

AI IDEs or Autonomous Agents? Measuring the Impact of Coding Agents on Software Development

Shyam Agarwal

Carnegie Mellon University
Pittsburgh, Pennsylvania, USA

Hao He

Carnegie Mellon University
Pittsburgh, Pennsylvania, USA

Bogdan Vasilescu

Carnegie Mellon University
Pittsburgh, Pennsylvania, USA

Abstract

Large language model (LLM) based coding agents increasingly act as autonomous contributors that generate and merge pull requests, yet their real-world effects on software projects are unclear—especially compared with widely adopted IDE-based AI assistants. We present a longitudinal causal study of agent adoption in open-source repositories using staggered difference-in-differences with matched controls. Using the AIDev dataset, we define adoption as the first agent-generated pull request and analyze monthly repository-level outcomes spanning development velocity (commits, lines added) and software quality (static-analysis warnings, cognitive complexity, duplication, and comment density). Results show large, front-loaded velocity gains only when agents are the first observable AI tool in a project; repositories with prior AI IDE usage experience minimal or short-lived throughput increases. In contrast, quality risks are persistent across settings, with static-analysis warnings and cognitive complexity rising by roughly 18% and 39%, indicating sustained agent-induced technical debt even when velocity advantages fade. These heterogeneous effects suggest diminishing returns to AI assistance and highlight the need for quality safeguards, provenance tracking, and selective deployment of autonomous agents. Our findings establish an empirical basis for understanding how agentic and IDE-based tools interact, and motivate research on balancing acceleration with maintainability in AI-integrated development workflows. The replication package for this study is publicly available at <https://github.com/shyamagarwal13/agentic-coding-impact>.

CCS Concepts

- Software and its engineering → *Development frameworks and environments*; • Computing methodologies → *Intelligent agents*.

Keywords

Autonomous coding agents, Agentic AI, AI-assisted programming, Software quality, Longitudinal study, Causal inference

ACM Reference Format:

Shyam Agarwal, Hao He, and Bogdan Vasilescu. 2026. AI IDEs or Autonomous Agents? Measuring the Impact of Coding Agents on Software Development. In *23rd International Conference on Mining Software Repositories (MSR '26), April 13–14, 2026, Rio de Janeiro, Brazil*. ACM, New York, NY, USA, 5 pages. <https://doi.org/10.1145/3793302.3793589>



This work is licensed under a Creative Commons Attribution 4.0 International License.
MSR '26, Rio de Janeiro, Brazil

© 2026 Copyright held by the owner/authors(s).
ACM ISBN 979-8-4007-2474-9/2026/04
<https://doi.org/10.1145/3793302.3793589>

1 Introduction

The landscape of software development is rapidly changing with the growing use of AI-powered coding tools. Importantly, AI-assisted development is not uniform: two distinct paradigms have emerged. The first consists of *pre-agentic IDE-based coding assistants* such as early GitHub Copilot and Cursor, which integrate directly into developers' environments to provide real-time code suggestions and inline assistance [21, 23, 33]. These tools operate synchronously within the editing workflow, offering suggestions that developers can accept, modify, or reject as they type [24, 30].

The second, more recent paradigm involves *IDE and web-based coding agents*—autonomous AI systems that operate at the repository level to generate entire pull requests, implement features, and make substantial code contributions with minimal human intervention. Tools such as OpenAI Codex [4], Anthropic's Claude Code Agent [2], Devin [6], and the Cursor Agent [5] exemplify this emerging category, where AI systems act more like autonomous contributors. Unlike pre-agentic assistants that augment individual coding sessions, coding agents can work asynchronously, potentially completing full features or bug fixes independently [37]. They differ from traditional assistants in at least four key dimensions: (1) *Autonomy*—ability to complete tasks without continuous human guidance; (2) *Scope*—operation across multiple files and entire codebases; (3) *Planning*—breaking down requirements into subtasks and executing multi-step solutions; and (4) *Interaction*—primarily contributing through pull requests rather than inline suggestions.

Despite the growing use of agentic coding tools in open-source development, empirical research has largely focused on pre-agentic assistants, in part due to the recency of agentic tools as a technology category. Existing evaluations of coding agents primarily rely on benchmarks without humans-in-the-loop [16], limiting insight into impacts on user experience and productivity [20]. In contrast, pre-agentic coding assistants like GitHub Copilot have been extensively studied, with work documenting positive effects on perceived productivity [20, 41, 47]. Research on Copilot spans productivity [22, 39, 48], code quality [32, 36, 46], and developer experience [29, 43], yet the impacts of autonomous, repository-level agents remain largely unexplored. This gap is concerning given that agentic tools operate with greater autonomy, make larger-scale contributions, and introduce distinct challenges [38].

Understanding the impact of agentic coding tools is critical for several reasons. First, these tools are rapidly gaining traction in open-source development, with an increasing number of repositories accepting AI-generated pull requests [44]. Second, the autonomous nature of agentic tools raises unique questions about code quality, maintainability, and technical debt that may not generalize from IDE-based tool studies [25]. Third, the scale of contributions,

entire features versus individual code snippets, necessitates different evaluation frameworks and metrics [20]. Finally, as organizations consider adopting these tools, evidence-based understanding of their impact on development velocity and software quality becomes essential for informed decision-making [14, 19].

In this paper, we use the AIDev dataset [28] to examine how agentic coding tools affect software development velocity and quality at the repository level, reflecting their role as autonomous contributors rather than inline assistants. Focusing on repository-level outcomes, rather than individual-level, captures the fundamentally different operational model of agentic tools. Prior work finds short-term velocity gains but also increased technical debt for repositories adopting the Cursor AI IDE [25]. Using similar causal inference methods [18], we estimate the effects of adopting a wide range of agentic coding tools, comparing the development trajectories of repositories that used pre-agentic coding tools to those that directly adopted coding agents. Our methodology accounts for staggered adoption, heterogeneous treatment effects, and time-varying confounders, enabling a causal assessment of the marginal impact of autonomous coding agents. Our research addresses:

- **RQ1: Development Velocity Impact** – How does the adoption of coding agents affect development velocity?
- **RQ2: Software Quality Impact** – How does the adoption of coding agents affect software quality?
- **RQ3: Prior AI Exposure and Transition Effects** – How do the effects of adopting coding agents differ between repositories with and without prior IDE-based AI assistance?

Our contributions are twofold: (1) We replicate and extend prior results on newer data and a broader ecosystem of autonomous coding agents, providing the first large-scale longitudinal evidence on agentic contributions at the repository level, and (2) We provide the first causal evidence on the differential effects of transitioning from IDE-based AI assistants to autonomous coding agents, showing how prior AI tool adoption affects both velocity and code quality.

2 Related Work

Prior work evaluates AI coding tools using development velocity and software quality metrics. Velocity has been measured via task completion time [35], code volume and throughput [27], and cycle-time reductions [22]. Quality outcomes span security vulnerabilities [36], code churn and revision requirements [44], static analysis warnings and code complexity [25], and broader indicators of technical debt. Methodologically, studies employ randomized trials [35], observational field studies [22], and quasi-experimental designs including difference-in-differences [25].

The vast majority of real-world research focuses on IDE-based AI assistants [15, 17, 22, 31, 32, 34, 36, 39, 40, 44–46], typically reporting modest velocity improvements [25, 26, 41, 47]. In contrast, early studies of agentic tools show mixed results: maintainers merge the majority of Claude Code pull requests (83.8% of 567) [42], yet controlled experiments with tools such as Cursor find limited productivity benefits for experienced open-source developers [16], potentially due to over-optimism, unreliability, and high task complexity. These inconsistencies underscore the need for longitudinal evidence on the real-world effects of autonomous coding agents.

Table 1: Descriptive statistics of treated repositories by prior AI exposure. AF repositories are older but smaller and less popular, while IF repositories are more starred, forked, and active; both exhibit substantial agentic contributions.

Outcome	Mean (AF/IF)	Min (AF/IF)	Median (AF/IF)	Max (AF/IF)
Age (days) [†]	1537.8/1215.2	131/199	1069/926	6141/5570
Stars ^{††}	1460.9/8123.4	10/10	68/423	48110/177379
Forks ^{††}	236.8/1340.2	0/0	12/120	8940/45908
Pull Requests [†]	696.7/2146.0	10/21	122/740	31836/35389
Agentic Pull Requests [†]	121.5/101.5	10/10	28/37	13462/912

Note: [†] as of November end; ^{††} as in original dataset.

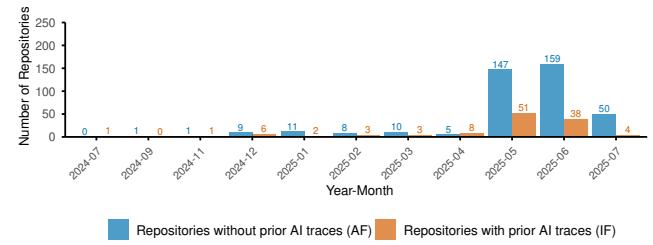


Figure 1: Monthly distribution of agent adoption dates, separated by prior AI exposure. Most adoptions occur between May–July 2025.

3 Methods

We estimate the causal effects of adopting *LLM-based coding agents* on project-level development velocity and software quality using a quasi-experimental difference-in-differences (DiD) design with staggered adoption. Our empirical strategy, outcome definitions, matching procedure, and estimation methods follow similar work on Cursor adoption [25], unless otherwise specified, enabling direct comparability. We summarize the methodology and highlight deviations specific to coding agents.

3.1 Data Collection and Treatment Definition

We build on the AIDev dataset (v3) [28], which links GitHub repositories to AI-generated pull requests and provides explicit evidence of agentic contributions (e.g., multi-file edits, autonomous refactorings, test and documentation generation). For each repository, we define the *agent adoption date* as the earliest month containing an agent-attributed PR. Because AIDev coverage begins in December 2024 while most agentic tools were released earlier that year, we mitigate left-censoring by retrospectively parsing all PRs from January 2024 through November 2025, ensuring that the earliest observed agentic PR closely approximates true adoption timing—crucial for valid temporal ordering in our staggered DiD design.

We extend the original agent taxonomy (Claude [2], Copilot [7], Cursor [5], Devin [6], Codex [4]) to mutually exclusive labels: *human*, *bot*, *codex* [4], *devin* [6], *jules* [8], *cursor* [5], *claude* [2], *copilot* [7], *openhands* [9], *codegen* [3], *cosine* [10], and *tembo* [12]. Attribution follows a cascading strategy prioritizing agent-specific signals before generic labels: (1) branch prefixes (e.g., *cursor/*, *claude/*);

(2) PR author logins (e.g., *codegen-sh*, *tembo-io*); (3) first-commit author names (e.g., *google-labs-jules[bot]*, *claude[bot]*); (4) GitHub actor type *bot*; and (5) default classification as *human*. For Claude-specific attribution, we additionally search PR descriptions, comments, and reviews for distinctive co-authorship patterns (“Generated with Claude Code”, optionally including “Co-Authored-By: Claude <no-reply@anthropic.com>”). This multi-signal approach improves recall for agentic PRs lacking explicit bot authorship or branch naming.

By following this convention, we identified several misclassifications and missing PRs in the original dataset (replication package). Correctly recovering the *earliest* agentic PR is essential because adoption timing determines treatment assignment in our causal design; any remaining attribution errors primarily introduce noise in treatment timing rather than systematic bias, likely attenuating effects toward zero.

We restrict analysis to repositories with ≥ 10 stars to exclude toy and spam projects and require ≥ 10 agentic PRs per treated repository to ensure non-trivial exposure. Repositories are observed monthly before and after adoption, forming an unbalanced panel with staggered timing. To examine moderation by prior AI usage, we partition treated repositories into *agent-first* (AF), which show no traces of AI IDEs throughout the collection time period, and *IDE-first* (IF), which exhibit IDE activity prior to agent adoption. Prior IDE usage is detected monthly by scanning the most recent commit for configuration artifacts associated with agent-enabled IDEs (GitHub Copilot [7], Cursor [5], WindSurf [13]). A repository is IF if such artifacts appear before its first agentic PR and AF if they don’t exist at all. Analyses are conducted separately for AF and IF using control repositories that mirror the same IDE exposure pattern. Figure 1 shows both groups adopt primarily between April–June 2025, with AF more numerous. Table 1 shows AF repos are older but smaller and less popular, whereas IF repos are more starred, forked, and PR-active; both groups nonetheless exhibit substantial agentic PR volume, motivating separate analyses.

Our control set consists of all GitHub repositories with ≥ 10 stars at collection time. For each month with agent adoption, we extract monthly activity series from GHArchive [1] for repositories with at least one event (age, active users, stars, forks, releases, pull requests, issues, comments, total events). Because inferring prior IDE exposure at this scale is initially infeasible, we first perform propensity-score matching over the full control population without conditioning on prior exposure. We then infer prior exposure only for matched candidates and retain matched control sets that mirror treated repositories’ AF/IF status. After filtering, our heterogeneous samples contain 401 AF repositories matched to 606 controls and 117 IF repositories matched to 73 controls.

3.2 Outcomes and Covariates

We measure development velocity via monthly commit counts and lines added; software quality via static-analysis warnings, duplicated-line density, and cognitive complexity using SonarQube[11]. All outcomes are aggregated at the repository-month level. We control for time-varying covariates: lines of code, repository age (days), contributors, stars received, issues opened, and issue comments added in the observation month. Lines of code and outcome metrics

Table 2: The Borusyak et al. [18] estimated average post-adoption treatment effects of agentic coding tools separated by prior AI exposure. The estimate and standard error are log-transformed to facilitate easy comparison.

Outcome	β (AF / IF)	Std. Err. (AF / IF)	% Change (AF / IF)
Commits	0.309***/0.030	(0.051)/(0.092)	36.25 / 3.06
Lines Added	0.569***/-0.066	(0.103)/(0.189)	76.59 / -6.34
Duplicate Line Density	0.076/-0.009	(0.044)/(0.056)	7.92 / -0.94
Comment Line Density	0.042/0.201***	(0.028)/(0.055)	4.34 / 22.30
Static Analysis Warnings	0.163***/.0174	(0.048)/(0.114)	17.73 / 19.00
Code Complexity	0.299***/0.357**	(0.059)/(0.114)	34.85 / 42.87

Note: * $p < 0.05$; ** $p < 0.01$; *** $p < 0.001$.

come from SonarQube [11], contributor counts from version history, and activity covariates from GHArchive [1]. These covariates are collected consistently for both treated and control repositories across the full observation window, ensuring comparable temporal coverage and supporting adjustment for confounding dynamics unrelated to agent adoption.

3.3 Causal Inference Strategy

We estimate effects using DiD with staggered adoption: later adopters serve as controls for earlier adopters prior to adoption; never-adopters provide additional counterfactuals. To address selection bias of control repositories and satisfy the conditional independence assumption to establish the quasi-experimental settings, we perform propensity score matching prior to estimation. Propensity scores use logistic regression over dynamic pre-treatment characteristics, incorporating both activity *levels* (recent covariates) and *trajectories* (lagged covariates) to capture whether repositories are growing, declining, or stable. Specifically, we model adoption likelihood using repository age at the month prior to observation, six monthly covariate lags, and cumulative historical covariates, enabling discrimination between repositories with similar recent activity but different long-term trends. Because candidates outnumber adopters, we subsample at most 10,000 candidates/month, yielding propensity models with AUC 0.92–0.99. Treated repositories are matched to maximum three controls with similar propensity scores and the same primary language, ensuring comparable pre-adoption activity, exposure histories, and language-specific performance characteristics. We estimate average and dynamic treatment effects using the imputation-based DiD estimator of Borusyak et al. [18], which avoids biases from traditional two-way fixed effects models under staggered adoption and heterogeneous effects. Event-study specifications are used to assess pre-treatment trends and to characterize how effects evolve over time. Standard errors are clustered at the repository level. As in prior work, our estimates should be interpreted as intent-to-treat effects of observable agent adoption. While we cannot directly measure usage intensity or developer-level interactions, staggered adoption and matching on pre-treatment dynamics strengthen the causal interpretation of our findings.

4 Results and Discussion

Table 2 summarizes average post-adoption effects, while Figure 2 shows six-month dynamic estimates.

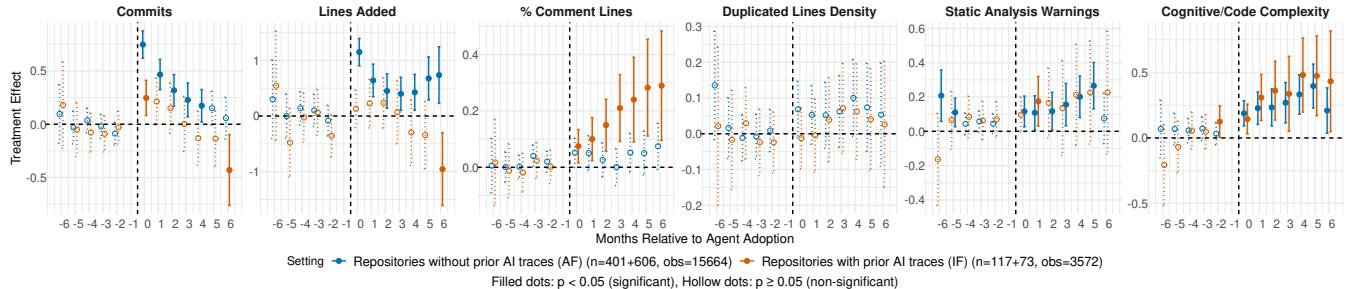


Figure 2: Estimated post-adoption effects of agentic coding tools by prior AI exposure. AF repositories gain velocity and accumulate maintainability risks; IF repositories show minimal velocity gains but comparable maintainability increases.

Development Velocity. Agentic tools substantially accelerate development only when introduced as a repository’s first observable AI tool. AF repos see large average gains (+36.3% commits; +76.6% lines added), whereas IF repos show minimal changes (+3.1%; −6.3%). Dynamics reveal a sharper contrast: AF repositories experience a spike at $t=0$ (about +111% commits; +216% lines added) and persistently elevated activity, with lines added remaining roughly +49–+109% through $t=6$. IF repositories show only a short-lived bump around adoption (commits +16–28% at $t=0$ –2) before estimates return near zero and eventually turn negative (lines $\sim -61\%$, commits $\sim -35\%$ by $t=6$).

These patterns indicate that agentic tools act as high-throughput contributors primarily in new-to-AI workflows, but yield diminishing returns in AI-saturated ones. AF repositories appear able to harvest “first AI” acceleration, while IF repositories, having already absorbed productivity gains from AI IDEs, likely face higher coordination and integration costs that limit throughput. The greater maturity of IF repos (higher stars, forks, PR volume; from Table 1) likely constrains how aggressively agentic changes can be merged, so that localized speedups are offset by triage and review overhead. Overall, agent adoption yields sustained velocity gains only when not preceded by IDE-based AI tooling.

Software Quality. Regardless of prior AI exposure, adoption is associated with increased maintainability risks. Across both AF and IF repositories, static-analysis warnings rise by about 18% and cognitive complexity by roughly 39% (Table 2). Dynamics show persistent complexity accumulation: AF repos increase +20.7% at $t=0$ and reach $\sim +49\%$ by $t=5$, while IF repositories become significantly elevated as early as $t=-2$ and remain $\sim +15$ –+62% through $t=6$. Warning growth is somewhat noisier but similarly persistent ($\sim +22$ –+31% in AF by $t=4$ –5; IF values consistently positive around +25% at $t=4$ –6).

These simultaneous increases in complexity and warnings, even when net velocity gains are weak or negative (IF), indicate *agent-induced complexity debt*: agents accelerate the introduction of code that raises long-term cognitive and maintenance load. Duplication effects are small and inconsistent, suggesting quality risks stem from structural complexity rather than copy–paste proliferation. Comment density diverges across groups: IF repositories show substantial and sustained increases (about +22% on average and $> +30\%$ by $t=6$), whereas AF effects are muted, hinting that teams already using AI IDEs rely on agents for documentation as well as

code. While additional comments can aid comprehension, they do not counterbalance persistent complexity growth. Relative to prior work on AI IDEs [25], which finds modest productivity gains and mixed quality effects, these findings suggest autonomous agents amplify the speed–maintainability trade-off, and in AI-rich environments may magnify complexity without delivering sustained velocity benefits. We also observe isolated significant pre-treatment coefficients in static-analysis warnings and code complexity, suggesting that untreated potential outcomes are not fully captured by additive fixed effects, reflecting systematic mean differences between treated and matched controls. These are not pervasive enough to indicate a sustained pre-trend or clear violation of parallel trends, but they are concerning and highlight a limitation of our quasi-experimental design and the underlying data.

Implications and Ethical Considerations Autonomous coding agents function as powerful but risky accelerators whose value depends on prior AI exposure: in our sample substantial front-loaded velocity gains materialize only when agents are a project’s first AI tool; more broadly, velocity increases likely won’t suffice, and AI adoption will need to be paired with strong quality safeguards (e.g., complexity-aware review of agent pull requests, routine refactoring, comprehensive automated tests) to prevent accumulating complexity debt. Teams using AI IDEs should not assume additive productivity and may instead deploy agents selectively or for tightly scoped tasks. Persistent complexity growth underscores the need to surface maintainability metrics directly in agent planning and prompting, while modulating behavior according to existing AI usage. Ethically, increased automation shifts accountability and maintainability burdens; provenance tracking, transparency of agent-generated changes, and review practices that emphasize human oversight are still essential to avoid debt.

5 Conclusion

Autonomous agents offer meaningful velocity gains only in new-to-AI settings while consistently raising complexity and warning levels across contexts, reinforcing a speed–maintainability trade-off. Prior exposure to AI IDEs moderates benefits but not risks, underscoring the need for selective deployment and active oversight. Future work should follow post-adoption trajectories over longer horizons and examine collaborative patterns that balance acceleration with sustained code quality.

References

- [1] 2011. *GHArchive*. Retrieved Nov 30, 2024 from <https://www.gharchive.org/>
- [2] 2025. *Claude Code / Claude*. Retrieved Nov 30, 2025 from <https://claude.com/product/claude-code>
- [3] 2025. *Codegen / The OS for Code Agents*. Retrieved Nov 30, 2025 from <https://codegen.com/>
- [4] 2025. *Codex / OpenAI*. Retrieved Nov 30, 2025 from <https://openai.com/codex/>
- [5] 2025. *Cursor - The AI Code Editor*. Retrieved Nov 30, 2025 from <https://www.cursor.com/>
- [6] 2025. *Devin / The AI Software Engineer*. Retrieved Nov 30, 2025 from <https://devin.ai/>
- [7] 2025. *GitHub Copilot - Your AI pair programmer*. Retrieved Nov 30, 2025 from <https://github.com/features/copilot>
- [8] 2025. *Jules - An Autonomous Coding Agent*. Retrieved Nov 30, 2025 from <https://jules.google/>
- [9] 2025. *OpenHands*. Retrieved Nov 30, 2025 from <https://app.all-hands.dev/>
- [10] 2025. *Self sufficient agentic AI software engineer / Cosine AI*. Retrieved Nov 30, 2025 from <https://cosine.sh/>
- [11] 2025. *SonarQube Community Build Documentation*. Retrieved Nov 30, 2025 from <https://docs.sonarsource.com/sonarqube-community-build/>
- [12] 2025. *Tembo – Delegate work to any coding agent*. Retrieved Nov 30, 2025 from <https://www.tembo.io/>
- [13] 2025. *Windsurf - The best AI for Coding*. Retrieved Nov 30, 2025 from <https://windsurf.com>
- [14] Magnus Chukwuebuka Ahuchogu, Pravin Ganpatrao Gawande, Dr Charu Mohla, Dr Deepak A. Vidhate, and Nidal Al Said. 2025. Evaluating the Impact of Generative AI on Intelligent Programming Assistance and Code Quality. *Power System Technology* (2025).
- [15] Owura Asare, Meiyappan Nagappan, and Nirmal Asokan. 2022. Is GitHub's Copilot as bad as humans at introducing vulnerabilities in code? *Empirical Software Engineering* 28 (2022), 1–24.
- [16] Joel Becker, Nate Rush, Elizabeth Barnes, and David Rein. 2025. Measuring the Impact of Early-2025 AI on Experienced Open-Source Developer Productivity. *ArXiv abs/2507.09089* (2025).
- [17] Marta Borek and Robert Nowak. 2025. Quality evaluation of Tabby coding assistant using real source code snippets. *ArXiv abs/2504.08650* (2025).
- [18] Kirill Borusyak, Xavier Jaravel, and Jann Spiess. 2021. Revisiting event study designs: robust and efficient estimation.
- [19] Eric W. Bridgeford, Iain Campbell, Zijiao Chen, Zhicheng Lin, Harrison Ritz, Joachim Vandekerckhove, and R.A. Poldrack. 2025. Ten Simple Rules for AI-Assisted Coding in Science. *ArXiv abs/2510.22254* (2025).
- [20] Valerie Chen, Ameet Talwalkar, Robert Brennan, and Graham Neubig. 2025. Code with Me or for Me? How Increasing AI Automation Transforms Developer Workflows. *ArXiv abs/2507.08149* (2025).
- [21] Ruijia Cheng, Ruotong Wang, Thomas Zimmermann, and Denae Ford. 2022. "It would work for me too": How Online Communities Shape Software Developers' Trust in AI-Powered Code Generation Tools. *ACM Transactions on Interactive Intelligent Systems* 14 (2022), 1 – 39.
- [22] Lena Chretien and Nikolas Albarban. 2024. Impact of AI-tooling on the Engineering Workspace. *ArXiv abs/2406.07683* (2024).
- [23] Mariana Coutinho, Lorena Marques, Anderson Santos, Marcio Dahia, César França, and Ronnie de Souza Santos. 2024. The Role of Generative AI in Software Development Productivity: A Pilot Case Study. *Proceedings of the 1st ACM International Conference on AI-Powered Software* (2024).
- [24] Emanuela Guglielmi, Venera Arnoudova, Gabriele Bavota, Rocco Oliveto, and Simone Scalabrino. 2025. How do Copilot Suggestions Impact Developers' Frustration and Productivity? *ArXiv abs/2504.06808* (2025).
- [25] Hao He, Courtney Miller, Shyam Agarwal, Christian Kastner, and Bogdan Vasilescu. 2026. Speed at the Cost of Quality: How Cursor AI Increases Short-Term Velocity and Long-Term Complexity in Open-Source Projects. In *International Conference on Mining Software Repositories (MSR)*.
- [26] Saki Imai. 2022. Is GitHub Copilot a Substitute for Human Pair-programming? An Empirical Study. *2022 IEEE/ACM 44th International Conference on Software Engineering: Companion Proceedings (ICSE-Companion)* (2022), 319–321.
- [27] Anand Kumar, Vishal Khare, Deepak Sharma, Satyam Kumar, Vijay Saini, Anshul Yadav, Sachendra Jain, Ankit Rana, Pratham Verma, Vaibhav Meena, and Avinash Edubilli. 2025. Intuition to Evidence: Measuring AI's True Impact on Developer Productivity. *ArXiv abs/2509.19708* (2025).
- [28] Hao Li, Haoxiang Zhang, and Ahmed E. Hassan. 2025. The Rise of AI Teammates in Software Engineering (SE) 3.0: How Autonomous Coding Agents Are Reshaping Software Engineering. *arXiv preprint arXiv:2507.15003* (2025).
- [29] Jenny T Liang, Chenyang Yang, and Brad A. Myers. 2023. A Large-Scale Survey on the Usability of AI Programming Assistants: Successes and Challenges. *2024 IEEE/ACM 46th International Conference on Software Engineering (ICSE)* (2023), 616–628.
- [30] Hussein Mozannar, Gagan Bansal, Adam Fournier, and Eric Horvitz. 2022. Reading Between the Lines: Modeling User Behavior and Costs in AI-Assisted Programming. *ArXiv abs/2210.14306* (2022).
- [31] Kevin KB Ng, Liyana Fauzi, Leon Leow, and Jaren Ng. 2024. Harnessing the Potential of Gen-AI Coding Assistants in Public Sector Software Development. *ArXiv abs/2409.17434* (2024).
- [32] Nhan Ton Nguyen and Sarah Nadi. 2022. An Empirical Evaluation of GitHub Copilot's Code Suggestions. *2022 IEEE/ACM 19th International Conference on Mining Software Repositories (MSR)* (2022), 1–5.
- [33] Ruchika Pandey, Prabhat Singh, Raymond Wei, and Shaila Shankar. 2024. Transforming Software Development: Evaluating the Efficiency and Challenges of GitHub Copilot in Real-World Projects. *ArXiv abs/2406.17910* (2024).
- [34] Elise Paradis, Kate Grey, Quinn Madison, Daye Nam, Andrew Macvean, Vahid Meimand, Nan Zhang, Ben Ferrari-Church, and Satish Chandra. 2024. How Much Does AI Impact Development Speed? an Enterprise-Based Randomized Controlled Trial. *2025 IEEE/ACM 47th International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEP)* (2024), 618–629.
- [35] Elise Paradis, Ambar Murillo, Maulishree Pandey, Sarah D'Angelo, Andrew Macvean, Ben Ferrari-Church, and Matthew Hughes. 2025. Creating benchmarkable components to measure the quality of AI-enhanced developer tools. *Proceedings of the Extended Abstracts of the CHI Conference on Human Factors in Computing Systems* (2025).
- [36] Hammond A. Pearce, Baleegh Ahmad, Benjamin Tan, Brendan Dolan-Gavitt, and Ramesh Karri. 2021. Asleep at the Keyboard? Assessing the Security of GitHub Copilot's Code Contributions. *2022 IEEE Symposium on Security and Privacy (SP)* (2021), 754–768.
- [37] Abhil Roychoudhury. 2025. Agentic AI for Software: thoughts from Software Engineering community. *ArXiv abs/2508.17343* (2025).
- [38] Ranjan Sapkota, Konstantinos I. Roumeliotis, and Manoj Karkee. 2025. Vibe Coding vs. Agentic Coding: Fundamentals and Practical Implications of Agentic AI. *ArXiv abs/2505.19443* (2025).
- [39] Md. Istiak Hossain Shihab, Christopher Hundhausen, Ahsun Tariq, Summit Haque, Yunhan Qiao, and Brian Mulanda. 2025. The Effects of GitHub Copilot on Computing Students' Programming Effectiveness, Efficiency, and Processes in Brownfield Coding Tasks. *Proceedings of the 2025 ACM Conference on International Computing Education Research V.1* (2025).
- [40] Antero Taivalsaari, Tommi Mikkonen, and Cesare Pautasso. 2025. On the Future of Software Reuse in the Era of AI Native Software Engineering. *ArXiv abs/2508.19834* (2025).
- [41] Priyan Vaithilingam, Tianyi Zhang, and Elena L. Glassman. 2022. Expectation vs. Experience: Evaluating the Usability of Code Generation Tools Powered by Large Language Models. *CHI Conference on Human Factors in Computing Systems Extended Abstracts* (2022).
- [42] Miku Watanabe, Hao Li, Yutaro Kashiwa, Brittany Reid, Hajimu Iida, and Ahmed E Hassan. 2025. On the use of agentic coding: An empirical study of pull requests on GitHub. *CoRR abs/2509.14745* (2025).
- [43] Justin D. Weisz, Shraddha Kumar, Michael J. Muller, Karen-Ellen Browne, Arielle Goldberg, Katrin Ellice Heintze, and Shagun Bajpai. 2024. Examining the Use and Impact of an AI Code Assistant on Developer Productivity and Experience in the Enterprise. *Proceedings of the Extended Abstracts of the CHI Conference on Human Factors in Computing Systems* (2024).
- [44] Tao Xiao, Youmei Fan, Fabio Calefato, Christoph Treude, Raula Gaikovina Kula, Hideaki Hata, and Sebastian Baltes. 2025. Self-Admitted GenAI Usage in Open-Source Software. *ArXiv abs/2507.10422* (2025).
- [45] Feiyang Xu, Poonacha K. Medappa, Murat Mustafa Tunç, Martijn Vroegeindeweijs, and Jan C Fransoo. 2025. AI-assisted Programming May Decrease the Productivity of Experienced Developers by Increasing Maintenance Burden. *ArXiv abs/2510.10165* (2025).
- [46] Burak Yetistiren, Isik Ozsoy, and Eray Tüzün. 2022. Assessing the quality of GitHub copilot's code generation. *Proceedings of the 18th International Conference on Predictive Models and Data Analytics in Software Engineering* (2022).
- [47] Albert Ziegler, Eirini Kalliamvakou, Shawn Simister, Ganesh Sittampalam, Alice Li, Andrew Rice, Devon Rifkin, and Edward Aftandilian. 2022. Productivity assessment of neural code completion. *Proceedings of the 6th ACM SIGPLAN International Symposium on Machine Programming* (2022).
- [48] Albert Ziegler, Eirini Kalliamvakou, LI X.ALICE, Andrew Rice, Devon Rifkin, Shawn Simister, Ganesh Sittampalam, and Edward Aftandilian. 2024. Measuring GitHub Copilot's Impact on Productivity. *Commun. ACM* 67 (2024), 54 – 63.