"An Endless Stream of AI Slop": How Developers Discuss the Burden of AI-Assisted Software Development

Sebastian Baltes*, Marc Cheong†, Christoph Treude‡
*Heidelberg University, Germany
sebastian.baltes@uni-heidelberg.de
†University of Melbourne, Australia
marc.cheong@unimelb.edu.au
‡Singapore Management University, Singapore
ctreude@smu.edu.sg

***Abstract*—"AI slop", that is, low-quality AI-generated content, is increasingly affecting software development, from generated code and pull requests to documentation and bug reports. However, there is limited empirical research on how developers perceive and respond to this phenomenon. We qualitatively analyzed how developers discuss AI slop in 1,154 Reddit and Hacker News posts, developing a codebook of 15 codes organized into three thematic clusters: *Review Friction* (how AI slop burdens reviewers, erodes trust, and prompts countermeasures), *Quality Degradation* (damage to codebases, knowledge resources, and developer competence), and *Forces and Consequences* (systemic incentives, mandated adoption, craft erosion, and workforce disruption). Our findings frame AI slop as a *tragedy of the commons*, where individual productivity gains externalize costs onto reviewers, maintainers, and the broader community. We report the concerns developers raise and the mitigation strategies they propose, with implications for tool developers, team leads, and educators.**

***Index Terms*—AI slop, generative AI, AI-assisted software development, developer discourse, qualitative study**

I. Introduction

"AI slop" was named Merriam-Webster's 2025 Word of the Year.[1] The term describes low-quality digital content produced, usually in quantity, with AI. Like "spam" before it, the term names a category of unwanted digital content. Kommers et al. identify three prototypical properties of AI slop [1]: *superficial competence* (a veneer of quality belied by a deeper lack of substance), *asymmetry of effort* (creation requires vastly less labor than would be needed without AI), and *mass producibility* (content exists within a digital ecosystem of widespread generation and consumption). These properties separate AI slop from ordinary low-quality code and related terms: Slop is superficially competent, effort-asymmetric, and mass-producible. A hallucination, i.e., "*a plausible but false or misleading response generated by an artificial intelligence algorithm*",[2] names a correctness failure of model output—slop can be free of hallucinations and still waste its recipients' time. Spam, i.e., "*unsolicited usually commercial messages*",[3] serves a commercial or criminal interest. Slop requires no such motive and can arise from good-faith contributions such as pull requests or bug reports. Niederhoffer et al. coined the related term "workslop" for AI-generated content that destroys workplace productivity.[4]

Much of the public discourse around AI slop has focused on social media[5] and search engine pollution.[6] However, AI slop has tangible impact on software development as well. It pervades code, bug reports, pull requests, documentation, and Stack Overflow answers. In open source, the problem is acute: "*AI slop is ripping up the social contract between maintainers and contributors essential to open source development*".[7]

The consequences are already visible. The *curl* project shut down its bug bounty program after AI-generated vulnerability reports consumed maintainer time without producing valid findings.[8] Apache *Log4j 2*[9] and the Godot game engine[10] reported similar problems with AI-generated contributions that drained maintainer capacity. Koren et al. argue more broadly that *vibe coding* threatens the sustainability of open-source ecosystems by undermining maintainer incentives [2].

This pattern resembles a *tragedy of the commons* [3], especially in open source where shared resources are maintained by volunteers. Individual developers and organizations benefit from AI-generated content, but the cumulative effect degrades the shared resources that collaborative development depends on. Codebases accumulate technical debt, knowledge resources become polluted, reviewer capacity is exhausted, and interpersonal trust erodes. Each AI-generated submission that skips quality review *externalizes* its costs onto reviewers, maintainers, and the broader community. Prause previously

[1] merriam-webster.com/slang/slop
[2] merriam-webster.com/dictionary/hallucination
[3] merriam-webster.com/dictionary/spam
[4] hbr.org/2025/09/ai-generated-workslop-is-destroying-productivity
[5] 9news.com.au/world/openai-sora-...
[6] nymag.com/intelligencer/.../google-amazon-slop-internet
[7] redmonk.com/kholterhoff/2026/02/03/ai-slopageddon-...
[8] lists.haxx.se/pipermail/daniel/2026-January/000143.html
[9] github.com/apache/logging-log4j2/discussions/4052
[10] bsky.app/profile/.../post/3meyerixvhs2p

applied this framing to documentation quality in software projects, arguing that documentation has low value to individual developers but high social cost when absent [4].

Yet there is limited empirical research on how developers perceive and react to AI slop, even as practitioners report that mandated AI coding tools are "*rapidly becoming an endless stream of AI slop*" [R07]. We study AI slop as a category in developer discourse, that is, how practitioners name, frame, and respond to it. We conducted a qualitative analysis of developer discourse on Reddit and Hacker News, guided by the following research question:

> **RQ:** *How do software developers perceive and discuss "AI slop"?*

We analyzed 1,154 posts across 15 documents, developing a codebook of 15 codes organized into three thematic clusters: *Review Friction*, *Quality Degradation*, and *Forces and Consequences*. Our findings capture the concerns developers raise and the strategies they propose, offering practical guidance for tool developers, team leads, and educators.

II. Related Work

Prior work has examined AI-generated code quality, finding that LLM output frequently contains bugs and security vulnerabilities [5, 6]. Surveys explore how developers perceive and adopt AI coding assistants as productivity aids [7]. Recent work analyzing Hacker News examined how AI-powered GitHub projects are promoted and received in developer communities [8]. The degradation of online information quality through AI-generated content has been documented from model collapse on synthetic data [9] to incorrect AI-generated programming answers [10]. Research on open-source sustainability has long examined maintainer attention, maintenance burden, and burnout in collaborative development [11, 12]. AI slop introduces a new dimension by lowering the effort required to produce contributions while shifting the burden of quality assurance onto maintainers.

Our work differs from these studies in two ways. First, we study AI slop as a *named* phenomenon, capturing how developers themselves frame the problem. Second, our qualitative approach captures the interplay between technical, social, and economic dimensions as they surface in practitioner discourse.

III. Research Method

A. Data Collection

We collected discussions from *Reddit*[11] and *Hacker News*.[12] On September 26–27, 2025, we searched for the exact phrase "ai slop," with no date restriction, in three subreddits (`r/programming`, `r/learnprogramming`, `r/ExperiencedDevs`) and on Hacker News, where we narrowed the topical scope by adding the keyword "software," because it has no equivalent of subreddits. For both sources, we screened all returned results and captured every discussion with at least one vote and one comment. This yielded 16 documents: 13 Reddit threads (R01, R02, R03, R04, R05, R06, R07, R08, R09, R10, R11, R12, R13) and three Hacker News result pages (H01, H02, H03). One Reddit thread (R09) was later excluded because it focused on successful AI use without engaging with AI slop as a problem. This left 15 documents (1,154 posts) in the final corpus.

We captured all threads and result pages as PDF files, which were the basis for our qualitative analysis. The earliest Reddit thread is from January 2025, and the cited Hacker News comments are from April to September 2025.

B. Coding Procedure

Our analysis followed an iterative qualitative coding approach. The second author performed open coding on five documents (R01–R05), generating 40 initial codes. All three authors then performed axial coding, grouping related open codes into 8 consolidated codes by comparing their meaning, scope, and overlap. The first author refined the codebook through ten revisions,[13] assisted by Claude Code (Opus 4.6, high effort[14]). Recent work cautions that claims about automating qualitative analysis with generative AI overgeneralize from narrow successes [13], although studies have used LLM-assisted qualitative coding [14]. We report Claude Code's version, its role at each step, and the human verification of its output, following the community guidelines for empirical studies involving LLMs.[15] The first author retained decision authority over all structural changes. The other two authors validated the final code set using an interactive visualization.[16]

The first author then annotated the full corpus with Claude Code in two passes (filtering for relevant sub-discussions, then coding) and reviewed the output at each step. Claude Code assigned the codes, but the authors held authority over every coding decision. Rather than checking only a sample of its labels, as is common, the second author then checked every label across all 15 documents over four review rounds,[16] changing 234 of them. A detailed account is available in the online supplementary material.[13,16]

C. Final Codebook

The final codebook consists of 15 codes: one *rhetorical* code (`sarcastic-skepticism`) and 14 *topical* codes. During the codebook development, we recorded relationships between codes as a graph whose nodes are codes and whose edges are conceptual links the authors identified by hand, capturing causal connections, scope distinctions, and thematic overlaps. We applied Louvain community detection, which partitions a graph by maximizing modularity, to identify three thematic clusters (Figure 1), which we use to organize the presentation of results: *Review Friction*, *Quality Degradation*, and *Forces and Consequences*. The codebook, corpus, and all annotated data are available online.[13,16]

[11] https://www.reddit.com
[12] https://news.ycombinator.com/
[13] https://doi.org/10.5281/zenodo.19283651
[14] We used the same tool/model to proofread and tighten this manuscript.
[15] llm-guidelines.org
[16] https://se-uhd.de/ai-slop/

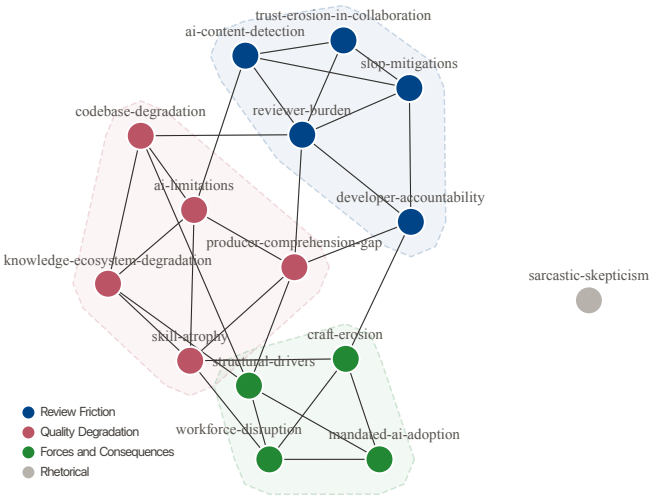

Fig. 1. Code relationship network with Louvain community clusters. Edges represent conceptual relationships between codes, defined by the authors, capturing causal links, scope distinctions, and thematic overlaps. Node colors indicate cluster membership. An interactive version is available online.[16]

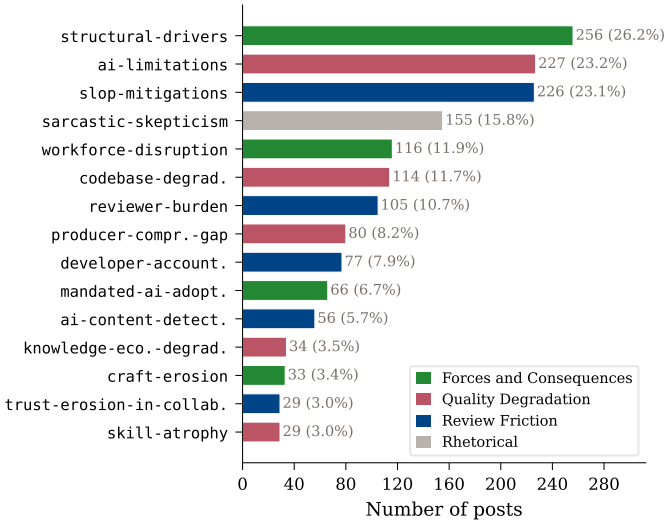

Fig. 2. Code frequency distribution across 978 coded posts. Bar colors indicate cluster membership. An interactive version with co-occurrence data is available online.[16]

IV. Results

We present findings along the three topical clusters (Table I), plus the rhetorical code `sarcastic-skepticism`. The source IDs (e.g., R02) link to the original online posts.

A. Annotation Overview

We annotated all 1,154 posts across 15 documents using the final codebook. Of all 1,154 posts, 978 (84.7%) received at least one code, yielding 1,603 codings. On average, these posts received 1.6 codes, showing that developers often address multiple themes in one post.

Figure 2 shows the frequency distribution. The three most frequent topical codes are `structural-drivers` (256, or 26.2% of coded posts), `ai-limitations` (227), and `slop-mitigations` (226), which together account for 44.2% of all 1,603 codings. They reflect the fundamental questions: *why* slop is produced, *what* makes it problematic, and *how* to counter it. The rhetorical code `sarcastic-skepticism` (155) ranks fourth, indicating that ironic framing was common in this discourse. Co-occurrence analysis shows cross-cutting patterns (`sarcastic-skepticism` pairs most often with `structural-drivers` and `ai-limitations`) and within-cluster pairings consistent with the Louvain clustering.

B. Review Friction

This cluster spans five codes addressing how AI slop disrupts collaborative development: detecting AI content (`ai-content-detection`), the workload it creates (`reviewer-burden`), the trust it erodes (`trust-erosion-in-collaboration`), and how developers respond through accountability norms (`developer-accountability`) and concrete countermeasures (`slop-mitigations`).

Detection. Reviewers developed pattern recognition for AI-generated code. Explicit markers include emojis in code comments [R06], step-by-step commenting, verbose style, and Unicode artifacts [R05]. Others relied on tacit recognition: "*Someone on my team will publish a PR, it's like 1–2k lines of code and after looking at it for 5 minutes I can tell it's pretty much entirely AI generated*" [R05].

Reviewer burden. The workload asymmetry between AI generation and human review was a dominant theme. "*The development time has been shortened but the team now needs to spend more time to review. Doesn't look like any benefit*" [R05]. One team reported receiving 30 PRs per day across 6 reviewers [R07]. Reviewers described feeling like "*the first human being to ever lay eyes on this code*" [R05] and being turned into unpaid prompt engineers: "*They're literally just using you to do their job (i.e., critically evaluate and understand their AI slop and give it the next prompt)*" [R05].

Trust erosion. AI-generated contributions eroded trust. One reviewer described an AI agent's PR: "*I don't know how you could trust any of it [. . .] No real understanding of what it's doing, it's just guessing*" [R13]. Proving AI authorship is hard: "*When the comment smells like AI but I just can't prove it*" [R11].

Accountability and mitigations. In response, developers articulated accountability norms. The norm "*It's not AI's code, it's my code*" [R05] was widely endorsed, with some organizations formalizing it: "*Ownership of a PR always rests with you. It is never acceptable to shift responsibility to AI.*" [R05]. Concrete mitigations included PR size limits ("*less than 500 LOC per PR or they won't review it*" [R05]), requiring self-review before peer review, [R05] synchronous code walkthroughs [R05], and dual code reviews with outside teams [R04].

TABLE I
CODEBOOK OF 15 CODES ORGANIZED INTO THREE TOPICAL AND ONE RHETORICAL CLUSTERS. SOURCE IDS LINK TO THE POSTS ON REDDIT AND HACKER NEWS THAT MOTIVATED THE CORRESPONDING CODE. THE FULL CODEBOOK IS AVAILABLE IN AN INTERACTIVE ONLINE TOOL.[16]

Code	Description	Representative Quote	Examples
Cluster: **Review Friction** (5 codes)			
`ai-content-detection`	Recognizing, flagging, or proving that content is AI-generated.	"*if the comment has an emoji it's a guarantee*"	R05 R05 R06 R06
`reviewer-burden`	Workload asymmetry where AI saves the author time while the reviewer bears the cost.	"*The development time has been shortened but the team now needs to spend more time to review. Doesn't look like any benefit.*"	R05 R05 R05 R05 R05 R13 R07
`trust-erosion-in-collaboration`	Degraded trust in contributors or collaborative processes due to AI content.	"*When the comment smells like AI but I just can't prove it*"	R02 R04 R13 R11 R12
`developer-accountability`	Normative principle: developers bear full responsibility for submitted code, regardless of AI.	"*It's not AI's code, it's my code*"	R05 R05 R05 R05
`slop-mitigations`	Concrete actions taken or proposed to reduce, contain, or respond to AI slop.	"*less than 500 LOC per PR or they won't review it*"	R04 R05 R05 R05 R05
Cluster: **Quality Degradation** (5 codes)			
`ai-limitations`	Poor output quality attributed to what the AI tool itself cannot do, independent of user skill.	"*almost all code ever written is mediocre (or worse) and that's what LLMs have been trained to replicate without understanding*"	R05 R03 R06 R06 R06 R13 R13 R13 R07
`codebase-degradation`	Degraded technical quality within a project or codebase as a consequence of AI slop.	"*You can go very fast with AI, but you accrue technical debt at a much higher speed too*"	R06 R06 R05 R04 R13 R07 H03
`knowledge-ecosystem-degradation`	Degraded quality of external knowledge resources (docs, tutorials, Q&A sites).	"*I'm starting to see documentation and tutorials missing key information and code samples needed to be able to implement something now*"	R11 R11 R11 R11
`producer-comprehension-gap`	Producers who lack the understanding needed to evaluate their own AI-generated output.	"*I straight up asked them if they know what their code does. They didn't. The PR was not approved.*"	R02 R01 R05 R06
`skill-atrophy`	Declining developer capability over time due to AI reliance.	"*many in our field are happily contributing to that future themselves by letting AI do their own job and slowly let their brains rot away*"	R02 R06 R05 R10 H02
Cluster: **Forces and Consequences** (4 codes)			
`structural-drivers`	Structural forces (incentives, corporate actors, cost-cutting) that drive AI slop production.	"*This will just be weaponised by people trying to boost their profile in an ironically shrinking job market.*"	R02 R05 R05 R07 H02 R02 R05 R06 H02
`mandated-ai-adoption`	AI features or workflows imposed on developers without meaningful choice.	"*[There has been] a huge push for teams to adopt tools like Cursor [. . .] rapidly becoming an endless stream of AI slop.*"	R02 R06 R06 R08 R07
`craft-erosion`	Loss, grief, or disillusionment about what software development has become.	"*What does saving an hour or two writing code, which I actually like, give me if I have to spend that same hour or two deshittifying it?*"	R05 R05 R05 R06 H01 H03
`workforce-disruption`	How AI slop affects the software workforce: jobs, hiring, or career trajectories.	"*Fake LinkedIn profiles, fake GitHub, fake resumes. In a few cases even fake people*"	R06 R06 R06 R02 R06 R12 R12 R12
Cluster: **Rhetorical** (1 code)			
`sarcastic-skepticism`	Irony, mock enthusiasm, or absurdist framing. Applied alongside topical codes.	"*the final boss: outsourced AI slop*"	R01 R02 R05 R06

C. Quality Degradation

This cluster captures how AI slop degrades technical quality: the limitations of AI tools (`ai-limitations`), their downstream impact on codebases (`codebase-degradation`), the pollution of external knowledge resources (`knowledge-ecosystem-degradation`), knowledge deficits in slop producers (`producer-comprehension-gap`), and the trajectory of declining developer skills (`skill-atrophy`).

AI tool limitations. Developers described characteristic failure modes. Common patterns included using `setTimeout` as a band-aid fix, [R06] casting to `any` to silence type errors, [R06] and deleting methods instead of fixing them [R06]. One developer summarized: "*Almost all code ever written is mediocre (or worse) and that's what LLMs have been trained to replicate without understanding*" [R03]. AI agents showed concerning behavior rooted in overconfident hallucination: "death loops" of confident-but-wrong fixes [R13] and test subversion, changing tests to pass broken code [R13]. In one case, an AI agent "*hallucinated external services, then mocked out the hallucinated external services,*" creating a coherent but fictional integration [R07].

Codebase impact. Downstream consequences were widely discussed. "*You can go very fast with AI, but you accrue technical debt at a much higher speed too,*" [R06] captured the tradeoff between generation velocity and maintenance cost. Security concerns were prominent: In one PR the AI "*did something extremely dangerous [. . .] where it aborted early in a middleware basically skipping most of AuthZ*" [R07]. Even AI proponents expressed concern: "*I'm actually pro-AI [. . .] but I'm also very concerned that the way those things will be deployed at scale [. . .] is likely to lead to severe degradation of software quality across the board*" [H03].

Knowledge ecosystem. Developers also reported degradation of external knowledge resources. "*I'm starting to see documentation and tutorials missing key information [...] Or it's just completely wrong or using a class that doesn't exist*" [R11]. The layoff of developer relations (DevRel) teams compounded the problem: "*The DevRel field was absolutely gutted in the layoffs starting in 2022. [...] They were the ones maintaining docs and code examples and demo repos [...] making sure the SEO'd articles and blog posts actually had quality content with code snippets that ran*" [R11].

Comprehension gaps. Producers of AI slop often lacked the understanding needed to evaluate their *own* output. "*I straight up asked them if they know what their code does. They didn't. The PR was not approved*" [R05]. In one case, a designer used AI to build a full React app: "*The code was a mess so they hired a freelancer React guy to fix it up and he just removed 90+ files out of 100*" [R06].

Skill atrophy. Developers described collective deskilling. "*Many in our field are happily contributing to that future themselves by letting AI do their own job and slowly let their brains rot away*" [R02]. One Hacker News commenter called this a "*Catch-22*": "*If to make actually good use of AI you have to be an experienced engineer, but to become an experienced engineer you had to get there without AI doing all your work for you, then how are we going to get new experienced engineers?*" [H02]

D. Forces and Consequences

This cluster captures the broader forces and human toll of AI slop: the systemic drivers behind slop proliferation (`structural-drivers`), the experience of having AI imposed without meaningful choice (`mandated-ai-adoption`), loss of professional meaning (`craft-erosion`), and effects on the labor market (`workforce-disruption`).

Incentive mechanisms. Developers identified structural forces that reward slop. Gameable metrics (GitHub contribution graphs, bug bounty payouts, SEO rankings) reward quantity over quality, an instance of Goodhart's law. As one commenter observed: "*This will just be weaponised by people trying to boost their profile in an ironically shrinking job market*" [R02]. Another framed it as disrespect: "*If someone wants that green Github contribution graph, they should at least take the time [...] to learn software engineering. They shouldn't steal open source maintainers' time with AI slop*" [H02].

Corporate pressure and historical parallels. Multiple discussions drew parallels to corporate offshoring. One developer noted: "*AI is strangely similar to offshoring: Get the nominal task completed more cheaply (yay!), while ballooning administrative oversight labor to fix the greater number of issues (boo!)*" [R05]. The speed narrative creates its own pressure: "*AI is definitely sold as a supposed competitive advantage in development time. I do think that adds an extra subconscious incentive to push work through to production as quickly as possible*" [R05].

Reduced developer agency. Developers described AI workflows imposed by management. C-level executives were "*running parts of our codebase through AI tools and literally copy pasting the response as an answer to every technical problem*" [R08]. Another reported: "*Lately there has been a huge push for teams to adopt tools like Cursor, the problem is that while yes they can generate code, it is just lately rapidly becoming an endless stream of AI slop*" [R07].

Craft erosion. Some developers expressed grief over what software development was becoming, often as an inversion of creative and tedious work: "*What does saving an hour or two writing code, which I actually like, give me if I have to spend that same hour or two deshittifying it, which I don't like at all?*" [R05] One developer described the loss of craft value in review: "*If I review (and rework) their code I can find the pieces of brilliance, where they encoded their deep understanding [...] even if the C is bad. But with AI slop there is nothing*" [R05]. Some expressed deep disillusionment: "*I'm almost 40 and I'm really not interested in continuing the AI slop treadmill [...] I used to love technology. Now I'm coming to loathe it*" [H01].

Workforce disruption. AI slop affected the labor market in both negative and positive ways. Hiring pipelines were contaminated by AI-generated fraud: "*Fake LinkedIn profiles, fake GitHub, fake resumes. In a few cases even fake people*" [R12]. Legitimate developers were collateral damage. One obscured their LinkedIn details after malicious actors copied them into fake profiles [R12]. On the positive side, some developers saw opportunity: "*I am happy keeping my development skills sharp. There will be a huge demand for people who can clean up the mess and I will be happy to help with it, for a good price*" [R06].

E. Rhetorical

Across all themes, developers used irony, mock enthusiasm, and absurdist framing toward AI slop. Examples ranged from deadpan sarcasm ("*I guess you'll just never get to experience the joy of spending your limited free time reading and dealing with bug reports that the authors couldn't even be bothered to write*" [R02]) to gaming metaphors ("*the final boss: outsourced AI slop*" [R06]). The prevalence of sarcasm suggests that developers use humor as a coping mechanism.

V. Discussion

A. AI Slop as a Tragedy of the Commons

Our findings support framing AI slop as a tragedy of the commons [3]. Independent evidence is consistent with these concerns: A large-scale comparison found that AI-generated code carries more high-risk security vulnerabilities than human-written code [15], and a survey of AI programming assistants found that developers value them for speed but struggle to get output that is correct and controllable [7]. As outlined in the introduction, individual gains externalize costs onto the shared resources of collaborative development. The structural forces identified in our analysis are the mechanisms through which the commons is overexploited: gameable metrics (a form of Goodhart's law), corporate speed mandates, and reduced developer agency. The disconnect is visible at the highest levels: The CEO of one major AI tool vendor publicly

objected to the term itself,[17] even as developers in our data described being overwhelmed by the output of that vendor's tools.

B. Implications for Practice

For tool developers. Current AI tools support code generation more than the verification and review of generated artifacts (`reviewer-burden`, `ai-limitations`). Tool support should help developers understand and evaluate generated code: surfacing uncertainty indicators, flagging changes to tests, security mechanisms, or external dependencies, and explaining changes. These targets address observed failure modes such as test subversion, unsafe shortcuts, and incorrect integrations rather than further reducing generation time. Tools should also encourage smaller, incremental changes, structure output to support inspection (`slop-mitigations`), and make AI assistance visible through provenance (`trust-erosion-in-collaboration`, `ai-content-detection`).

For team leads and organizations. The structural drivers tie low-quality, high-volume contributions to existing incentives. Organizations should reconsider evaluation criteria that reward output volume and instead weigh downstream cost, such as review effort, defect rates, and rework. Letting developers judge when and how to use AI reduces management-driven adoption that produces unchecked low-quality output (`mandated-ai-adoption`). Teams should require contributors to understand and explain their changes, via PR size limits and code walkthroughs (`developer-accountability`, `slop-mitigations`).

For educators. Comprehension gaps (`producer-comprehension-gap`) suggest that correctness alone does not indicate competence. Assessments should require understanding through formats AI cannot deliver, such as oral exams, live coding, or in-person walkthroughs. The `skill-atrophy` code supports restricting AI use in early coursework so students build foundational skills first.

C. Threats to Validity

We study naturally occurring discourse rather than elicited responses. Our corpus is limited to discussions explicitly mentioning "AI slop," which biases toward participants who have adopted this framing. The exact-phrase search excludes critiques that avoid the term as well as neutral and positive accounts. Reddit and Hacker News attract a particular demographic. Perspectives from other communities may differ. Public posts avoid the recall and social-desirability biases of interviews and surveys, but they lack demographic detail and the chance to ask follow-up questions. Interviews, focus groups, or surveys could complement our findings. Our qualitative approach captures the range of perceptions but does not quantify their prevalence, and we do not claim that the code frequencies generalize to the developer population. The corpus is also a September 2025 snapshot of the discourse. Later edits are not reflected, and comments created and then deleted before our data collection were excluded. AI tools change so quickly that findings can be outdated within months. Specific failure modes may no longer apply even as the structural dynamics persist, and newer discussions may contain themes our codebook misses. The codebook refinement and annotation involved AI assistance, which could introduce bias. We mitigated this by having the second author check every AI-assigned label rather than a sample.

Data availability. The corpus, codebook, and all annotated data are publicly available, archived on Zenodo[13] and browsable through an interactive tool that shows every coded post.[16]

[17] windowscentral.com/.../microsoft-ceo-satya-nadella-...

VI. Conclusion

Our analysis of developer discourse about AI slop identified 15 codes in three thematic clusters. AI slop is not merely a code quality issue: It is a sociotechnical problem spanning incentive structures, knowledge ecosystems, collaborative trust, and labor markets. The discourse also reveals a constructive dimension: Practitioners are articulating accountability norms, proposing mitigations, and developing detection heuristics. Tools should support the verification and review of generated code, not only its generation. Teams should reward downstream quality over output volume. Educators should assess understanding through formats AI cannot deliver.

Future work should expand beyond the "AI slop" keyword to capture subtler critiques, validate findings through surveys and interviews, and investigate the real-world impact of AI slop through longitudinal studies.

References

[1] C. Kommers, E. Duede, J. Gordon, A. Holtzman, T. McNulty, S. Stewart, L. Thomas, R. J. So, and H. Long, "Why slop matters," *ACM AI Letters*, vol. 1, no. 1, pp. 1–6, 2026. [Online]. Available: https://doi.org/10.1145/3786777

[2] M. Koren, G. Békés, J. Hinz, and A. Lohmann, "Vibe coding kills open source," 2026, arXiv:2601.15494. [Online]. Available: https://arxiv.org/abs/2601.15494

[3] G. Hardin, "The tragedy of the commons," *Science*, vol. 162, no. 3859, pp. 1243–1248, 1968. [Online]. Available: https://doi.org/10.1126/science.162.3859.1243

[4] C. Prause, "Reputation-based self-management of software process artifact quality in consortium research projects," in *SIGSOFT/FSE 2011: 19th ACM SIGSOFT Symposium on the Foundations of Software Engineering and 13th European Software Engineering Conference, Szeged, Hungary, September 5-9, 2011*, T. Gyimóthy and A. Zeller, Eds. ACM, 2011, pp. 380–383. [Online]. Available: https://doi.org/10.1145/2025113.2025166

[5] 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 *43rd IEEE Symposium on Security and Privacy, SP 2022, San Francisco, CA, USA, May 22-26, 2022*. IEEE, 2022, pp. 754–768. [Online]. Available: https://doi.org/10.1109/SP46214.2022.9833571

[6] K. Jesse, T. Ahmed, P. T. Devanbu, and E. Morgan, "Large language models and simple, stupid bugs," in *20th*

IEEE/ACM International Conference on Mining Software Repositories, MSR 2023, Melbourne, Australia, May 15-16, 2023. IEEE, 2023, pp. 563–575. [Online]. Available: https://doi.org/10.1109/MSR59073.2023.00082

[7] J. T. Liang, C. Yang, and B. A. Myers, "A large-scale survey on the usability of AI programming assistants: Successes and challenges," in *Proceedings of the 46th IEEE/ACM International Conference on Software Engineering, ICSE 2024, Lisbon, Portugal, April 14-20, 2024*. ACM, 2024, pp. 52:1–52:13. [Online]. Available: https://doi.org/10.1145/3597503.3608128

[8] P. Meakpaiboonwattana, W. Tarntong, T. Mekratanavorakul, C. Ragkhitwetsagul, P. Sangaroonsilp, R. G. Kula, M. Choetkiertikul, K. Matsumoto, and T. Sunetnanta, "Social media reactions to open source promotions: AI-powered GitHub projects on hacker news," in *IEEE International Conference on Software Maintenance and Evolution, ICSME 2025, Auckland, New Zealand, September 7-12, 2025*. IEEE, 2025, pp. 344–355. [Online]. Available: https://doi.org/10.1109/ICSME64153.2025.00039

[9] I. Shumailov, Z. Shumaylov, Y. Zhao, N. Papernot, R. J. Anderson, and Y. Gal, "AI models collapse when trained on recursively generated data," *Nature*, vol. 631, no. 8022, pp. 755–759, 2024. [Online]. Available: https://doi.org/10.1038/s41586-024-07566-y

[10] S. Kabir, D. N. Udo-Imeh, B. Kou, and T. Zhang, "Is stack overflow obsolete? an empirical study of the characteristics of chatgpt answers to stack overflow questions," in *Proceedings of the CHI Conference on Human Factors in Computing Systems, CHI 2024, Honolulu, HI, USA, May 11-16, 2024*, F. F. Mueller, P. Kyburz, J. R. Williamson, C. Sas, M. L. Wilson, P. O. T. Dugas, and I. Shklovski, Eds. ACM, 2024, pp. 935:1–935:17. [Online]. Available: https://doi.org/10.1145/3613904.3642596

[11] N. Eghbal, *Working in Public: The Making and Maintenance of Open Source Software*. Stripe Press, 2020.

[12] N. Raman, M. Cao, Y. Tsvetkov, C. Kästner, and B. Vasilescu, "Stress and burnout in open source: Toward finding, understanding, and mitigating unhealthy interactions," in *ICSE-NIER 2020: 42nd International Conference on Software Engineering, New Ideas and Emerging Results, Seoul, South Korea, 27 June - 19 July, 2020*, G. Rothermel and D. Bae, Eds. ACM, 2020, pp. 57–60. [Online]. Available: https://doi.org/10.1145/3377816.3381732

[13] N. A. Ernst and C. Treude, "GenAI is no silver bullet for qualitative research in software engineering," *CoRR*, vol. abs/2603.08951, 2026, arXiv:2603.08951. [Online]. Available: https://arxiv.org/abs/2603.08951

[14] Z. O. Dunivin, "Scaling hermeneutics: A guide to qualitative coding with LLMs for reflexive content analysis," *EPJ Data Science*, vol. 14, no. 1, p. 28, 2025. [Online]. Available: https://doi.org/10.1140/epjds/s13688-025-00548-8

[15] D. Cotroneo, C. Improta, and P. Liguori, "Human-written vs. ai-generated code: A large-scale study of defects, vulnerabilities, and complexity," in *36th IEEE International Symposium on Software Reliability Engineering, ISSRE 2025, São Paulo, Brazil, October 21-24, 2025*, A. Brito, R. L. de Oliveira Moraes, Y. Sui, C. Trubiani, F. Aires, and N. Ivaki, Eds. IEEE, 2025, pp. 252–263. [Online]. Available: https://doi.org/10.1109/ISSRE66568.2025.00035

Sebastian Baltes is a Professor of Software Engineering at Heidelberg University, Germany, and an Adjunct Professor at the University of Adelaide, Australia. He received his Ph.D. in Computer Science from the University of Trier, Germany, in 2019 and spent 3.5 years in the software industry before returning to academia. His research focuses on software analytics, that is, processing, analyzing, and visualizing software engineering data to monitor, govern, and improve development processes and tools. He is also interested in interdisciplinary research and the methodological foundations of empirical software engineering. His work has been published in leading venues including ICSE, FSE, TSE, TOSEM, and EMSE, has received multiple best paper awards, including two ACM SIGSOFT Distinguished Paper Awards, and has received funding from Google and SAP. Contact him at sebastian.baltes@uni-heidelberg.de.

Marc Cheong is a Senior Lecturer of Information Systems (Digital Ethics) and Deputy Director, Centre for AI and Digital Ethics (CAIDE), at the University of Melbourne, Australia. His research interests are at the intersection of philosophy and technology, in particular the ethics of social media technologies and sociotechnical issues in software development and deployment. He is the co-author of the textbook on digital ethics for technologists and practitioners, *Transition to Digital Ethics: A Primer from Philosophy to Practice* (Chapman & Hall / CRC Press). His work has also been featured in news outlets such as The Conversation, ABC (Australia), The Age, and The New York Times. Cheong received his Ph.D. from the Clayton School of Information Technology, Monash University. He is also an honorary Burnet Institute Senior Fellow, working in social media and public health research. He has received funding from Google as part of institutional support for CAIDE. Contact him at marc.cheong@unimelb.edu.au.

Christoph Treude is an Associate Professor of Computer Science at Singapore Management University (SMU). His work spans empirical and automated software engineering, human and social aspects of software engineering, human-AI collaboration, and AI for science. He has authored over 200 scientific publications in collaboration with more than 300 co-authors. His research has been recognized with five best paper awards, including three ACM SIGSOFT Distinguished Paper Awards, and has received funding from Google, Facebook, DST, and through an ARC Discovery Early Career Researcher Award (2018–2020). He currently serves as Associate Editor-in-Chief of IEEE Transactions on Software Engineering, on the editorial board of Empirical Software Engineering (Springer), and as Open Science Editor for the Journal of Systems and Software (Elsevier). Contact him at ctreude@smu.edu.sg.