis (EDA) of Core Variables

Before conducting inferential hypothesis testing, an EDA was performed on the core independent variables. Also, with its mediating variables were analysed to assess the current state of AI Induced Technical Debt (AITD).

Influx of AI Code

The frequency analysis of the variables regulating AI output reveals a massive acceleration in code generation. The data indicates that 64% of respondents report Pull requests volume has significantly increased since the adoption of AI coding tools. Other 35% reported slightly increase marking there is increase of Pull Requests (PRs) after introduction of AI. This exploratory finding establishes the physical foundation of AITD.

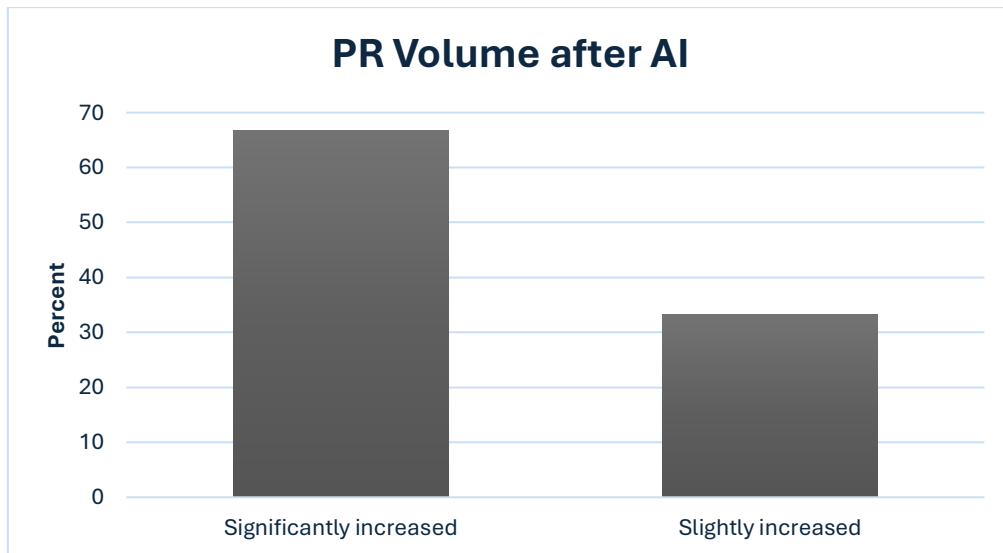


Figure 3 PR Volume After AI

Quality Deficit

Despite the high volume of AI-generated code, the EDA reveals a severe lack of automated organisational governance. The data reveals that only 25.5% of the engineers and professionals rely on basic automated unit tests to catch bad AI code, and unit tests are not built to flag AI generated long term technical debt.

Burden Shift

The examined data showed that burden shift is significant and more than quarter of the respondent's state that the burden is shifted to Senior and Lead engineers. One of the most expensive resources are paying the high prices for AI Acceleration. And the data suggests that only about 15% of the junior developers are absorbing the review load.

Operational Impact

To test first hypothesis; an increase in AI-driven pull requests leads to increase in code churn, a linear regression analysis was conducted. The results clearly show a highly significant predictive relationship. The model shows that there is variance of 23% in code churn. This is directly linked with volume of AI-generated pull requests created. This provides strong evidence that "illusion of correctness" is actively degrading software pipelines. As I tool help in generate code and accelerate the volume of code submissions,

teams experience significant increase in broken code that must be rewritten shortly after merging. Ultimately, the data proved that AI is not just making teams write code faster but also pushing developers into accelerated maintenance loop.

Model Summary

Model	R	R Square	Adjusted R Square	Std. Error of the Estimate
1	.476 ^a	.226	.211	.444

a. Predictors: (Constant), PR_Vol_Num

Figure 4 Regression Model Summary

ANOVA^a

Model		Sum of Squares	df	Mean Square	F	Sig.
1	Regression	2.833	1	2.833	14.348	<.001 ^b
	Residual	9.676	49	.197		
	Total	12.510	50			

a. Dependent Variable: Churn_Num

b. Predictors: (Constant), PR_Vol_Num

Figure 5 ANOVA Table for Regression

Cognitive Burden

Another hypothesis which examined whether the mental effort required to review AI generate code have variance based on the level of engineer's experience. Kruskal-Wallis's test was used to compare perceived mental strain across the different experience groups.

The test confirmed a significant difference in mental effort. And this effort was highly dependent on engineer's seniority. This has a distinct socio-technical divide. Mid-level engineers report the lowest cognitive strain; the most experienced cohort reports the highest levels of mental effort required to verify AI-assisted code.

Ranks

	Experience	N	Mean Rank
Mental Burnout	11–15 years	14	16.96
	3–5 years	15	25.10
	6–10 years	22	32.36
	Total	51	

Figure 6 Ranks Table for Kruskal Wallis Test

Test Statistics^{a, b}

	Mental_Num
Kruskal-Wallis H	12.577
df	2
Asymp. Sig.	.002

a. Kruskal Wallis Test

b. Grouping Variable:

Exp_Num

Figure 7 Kruskal Wallis Test Statistics Table

This finding empirically validates the concept of “epistemic debt”. While less experienced developers may comfortably trust AI’s output, experience engineers argue this claim. This argument is valid as the perspective of senior engineers is from massive cognitive burden attempting to verify complex, machine-hallucinated logic. The data demonstrates that AI tools are inadvertently weaponizing the review process, directly contributing to severe senior-level burnout.

Productivity Paradox Consensus

A descriptive frequency check was run on the trade-off statement to measure productivity paradox. Respondents were asked if the short-term coding speed gained from AI is eventually lost due to extra time spent in reviewing, debugging and maintaining that code.

AI Induced Flaws

		Frequency	Percent	Valid Percent	Cumulative Percent
Valid	Complex logic or architectural flaws	16	31.4	31.4	31.4
	Complex logic or architectural flaws, Security vulnerabilities	14	27.5	27.5	58.8
	Security vulnerabilities	3	5.9	5.9	64.7
	Simple syntax errors or typos, Complex logic or architectural flaws	10	19.6	19.6	84.3
	Simple syntax errors or typos, Complex logic or architectural flaws, Security vulnerabilities	4	7.8	7.8	92.2

Simple syntax errors or typos, Security vulnerabilities	4	7.8	7.8	100.0
Total	51	100.0	100.0	

Figure 8 AI Induced Flaws Table

The data revealed extraordinary reality: 100% of the surveyed professionals agreed with the statement that in long term due to reduced maintainability and scalability of the software system built the net productivity is lost. Specifically, 45 out of 51 respondents strongly agreed, while the remaining 6 others somewhat agreement making cumulative consensus to 100%. In quantitative research, this kind of absolute consensus is rare. It mathematically proves that across all measured demographics, regions, and company sizes, AI-induced technical debt is not a theoretical risk, but a universally recognised industry challenge.

AI Induced Trade off					
		Frequency	Percent	Valid Percent	Cumulative Percent
Valid	Somewhat agree	6	11.8	11.8	11.8
	Strongly Agree	45	88.2	88.2	100.0
	Total	51	100.0	100.0	

Figure 9 AI Induced Trade Offs Table

Discussion

The first major finding of this study provided an empirical validation for the operational bottleneck caused by AI tools. The first hypothesis tested the relationship between the volume of AI-generated pull requests and frequency of subsequent code fixes. The regression analysis proved that the relation is highly significant and highly predictive. It showed that nearly 23% of the variance in code churn is directly caused by the influx of AI code.

The findings directly support the assertion made by (Yetiştiren *et al.*, 2023) , who theorised that the speed of AI generation does not equate to first-pass quality. The data proves that with introduction of AI, it lowers the barriers to generate code, they simultaneously flood the CI/CD pipeline with fragile architecture. They are not simply writing code faster; they are entering an accelerated, endless loop of writing, merging, and immediately rewriting. This confirms that AITD is fundamentally altering the physical operations of software development.

Perhaps the most critical socio-technical finding of this study relates to cognitive load and engineer burnout. Second hypothesis studied the relation of mental effort required to review AI code and engineer's years of experience. The Kruskal-Wallis test revealed a highly significant disparity, proving that the cognitive burden of AI falls disproportionately on the most experienced members of the team. The empirical finding validates the concept of "epistemic debt" as discussed by (Storey, 2026). When junior developers use AI, they often generate syntactically correct code that lacks deep business context.

Conclusion

There is indeed a paradigm shift as engineering domain have started introducing AI tool chains in their development activity. But the empirical study has demonstrated that the rush to adopt these tools for short-term coding velocity has masked a severe and long-term maintainability and scalability risk. With the quantitative analysis of 51 software development stakeholders this research validates the existence of "illusion of correctness". The study also reveals socio-technical divide, proving that the burden of verifying machine-hallucinated logic falls disproportionately on experienced engineers. This creates an unsustainable "epistemic debt" within enterprise teams.

Generative AI is an incredibly powerful tool, but it is not an autonomous engineering solution. This study provides empirical proof that treating AI as a "silver bullet" for productivity creates a massive technical hidden debt. This study also mathematically verified Productivity Paradox, with absolute industry consensus. The initial coding speed gained from AI is ultimately lost to maintenance, manual process of reviewing and debugging that fragile code.

References

- 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>.
- Dargo, Z. (2019) *Technical Debt Management: Definition of a Technical Debt Reduction Software Engineering Methodology for SMEs*.
- Ehsani, R. et al. (2026) "Where Do AI Coding Agents Fail? An Empirical Study of Failed Agentic Pull Requests in GitHub." Available at: <https://doi.org/10.48550/arxiv.2601.15195>.
- Kalai, A.T. et al. (2025) *Why Language Models Hallucinate*. Ithaca: Cornell University Library, arXiv.org. Available at: <https://www.proquest.com/docview/3302647497>.
- Kashif, S.M. et al. (2026) "Beyond Functional Correctness: Design Issues in AI IDE-Generated Large-Scale Projects." Available at: <https://doi.org/10.48550/arxiv.2604.06373>.
- Letouzey, J.-L. and Whelan, D. (2016) "Introduction to the TechnicalDebt Concept." Available at: <https://agilealliance.org/wp-content/uploads/2016/05/IntroductiontotheTechnicalDebtConcept-V-02.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>.
- Paradis, E. et al. (2024) "How much does AI impact development speed? An enterprise-based randomized controlled trial." Available at: <https://doi.org/10.48550/arxiv.2410.12944>.
- 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>.

Rahman, M. and Shihab, E. (2026) "Will It Survive? Deciphering the Fate of AI-Generated Code in Open Source." Available at: <https://doi.org/10.48550/arxiv.2601.16809>.

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>.

Storey, M.-A. (2026) "From Technical Debt to Cognitive and Intent Debt: Rethinking Software Health in the Age of AI." Available at: <https://doi.org/10.48550/arxiv.2603.22106>.

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>.

Yetiştirilen, B. *et al.* (2023) "Evaluating the Code Quality of AI-Assisted Code Generation Tools: An Empirical Study on GitHub Copilot, Amazon CodeWhisperer, and ChatGPT." Available at: <https://doi.org/10.48550/arxiv.2304.10778>.

Appendices

Survey Questions

Q1 The AI Coding Paradox: Are we trading coding speed for an endless maintenance loop?

While AI tools like GitHub Copilot and ChatGPT help us write code faster than ever, we don't yet fully understand the long-term "hidden costs" they might create. This study explores whether the speed we gain today is leading to an invisible buildup of technical debt that will be difficult to manage tomorrow. My research aims to develop a strategic framework to help organizations balance the benefits of AI-augmented coding with the need for sustainable, high-quality software maintenance. Your Participation & Consent By proceeding and submitting this survey, you agree to participate in this study. Your responses are completely anonymised and will be used strictly for academic purposes as part of my Doctoral (DBA) research. This data will not be used for commercial purposes, and no personally identifiable information is collected. You are free to stop the survey at any time.

Q2 How many years of professional experience do you have in software development or technical management?

- ☐ 0 – 2 years (1)
- ☐ 3 – 5 years (2)
- ☐ 6 – 10 years (3)
- ☐ 11 – 15 years (4)
- ☐ 16+ years (5)

Q3 What is your current primary role?

☐ Software Engineer (1)

☐ Team Lead (2)

☐ Software Architect (3)

☐ Product Manager (4)

☐ Executive (CTO, Director, VP) (5)

☐ Consultant (6)

Other: (7) _____

Q4 Organization Size

- ☐ Startup (1–10 employees) (1)
- ☐ Small to Medium Enterprise (11–250 employees) (2)
- ☐ Large Enterprise (251+ employees) (3)

Q5 Click to write the question text

- ☐ United Kingdom & Europe (e.g., UK, Germany, France, Poland) (1)
- ☐ North America (e.g., USA, Canada, Mexico) (2)
- ☐ South Asia (e.g., Nepal, India, Pakistan, Bangladesh, Sri Lanka) (3)
- ☐ East & Southeast Asia (e.g., China, Japan, Singapore, Vietnam, Philippines) (4)
- ☐ Middle East (e.g., UAE, Saudi Arabia, Israel, Qatar, Jordan) (5)
- ☐ Africa (e.g., Nigeria, Kenya, South Africa, Egypt, Ethiopia) (6)
- ☐ South & Central America (e.g., Brazil, Argentina, Colombia, Costa Rica) (7)
- ☐ Oceania (e.g., Australia, New Zealand, Fiji) (8)

Q6 PR Volume: Since your team adopted AI coding tools, the number of pull requests (PRs) or code submissions you are asked to review has:

- ☐ Significantly increased (1)
- ☐ Slightly increased (2)
- ☐ Stayed the same (3)
- ☐ Decreased (4)

Q7 PR Size: The average size (lines of code) of the changes you are reviewing has:

- ☐ Significantly increased (1)
- ☐ Slightly increased (2)
- ☐ Stayed the same (3)
- ☐ Decreased (4)

Q8 Mental Effort: Compared to human-written code, the mental focus required to verify that AI-assisted code is structurally sound and secure is:

- ☐ Much higher (It requires deeper scrutiny) (1)
- ☐ Slightly higher (2)
- ☐ About the same (3)
- ☐ Lower (It is easier to review) (4)

Q9 The Trust Gap: How comfortable are you approving AI-generated code without doing a deep, manual, line-by-line verification?

- ☐ Extremely uncomfortable (1)
- ☐ Somewhat uncomfortable (2)
- ☐ Neither comfortable nor uncomfortable (3)
- ☐ Somewhat comfortable (4)
- ☐ Extremely comfortable (5)

Q10 Code Churn: How often do you see AI-assisted code get approved and merged, only to require fixes or rewrites within a few weeks?

☐ Very frequently (1)

☐ Occasionally (2)

☐ Rarely (3)

☐ Never (4)

Q11 Nature of Flaws: When you do find issues in AI-generated code during a review, they are most often:

☐ Simple syntax errors or typos (1)

☐ Complex logic or architectural flaws (2)

☐ Security vulnerabilities (3)

Other (4) _____

Q12 The Burden Shift: In your observation, who on the team is absorbing the most time verifying, fixing, and maintaining AI-generated code?

- ☐ Mostly Senior / Lead Engineers (1)
- ☐ Mostly Junior / Mid-level Engineers (2)
- ☐ The burden is evenly distributed (3)

Q13 The Trade-off: To what extent do you agree: *"The coding speed our team gains from AI is eventually lost to the extra time spent reviewing, debugging, and maintaining that code."*

- ☐ Strongly Agree (1)
- ☐ Somewhat agree (2)
- ☐ Neither agree nor disagree (3)
- ☐ Somewhat disagree (4)
- ☐ Strongly disagree (5)

Q14 Current Safeguards: How is your team primarily catching bad AI code before it hits the main branch?

- ☐ Manual human review only (1)
- ☐ Basic automated unit tests (2)
- ☐ Advanced automated quality and security gates in the CI/CD pipeline (3)
- ☐ We do not have a reliable safeguard yet (4)

Other (5) _____

Q15 The Main Friction Point: In one short sentence, What **is the single most exhausting part of managing or reviewing AI-generated code?**

IBM SPSS Generated Reports

Script Log

Churn into Churn_Num

Old Value	New Value	Value Label
-----------	-----------	-------------

Occasionally	1	Occasionally
Very frequently	2	Very frequently

Exp into Exp_Num

Old Value	New Value	Value Label
-----------	-----------	-------------

11-15 years	1	11-15 years
3-5 years	2	3-5 years
6-10 years	3	6-10 years

Mental into Mental_Num

Old Value	New Value	Value Label
-----------	-----------	-------------

Much higher (It requires deeper	1	Much higher (It
requires deeper		
scrutiny)	1	scrutiny)
Slightly higher	2	Slightly higher

AI_Trust into Trust_Num

Old Value	New Value	Value Label
-----------	-----------	-------------

Extremely uncomfortable	1	Extremely uncomfortable
Somewhat uncomfortable	2	Somewhat uncomfortable

Trade_Off into Tradeoff_Num

Old Value	New Value	Value Label
-----------	-----------	-------------

Somewhat agree	1	Somewhat agree
Strongly Agree	2	Strongly Agree

Frequency Tables

Statistics

	Exp_ Num	R ole	Comp any	Reg ion	PR_Vol_ Num	PR_ Size	Safe_G uard	Flaw_ Type	AI_T rust	Budern_ Shift
N Valid	51	51	51	51	51	51	51	51	51	51
Missing	0	0	0	0	0	0	0	0	0	0

Exp_Num

		Frequency	Percent	Valid Percent	Cumulative Percent
Valid	11–15 years	14	27.5	27.5	27.5
	3–5 years	15	29.4	29.4	56.9
	6–10 years	22	43.1	43.1	100.0
	Total	51	100.0	100.0	

Role

		Frequency	Percent	Valid Percent	Cumulative Percent
Valid	Executive (CTO, Director, VP)	13	25.5	25.5	25.5
	Product Manager	8	15.7	15.7	41.2
	Software Architect	3	5.9	5.9	47.1
	Software Engineer	16	31.4	31.4	78.4
	Team Lead	11	21.6	21.6	100.0
	Total	51	100.0	100.0	

Company

		Frequency	Percent	Valid Percent	Cumulative Percent
Valid	Large Enterprise (251+ employees)	12	23.5	23.5	23.5
	Small to Medium Enterprise (11–250 employees)	28	54.9	54.9	78.4
	Startup (1–10 employees)	11	21.6	21.6	100.0
	Total	51	100.0	100.0	

		Region			
		Frequency	Percent	Valid Percent	Cumulative Percent
Valid	East & Southeast Asia (e.g., China, Japan, Singapore, Vietnam, Philippines)	3	5.9	5.9	5.9
	Middle East (e.g., UAE, Saudi Arabia, Israel, Qatar, Jordan)	10	19.6	19.6	25.5
	North America (e.g., USA, Canada, Mexico)	9	17.6	17.6	43.1
	South Asia (e.g., Nepal, India, Pakistan, Bangladesh, Sri Lanka)	13	25.5	25.5	68.6
	United Kingdom & Europe (e.g., UK, Germany, France, Poland)	16	31.4	31.4	100.0
	Total	51	100.0	100.0	

		PR_Vol_Num			
		Frequency	Percent	Valid Percent	Cumulative Percent
Valid	Significantly increased	34	66.7	66.7	66.7
	Slightly increased	17	33.3	33.3	100.0
	Total	51	100.0	100.0	

		PR_Size			
		Frequency	Percent	Valid Percent	Cumulative Percent
Valid	Significantly increased	38	74.5	74.5	74.5
	Slightly increased	13	25.5	25.5	100.0
	Total	51	100.0	100.0	

PR_Vol_Num

		Frequency	Percent	Valid Percent	Cumulative Percent
Valid	Significantly increased	34	66.7	66.7	66.7
	Slightly increased	17	33.3	33.3	100.0
	Total	51	100.0	100.0	

Safe_Guard

		Frequency	Percent	Valid Percent	Cumulative Percent
Valid	Advanced automated quality and security gates in the CI/CD pipeline	15	29.4	29.4	29.4
	Basic automated unit tests	16	31.4	31.4	60.8
	Manual human review only	17	33.3	33.3	94.1
	We do not have a reliable safeguard yet	3	5.9	5.9	100.0
	Total	51	100.0	100.0	

Flaw_Type

		Frequency	Percent	Valid Percent	Cumulative Percent
Valid	Complex logic or architectural flaws	16	31.4	31.4	31.4
	Complex logic or architectural flaws, Security vulnerabilities	14	27.5	27.5	58.8
	Security vulnerabilities	3	5.9	5.9	64.7
	Simple syntax errors or typos, Complex logic or architectural flaws	10	19.6	19.6	84.3

Simple syntax errors or typos,Complex logic or architectural flaws,Security vulnerabilities	4	7.8	7.8	92.2
Simple syntax errors or typos,Security vulnerabilities	4	7.8	7.8	100.0
Total	51	100.0	100.0	

AI_Trust

		Frequency	Percent	Valid Percent	Cumulative Percent
Valid	Extremely uncomfortable	35	68.6	68.6	68.6
	Somewhat uncomfortable	16	31.4	31.4	100.0
	Total	51	100.0	100.0	

Variables Entered/Removed^a

Model	Variables Entered	Variables Removed	Method
1	PR_Vol_Num ^b	.	Enter

a. Dependent Variable: Churn_Num

b. All requested variables entered.

Model Summary

Model	R	R Square	Adjusted R Square	Std. Error of the Estimate
1	.476 ^a	.226	.211	.444

a. Predictors: (Constant), PR_Vol_Num

ANOVA^a

Model		Sum of Squares	df	Mean Square	F	Sig.
1	Regression	2.833	1	2.833	14.348	<.001 ^b
	Residual	9.676	49	.197		
	Total	12.510	50			

a. Dependent Variable: Churn_Num

b. Predictors: (Constant), PR_Vol_Num

Coefficients ^a					
Model		Unstandardized Coefficients		Standardized Coefficients	
		B	Std. Error	Beta	t
1	(Constant)	2.235	.187		11.974
	PR_Vol_Num	-.500	.132	-.476	-3.788
					Sig.
					<.001
					<.001

a. Dependent Variable: Churn_Num

Ranks			
	Exp_Num	N	Mean Rank
Mental_Num	11–15 years	14	16.96
	3–5 years	15	25.10
	6–10 years	22	32.36
	Total	51	

Test Statistics ^{a,b}	
	Mental_Num
Kruskal-Wallis H	12.577
df	2
Asymp. Sig.	.002

a. Kruskal Wallis Test

b. Grouping Variable:

Exp_Num

Diagrams

