

When AI Code Doesn't Stick: An Empirical Study on Reverted Changes Introduced by AI Coding Agents

Issam Oukhay

École de technologie supérieure - ÉTS
Montreal, Quebec, Canada
issam.oukhay.1@ens.etsmtl.ca

Moataz Chouchen

Concordia University
Montreal, Quebec, Canada
moataz.chouchen@concordia.ca

Mahi Begoug

ETS Montreal
Montreal, Quebec, Canada
mahi.begoug.1@ens.etsmtl.ca

Ali Ouni

ÉTS Montreal, University of Quebec
Montreal, Quebec, Canada
Ali.Ouni@etsmtl.ca

Abstract

Agentic AI systems are increasingly integrated into software development workflows, contributing code alongside human developers. However, some AI-authored changes are later reverted, reflecting situations where agent-generated contributions are judged unsuitable after integration. This paper presents a large-scale empirical study of reverted changes introduced by AI coding agents to better understand the causes behind their rejection. We analyze 33,580 agentic pull requests comprising 86,315 commits authored by five major AI coding agents: Claude, Copilot, Cursor, Devin, and OpenAI Codex. Our results show that 2.66% of agentic pull requests contain at least one reverting commit, with substantial variation across agents, ranging from 0.7% for OpenAI Codex to 7.6% for GitHub Copilot indicating notable differences in code generation reliability. Through a manual analysis of 500 reverting commits, we derive a taxonomy comprising eight categories and 25 themes that explain why agent-generated code is reverted. The most common causes are unintended side effects (22.33%), overengineering (22.13%), functional incorrectness (17.71%), and dependency management problems (12.47%). Overall, our findings indicate that AI coding agents struggle primarily with scope management and contextual understanding, rather than purely functional defects. This study provides actionable guidance for practitioners, informs the design of human-AI collaboration workflows, and highlights priority areas for improving agentic code generation systems.

CCS Concepts

• **Software and its engineering** → **Software verification and validation.**

Keywords

Agentic AI, Mining software repositories

ACM Reference Format:

Issam Oukhay, Mahi Begoug, Moataz Chouchen, and Ali Ouni. 2026. When AI Code Doesn't Stick: An Empirical Study on Reverted Changes Introduced

by AI Coding Agents. 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/10.1145/3793302.3793587>

1 Introduction

AI-powered coding agents have become a prominent component of software development workflows by automating code implementation tasks [20, 33]. AI agents like Claude¹, Copilot², Cursor³, Devin⁴, and OpenAI Codex⁵ generate and propose non-trivial code changes, which are reviewed and merged by human developers via commits and pull requests (PRs) [26].

Despite their growing adoption, AI agents may introduce incorrect or undesired changes, including out-of-scope or unintended modifications, which may lead to defects and increase the cognitive load of developers during review, debugging, and maintenance tasks [19, 39, 40, 47]. As in human-driven software development, such changes are commonly mitigated through *reverting* commits (i.e., commits that revert changes), which provide a rollback mechanism to undo prior modifications [37]. For instance, in the GitHub project *shadenblocks/kibo*, the *Copilot* AI coding agent created a PR (PR#171[35]) titled “*Add Tree component for displaying collapsible hierarchical data*” to introduce new functionality. As shown in Figure 1, during code review, a maintainer asked the agent to undo changes that affected unrelated packages, noting that these modifications were out of scope. Subsequently, Copilot reverted the unintended changes [34] and addressed compilation issues. However, the PR was eventually abandoned after nearly two months of review (from June 25 to August 20, 2025). This example shows how AI-generated commits can cause unintended side effects, leading to partial or complete reverts, often due to a mismatch between the agent's behavior and the developer's intent.

To understand the factors that lead to reverted changes, in the code review process, prior studies [37, 43, 45] have characterized the causes of reverted commits in human-driven software development. However, the prevalence and underlying causes of reverted changes in PRs generated by AI coding agents remain underexplored. In agentic PRs, code change reverts may arise not only from defect correction but also from unintended side effects, deviations



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/author(s).

ACM ISBN 979-8-4007-2474-9/26/04

<https://doi.org/10.1145/3793302.3793587>

¹<https://claude.com/solutions/coding>

²<https://copilot.microsoft.com/>

³<https://cursor.com/>

⁴<https://devin.ai/>

⁵<https://chatgpt.com/features/codex>

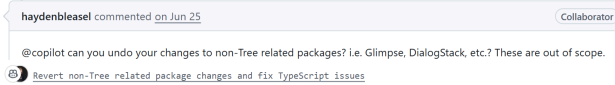


Figure 1: An example Reverted Copilot-Authored Change.

from developer intent, or an expanded review scope. Such reverts typically translates to a substantial number of instances where AI-generated code fails to meet expectations, requiring intensive manual intervention [18, 32, 44]. The lack of empirical evidence on the prevalence of such reverts and the factors that trigger them limits our understanding of how AI coding agents behave, and how they are trusted in real-world development workflows.

In this paper, we aim to *investigate the prevalence of reverting commits and understand why AI-assisted changes are reverted in agent-generated PRs*. To achieve this goal, we conduct a mixed-methods empirical study that combines quantitative and qualitative analyses. Using the AIDev dataset [26], which includes more than 33K agentic PRs spanning 86,315 commits from five major AI agents, we analyze 1,141 reverting commits and their corresponding 893 agent-generated pull requests. We define two research questions:

— **RQ1: What is the prevalence of reverted changes in agentic PRs?** We quantify revert rates across five AI agents to understand how frequently AI-generated code requires rollback. We find 2.66% of PRs contain reverts, with rates varying tenfold across AI agents (0.7%–7.6%).

— **RQ2: Why are agentic changes being reverted?** We develop a taxonomy of revert reasons to characterize the specific failure modes of agentic code generation using thematic analysis. We identify 8 categories and 25 themes, with overengineering/side effects (22.33%) and functional incorrectness (22.13%) as dominant failure modes.

These findings reveal that current agentic coding tools exhibit quality variations and struggle primarily with software engineering concerns. For practitioners, our results suggest prioritizing human review of agentic generated changes. For researchers and tool developers, we identify scope control and contextual awareness as critical areas for improvement in next-generation agentic systems.

Replication Package. We provide our custom scripts and data in a comprehensive replication package [6].

2 Study Design

Our goal is to investigate reverted changes in AI-generated pull requests, focusing on their prevalence and underlying causes. Figure 2 provides an overview of our approach.

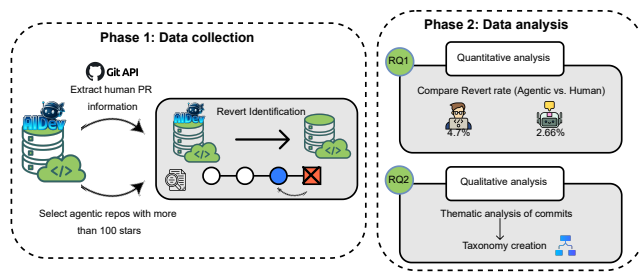


Figure 2: Approach Overview.

2.1 Data Collection

We use the AIDev dataset [25], comprising 33,580 AI-generated PRs with over 80K commits from five major coding agents: GitHub Copilot, OpenAI Codex, Anthropic Claude Code, Cursor, and Devin. Each PR includes metadata such as patches and commit messages. Human PR data was extracted using the GitHub REST API⁶ for comparative analysis.

2.2 Data Analysis

2.2.1 Quantitative Analysis. To address RQ1, we identify reverting commits using: (1) Git Revert Detection: regular expressions matching standard format (e.g., “Revert ”... This reverts commit <hash>”) [37], and (2) Semantic Revert Detection: NLP identifying phrases like “revert”, “undo”, “rollback”, and “backing out”. Thereafter, we manually validated 500 commits (Section 2.2.2) achieving 96.2% precision [45], false positives were primarily metaphorical uses. This yields 1,141 revert commits across 893 PRs. We calculate: $\text{PR Revert Rate} = \frac{\# \text{ PRs having at least one revert}}{\text{Total PRs}} \times 100$ overall and per agent, enabling comparison across AI agents.

2.2.2 Qualitative Analysis. To answer RQ2, we apply thematic analysis [5, 15, 38] to construct a revert taxonomy. We randomly sample 500 commits stratified by agent distribution, exceeding thresholds for 95% confidence and theoretical saturation [5, 7, 8, 11, 12, 38]. Our taxonomy comprises *categories* (high-level failure modes) and *themes* (specific revert reasons), developed through four steps:

Step 1: Initial Reading. Two authors independently analyzed all 500 commits, examining: (1) commit messages and diffs, (2) PR discussions and agent interactions (e.g., “@copilot”), and (3) surrounding context to identify primary reversion reasons.

Step 2: Label Generation and Refinement. Each author independently assigned descriptive labels. Authors then met to discuss and refine labels, identifying 19 initial labels. Semantically distinct cases were split (e.g., extracting “merge conflict” from “breaking changes”), yielding 25 final labels.

Step 3: Disagreement Resolution. Initial coding achieved substantial agreement ($\kappa = 0.62$) [13, 23]. Through consensus discussion, we reassigned themes to appropriate categories, identified missing themes (e.g., “wrong naming”), and refined labels for clarity (e.g., “CI & Integration Failures” became “Workflow & Pipeline Issues”). Final agreement reached $\kappa = 1.0$ [28].

Step 4: Hierarchical Categorization. Through iterative organization, similar themes were grouped into higher-level categories. For example, “Violations of conventions,” “Redundant logic,” “Formatting issues,” “Wrong naming,” and “Documentation Issues” were grouped under “Code Quality & Maintainability Issues.” This ensured coverage and mutual exclusivity, yielding 8 categories and 25 themes.

3 Empirical Study Results

3.1 RQ1: What is the prevalence of reverted changes in agentic PRs?

Our analysis reveals that 2.66% of agentic PRs (893 out of 33,580) contain at least one AI authored reverting commit, indicating a non-negligible prevalence. Although this percentage appears low, reverted commits often incur disproportionate effort due to rework, CI

⁶<https://docs.github.com/en/rest?apiVersion=2022-11-28>

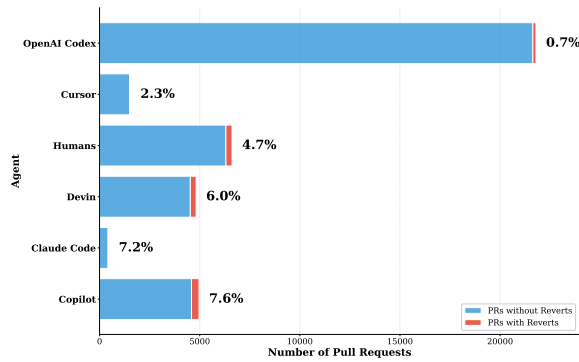


Figure 3: Percentage of PRs with Reverts for each AI Agent

re-execution, and additional review [37]. 1,141 distinct revert commits distributed across the 893 affected PRs, averaging 1.27 reverts per affected PR. This finding suggests that when AI-generated code requires correction, developers typically address issues through a single revert, although some PRs undergo multiple revert iterations. Comparing AI agents to human developers provides important context for evaluating code quality. Human-authored PRs exhibit a revert rate of 4.68% (310 of 6,618 PRs), approximately 1.76 times higher than the aggregate AI agent rate of 2.66%. This suggests that, overall, AI-generated code requires less correction through reverts than human-written code. However, as illustrated in Figure 3, revert rates vary substantially across individual AI agents, spanning more than a tenfold range from 0.72% to 7.57%. GitHub Copilot exhibits the highest revert rate at 7.57% (376 of 4,967 PRs), exceeding the human by 62%. Claude Code follows closely at 7.21% (33 of 458 PRs), also exceeding the human rate. Devin (6.03%, 291 of 4,822 PRs) and Cursor (2.34%, 36 of 1,540 PRs) fall below the human baseline, while OpenAI Codex demonstrates a remarkably low revert rate of 0.72% (157 of 21,793 PRs), more than six times lower than humans and over ten times lower than Copilot, despite contributing the largest volume of PRs to the dataset. These disparities suggest that the tendency for generating revert-triggering code varies considerably across agents. OpenAI Codex’s notably low revert rate may indicate more conservative code-generation strategies, better alignment with established development patterns, or specialization in well-defined tasks with precise requirements. Conversely, higher rates for Copilot and Claude Code might reflect more aggressive feature suggestions, broader task scopes, or different interaction paradigms that encourage exploratory code generation. The fact that two agents (Copilot / Claude Code) exceed human revert rates while three others fall below suggests that agent architecture and design choices significantly impact code quality.

3.2 RQ2: Why are agentic changes being reverted?

Our manual analysis revealed 8 categories and 25 themes characterizing why AI-generated changes are reverted. As shown in Figure 4, our taxonomy includes: (1) **Overengineering Side Effects**, (2) **Functional Incorrectness**, (3) **Code Quality Issues**, (4) **Dependency Management Issues**, (5) **Test Failures**, (6) **Workflow**

& Pipeline Issues, (7) **Security Violations**, and (8) **Ambiguous Revert**.

(C1) Overengineering Side Effects account for 22.33% of reverts, representing significant integration challenges. AI agents frequently introduce unnecessary complexity, “gold plating” [27], manifesting as unintended behavioral changes (12.47%), unrelated file changes (7.04%), and excessive refactoring (2.82%). These subtle alterations reveal AI’s struggle with restraint and scope management. Agents optimize for comprehensiveness when developers need precision, creating technical debt where maintenance costs outweigh productivity gains [22]. Current AI models lack the holistic system understanding to judge when “good enough” suffices and when touching working code introduces more risk than value [10].

(C2) Functional Incorrectness represents the second major challenge, accounting for 22.13% of all reverted changes. These are traditional bugs in which AI-generated code does not work as intended. The breakdown reveals the specific failure modes: logical errors dominate at 11.67%, followed by partial implementations that leave functionality incomplete (7.44%), misinterpreted requirements in which the AI solves the wrong problem (1.61%), and edge-case failures in which the code breaks under unexpected conditions (1.41%). This distribution shows that, despite their sophistication, AI models still struggle with the fundamental task of translating intent into executable code. The prevalence of logical errors and partial implementations suggests that AI agents often understand what to build, but falter in the how [17]. These findings underscore a critical reality: AI can accelerate code generation, but it cannot yet replace human oversight, rigorous testing, and critical thinking required to ensure functional correctness [46], making verification processes more essential than ever in AI-assisted development workflows.

(C3) Code Quality Issues drive 17.71% of reverts, demonstrating functional correctness alone is insufficient. Documentation issues lead at 6.04%, followed by convention violations at 5.43%. For example, PR #58076 [3] required reverting “docs: revert incorrect platform/release_notes links” to correct misleading documentation. AI agents deliver working code that fails professional standards: proper structure, consistent naming, adequate documentation, and established conventions [30]. The code executes correctly but creates friction for developers who must later maintain it. This creates subtle technical debt where AI solves today’s problem while creating tomorrow’s maintenance burden [29]. Documentation issues suggest agents struggle to explain their code, while convention violations indicate failure to internalize project-specific standards despite diverse training [7, 41].

(C4) Dependency Management Issues account for 12.47% of reverts, exposing critical weaknesses in handling external libraries and version compatibility. Breaking changes from upgrades dominate at 6.64%, where agents introduce incompatible version updates without understanding downstream consequences. For instance, in PR #62036 [4], an agent upgraded `@microsoft/signalr-protocol-msgpack` from version 5.0.0 to 10.0.0 without verifying compatibility with existing infrastructure. The revert removed the version specification entirely, reverting to the stable version. Lock file restorations (3.42%) and missing or incompatible dependencies (2.41%) reveal how AI treats dependencies as simple additions rather than interconnected ecosystem components [16]. Agents confidently upgrade

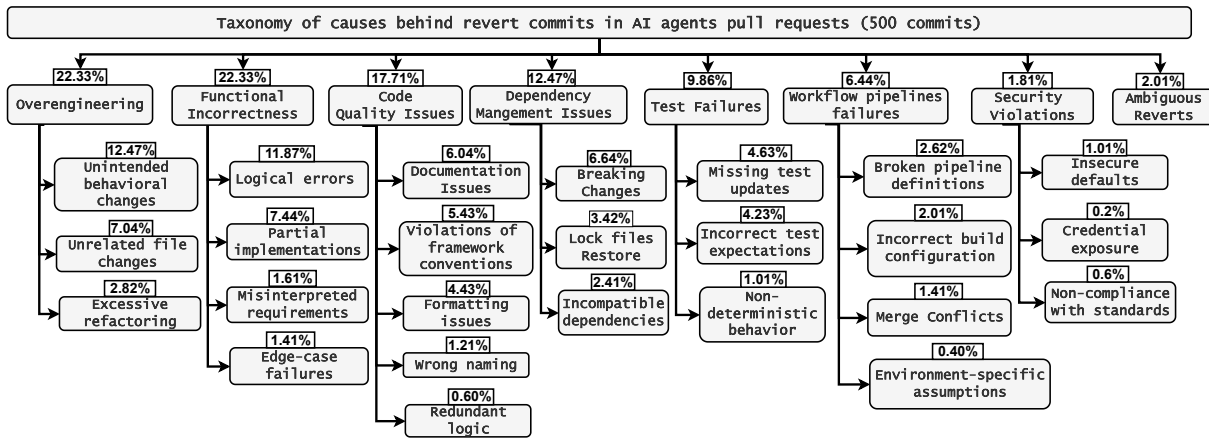


Figure 4: Taxonomy of causes behind revert commits in AI agents pull requests.

libraries without recognizing compatibility breaks or add packages without checking conflicts. A single mismanaged dependency can trigger cascading build failures, runtime errors, and subtle bugs [14].

■ **(C5) Test Failures** comprise 9.86% of reverts, involving test maintenance and validation issues. Missing test updates lead at 4.63%, where agents modify code without updating tests, followed by incorrect test expectations (4.23%) and nondeterministic behavior (1.01%). For example, in PR #18584 [2], an agent modified CLIEvent handling in the F# compiler service, breaking three existing tests. The developer reverted with: “Revert changes to CLIEvent handling to fix test failures,” directly addressing the agent: “@copilot: You caused 3 test failures”, detailing how modifications altered event classification behavior violating existing test expectations. Agents either neglect test updates entirely or introduce changes breaking existing expectations, creating false security where modifications appear valid until the test suite reveals fundamental behavioral changes [9]. This highlights the need for AI models capable of understanding and maintaining the contract between code behavior and test expectations [24].

■ **(C6) Workflow & Pipeline Issues** account for 6.44% of reverts, encompassing build configurations, CI/CD pipelines, and development workflows. Broken CI scripts or pipeline definitions (2.62%) and incorrect build configurations (2.01%) dominate. Agents inadvertently disrupt automated systems by making incorrect modifications without understanding full implications [42]. Given the complexity and project-specific nature of pipelines, agents struggle to grasp how modifications ripple through the entire software development ecosystem beyond individual code files [21], where AI’s narrow focus on immediate code generation conflicts with broader context needed for successful delivery [36].

■ **(C7) Security Violations** represent only 2.01% of reverts but carry disproportionate risk. Insecure defaults (1.01%), non-compliance with standards (0.80%), and credential exposure (0.20%) reveal how AI models can inadvertently learn and replicate insecure coding patterns from public code [31]. Agents can also introduce issues by misapplying protective mechanisms. In PR #1270 [1], Codex attempted to clean user input before security validation by sanitizing a WordPress nonce. The developer reverted with: “We shouldn’t clean up a submitted nonce. If we have to, it should be considered invalid and rejected.” This illustrates how agents may misunderstand security best practices, sanitizing a nonce before verification

undermines the security check itself, as valid nonces should not require sanitization. Such misapplications can introduce vulnerabilities including SQL injection, cross-site scripting, or weakened cryptographic guarantees.

■ **(C8) Ambiguous Revert** accounts for 2.21% of cases where reversion reasons couldn’t be determined. This encompasses commits with: (1) uninformative revert messages (e.g., simply “revert” without explanation), (2) deleted or unavailable PR discussions, (3) insufficient context, or (4) multiple potential reasons with no clear primary cause. Despite thorough examination of commit diffs, PR history, and related discussions, some reverts remain unclear.

4 Conclusion and Takeaways

As AI coding agents become more integrated into software development workflows, understanding how and why their contributions fail in practice is increasingly important. In this study, we analyzed 1,141 reverting commits across 893 PRs authored by five widely used AI coding agents. While reverts remain relatively uncommon overall (2.67%), they reveal systematic reliability differences and recurring challenges beyond conventional bug fixing.

Takeaway #1: Revert rates vary substantially across agents. Although only a small fraction of agentic PRs contain reverting commits, the observed rates differ markedly across tools, ranging from 0.7% (OpenAI Codex) to 7.6% (GitHub Copilot). This variability suggests that agent behavior and failure profiles are not uniform, and that practitioners should account for agent-specific reliability when integrating AI agents into production pipelines.

Takeaway #2: Failures often stem from scope and boundary misalignment, not just incorrect code. Overengineering and unintended side effects (22.33%) occur at a rate comparable to functional defects (22.13%). In many cases, the reverted changes were technically correct but exceeded the intended scope by modifying unrelated components or introducing unnecessary complexity. This indicates that current AI agents struggle to interpret the contextual boundaries of change, underscoring the importance of explicitly reviewing scope when assessing AI-generated contributions.

Takeaway #3: Infrequent security-related errors still warrant special attention. Security-related reverts represent a small proportion of cases (2.01%), yet they include errors with potentially high impact, such as weakening authentication or misapplying sanitization logic. These findings suggest that AI-generated code

should be subject to targeted security checks, combining automated analysis with focused human review.

Overall, our results show that the reverted AI-generated changes provide a valuable lens into the practical limitations of current agentic systems. By characterizing where and how these failures occur, this study provides evidence to inform safer deployment practices, more effective review workflows, and future improvements to agent design.

References

- [1] 2025. Pull Request #1270. <https://github.com/siteorigin/siteorigin-panels/pull/1270/commits/ae2e1e5e5685a4024661695869b0903875d98bd7>. Dec. 2025.
- [2] 2025. Pull Request #18584. <https://github.com/dotnet/fsharp/pull/18584/commits/a7c9a7ce0ab1a4f104cbc7fa9b427a78785dccb5>. Accessed Dec. 2025.
- [3] 2025. Pull Request #58076. <https://github.com/airbytehq/airbyte/pull/58076/commits/5fb3ba4afe8c3471effad4070478e16e704dcbca>. Accessed Dec. 2025.
- [4] 2025. Pull Request #62036. <https://github.com/dotnet/aspnetcore/pull/62036/commits/193c6a75b0d0c71fd8a399204aa6924348242d1f>. Accessed Dec. 2025.
- [5] Eman Abdullah AlOmar, Moataz Chouchen, Mohamed Wiem Mkaouer, and Ali Ouni. 2022. Code review practices for refactoring changes: An empirical study on openstack. In *international conference on mining software repositories*. 689–701.
- [6] Anonymous. 2025. *Replication package for "When AI Code Doesn't Stick: An Empirical Study on Reverted Changes Introduced by AI Coding Agents"*. <https://figshare.com/s/bbd010fbfc71a63781f5>
- [7] Alberto Bacchelli and Christian Bird. 2013. Expectations, outcomes, and challenges of modern code review. (2013), 712–721.
- [8] Gabriele Bavota, Bernardino De Carluccio, Andrea De Lucia, Massimiliano Di Penta, Rocco Oliveto, and Orazio Strollo. 2012. When does a refactoring induce bugs? an empirical study. In *International Working Conference on Source Code Analysis and Manipulation*. IEEE, 104–113.
- [9] Jonathan Bell, Owlabi Legunsen, Michael Hilton, Lamyaa Eloussi, Tiffany Yung, and Darko Marinov. 2018. DeFlaker: Automatically detecting flaky tests. In *international conference on software engineering*. 433–444.
- [10] Moataz Chouchen, Narjes Bessghaier, Mahi Begoug, Ali Ouni, Eman Alomar, and Mohamed Wiem Mkaouer. 2024. How do software developers use chatgpt? an exploratory study on github pull requests. In *International Conference on Mining Software Repositories*. 212–216.
- [11] Moataz Chouchen, Ali Ouni, Raula Gaikovina Kula, Dong Wang, Patanamom Thongtanunam, Mohamed Wiem Mkaouer, and Kenichi Matsumoto. 2021. Anti-patterns in modern code review: Symptoms and prevalence. In *international conference on software analysis, evolution and reengineering*. IEEE, 531–535.
- [12] Flávia Coelho, Nikolaos Tsantalis, Tiago Massoni, and Everton LG Alves. 2021. An empirical study on refactoring-inducing pull requests. In *International Symposium on Empirical Software Engineering and Measurement*. 1–12.
- [13] Jacob Cohen. 1960. A coefficient of agreement for nominal scales. *Educational and psychological measurement* 20, 1 (1960), 37–46.
- [14] Russ Cox. 2019. Surviving software dependencies. *Commun. ACM* 62, 9 (2019), 36–43.
- [15] Daniela S Cruzes and Tore Dyba. 2011. Recommended steps for thematic synthesis in software engineering. In *international symposium on empirical software engineering and measurement*. IEEE, 275–284.
- [16] Alexandre Decan, Tom Mens, and Philippe Grosjean. 2019. An empirical comparison of dependency network evolution in seven software packaging ecosystems. *Empirical Software Engineering* 24, 1 (2019), 381–416.
- [17] Yihong Dong, Xue Jiang, Jiaru Qian, Tian Wang, Kechi Zhang, Zhi Jin, and Ge Li. 2025. A survey on code generation with llm-based agents. *arXiv preprint arXiv:2508.00083* (2025).
- [18] Dena Ford, Mahnaz Behrooz, Alexander Serebrenik, and Chris Parnin. 2019. Beyond the code itself: how programmers really look at pull requests. In *International Conference on Software Engineering: Software Engineering in Society*. IEEE, 51–60.
- [19] Ruofan Gao, Amjed Tahir, Peng Liang, Teo Susnjak, and Foutse Khomh. 2025. A Survey of Bugs in AI-Generated Code. *arXiv preprint arXiv:2512.05239* (2025).
- [20] Ahmed E Hassan, Hao Li, Dayi Lin, Bram Adams, Tse-Hsun Chen, Yutaro Kashiwa, and Dong Qiu. 2025. Agentic Software Engineering: Foundational Pillars and a Research Roadmap. *arXiv preprint arXiv:2509.06216* (2025).
- [21] Michael Hilton, Timothy Tunnell, Kai Huang, Darko Marinov, and Danny Dig. 2016. Usage, costs, and benefits of continuous integration in open-source projects. In *international conference on automated software engineering*. 426–437.
- [22] Philippe Kruchten, Robert L Nord, and Ipek Ozkaya. 2012. Technical debt: From metaphor to theory and practice. *Ieee software* 29, 6 (2012), 18–21.
- [23] J Richard Landis and Gary G Koch. 1977. The measurement of observer agreement for categorical data. *biometrics* (1977), 159–174.
- [24] Caroline Lemieux, Jeevana Priya Inala, Shuvendu K Lahiri, and Siddhartha Sen. 2023. Codamosa: Escaping coverage plateaus in test generation with pre-trained large language models. In *45th International Conference on Software Engineering*. IEEE, 919–931.
- [25] Hao Li et al. 2024. AIDev: A Large-Scale Dataset of AI-Generated Pull Requests. Hugging Face Datasets. <https://huggingface.co/datasets/hao-li/AIDev>
- [26] 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).
- [27] Steve McConnell. 2004. *Code complete*. Pearson Education.
- [28] Mary L McHugh. 2012. Interrater reliability: the kappa statistic. *Biochemia medica* 22, 3 (2012), 276–282.
- [29] David O'Brien, Sumon Biswas, Sayem Imtiaz, Rabe Abdalkareem, Emad Shihab, and Hridesh Rajan. 2022. 23 shades of self-admitted technical debt: an empirical study on machine learning software. In *ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 734–746.
- [30] Sebastian Panichella and Nik Zaugg. 2020. An empirical investigation of relevant changes and automation needs in modern code review. *Empirical Software Engineering* 25, 6 (2020), 4833–4872.
- [31] Hammond Pearce, Baleegh Ahmad, Benjamin Tan, Brendan Dolan-Gavitt, and Ramesh Karri. 2025. Asleep at the keyboard? assessing the security of github copilot's code contributions. *Commun. ACM* 68, 2 (2025), 96–105.
- [32] Achyudh Ram, Anand Ashok Sawant, Marco Castelluccio, and Alberto Bacchelli. 2018. What makes a code change easier to review: an empirical investigation on code change reviewability. In *ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 201–212.
- [33] Abhik Roychoudhury, Corina Pasareanu, Michael Pradel, and Baishakhi Ray. 2025. Agentic ai software engineer: Programming with trust. *arXiv preprint arXiv:2502.13767* (2025).
- [34] shadcnblocks. 2024. Commit b837b5112c0602e25896a8eac3318dc6b9f9d747. <https://github.com/shadcnblocks/kibo/pull/171/commits/b837b5112c0602e25896a8eac3318dc6b9f9d747>. Accessed: 2025-01.
- [35] shadcnblocks. 2024. Pull request #171. <https://github.com/shadcnblocks/kibo/pull/171>. Accessed: 2025-01.
- [36] Mojtaba Shahin, Muhammad Ali Babar, and Liming Zhu. 2017. Continuous integration, delivery and deployment: a systematic review on approaches, tools, challenges and practices. *IEEE access* 5 (2017), 3909–3943.
- [37] Junji Shimagaki, Yasutaka Kamei, Shane McIntosh, David Purchase, and Naoyasu Ubayashi. 2016. Why are commits being reverted?: A comparative study of industrial and open source projects. In *International Conference on Software Maintenance and Evolution*. IEEE, 301–311.
- [38] Danilo Silva, Nikolaos Tsantalis, and Marco Tulio Valente. 2016. Why we refactor? confessions of github contributors. In *international symposium on foundations of software engineering*. 858–870.
- [39] Joseph Spracklen, Raveen Wijewickrama, AHM Nazmus Sakib, Anindya Maiti, and Bimal Viswanath. 2025. We have a package for you! a comprehensive analysis of package hallucinations by code generating {LLMs}. In *USENIX Security Symposium*. 3687–3706.
- [40] Florian Tambon, Arghavan Moradi-Dakheel, Amin Nikanjam, Foutse Khomh, Michel C Desmarais, and Giuliano Antoniol. 2025. Bugs in large language models generated code: An empirical study. *Empirical Software Engineering* 30, 3 (2025), 65.
- [41] Rosalia Tufano. 2023. Automating code review. In *International Conference on Software Engineering: Companion Proceedings*. IEEE, 192–196.
- [42] Carmine Vassallo, Gerald Schermann, Fiorella Zampetti, Daniele Romano, Philipp Leitner, Andy Zaidman, Massimiliano Di Penta, and Sebastiano Panichella. 2017. A tale of CI build failures: An open source and a financial organization perspective. In *conference on software maintenance and evolution*. IEEE, 183–193.
- [43] Fengcai Wen, Csaba Nagy, Michele Lanza, and Gabriele Bavota. 2020. An empirical study of quick remedy commits. In *Proceedings of the 28th International Conference on Program Comprehension*. 60–71.
- [44] Mairieli Wessel, Alexander Serebrenik, Igor Wiese, Igor Steinmacher, and Marco A Gerosa. 2020. Effects of adopting code review bots on pull requests to OSS projects. In *international conference on software maintenance and evolution*. IEEE, 1–11.
- [45] Meng Yan, Xin Xia, David Lo, Ahmed E Hassan, and Shanping Li. 2019. Characterizing and identifying reverted commits. *Empirical Software Engineering* 24, 4 (2019), 2171–2208.
- [46] Huaen Zhang, Yu Pei, Junjie Chen, and Shin Hwei Tan. 2023. Statfier: Automated testing of static analyzers via semantic-preserving program transformations. In *ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 237–249.
- [47] Ziyao Zhang, Chong Wang, Yanlin Wang, Ensheng Shi, Yuchi Ma, Wanjuan Zhong, Jiachi Chen, Mingzhi Mao, and Zibin Zheng. 2025. Llm hallucinations in practical code generation: Phenomena, mechanism, and mitigation. *ACM on Software Engineering* 2, ISSTA (2025), 481–503.