Do These Violent Delights Have Violent Ends? Measuring the Post-Merge Fate of Agentic Code

Chunqiu Steven Xia
University of Illinois Urbana-Champaign
Urbana, IL, USA
chunqiu2@illinois.edu

Courtney Miller
George Washington University
Washington, DC, USA
courtney.miller@gwu.edu

***Abstract*—Agentic coding tools are increasingly used to make autonomous repository-level changes to real-world projects. Prior work has largely evaluated these contributions at the pre-merge stage, through outcomes such as pull request acceptance and review effort. Far less is known about what happens to agentic code post-merge. Yet merge success alone does not reveal whether a contribution will remain stable or require bug fixes and other corrective maintenance downstream. We conduct a longitudinal empirical analysis of agentic and human contributions across 182 repositories, tracking their post-merge fate over time, characterizing the intent of subsequent modifications, and analyzing the defects and vulnerabilities they introduce. While the overall maintenance rates are similar, agentic contributions require significantly higher rates of corrective maintenance and introduce more security weaknesses and dependency vulnerabilities. We also find statistically significant evidence that agentic maintenance burden is associated with repository characteristics. In particular, each 10 percentage-point increase in a project's no-review rate is associated with roughly a 6% increase in agentic maintenance burden on average. As coding agents become pervasive in software development, our findings highlight the need to evaluate and design agentic tools not only to produce mergeable changes, but to produce contributions that remain secure and maintainable.**

I. Introduction

The case for adopting Generative AI (GenAI) agentic coding tools is made almost entirely based on tool-centric metrics: how much the tools produce and what their output looks like at merge. Microsoft reports GenAI produces close to 30% of its codebase [1], Meta aims for agentic coding tools to handle half of its development by late 2026 [2], and Google reports that 75% of their new code is produced by GenAI [3]. In a recent survey of more than 200 technology decision-makers, 67% claim that agentic coding tools write over half of their organization's weekly code [4]. These are the numbers used to justify adoption. They say little about that code's sustainability, reliability, security, or maintainability in the long term post-merge. What happens to agentic code once it becomes part of the software system remains largely unstudied.

This tool-centric framing has revived evaluation metrics that are weak proxies for the quality of what actually ships: lines of code written, number of merged pull requests, percentage of code written by GenAI, and even "tokenmaxxing" (amount of vendor tokens spent) [5]. Such metrics make for impressive growth graphs, but they reveal little about how that change behaves as it evolves with the project.

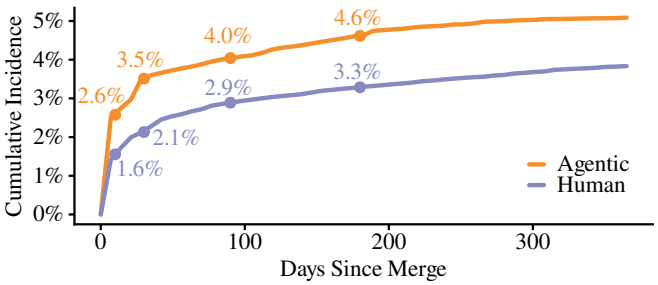


Fig. 1: CIF for corrective terminations. Dots represent time points of 10/30/90/180 days since merge.

In fields like medicine and aviation, performance claims about tools that carry safety risks are expected to be substantiated before the tool is relied on. Medical devices like surgical robots must clear thorough regulatory review processes for safety and effectiveness before they can get FDA approval for use. Agentic coding tools face no comparable bar before bold sweeping promises are made and they are deployed to write code for banks, hospitals, and other critical systems. And whatever scrutiny the output receives largely occurs post-merge by the people who already paid for the tool and now potentially face another cost much greater. The actualized burden of this is already a common complaint among developers [6], with many describing a dynamic where generation is cheap and then the arduous work of review and clean up falls on their shoulders [7]. While practitioners have articulated this loudly and clearly, it has been argued largely from anecdote rather than measured in the code itself.

Almost everything we know about agentic code quality comes from measurements taken at or before merge: performance on benchmarks [8], pre-merge checks, and pull request acceptance rates and review outcomes [9], [10]. But merge only marks the beginning of a contribution's life in a project. What happens to agentic code after merge, and how it compares to human code over time, remains largely unstudied.

Our work. To bridge this gap, we present the first longitudinal empirical study of the post-merge lifecycle of agentic and human code in real-world open source projects by exploring the following research questions (RQs):

RQ1a Does agentic code require more post-merge mainte-

nance compared to human code?

RQ1b Does agentic code require different types of post-merge maintenance compared to human code?

RQ1c Does agentic code require different types of corrective maintenance and introduce more defects compared to human code?

RQ2 Does corrective maintenance burden increase in projects as the proportion of agentic code increases over time?

RQ3 What project characteristics are associated with increased agentic maintenance burden?

We study 182 repositories with sustained agentic contributions, tracking what happens to agentic and human contributions over time. We collect commits merged over a one-year window from May 1st 2025 to May 31st 2026. For each commit, we identify whether it was contributed by an agentic coding tool or human, and we classify its maintenance purpose at two levels: a high-level *maintenance class* (e.g., corrective or perfective) and a finer-grained *operational intent* (e.g., bug fix, feature, or refactor). For each line in a commit, we perform a line-level lifecycle analysis that tracks whether the line survived or was terminated, when, and by which later commit, letting us connect every termination to the intent of the commit responsible for it. For each commit we also run Semgrep to identify source-code vulnerabilities and OSV-Scanner to identify dependency vulnerabilities, and we use SonarQube to take weekly snapshots of each repository over the observation window. We then apply survival analysis, longitudinal panel models, and meta-regression to answer our RQs.

Our findings show that while agentic code does not have a different maintenance burden in aggregate, that flat trend results from significant heterogeneity in the agentic maintenance burden across projects. The composition of maintenance also differs: agentic code receives a 46% higher corrective maintenance rate and a 45% higher bug-fixing rate on average (cf. Figure 1), and agentic contributions introduce more security weaknesses and dependency vulnerabilities at merge. Agentic corrective maintenance burden also accumulates: within a project, a higher share of agentic code in one month is associated with a higher corrective maintenance rate the following month, including corrective work performed by human developers.

Our model demonstrates the cross-project variation in maintenance burden is not random: a project's agentic maintenance burden is associated with characteristics of the project itself. Of the project characteristics we evaluate, a project's no-review merge rate has the strongest relationship with agentic maintenance burden, with each 10 percentage-point increase in a project's no-review rate being associated with a roughly 6% increase in agentic maintenance burden on average. Weaker test infrastructure and higher technical debt also both point in the same direction without reaching significance, together suggesting a picture where weaker engineering safeguards are associated with higher agentic maintenance burden.

In summary, this paper makes the following contributions: (1) A longitudinal dataset of agentic and human contributions across 182 repositories, linking each commit to its maintenance intent, line-level survival outcome, and static analysis findings. (2) A comprehensive post-merge comparison of agentic and human contributions, covering overall maintenance activity, the composition and rate of corrective maintenance, the accumulation of corrective burden as agentic code share grows over time, and some of the project characteristics associated with agentic maintenance burden.

II. Background and Related Work

A. LLMs and Coding Agents

With the recent rise of large language models (LLMs) and their high benchmark performance on software engineering tasks [11], developers have started adopting LLMs as part of their development workflows. Initially, *LLM-based coding assistants* such as GitHub Copilot [12] and Cursor [13] were developed to be integrated into developers' IDEs. These coding assistants provide inline code completion suggestions based on the surrounding code context which users could accept, modify, or reject. More recently, *autonomous coding agents* have emerged to move beyond inline completion to perform repository-level development tasks, e.g., feature implementation, refactoring, code execution, and pull request submission. Tools like Anthropic's Claude Code [14] and OpenAI's Codex [15] have access to the full codebase with a chat-based interface that allows the user to interact and collaborate to perform development tasks. In this sense, the usage of coding agents shifts the developer's role from manual coding toward high-level task orchestration with the promise of drastically improving developer efficiency [16]. GitHub reports more than 1 million open-source repositories used GenAI coding tools between 2024 and 2025 [17], and OpenAI reports more than 5 million weekly Codex users as of June 2026 [18], with a similar number of users being estimated for Claude Code [19]. These numbers suggest the adoption of coding agents shows no sign of slowing down. However, despite this rapid adoption and the promise of improved developer productivity, our understanding of how coding agents affect software development remains limited. Questions about how developers interact with agentic contributions, how those contributions differ from human code, and what longitudinal maintenance, correctness, and security risks they introduce must be continually examined.

B. Coding Agents Impact on Software Development

Driven by the recent adoption of GenAI-based tools, researchers have begun evaluating their impact on software development. Initial research mainly focuses on LLM-based coding assistants across areas of code quality [20], [21], [22], code security [23], [24], [25], maintainability [26], [27], and usability and productivity [28], [29], [30], [31], [32], [33], [34], [35]. These studies report mixed productivity effects, ranging from moderate gains [35], [36], [37], [38] to measurable decreases [39], [40]. At the same time, growing evidence raises concerns about the trustworthiness of generated code, including security vulnerabilities [41], [23], regressions [42], code smells [43], and outdated API usage [44].

Metric	Mean	Min	Max
Stars	4,867.2	10	45,401
Forks	945.6	0	15,123
Human contributors	70.1	1	443
Unique AI agents used	2.4	1	5
Commits	2,273.5	153	19,934
AI-coauthored commits	169.8	1	2,397
Human commits	1,860.7	42	18,711

TABLE I: Descriptive statistics of our dataset.

Agents considered
Claude Code, Codex, GitHub Copilot, Cursor, Devin, Aider, OpenHands, Cline, Roo Code, Windsurf, Sweep, Replit Agent, Jules, Gemini, Amazon Q

TABLE II: GenAI coding agents targeted in study.

Maintenance Class	Operational Intent
Corrective	Security fix, Revert, Bug fix
Adaptive	Dependency update
Perfective	Performance, Feature, Documentation, Resource
Preventive	Refactor, Test, Style/Formatting
Management	Merge/Release/Versioning, Build/Config/CI

TABLE III: Two levels of maintenance intent categories.

More recently, following the adoption trend, researchers have begun studying the impact of autonomous coding agents on software development [45], [46]. Watanabe et al. [9] found that more than 80% of Claude Code pull requests were eventually merged, with less than 50% of agentic pull requests requiring additional human revisions. He et al. [47] studied the impact of Cursor adoption across 807 GitHub repositories and found that although project development velocity increased temporarily, code complexity and static analysis warnings also increased persistently, ultimately slowing project velocity in the long term. Researchers have also begun collecting datasets of real-world agentic coding tool usage, including AIDev [48], which captures more than 450,000 agentic pull requests, and Agents in the Wild [49], which continuously tracks agentic pull requests on GitHub. Using and building on these datasets, recent studies have examined agentic pull-request outcomes and failure modes [50], [51], review dynamics [52], [53], [10], code change characteristics [54], [55], and repository-level outcomes [56], [57]. Taken together, these studies show that while autonomous coding agents can produce mergeable pull requests at a high rate, they may also strain the review process and increase project complexity.

However, most existing work focuses on the pre-merge stage. A few studies include brief post-merge analyses, but only through narrow windows. Rahman and Shihab [58] focus purely on agentic code survival and use only coarse modification intent while Liu et al. [46] focuses on the persistence of static-analysis issues introduced by agentic commits. Both provide post-merge signals, but neither reconstructs the broader maintenance trajectory of agentic contributions. Our study addresses this gap by following agentic and human code after merge, identifying the later commits that modify them, characterizing the intent of those changes, measuring defects and vulnerabilities introduced, and studying how agentic code share and project characteristics affect maintenance burden.

III. Methods

Table I shows the overall summary characteristics of the repositories in our dataset. We conduct our longitudinal analysis over an observation window from May 1st 2025 to May 31st 2026. We start at May 2025 because that is when we observe a meaningful increase in agentic contributions. Figure 2 shows the overview of our data collection approach.

1 ***Project Identification.*** We start by identifying projects with agentic contributions. We use the Agents in the Wild dataset [49] which contains pull requests (PRs) made by AI-coding agents on public GitHub repositories. This yields an initial set of 2,860 projects with at least 10 stars and at least one human and one agentic PR. We remove 29 projects from the top and bottom 1% of the star distribution to reduce the influence of extreme popularity outliers. Next, to filter out projects without sustained agentic activity, we exclude 2,631 projects without at least 10 agentic and 10 human PRs overall, at least three months of agentic PRs, and at least 10 commits per month over the past six months. We further filter out one non-source-code project. Lastly, we remove 17 extremely large projects, whose size hinders our lifecycle and static analysis to end with 182 total projects in our dataset.

2 ***Commit & PR Collection & Classification.*** For each project in our dataset, we collect all commits merged during our observation window. To identify agentic commits, we use a simple classification procedure that matches commit author name, email, and commit signature against known agent names (e.g., `Co-authored-by: Claude <noreply@anthropic.com>`). Table II lists the AI coding agents that we specifically target. In addition, we collect basic PR review metrics including the number of comments and review threads.

3 ***Commit Maintenance Intent Classification.*** We classify the maintenance intent of each commit at two levels (cf. Table III): (1) high-level *maintenance class* and (2) low-level *operational intent*. For maintenance class, we extend the classic Swanson's taxonomy [59], [60] with an additional *management* category for commits related to merging, release, and versioning which are not captured by the original taxonomy. For operational intent, we borrow concepts from prior large-scale study [61], [62] as classification signals.

We first classify each commit into the low-level operational intent and then map that intent to the high-level maintenance class according to Table III. We begin by tagging the modified files by file types (*source code*, *test*, *doc*, *config*, *dependency*, and *resource*) based on file extensions and paths. We then use these file-type tags to assign scores to specific intents. For example, a commit that modifies only test files will have scores added to the test intent. Next, we analyze the commit

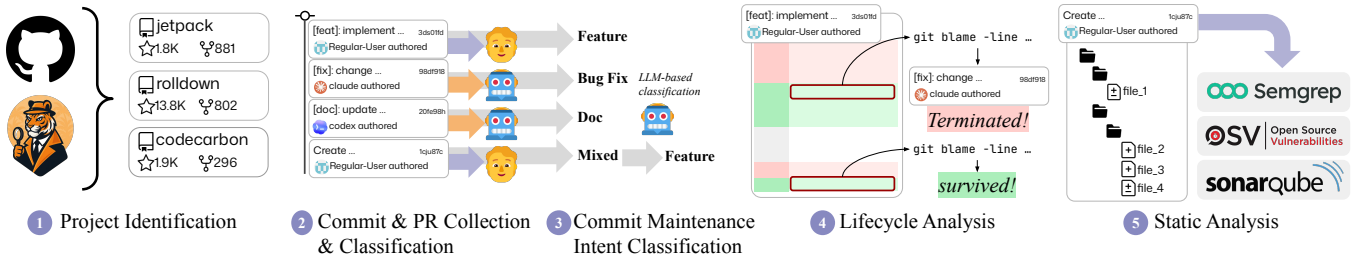


Fig. 2: Overview of the data collection process.

message subject by first checking for conventional commit prefixes [63] and mapping them to operational intents. We then apply keyword regexes commonly associated with each operational intent on the commit subject text. Finally, we obtain a score distribution over all operational intents and determine a primary intent by selecting the intent with the highest score. We also categorize commits as *mixed* if there are multiple intents with the same score or as *unknown* if no intent receives a sufficiently high score.

To handle unknown or mixed commits, we perform an LLM-based reclassification. Specifically, we prompt the LLM (we use MiniMax-M2.7 [64]) with the commit message subject, body, and modified filenames and ask it to classify the operational intent. To validate our commit intent classifier, one author manually labeled a random sample of 100 commits and compared the human labels against the classifier outputs, yielding a Cohen's kappa [65] of 0.94.

4 ***Lifecycle Analysis.*** To study the lifecycle of agentic code at scale, we track all introduced lines and determine whether they have been modified during our observation window. For each line added by a commit, we classify it as (1) *survived*: not modified or removed by any future commits; or (2) *terminated*: modified or removed by a future commit. For each terminated line, we also identify the exact commit, referred to as the terminal commit, that performed the modification.

At a high level, our lifecycle analysis consists of a diff-tracking pass followed by a `git blame`-based verification pass. First, for each added line in a commit, we replay subsequent commits in topological order and update each line's current file and line number as later diffs shift the surrounding code. When a later diff touches the tracked line's current location, we provisionally mark the line as *terminated* and record the modifying commit as the terminal commit; otherwise, the line remains *survived*. The diff-tracking pass provides an efficient first approximation of each line's location and possible terminal commit. Next, we use `git blame` at the end of the observation window to determine whether the original line is still present. If the line is still attributed to the origin commit, we mark it as survived. For lines that are not found in the final blame output, we verify or recover the earliest terminal commit by checking candidate commits via iterative `git blames`. To make this tractable at scale, we cache blame results, batch full-file blame calls, and parallelize blame verification across threads.

5 ***Static Analysis.*** To study changes in code quality metrics and static warnings, we apply static analysis tools to our collected dataset. We use three widely used static analysis tools: (1) *Semgrep* [66], rule-based analysis tool for detecting security weaknesses in source code; (2) *OSV-Scanner* [67], dependency scanner for identifying known dependency vulnerabilities; and (3) *SonarQube* [68], software quality analysis tool for tech-debt metrics. We apply both Semgrep and OSV-Scanner after each commit in our dataset. For each commit, we identify all changed files and create two versions of those files: before the commit and after the commit. We then run the analysis tools on both versions and obtain two sets of static findings. To measure the overall repository maintenance metrics, we use SonarQube to analyze weekly snapshots of each repository during our observation window.

IV. RQ1: Does agentic code require more maintenance post-merge?

We measure differences in overall maintenance (RQ1a), types of maintenance (RQ1b), and types of corrective maintenance received and introduced defects (RQ1c).

A. Research Methods

Does agentic code require more maintenance? (RQ1a). We first ask whether agentic contributions are maintained at a different overall rate than human contributions. We model post-merge maintenance as line-level time-to-event data, treating each line introduced by an origin commit as a subject observed from its date of merge until it is either: (1) *terminated* by a later commit, our event-of-interest representing maintenance activity, or (2) *right-censored*, meaning that it does not experience an event before the end of our observation window.

We use survival analysis, a branch of statistical analysis for modeling time-to-event data [69]. Specifically, we use the Kaplan-Meier estimator [70], a standard non-parametric method for estimating survival functions [71]. We estimate the agentic maintenance burden with a Cox proportional hazards model and report the hazard ratio (HR) where HR $>$ 1 indicates a higher termination rate for agentic lines than for human lines and HR $<$ 1 indicates a lower termination rate.

Does agentic code require different types of maintenance? (RQ1b). We next ask whether the *composition* of maintenance differs between agentic and human code. We first test whether the distribution of maintenance types differs using a Rao-Scott

cluster-corrected chi-square test [72]. We perform this test at the maintenance-action level where each terminal commit that terminates either agentic or human lines counts as one maintenance action. This captures the composition of maintenance work performed on each group of code. We exclude terminal commits that maintain code from both groups in the same action, because such commits would violate the independence-of-observations assumption. In total, we excluded 46,320 cross-group actions and retained 157,388 human and 13,729 agentic maintenance actions for analysis.

To estimate whether the *risk* of experiencing each maintenance type differs by authorship, we use Fine-Gray competing-risks regression [73]. Unlike standard survival analysis, which models a single event-of-interest, our setting allows each subject, i.e., line, to be terminated by exactly one of several mutually exclusive maintenance intent types that compete with one another. Fine-Gray regression addresses this by modeling the subdistribution hazard, i.e., the effect of authorship group on the cumulative incidence of a given type while treating the other types as competing events. We fit one Fine-Gray model for each maintenance class and operational intent. We pair each model with its cumulative incidence function (CIF) which estimates the probability that a line has been terminated by that maintenance intent over time.

Does agentic code require different types of corrective maintenance and introduce more defects? (RQ1c). Lastly, we focus specifically on differences in corrective maintenance and introduced defects between agentic and human contributions. We test whether the distribution of corrective operational intents differs between the two groups using the same chi-square test as RQ1b. We then estimate corrective maintenance burden using four Cox proportional hazards models: one for overall corrective maintenance and one for each of the three corrective operational intents to compare the corrective termination hazard of agentic and human lines.

To measure defects directly, we use static analysis to count the static findings each contribution introduces, with Semgrep for source-code security-related findings and OSV-Scanner for dependency findings. For each tool, we fit two Poisson generalized linear mixed models (GLMMs) [74] to answer whether agentic commits introduce more findings per unit of code changed overall, and whether they introduce more high-severity findings.

Modeling Considerations. Because we estimate the *total* effect of authorship, we adjust for confounders in our modeling formulas [75]. We treat file primary role and repository as confounders: file role reflects the task a contribution serves, while repositories differ in both their level of agentic adoption and their baseline maintenance behavior. Repository is a high-cardinality confounder, which we handle using the mechanism appropriate for each model family: stratification by repository in the Cox and Fine-Gray models [76], and a repository random intercept in the Poisson GLMMs [77]. Within-repository correlation additionally threatens the independence-of-observations assumption. We address this by clustering standard errors by repository in the survival models, applying the Rao-Scott correction in the chi-square tests, and including the random intercept in the GLMMs.

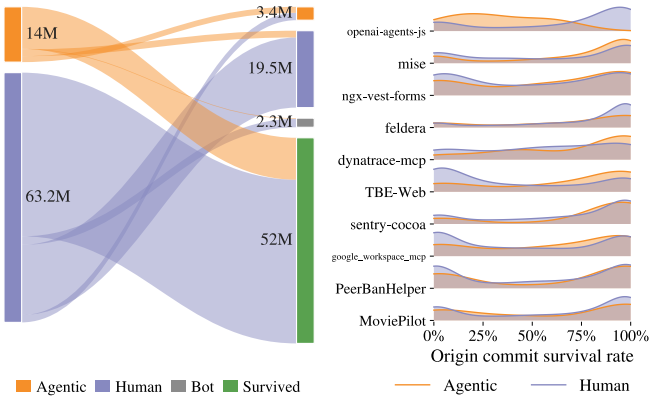
The Cox and Fine-Gray models use the same specification: authorship group as the focal covariate, file primary role as a covariate, stratification by repository, and repository-clustered standard errors. We assess the proportional-hazards assumption for every survival model with the Grambsch-Therneau test and scaled Schoenfeld residual plots [78]. For the overall corrective Cox model, the test flagged a minor proportional-hazards violation. A time-split robustness check at 21 days found no statistically distinguishable difference between the early and late agentic effects, so we report the time-averaged hazard ratio as the primary estimate. For each Fine-Gray model, we verify the minimum events-per-variable threshold of 10 following common statistical best practices [79], which all categories meet. Because agentic lines skew younger, we conduct a calendar-period sensitivity check by re-estimating survival on the overlapping window in which both groups are well represented (August 2025 to May 2026) and comparing the agentic-human survival gap at 90, 180, and 270 days against the full-window model. The gaps match in sign and magnitude, indicating that no calendar adjustment is needed.

The static analysis GLMMs are fit at the commit level with a repository random intercept and a log-exposure offset for the number of relevant changed lines: source-code lines for Semgrep findings and dependency-file lines for OSV-Scanner findings. We assess fit with DHARMa simulated residuals and a Pearson overdispersion check [80].

Limitations. First, our authorship classification relies on commit-level heuristic signals which may misclassify authorship in either direction. In addition, human contributions in our dataset may have unobservable AI assistance such as IDE inline suggestions or pasted agentic outputs. This makes the human baseline a lower bound on true AI involvement and may bias estimated differences between agentic and human contributions toward the null. Second, our maintenance intent labels are obtained from a rule-based and LLM-assisted classification pipeline. Particularly, commits near category boundaries (e.g., corrective versus perfective) may be mislabeled. Third, not every line termination necessarily reflects a defect or quality problem; we mitigate this by separately analyzing different maintenance intents. Fourth, our static analysis results are limited to weaknesses and vulnerabilities detectable by the rule-based tools we use and may include false positives. Finally, our study spans from May 2025 to May 2026 and focuses on active open-source projects with sustained agentic contributions. As such, longer-horizon maintenance beyond our observation window is unobserved, and our findings may not generalize to closed-source software or projects with minimal agentic adoption.

B. Results

Does agentic code require more maintenance? (RQ1a). Our results indicate that agentic code does not carry a uniformly



(a) Flow of survival and termination events.

(b) Survival rates of 10 selected repositories.

Fig. 3: Post-merge maintenance rate.

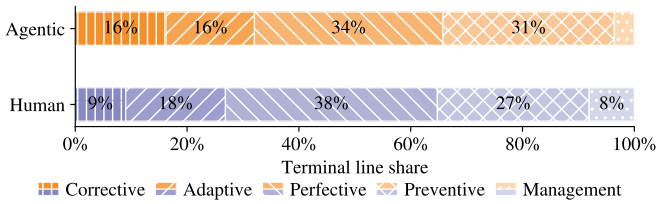


Fig. 4: Line-level termination maintenance class proportion.

Maintenance Intent	sHR (95% CI)
Corrective	1.46 [1.02, 2.08]*
Security fix	1.18 [0.81, 1.72]
Revert	1.39 [0.55, 3.49]
Bug fix	1.45 [1.02, 2.07]*
Adaptive	1.00 [0.77, 1.31]
Dependency update	1.00 [0.77, 1.31]
Perfective	0.98 [0.65, 1.50]
Performance	1.34 [0.65, 2.80]
Feature	0.71 [0.49, 1.02]
Documentation	1.17 [0.86, 1.59]
Resource	1.81 [0.68, 4.83]
Preventive	1.11 [0.69, 1.78]
Refactor	0.75 [0.54, 1.04]
Test	1.54 [0.96, 2.47]
Style/Formatting	0.48 [0.33, 0.68]***
Management	0.39 [0.25, 0.60]***
Merge/Release/Versioning	0.28 [0.16, 0.51]***
Build/Config/CI	0.58 [0.34, 0.99]*

Note: $^{*}p < 0.05;^{**}p < 0.01;^{***}p < 0.001$

TABLE IV: Fine-Gray regression modeling summary.

higher overall maintenance burden. Agentic lines are not significantly more or less likely to be terminated than human lines (HR = 1.11, 95% CI: 0.85–1.45, p = 0.45). Although the HR suggests an ∼11% higher termination rate for agentic lines, the confidence interval includes 1, so the difference is not statistically significant. This null result is not due to sparse data: in the raw survival rate, agentic lines survive at least as often as human lines (75.8% vs. 65.6%, cf. Figure 3a).

The flat HR is not evidence that authorship does not matter but rather that its effect is highly project-dependent. Authorship effects vary widely across per-repository survival distributions (cf. Figure 3b) and this heterogeneity washes out in the aggregate. We treat this heterogeneity as a key finding in its own right and return to it in more detail in RQ2.

The Sankey flow reveals an asymmetry in who performs terminations: although human commits vastly outnumber agentic commits, nearly half of agentic line terminations are performed by agentic commits themselves (cf. Figure 3a). This suggests that agents tend to operate on existing agentic code and in a different part of the codebase than humans. Thus, similar overall termination rates do not imply that the underlying maintenance patterns are the same. We therefore next examine *what kinds* of maintenance each group attracts.

Key Insight: Agentic contributions do not carry a uniform post-merge maintenance burden. Their termination behavior varies substantially across repositories, making project context critical for interpreting post-merge maintenance.

Does agentic code require different types of maintenance? (RQ1b). We find that the *composition* of maintenance does differ significantly. Agentic and human code receive significantly different mixes of maintenance (Rao-Scott chi-square = 910.6, $p < 0.001$, Cramér's V = 0.07), driven by agentic code receiving a higher share of bug fixes and a lower share of feature work and refactoring (cf. Figure 4).

The Fine-Gray models confirm this difference appears in *risk* as well: agentic lines have a significantly higher risk of experiencing corrective maintenance and lower risk of management maintenance (cf. Table IV). These are the only maintenance classes in which the two groups diverge significantly. Adaptive, perfective, and preventive maintenance, by contrast, are indistinguishable once file role is accounted for. At the operational intent level, the elevated corrective risk is specific to bug fixes: agentic lines reach a higher cumulative incidence of bug-fix termination almost immediately after merge and stay there, with 4.0% of agentic lines changed by a bug-fixing commit by 180 days compared with 2.7% of human lines (cf. Figure 5). Conversely, agentic lines are significantly *less* likely to be terminated by style/formatting, merge/release/versioning, and build/config/CI commits.

Key Insight: Agentic and human contributions undergo different types of maintenance, with agentic code receiving more corrective maintenance.

Does agentic code require different types of corrective maintenance and introduce more defects? (RQ1c). We find that agentic lines receive corrective maintenance at a 49% higher rate than human lines (cf. Table V). This gap appears throughout the post-merge period: both groups accumulate

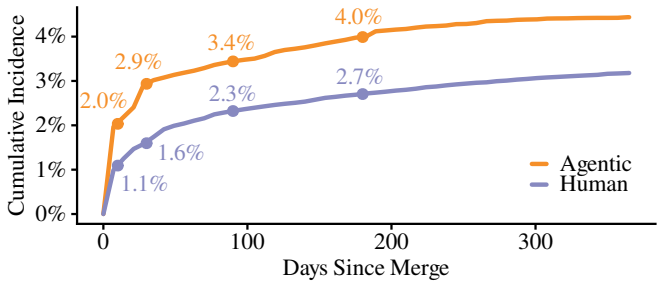
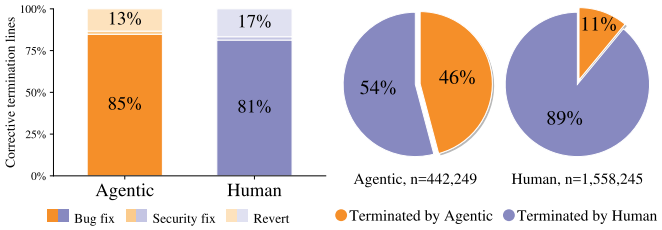


Fig. 5: CIF for bug fix. Dots represent time points of 10/30/90/180 days.



(a) Proportion of the line-level corrective operational intent.

(b) Authorship of bug-fix terminations.

Fig. 6: Post-merge corrective maintenance.

fixes fastest in the first few weeks, the agentic curve rises above the human curve early, and the two stay roughly parallel thereafter (cf. Figure 1). This elevated corrective maintenance rate is concentrated in bug fixes. Agentic lines show a 51% higher bug-fix termination rate, while the security-fix and revert effects are directionally higher but not significant (cf. Table V). The *composition* of corrective work, however, is similar across groups: the bug-fix/security-fix/revert split does not differ significantly (Rao-Scott chi-square = 8.45, $p = 0.385$, Cramér's V = 0.01; cf. Figure 6a). Together these findings suggest agentic code is corrected *more* but not *differently*.

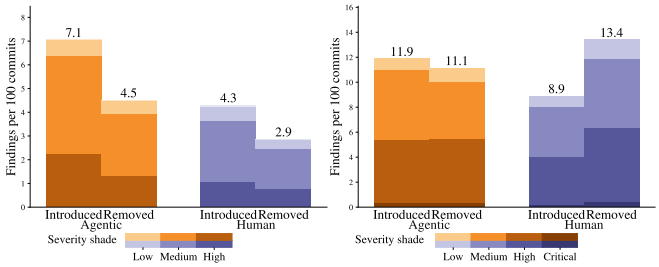
We also find close to half of the bug-fix terminations on agentic lines come from agentic commits (cf. Figure 6b), echoing the RQ1a pattern of agents operating on existing agentic code. Still, more than half of agentic bug fixes are performed by humans, indicating that agentic contributions continue to draw substantial human corrective effort.

Additionally, we examine the source-code and dependency findings introduced by agentic and human code. Agentic contributions introduce Semgrep security-related findings at 1.14 times and high-severity findings (`ERROR`) at 1.51 times the human per-source-line rate (cf. Figure 7a and Table V). The same holds for dependency vulnerabilities: agentic contributions introduce OSV-Scanner findings at 1.10 times and high-severity findings (`HIGH`/`CRITICAL`) at 1.15 times the human per-dependency-line rate (cf. Figure 7b and Table V).

Cox PH	HR (95% CI)	Poisson GLMM	RR (95% CI)
Corrective	1.49 [1.05, 2.12]*	Static find.	1.14 [1.08, 1.21]***
Bug fix	1.51 [1.06, 2.16]*	High-sev. static find.	1.51 [1.33, 1.70]***
Security fix	1.20 [0.82, 1.75]	Dep. find.	1.10 [1.06, 1.15]***
Revert	1.42 [0.57, 3.54]	High-sev. dep. find.	1.15 [1.08, 1.23]***

Note: $^{*}p < 0.05$;$^{**}p < 0.01$;$^{***}p < 0.001$

TABLE V: Cox PH and Poisson GLM models for corrective maintenance and static analysis findings.



(a) Semgrep findings.

(b) OSV-Scanner findings.

Fig. 7: Static analysis findings per 100 commits.

Key Insight: Agentic code faces higher corrective-maintenance risk, driven primarily by bug fixes. Agentic commits also introduce weaknesses and vulnerabilities at higher rates, especially high-severity findings.

V. RQ2: Does corrective maintenance burden increase in projects as the proportion of agentic code increases over time?

We now examine whether corrective maintenance accumulates as agentic code share increases. To do so, we conduct a longitudinal panel study that analyzes how increases in agentic code share affect repository-level corrective maintenance.

A. Research Methods

We ask whether repositories experience more corrective maintenance as agentic code accounts for a larger share of their codebase over time. We construct a longitudinal panel dataset that follows repositories month by month, allowing us to relate changes in agentic code share to subsequent corrective maintenance activity. We use repository-month as the unit of analysis. For each repository i and month m, we measure corrective maintenance using two outcomes: (1) $corrective\_commit\_rate_{i,m}$: number of corrective commits / total number of commits and (2) $human\_corrective\_commit\_rate_{i,m}$: the number of human corrective commits / total number of human commits. At the same time, we compute the $agentic\_code\_share_{i,m}$ as the percentage of the repository's code, measure at the end of the month, that was written by agents. Using these monthly measures, we fit panel models based on Binomial generalized linear mixed models (GLMMs) [74] and report odds ratios (ORs). In our setting, OR > 1 indicates that the predictor increases corrective maintenance rate, while OR < 1 means predictor decreases corrective maintenance rate. Our key predictor is

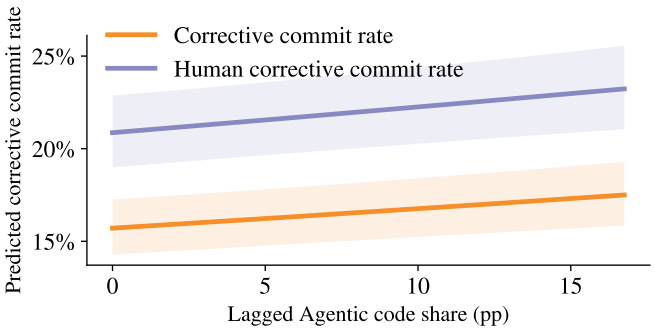


Fig. 8: Predicted corrective maintenance rates as a function of lagged agentic code share. Slopes show Binomial GLMM's predicted rates for all corrective commits and human corrective commits across repo-months, with shaded bands indicating 95% confidence intervals.

lagged agentic code share ($agentic\_code\_share_{i,m-1}$) which captures the agentic code share at the end of the previous month. We use lagged agentic code share because our goal is to measure how prior agentic code presence is associated with corrective maintenance activity in the following month.

Modeling Considerations. For the Binomial GLMMs used in our panel analysis, we first control for commit volume using the log-transformed number of commits: total and human commits for the overall and human corrective commit rate models respectively. We include a random intercept for each repository, allowing each repository to have its own baseline corrective commit rate. We also include an observation-level random effect (OLRE) [81] to account for overdispersion in the binomial response, absorb extra-binomial variation and reduce the risk of understated standard errors. Additionally, we performed model diagnostics to assess fit and robustness, including singular-fit checks, residual diagnostics, Pearson overdispersion tests, and grouped-binomial zero-success checks comparing the observed number of zero-corrective repository-months against the fitted model expectation.

Limitations. Our model uses one-month lag which captures only a coarse temporal relationship where we do not model corrective maintenance that may respond to agentic code over shorter or longer windows. Additionally, our metric of corrective commit rate only measures one aspect of corrective burden but not the size, complexity, or review effort to perform those fixes. As such, our findings should be generalized cautiously.

B. Results

Both the overall corrective commit rate and the human corrective commit rate increase as lagged agentic code share rises (cf. Figure 8). Repositories with more existing agentic code are more likely to spend the following month on both overall corrective maintenance (OR = 1.08, 95% CI: 1.05–1.12, $p < 0.001$) and human corrective maintenance (OR = 1.09, 95% CI: 1.05–1.13, $p < 0.001$). A 10 percentage-point increase in lagged agentic code share is associated with ∼8.0% and ∼8.6% higher odds of corrective commit overall and human corrective commits in the following month respectively. This indicates greater agentic code presence is associated not only with higher repository-level corrective burden but also with higher corrective burden for human developers.

Key Insight: Corrective maintenance burden increases with agentic code share, including corrective maintenance performed by human developers.

VI. RQ3: What project characteristics are associated with agentic maintenance burden?

RQ1 shows that the maintenance burden of agentic code varies substantially across projects. This heterogeneity suggests that project characteristics may affect the agentic maintenance burden. In this research question, we concretely examine which project characteristics explain differences in post-merge maintenance burden between agentic and human contributions.

A. Research Methods

To examine which project characteristics explain variation in agentic maintenance burden, we use a two-stage meta-regression approach. The key idea is to first estimate a separate agentic maintenance burden for each project and then test whether project characteristics explain why this burden is larger in some projects than in others. In stage one, for each project in our dataset, we fit a Cox PH model to measure the agentic maintenance burden relative to human lines, expressed as a hazard ratio (HR). This follows RQ1a, but instead of estimating a single HR for the entire dataset, we compute an HR and Standard Error (SE) for each project. In stage two, we use each project's HR as the outcome weighted by their uncertainty (SE) to model whether the different project characteristics explain variation in agentic maintenance burden using a random-effects meta-regression model. We operationalize six project characteristics, each corresponding to a hypothesis about factors that may affect agentic maintenance burden:

Technical Debt: We use the technical debt index computed by SonarQube at the start of the observation window. This is a proxy for the project's baseline maintenance practices. We expect projects with high technical debt indexes to have a harder time integrating new agentic code cleanly.

Number of Contributors: We count the number of non-bot contributors with at least one commit during the observation window. We expect projects with more contributors to have lower agentic maintenance burden as more maintainers may have more review bandwidth to catch issues pre-merge.

Agentic Source Code Share: We measure the fraction of agentic churn that occurs in files classified as source code. We expect projects with higher agentic source-code share to have higher agentic maintenance burden, because agentic activity concentrated in source code is more directly exposed to post-merge maintenance than activities in lower-stake documentation, configuration, or resource changes.

Test File Change Share: We measure the share of changed files classified as tests. We expect projects with higher test file change share to have lower agentic maintenance burden

Covariate	HR multiplier (95% CI)
Technical Debt	1.47 [0.92, 2.36]
Number of Contributors	1.00 [1.00, 1.00]
Agentic Source Code Share	0.72 [0.46, 1.13]
Test File Change Share	1.90 [0.85, 4.26]
Agentic Line Churn Share	0.73 [0.40, 1.34]
No Review Rate	1.75 [1.01, 3.04]*

Note: $^{*}p < 0.05;^{**}p < 0.01;^{***}p < 0.001$

TABLE VI: Random-effect meta-regression model with I^2 = 100%, R^2 = 6.0%.

since projects with stronger test infrastructure can catch and stabilize agentic code faster.

Agentic Line Churn Share: We measure the fraction of total line churn in the project by agentic contributions. We expect projects with higher agentic line-churn share to have higher agentic maintenance burden, because heavier agentic adoption can become load-bearing in the codebase and therefore more likely to be revisited and modified.

No Review Rate: We measure the proportion of merged pull requests during the observation window that received no code review. We expect projects with higher no-review rate to have higher agentic maintenance burden, because weaker review processes are less able to catch problematic or low quality agentic contributions (i.e., AI slop).

Modeling Considerations. For stage one, we fit the Cox PH models similar to RQ1a. However, we do not stratify or cluster based on repository because we compute a separate HR for each repository. We then filter out projects with fewer than 100 tracked lines or 10 termination events in either groups (5 projects). This avoids unstable project-level HR estimates driven by sparse data. For the stage two random-effects meta-regression model, we use the Knapp-Hartung correction [82] following standard best practices. To determine which covariates to operationalize, we conduct several exploratory analyses, including data visualization and LASSO-based screening [83] of candidate covariates. Based on these analyses, we select the six covariates described above. We perform standard model diagnostics, including checks for multicollinearity, influential projects, and residual normality.

Limitations. Our covariates are necessarily incomplete and proxy-based: some relevant project characteristics, such as review quality, are not directly observable in our dataset. Accordingly, the meta-regression should be interpreted as identifying associations rather than fully explaining cross-project variation in agentic maintenance burden.

B. Results

Across 177 projects, we find statistically significant evidence that agentic maintenance burden is associated with project characteristics rather than varying at random ($F(6, 170) = 2.86$, $p < 0.05$). The rate of merging code without review emerges as the strongest individual predictor (cf. Table VI), with higher no-review rates being associated with significantly higher agentic maintenance burden. In particular, each 10 percentage-point increase in a project's no-review rate is associated with roughly a 6% increase in agentic maintenance burden, on average. Technical debt and limited test infrastructure show non-significant effects in the same direction (cf. Table VI), suggesting a coherent pattern in which weaker engineering safeguards are associated with increased agentic maintenance burden.

Key Insight: Agentic maintenance burden is associated with project characteristics and particularly, having higher no-review rate is associated with increased agentic maintenance burden.

VII. Discussion and Implications

The Real Asymmetry. The adoption of agentic coding tools shattered a core symmetry between code generation and review that software development long relied on: code generation and code review were both constrained by human labor. Before GenAI, developers could only produce code as fast as they could understand, write, and revise it; reviewers, in turn, could only approve code as fast as they could understand its behavior and consequences. The entire pitch of agentic coding tools is that they decouple generation output velocity from human labor: a person can now produce far more code per unit of human effort than they could have by hand. But review did not get the same decoupling. The result is a generation-review velocity asymmetry that exists by construction: generation has been severed from the human constraint that used to pace it, while review remains bound to human comprehension.

This asymmetry is starkly visible in practice today. As teams adopt agentic tools, pull request volume increases while review time rises and review thoroughness and rates decline [84]. Our results help show the cost of this asymmetry.

Why a Full-Lifecycle View is Needed to See It. Although we do not measure the generation-review asymmetry directly, we do measure its consequences. Observing the consequences requires following agentic code throughout its entire lifecycle rather than stopping at merge [9], [50], [51], [52], [53], [10], or only looking after merge [46], [58]. Initially, agentic code looks reassuring with a similar overall maintenance rate to human code (Section IV-B). That similarity does not survive a closer look. Agentic code receives more corrective maintenance with more bug fixes (Section IV-B), introduces more security weaknesses and dependency vulnerabilities (Section IV-B), and is associated with higher overall corrective burden in projects as its share grows (Section V). Additionally, projects that merge agentic code without review show the largest agentic maintenance burden of all (Section VI).

Because these consequences are only visible when tracing agentic code across its full lifecycle, precisely where most current evaluation does not look, adoption-informing metrics must expand beyond generation velocity and pre-merge performance. Today, success is often measured through visible and easy-to-count signals: pull request volume, code generated, percentage of agentic code, and token consumption. But our

results show that those measures are blind to the outcome that matters most for software development: whether the code is reliable, secure, and maintainable after merge.

The Review Side Cannot Close the Gap. Faced with the generation-review asymmetry, the response across the software industry has been to treat it as a review-side problem and to adjust review processes until they keep pace with how fast code is produced by agentic tools [85]. These responses cluster into two broad strategies and neither closes the gap on its own.

The first is to scale human review, which runs into a limit at every point on its range no matter how much you attempt to scale it, because a human remains the final checkpoint. At the extreme, projects that merge agentic code without review at high rates are precisely the projects where agentic maintenance burden is the highest (Section VI). Short of that extreme, lighter and faster review removes fewer defects [86], and agentic code introduces more defects to begin with (Section IV).

The second option is to automate review. Automation can resolve the asymmetry, but the asymmetry is created exactly because review is the last checkpoint where a human decides whether an agents output is acceptable, and automating it surrenders that decision making autonomy to the same kind of system that produced the code rather than restoring human control of the generation velocity. We do not claim that automated review is inherently worse than human review; our data does not measure that. This point is instead structural: review is currently the only thing that keeps the rate of what enters a project tied, however loosely, to a rate a person can keep up with. Automating it removes all human control over development velocity and once a project depends on running at the automated rate, restoring a human-based checkpoint means falling behind on a volume of output the project has already organized itself around, which makes it an extremely difficult decision to walk back in practice.

Even if one is willing to cede human decision making autonomy, GenAI automated review systems do not magically solve the problem. These review systems are themselves software that must be evaluated and secured. Recently, the popular Claude Code GitHub action used for automated pull request review has been shown to expose CI/CD secrets after encountering untrusted content [87]. In other words, building automated review tools does not alleviate the problem, but instead can introduce additional development burdens.

Fix the Tool, Not the Process. At this point our argument moves from what we measured to what we believe the mechanism implies. If agentic tools increase generation velocity while human review continues to bound review velocity, then the two sides of the development pipeline are no longer paced by the same constraint. That mismatch cannot be absorbed by review indefinitely. This is why a hybrid configuration that keeps humans in control of review while GenAI controls generation is not sustainable. Eventually, the two process velocities must be brought under the same control, or the system breaks down under its own backlog. Adding GenAI to the review side may help with triage or prioritization, but it does not by itself resolve the deeper asymmetry if humans remain accountable for final judgment. Instead, it defers the cost while projects build a growing dependence on increasingly automated generation processes. We cannot come to a conclusion about the impacts of this tension with our data, but we raise it as a central question for longitudinal work on agentic tooling adoption.

If the asymmetry cannot be resolved until the entities controlling the output velocity of both processes are matched, and we continue to build a dependence on GenAI on the generation side, what does that mean for the moment when this asymmetry comes to a head and we are forced to make a decision? This question is not abstract. In a growing number of settings, declining to adopt is no longer a realistic option. Shopify and Duolingo have made agentic code use a baseline expectation [88], [89], Meta has set internal GenAI-use targets which include goals for agentic code changes and broad adoption among engineers [90], and developers at companies such as Microsoft report being evaluated in part by how much agentic tooling they use [91]. Where opting out is unavailable, teams cannot avoid the asymmetry; they can only manage it. If this mechanism holds, the choice is narrowed in advance: a team that leans on generation to stay ahead of review builds a dependence that makes the tool harder to question later, at the exact moment when such questioning and critical reflection would matter most.

The risk itself lies in the misconception that a stable hybrid approach exists without sufficient evidence while rapidly developing a dependence on the technology. If we as a collective are serious about preserving human control over digital infrastructure and decision making autonomy, then generation velocity must remain governable by humans. The tools themselves must be designed to respect the limits of human comprehension, validation, and accountability. We must fix the tools, not merely stretch the process around them.

A. Implications for Research and Practice

For researchers, our results argue that lifecycle-spanning evaluation of agentic code should become a standard complement to pre-merge benchmarks. Merge-time success is not enough: evaluations should also measure whether agentic contributions remain durable, secure, and maintainable. This aligns with recent practitioner calls for standardized first-class outcome metrics of post-merge maintenance durability, e.g., *code durability*, and *code turnover rate*.

For practitioners, our results argue for protecting review rather than treating it as something to compress. Recognize that review alone, so long as it remains a process humans have ultimate decision making autonomy over, cannot indefinitely absorb a source no longer bound by human effort.

For organizations making GenAI adoption decisions, our results urge caution and reflection on what metrics are being used to define the success and to ensure their visibility is in alignment with the metrics of success for the organization's software development practices.

For tool builders, our results argue that the decisive measure of a coding agent is not whether its output merges, but what it costs the project over time. The intervention is therefore to reduce that cost at the source: tools should be designed not just to generate more code, but to generate code that survives, imposes less maintenance burden, and remains sustainable post-merge.

VIII. Conclusion

In this work, we conducted a longitudinal post-merge study of agentic and human contributions across 182 repositories using line-level survival, maintenance intent, static analysis findings, and project characteristics. Our results show that while agentic code does not exhibit uniformly higher overall maintenance, it receives more corrective maintenance, and introduces more security weaknesses and dependency vulnerabilities than human contributions. At the project level, increasing agentic code share is associated with higher corrective maintenance rates and projects that merge more code without review show larger agentic maintenance burdens. As generation becomes cheaper, the relevant question is no longer whether agents can write mergeable code, but whether their contributions remain reliable, secure, and maintainable after merge.

IX. Data Availability

We provide our complete artifact for reproduction: https://github.com/post-merge-reality/post-merge-reality

References

[1] J. Novet and J. Vanian. (2025) Satya nadella says as much as 30% of Microsoft code is written by AI. CNBC. [Online]. Available: https://www.cnbc.com/2025/04/29/satya-nadella-says-as-much-as-30percent-of-microsoft-code-is-written-by-ai.html

[2] C. Mauran. (2025) Mark zuckerberg wants AI to do half of Meta's coding by 2026. Mashable. [Online]. Available: https://mashable.com/article/llamacon-mark-zuckerberg-ai-writes-meta-code

[3] H. Langley. (2026) Google says 75% of the company's new code is AI-generated. Business Insider. [Online]. Available: https://www.businessinsider.com/google-ai-generated-code-75-gemini-agents-software-2026-4

[4] J. Young. (2026) Introducing the state of AI coding 2026. New Relic. [Online]. Available: https://newrelic.com/blog/ai/state-of-ai-coding-2026

[5] E. Glover. (2026) What is tokenmaxxing? the AI workplace trend explained. Built In. [Online]. Available: https://builtin.com/articles/ai-tokenmaxxing

[6] GitLab Inc. (2026) GitLab research reveals organizations are generating AI code faster than they can control it. GitLab. [Online]. Available: https://ir.gitlab.com/news/news-details/2026/GitLab-Research-Reveals-Organizations-Are-Generating-AI-Code-Faster-Than-They-Can-Control-It/default.aspx

[7] S. Baltes, M. Cheong, and C. Treude, ""an endless stream of ai slop": The growing burden of ai-assisted software development," *arXiv preprint arXiv:2603.27249*, 2026.

[8] C. E. Jimenez, J. Yang, A. Wettig, S. Yao, K. Pei, O. Press, and K. R. Narasimhan, "SWE-bench: Can language models resolve real-world github issues?" in *The Twelfth International Conference on Learning Representations*, 2024.

[9] M. Watanabe, H. Li, Y. Kashiwa, B. Reid, H. Iida, and A. E. Hassan, "On the use of agentic coding: An empirical study of pull requests on github," *ACM Transactions on Software Engineering and Methodology*, 2025.

[10] K. Watanabe, R. Tsuchida, T. Monno, B. Huang, K. Yamasaki, Y. Fan, K. Shimari, and K. Matsumoto, "How ai coding agents communicate: A study of pull request description characteristics and human review responses," *arXiv preprint arXiv:2602.17084*, 2026.

[11] M. Chen, J. Tworek, H. Jun, Q. Yuan, H. P. de Oliveira Pinto, J. Kaplan, H. Edwards, Y. Burda, N. Joseph, G. Brockman, A. Ray, R. Puri, G. Krueger, M. Petrov, H. Khlaaf, G. Sastry, P. Mishkin, B. Chan, S. Gray, N. Ryder, M. Pavlov, A. Power, L. Kaiser, M. Bavarian, C. Winter, P. Tillet, F. P. Such, D. Cummings, M. Plappert, F. Chantzis, E. Barnes, A. Herbert-Voss, W. H. Guss, A. Nichol, A. Paino, N. Tezak, J. Tang, I. Babuschkin, S. Balaji, S. Jain, W. Saunders, C. Hesse, A. N. Carr, J. Leike, J. Achiam, V. Misra, E. Morikawa, A. Radford, M. Knight, M. Brundage, M. Murati, K. Mayer, P. Welinder, B. McGrew, D. Amodei, S. McCandlish, I. Sutskever, and W. Zaremba, "Evaluating large language models trained on code," *arXiv preprint arXiv:2107.03374*, 2021.

[12] GitHub, "Github copilot your ai pair programmer," https://github.com/features/copilot, accessed Jun. 2026.

[13] Cursor, "Cursor: Ai coding agent," https://cursor.com/, accessed Jun. 2026.

[14] Anthropic, "Claude code by anthropic | ai coding agent, terminal, ide," https://claude.com/product/claude-code, accessed Jun. 2026.

[15] OpenAI, "Codex | ai coding partner from openai | openai," https://openai.com/codex/, accessed Jun. 2026.

[16] C. Miller, R. Choudhuri, M. Ulloa, S. Haniyur, R. DeLine, M.-A. Storey, E. Murphy-Hill, C. Bird, and J. L. Butler, ""maybe we need some more examples:" individual and team drivers of developer genai tool use," in *Proc. Int'l Conf. Software Engineering (ICSE)*, 2026. ACM Distinguished Paper Award.

[17] GitHub, "Octoverse: A new developer joins github every second as ai leads typescript to #1," GitHub, Tech. Rep., 2025. [Online]. Available: https://github.blog/news-insights/octoverse/octoverse-a-new-developer-joins-github-every-second-as-ai-leads-typescript-to-1/

[18] OpenAI, "Codex for every role, tool, and workflow," https://openai.com/index/codex-for-every-role-tool-workflow/, 2026.

[19] J. Bort, "Anthropic's claude popularity with paying consumers is skyrocketing," *TechCrunch*, 2026, https://techcrunch.com/2026/03/28/anthropics-claude-popularity-with-paying-consumers-is-skyrocketing/.

[20] N. Nguyen and S. Nadi, "An empirical evaluation of github copilot's code suggestions," in *Proc. Conf. Mining Software Repositories (MSR)*, 2022, pp. 1–5.

[21] B. Yetistiren, I. Ozsoy, and E. Tuzun, "Assessing the quality of github copilot's code generation," in *Proceedings of the 18th international conference on predictive models and data analytics in software engineering*, 2022, pp. 62–71.

[22] M. Borek and R. Nowak, "Quality evaluation of tabby coding assistant using real source code snippets," *arXiv preprint arXiv:2504.08650*, 2025.

[23] O. Asare, M. Nagappan, and N. Asokan, "Is github's copilot as bad as humans at introducing vulnerabilities in code?" *Empirical Software Engineering*, vol. 28, no. 6, p. 129, 2023.

[24] H. Pearce, B. Ahmad, B. Tan, B. Dolan-Gavitt, and R. Karri, "Asleep at the keyboard? assessing the security of github copilot's code contributions," in *2022 IEEE Symposium on Security and Privacy (SP)*. IEEE Computer Society, 2022, pp. 754–768.

[25] N. Perry, M. Srivastava, D. Kumar, and D. Boneh, "Do users write more insecure code with ai assistants?" in *Proceedings of the 2023 ACM SIGSAC conference on computer and communications security*, 2023.

[26] M. Borg, D. Hewett, N. Hagatulah, N. Couderc, E. Söderberg, D. Graham, U. Kini, and D. Farley, "Echoes of ai: Investigating the downstream effects of ai assistants on software maintainability," *Empirical Software Engineering*, vol. 31, no. 6, p. 161, 2026.

[27] D. G. Paul, H. Zhu, and I. Bayley, "Investigating the smells of llm generated code," *arXiv preprint arXiv:2510.03029*, 2025.

[28] L. Chretien and N. Albarran, "Impact of ai-tooling on the engineering workspace," *arXiv preprint arXiv:2406.07683*, 2024.

[29] K. K. Ng, L. Fauzi, L. Leow, and J. Ng, "Harnessing the potential of gen-ai coding assistants in public sector software development," *arXiv preprint arXiv:2409.17434*, 2024.

[30] E. Paradis, K. Grey, Q. Madison, D. Nam, A. Macvean, V. Meimand, N. Zhang, B. Ferrari-Church, and S. Chandra, "How much does ai impact development speed? an enterprise-based randomized controlled trial," in *2025 IEEE/ACM 47th International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*, 2025.

[31] M. I. H. Shihab, C. Hundhausen, A. Tariq, S. Haque, Y. Qiao, and B. W. Mulanda, "The effects of github copilot on computing students' programming effectiveness, efficiency, and processes in brownfield coding tasks," in *Proceedings of the 2025 ACM Conference on International Computing Education Research V. 1*, 2025, pp. 407–420.

[32] A. Ziegler, E. Kalliamvakou, X. A. Li, A. Rice, D. Rifkin, S. Simister, G. Sittampalam, and E. Aftandilian, "Productivity assessment of neural code completion," in *Proceedings of the 6th ACM SIGPLAN international symposium on machine programming*, 2022, pp. 21–29.

[33] P. Vaithilingam, T. Zhang, and E. L. Glassman, "Expectation vs. experience: Evaluating the usability of code generation tools powered by large language models," in *CHI conference on human factors in computing systems extended abstracts*, 2022, pp. 1–7.

[34] S. Imai, "Is github copilot a substitute for human pair-programming? an empirical study," in *Proceedings of the ACM/IEEE 44th International Conference on Software Engineering: Companion Proceedings*, 2022.

[35] S. Peng, E. Kalliamvakou, P. Cihon, and M. Demirer, "The impact of ai on developer productivity: Evidence from github copilot," *arXiv preprint arXiv:2302.06590*, 2023.

[36] F. Song, A. Agarwal, and W. Wen, "The impact of generative ai on collaborative open-source software development: Evidence from github copilot," *arXiv preprint arXiv:2410.02091*, 2024.

[37] K. Z. Cui, M. Demirer, S. Jaffe, L. Musolff, S. Peng, and T. Salz, "The effects of generative ai on high-skilled work: Evidence from three field experiments with software developers," *Management Science*, 2026.

[38] R. Pandey, P. Singh, R. Wei, and S. Shankar, "Transforming software development: Evaluating the efficiency and challenges of github copilot in real-world projects," *arXiv preprint arXiv:2406.17910*, 2024.

[39] J. Becker, N. Rush, E. Barnes, and D. Rein, "Measuring the impact of early-2025 ai on experienced open-source developer productivity," *arXiv preprint arXiv:2507.09089*, 2025.

[40] F. Xu, P. K. Medappa, M. M. Tunc, M. Vroegindeweij, and J. C. Fransoo, "Ai-assisted programming decreases the productivity of experienced developers by increasing the technical debt and maintenance burden," *arXiv preprint arXiv:2510.10165*, 2025.

[41] S. H. Ambati, N. Ridley, E. Branca, and N. Stakhanova, "Navigating (in) security of ai-generated code," in *2024 IEEE international conference on cyber security and resilience (CSR)*. IEEE, 2024, pp. 1–8.

[42] S. Li, Y. Cheng, J. Chen, J. Xuan, S. He, and W. Shang, "Assessing the performance of ai-generated code: A case study on github copilot," in *Proc. Int'l Symp. Software Reliability Engineering (ISSRE)*. IEEE, 2024, pp. 216–227.

[43] M. L. Siddiq, S. H. Majumder, M. R. Mim, S. Jajodia, and J. C. Santos, "An empirical study of code smells in transformer-based code generation techniques," in *2022 IEEE 22nd International Working Conference on Source Code Analysis and Manipulation (SCAM)*. IEEE, 2022.

[44] M. F. Kharma, S. Choi, M. Alkhanafseh, and D. Mohaisen, "Security and quality in llm-generated code: A multi-language, multi-model analysis," *IEEE Transactions on Dependable and Secure Computing*, 2026.

[45] D. Ogenrwot and J. Businge, "Patchtrack: A comprehensive analysis of chatgpt's influence on pull request outcomes," *Empirical Software Engineering*, vol. 31, no. 5, p. 136, 2026.

[46] Y. Liu, R. Widyasari, Y. Zhao, I. C. Irsan, J. Chen, and D. Lo, "Debt behind the ai boom: A large-scale empirical study of ai-generated code in the wild," *arXiv preprint arXiv:2603.28592*, 2026.

[47] H. He, C. Miller, S. Agarwal, C. Kästner, and B. Vasilescu, "Speed at the cost of quality: How cursor ai increases short-term velocity and long-term complexity in open-source projects," in *Proc. Conf. Mining Software Repositories (MSR)*, 2026.

[48] H. Li, H. Zhang, and A. E. Hassan, "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.

[49] C. Mürtz and M. N. Müller, "Agents in the wild - dashboard," https://insights.logicstar.ai, 2025, interactive web dashboard. Code available at https://github.com/logic-star-ai/insights. [Online]. Available: https://doi.org/10.5281/zenodo.15846865

[50] S. R. O. Peralta, F. Hoshi, H. Washizaki, N. Ubayashi, I. Kondo, Y. Higo, H. Mukai, N. Yoshida, K. Kusama, H. Tanaka *et al.*, "Why are agentic pull requests merged or rejected? an empirical study," *arXiv preprint arXiv:2605.22534*, 2026.

[51] S. Nakashima, Y. Ishimoto, M. Kondo, S. Mclntosh, and Y. Kamei, "Why agentic-prs get rejected: A comparative study of coding agents," *arXiv preprint arXiv:2602.04226*, 2026.

[52] M. A. Haider and T. Zimmermann, "Understanding dominant themes in reviewing agentic ai-authored code," *arXiv preprint arXiv:2601.19287*, 2026.

[53] D. S. D. Minh, H. T. Kiet, N. L. P. Quy, P. P. Hoa, T. C. Nguyen, N. D. H. Duong, and T. B. Tran, "Early-stage prediction of review effort in ai-generated pull requests," *arXiv preprint arXiv:2601.00753*, 2026.

[54] S. Haque, S. Ingale, and C. Csallner, "Do autonomous agents contribute test code? a study of tests in agentic pull requests," *arXiv preprint arXiv:2601.03556*, 2026.

[55] K. Horikawa, H. Li, Y. Kashiwa, B. Adams, H. Iida, and A. E. Hassan, "Agentic refactoring: An empirical study of ai coding agents," *arXiv preprint arXiv:2511.04824*, 2025.

[56] S. Agarwal, H. He, and B. Vasilescu, "Ai ides or autonomous agents? measuring the impact of coding agents on software development," *arXiv preprint arXiv:2601.13597*, 2026.

[57] S. T. Cynthia, A. Muttakin, and B. Roy, "Beyond bug fixes: An empirical investigation of post-merge code quality issues in agent-generated pull requests," *arXiv preprint arXiv:2601.20109*, 2026.

[58] M. Rahman and E. Shihab, "Will it survive? deciphering the fate of ai-generated code in open source," *arXiv preprint arXiv:2601.16809*, 2026.

[59] E. B. Swanson, "The dimensions of maintenance," in *Proc. Int'l Conf. Software Engineering (ICSE)*, 1976.

[60] E. Varga, *Unraveling Software Maintenance and Evolution*. Springer, 2018.

[61] A. Hindle, D. M. German, and R. Holt, "What do large commits tell us? a taxonomical study of large commits," in *Proc. Conf. Mining Software Repositories (MSR)*, 2008.

[62] S. Wang, C. Bansal, and N. Nagappan, "Large-scale intent analysis for identifying large-review-effort code changes," *Information and Software Technology*, vol. 130, p. 106408, 2021.

[63] Q. Zeng, Y. Zhang, Z. Qiu, and H. Liu, "A first look at conventional commits classification," in *Proc. Int'l Conf. Software Engineering (ICSE)*, 2025.

[64] "MiniMax M2.7," https://www.minimax.io/models/text/m27/, accessed Jun. 2026.

[65] J. Cohen, "A coefficient of agreement for nominal scales," *Educational and psychological measurement*, vol. 20, no. 1, pp. 37–46, 1960.

[66] Semgrep, "Semgrep app security platform," https://semgrep.dev/, accessed Jun. 2026.

[67] OSV-Scanner, "Osv-scanner," https://google.github.io/osv-scanner/, accessed Jun. 2026.

[68] SonarQube, "Sonarqube: Fight ai slop & verify ai code | sonar," https://www.sonarsource.com/products/sonarqube/, accessed Jun. 2026.

[69] S. P. Jenkins, "Survival analysis," *Unpublished manuscript, Institute for Social and Economic Research, University of Essex, Colchester, UK*, vol. 42, pp. 54–56, 2005.

[70] E. L. Kaplan and P. Meier, "Nonparametric estimation from incomplete observations," *Journal of the American statistical association*, 1958.

[71] D. R. Cox and D. Oakes, *Analysis of survival data*. CRC press, 1984.

[72] J. N. Rao and A. J. Scott, "The analysis of categorical data from complex sample surveys: chi-squared tests for goodness of fit and independence in two-way tables," *Journal of the American statistical association*, vol. 76, no. 374, pp. 221–230, 1981.

[73] J. P. Fine and R. J. Gray, "A proportional hazards model for the subdistribution of a competing risk," *Journal of the American statistical association*, vol. 94, no. 446, pp. 496–509, 1999.

[74] C. E. McCulloch, S. R. Searle, and J. M. Neuhaus, *Generalized, linear, and mixed models*. John Wiley & Sons, 2008.

[75] T. J. VanderWeele, "Principles of confounder selection," *European journal of epidemiology*, vol. 34, no. 3, pp. 211–219, 2019.

[76] B. Zhou, A. Latouche, V. Rocha, and J. Fine, "Competing risks regression for stratified data," *Biometrics*, vol. 67, no. 2, pp. 661–670, 2011.

[77] B. M. Bolker, M. E. Brooks, C. J. Clark, S. W. Geange, J. R. Poulsen, M. H. H. Stevens, and J.-S. S. White, "Generalized linear mixed models: a practical guide for ecology and evolution," *Trends in ecology & evolution*, vol. 24, no. 3, pp. 127–135, 2009.

[78] P. M. Grambsch and T. M. Therneau, "Proportional hazards tests and diagnostics based on weighted residuals," *Biometrika*, 1994.

[79] P. Peduzzi, J. Concato, A. R. Feinstein, and T. R. Holford, "Importance of events per independent variable in proportional hazards regression analysis ii. accuracy and precision of regression estimates," *Journal of clinical epidemiology*, vol. 48, no. 12, pp. 1503–1510, 1995.

[80] F. Hartig, "Dharma: residual diagnostics for hierarchical (multi-level/mixed) regression models," *CRAN: contributed packages*, 2016.

[81] X. A. Harrison, "Using observation-level random effects to model overdispersion in count data in ecology and evolution," *PeerJ*, vol. 2, p. e616, 2014.
[82] J. Hartung and G. Knapp, "A refined method for the meta-analysis of controlled clinical trials with binary outcome," *Statistics in medicine*, vol. 20, no. 24, pp. 3875–3889, 2001.
[83] R. Tibshirani, "Regression shrinkage and selection via the lasso," *Journal of the Royal Statistical Society Series B: Statistical Methodology*, vol. 58, no. 1, pp. 267–288, 1996.
[84] F. Research. (2026) Ten takeaways from the ai engineering report 2026: The acceleration whiplash. Faros. [Online]. Available: https://www.faros.ai/blog/ai-acceleration-whiplash-takeaways
[85] S. Roy Choudhary, S. Mahajan, W. Bond, J. Wang, and A. Utture. (2025) uReview: Scalable, trustworthy GenAI for code review at Uber. Uber. [Online]. Available: https://www.uber.com/us/en/blog/ureview/
[86] S. McIntosh, Y. Kamei, B. Adams, and A. E. Hassan, "An empirical study of the impact of modern code review practices on software quality," *Empirical Software Engineering*, vol. 21, no. 5, 2016.
[87] Microsoft Defender Security Research Team, D. Edry, and A. Eliahu. (2026) Securing CI/CD in an agentic world: Claude Code GitHub action case. Microsoft Security Blog. [Online]. Available: https://www.microsoft.com/en-us/security/blog/2026/06/05/securing-ci-cd-in-agentic-world-claude-code-github-action-case/
[88] A. Palmer. (2025) Shopify ceo: Prove AI can't do jobs before asking for more headcount. CNBC. [Online]. Available: https://www.cnbc.com/2025/04/07/shopify-ceo-prove-ai-cant-do-jobs-before-asking-for-more-headcount.html
[89] S. Shibu. (2025) Duolingo launches 148 ai-written courses, replacing humans. Entrepreneur. [Online]. Available: https://www.entrepreneur.com/business-news/duolingo-will-replace-contract-workers-with-ai-ceo-says/490812
[90] H. Langley. (2026) Meta's AI push ties employee goals to AI tool adoption. Business Insider. [Online]. Available: https://www.businessinsider.com/meta-ai-push-employee-goals-tool-adoption-2-026-3
[91] A. Stewart. (2025) Microsoft internal memo: 'using AI is no longer optional.'. Business Insider. [Online]. Available: https://www.businessinsider.com/microsoft-internal-memo-using-ai-no-longer-optional-github-copilot-2025-6