Generative AI and Software Quality Dynamics

Edward G. Anderson
McCombs School of Business
The University of Texas at Austin
Austin, TX 78712
edward.anderson@mccombs.utexas.edu

Geoffrey G. Parker
Thayer School of Engineering
Dartmouth College
Hanover, NH 03755
geoffrey.g.parker@dartmouth.edu

Burcu Tan
The James & Gale Ellis School of Business Leadership
The University of New Mexico
Albuquerque, NM 87131
btan@unm.edu

Keywords: AI-Assisted Coding, Software Development, Technical Debt, System Dynamics, Human-AI Interaction

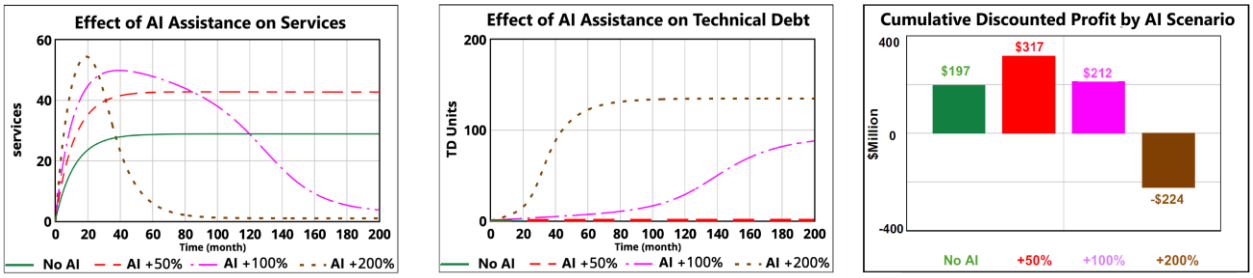
Motivation and Research Questions: AI-assisted coding tools are rapidly reshaping software development. Industry reports and controlled experiments document productivity gains of 20–55% (Peng et al. 2023, Cui et al. 2024), and prominent technology leaders have predicted that AI will soon write the vast majority of production code. Yet, the real-world experiences of professional software developers paint a more nuanced picture. Interviews that we conducted with veteran developers, discussions in the trade press and online software development forums, and early field studies (GitClear 2025, He et al 2026, Liu et al 2026) suggest that AI assistance can sometimes degrade code quality, increase complexity, and introduce new forms of risk, despite improving short-term output. We propose that technical debt (TD) — the accumulating maintenance burden arising from expedient but suboptimal coding decisions — provides a mechanism for explaining these divergent outcomes. We address four questions: (1) What are the long-term impacts of AI-assisted coding on software quality and firm profitability? (2) What level of AI assistance balances short-term productivity with long-term sustainability? (3) How do the programming environment, legacy-system interaction, market dynamism, and debt-management practices shape optimal AI adoption?

Model: We develop a system dynamics simulation model grounded in interviews with software engineers, industry cases, trade-press reporting, and prior research on technical debt and AI-assisted development. The model's core state variables are *Services* and *Technical Debt (TD)*. We consider a firm whose revenue depends on the software services it develops — a term used generically to encompass mobile apps, enterprise software, and internal code that supports revenue-generating functions.

Any software development activity generates some TD, which accumulates over time. As TD grows, service development productivity falls because developers must navigate tangled, outdated, or poorly structured ("spaghetti") code (Tom et al. 2013, Besker et al. 2019, 2020, Franco et al. 2023). Moreover, as the system grows more complex, even small changes require disproportionate effort to understand dependencies. This declining development rate in turn reduces the rate at which new TD is generated, closing a balancing feedback loop (*Bad code reducing productivity*). TD generation also depends on the current level of TD in the codebase: TD tends to compound on itself (Tom et al. 2013, McKinsey 2020, 2023, Franco et al. 2023) because accumulation triggers a downward spiral of design shortcuts that beget further accumulation (Ramasubbu & Kemerer 2016). This forms a reinforcing feedback loop (*TD begets TD*) — a vicious cycle acknowledged in prior work (Franco et al. 2023, Glaiel et al. 2014).

AI assistance accelerates this vicious cycle through two channels. First, AI-coding tools boost service development rate (Peng et al. 2023, Cui et al. 2024), expanding the codebase and thereby adding to TD even holding quality constant. Second, and more consequentially, AI tools directly elevate the rate of TD generation (GitClear 2025, He et al. 2026, Liu et al. 2026) due to AI's lack of architectural context awareness (Aslan 2024, Doerrfeld 2025), manifesting as code duplications, dependency conflicts, and poorly structured code. Both effects also increase total maintenance cost (Woodard et al. 2013, Ramasubbu & Kemerer 2016, Besker et al. 2019), further eroding firm profitability. A counterbalancing force exists: services obsolesce over time, rendering some of their associated TD irrelevant (Tom et al. 2013), which helps moderate further accumulation. Developers can also retire TD through refactoring; the model tracks this explicitly. The firm's objective is cumulative discounted profit over a finite horizon.

Results: Our simulations yield five main findings. **Result 1 (AI-assistance sweet spot):** Productivity gains from AI assistance follow an inverted-U pattern. Moderate AI use improves long-run profitability, but heavy reliance triggers a self-reinforcing debt spiral that ultimately collapses both service quality and revenue. **Result 2 (Project horizon):** Short-horizon projects can tolerate and benefit from higher levels of AI assistance, since technical debt requires time to accumulate sufficiently to generate downstream consequences. **Result 3 (Skill level):** Junior developers generate technical debt more rapidly than seniors, so pairing juniors with heavy AI assistance can backfire: early productivity gains erode as compounding maintenance burdens overtake development savings. Counterintuitively, junior developers require *lower* optimal AI assistance than their senior counterparts. **Result 4 (Brownfield environments):** Development atop legacy systems (brownfield) is substantially more vulnerable than greenfield settings. Pre-existing debt immediately lowers productivity and amplifies the AI-driven accumulation rate, shifting the optimal "sweet spot" downward and making technical debt bankruptcy more likely. **Result 5 (TD retirement as guardrail):** Dedicating a meaningful fraction of developer time to retiring technical debt — well above the industry norm of less than 20% — can sustain AI benefits over time. We also show that if AI tools themselves become more effective at supporting refactoring (e.g., identifying duplicate code or improving documentation), they can partially offset the debt they generate.



a) Services b) Technical Debt c) Cumulative Discounted Profit

Optimal AI Strategy: Using Vensim's built-in optimizer, we characterize optimal AI assistance over time under varying conditions. The optimal trajectory is U-shaped: firms deploy AI aggressively early (when no debt has accumulated), taper usage as the codebase matures and debt builds, and surge again near the end of the horizon ("profit-taking"). In brownfield settings, the initial surge is compressed and the taper begins earlier. Optimal AI usage is higher in fast-evolving markets (where debt obsolesce quickly), lower in high-complexity codebases, and lower for junior-heavy teams. Firms facing acute schedule pressure (high discount rates) rationally deploy more AI early, accepting the long-run risk — a finding that helps explain startup and first-mover adoption patterns.

Contributions and Implications: This paper makes three contributions. First, we provide a formal dynamic model showing how technical debt mediates the relationship between AI assistance and long-term software quality and firm profitability — an effect not yet explicitly modeled in the information systems or software engineering literatures. Second, we identify boundary conditions (project horizon, developer skill, legacy-system interaction, market dynamism, debt-management effort) under which AI assistance either enhances or undermines performance. Third, we derive optimal AI adoption strategies across environments, offering actionable managerial guidance beyond the binary "adopt or not" framing. For practitioners, the central message is that AI-assisted development is not merely a choice of productivity tool; it is a *strategic* decision requiring attention to system longevity, architectural complexity, team composition, and sustained investment in technical debt retirement.

References

Aslan M. 2024. The hidden costs of AI coding assistants: insights from a senior developer. Dev Genius (blog). https://blog.devgenius.io

Banker R, Liang Y, Ramasubbu N. 2021. Technical debt and firm performance. Management Science 67(5): 3174–3194.

Becker J, Rush N, Barnes E, Rein D. 2025. Measuring the impact of early-2025 AI on experienced open-source developer productivity. arXiv:2507.09089.

Besker T, Martini A, Bosch J. 2019. Software developer productivity loss due to technical debt—a replication and extension study examining developers' development work. Journal of Systems and Software 156: 41–61.

Besker T, Ghanbari H, Martini A, Bosch J. 2020. The influence of technical debt on software developer morale. Journal of Systems and Software 167: 110586.

Cui ZK, Demirer M, Jaffe S, Musolff L, Peng S, Salz T. 2024. The effects of generative AI on high skilled work: evidence from three field experiments with software developers. SSRN Working Paper 4945566.

Franco EF, Hirama K, Armenia S, dos Santos JR. 2023. A systems interpretation of the software evolution laws and their impact on technical debt management and software maintainability. Software Quality Journal 31(1): 179–209.

GitClear. 2025. AI copilot code quality: evaluating 2024's increased defect rate via code quality metrics. Technical Report. https://www.gitclear.com

Glaiel F, Moulton A, Madnick S. 2014. Agile project dynamics: a system dynamics investigation of agile software development methods. Working Paper CISL 2013-05, MIT Composite Information Systems Laboratory, Cambridge, MA.

He H, Miller C, Agarwal S, Kästner C, Vasilescu B. 2026. Speed at the cost of quality: how Cursor AI increases short-term velocity and long-term complexity in open-source projects. arXiv:2511.04427.

Liu Y, Widyasari R, Zhao Y, Irsan IC, Lo D. 2026. Debt behind the AI boom: a large-scale empirical study of AI-generated code in the wild. arXiv:2603.28592.

McKinsey & Company. 2020. Tech debt: reclaiming tech equity. McKinsey Digital. https://www.mckinsey.com

McKinsey & Company. 2023. Breaking technical debt's vicious cycle to modernize your business. McKinsey Digital. https://www.mckinsey.com

Medappa PK, Tunc MM, Vroegindeweij M, Fransoo JC, et al. 2025. AI-assisted programming may decrease the productivity of experienced developers by increasing maintenance burden. arXiv:2510.10165.

Peng S, Kalliamvakou E, Cihon P, Demirer M. 2023. The impact of AI on developer productivity: evidence from GitHub Copilot. arXiv:2302.06590.

Ramasubbu N, Kemerer CF. 2016. Technical debt and the reliability of enterprise software systems: a competing risks analysis. Management Science 62(5): 1487–1510.

Sterman JD. 2000. Business dynamics: systems thinking and modeling for a complex world. McGraw-Hill: New York.

Tom E, Aurum A, Vidgen R. 2013. An exploration of technical debt. Journal of Systems and Software 86(6): 1498–1516.

Woodard CJ, Ramasubbu N, Tschang FT, Sambamurthy V. 2013. Design capital and design moves: the logic of digital business strategy. MIS Quarterly 37(2): 537–564.