

Characterizing Self-Admitted Technical Debt Generated by AI Coding Agents

Zaki Brahmi

ETS Montreal, University of Quebec
Montreal, Canada
ezaki.ibrahmi.1@ens.etsmtl.ca

Mohammed Sayagh

ETS Montreal, University of Quebec
Montreal, Canada
Mohammed.Sayagh@etsmtl.ca

Ali Ouni

ETS Montreal, University of Quebec
Montreal, Canada
ali.ouni@etsmtl.ca

Mohamed Aymen Saied

Laval university
Québec, Canada
mohamed-aymen.saied@ift.ulaval.ca

Abstract

Large Language Models (LLMs) are increasingly used through autonomous agents (e.g., Copilot, Cursor, Devin, Claude) to perform complex software development tasks. However, little is known about how these agents introduce and document technical debt through Self-Admitted Technical Debt (SATD) comments. Understanding SATD in AI-generated code is critical, as such comments explicitly reveal acknowledged limitations and deferred fixes that affect long-term maintenance. In this study, we quantitatively and qualitatively analyze 525 SATD comments authored by AI agents using the AIDev dataset. Our results show that AI-generated SATD is slightly more technically detailed than human-authored SATD, yet both often describe problems without clear guidance on resolution. Through thematic analysis, we identify 34 SATD topics grouped into 10 categories, with AI agents predominantly documenting requirement- and design-related debt. While many SATD topics overlap between AI and humans, our taxonomy reveals new debt categories and emphases specific to AI-authored SATD, particularly related to infrastructure, pipelines, dependency management, and requirement interpretation driven by developer prompts. Overall, our findings suggest that AI- and human-authored SATD share common characteristics but differ in expression and focus, highlighting the need for deeper investigation into how agentic systems communicate and manage technical debt.

CCS Concepts

• **Software and its engineering;**

Keywords

Technical Debt, SATD, AI Agent, Mining Software Repositories

ACM Reference Format:

Zaki Brahmi, Ali Ouni, Mohammed Sayagh, and Mohamed Aymen Saied. 2026. Characterizing Self-Admitted Technical Debt Generated 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/10.1145/3793302.3793588>



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/10.1145/3793302.3793588>

1 Introduction

Software engineering has entered a new era, increasingly shaped by AI Agents, which can perform complex development activities such as code generation [34] and code review [27]. Modern AI coding Agents, like OpenAI Codex, and Cursor, are evolving beyond simple autocomplete tools to become active contributors that generate code, and submit pull requests [16]. Although they can rapidly produce functional code through natural language prompts, the generated code often contains logical errors, performance issues, or security flaws [32]. However, it remains unclear whether these agents explicitly acknowledge their limitations through SATD comments. Unlike human developers, AI agents are not expected to intentionally defer work or leave low-quality codes for future refinement. Analyzing SATD in AI-generated code enables quantifying when and how agents expose their failure modes, such as missing context, unavailable resources, or deferred decisions. This motivates a systematic investigation into whether AI agents generate SATD, how often they do so, and what types of SATD they produce.

Accordingly, we aim in this paper to examine the prevalence and characteristics of SATD generated by AI agents, and to compare these against human-authored SATD. We start by analyzing the prevalence and the quality of SATDs authored by AI Agent. Then, we aim to identify the main issues acknowledged by AI coding agents in their SATDs, providing insights into how these agents document limitations and reasoning in their code. The subsequent analysis is structured around the following research questions:

- **RQ1.** To what extent do AI Agents and human-authored SATD differ in prevalence and quality?
- **RQ2.** How are AI agent authored SATDs fixed ?
- **RQ3.** What types of SATD are produced by AI agents?

We explore the AIDev-Pop dataset [26], which consists of 856 GitHub repositories with more than 500 stars and involving contributions from Copilot, Cursor, Devin AI, and Claude. We limit our scope to commit-level data, analyzing file patches to extract SATD instances. To address our research questions, we analyze 820 agent-authored and 1,991 human-authored SATD instances extracted from 208k comments.

Our findings indicate that AI-generated SATD are slightly more technically detailed than those authored by humans; however, both predominantly describe problems without providing actionable guidance for remediation. Based on manual thematic analysis, we

find that while AI agents and human developers share common SATD topics, AI-authored SATD shows distinct emphases and introduces new debt categories, notably in infrastructure and pipelines, dependency management, and requirement interpretation.

Open Science. We provide our replication package publicly available for future extensions and replications [5].

2 STUDY DESIGN

2.1 Data collection

In this study, we use the AIDev-pop dataset [26], which contains 856 repositories. Following the findings of Li et al. [26], our analysis excludes the Codex agent. This exclusion is due to the difficulty of reliably assigning its authorship within commit metadata. In contrast, other agents explicitly indicate their authorship at the commit level, except for Claude, whose authorship is extracted from the “Co-Authored-By: Claude” message. From the complete set of commits in the database, we excluded those that either (i) did not include any modified files, or (ii) lacked source code changes (e.g., documentation-only updates). After filtering, the dataset comprises 34k commits and 84k comments from AI agents, and 46k commits and 124k comments from humans. We use the python library `comment-parser` [1] to extract comments from patches.

2.2 Data Analysis

To answer **RQ1**, we use the keywords of Xiao et al. [35] to identify SATD. We identified 820 SATD comments authored by the four studied AI agents. These instances are used to study the prevalence of SATD. We excluded 103 false positives ($\approx 12\%$) during manual validation, mostly from keywords like `fix` used out of context. After removing false positives and duplicated SATD instances, we retained 525 SATDs for quality evaluation. SATD prevalence is measured at the commit, file, and comment levels, defined as the proportion of units at each level that contain at least one SATD.

Unlike human, AI-generated SATD does not reflect intentional design trade-offs but emerges from the code generation process. Evaluating its quality is essential to determine whether it conveys actionable fix information or introduces ambiguity, and to mitigate maintenance risks in AI-assisted software development. Hence, we employ a comment-centric method by evaluating SATD comments without considering the surrounding code, following Wang et al. [33]. This comment-centric perspective provides a strict measure of the intrinsic clarity, readability, and usefulness of the comment itself [33]. For each SATD instance, we compute 4 classes of metrics:

(i) *Readability*: we use the Flesch Reading Ease (FRE) metric, a widely used measure for assessing text readability based on word and sentence length, word form, and syllable or letter count [18, 24]. A higher FRE score indicates easier readability, scores of 70–100 indicate easy-to-read, 60–70 suits standard readers, and 0–30 is suitable for specialized professionals,

(ii) *Text comprehensibility*: measures how easily a developer understands an SATD comment. Although developers are technically skilled, clarity depends on writing quality [25]. The Gunning–Fog Index (GFI) [21] measures this clarity through linguistic complexity. GFI estimates how many years of education a reader needs to understand a text on first reading. Lower scores indicate that the text is easier to read and requires less background knowledge.

(iii) *Technical Specificity (TS)*: measures how precise and actionable a SATD comment is (what, where, how), [33], to help developers locate, understand, and address debt more effectively.

(iv) *Comment Quality Levels (CQL)*: classifies comments into three levels according to the richness and clarity of the information they convey [24]. Level 1 comments offer no useful repair guidance and often contain vague remarks or parameter explanations. Level 2 comments describe the issue to some extent but lack concrete direction. Level 3 comments provide explicit suggestions or solutions, offering clear and actionable guidance for fixing the problem. Following Huang et al.’s method [24], we prompt LLM (Qwen2.5-7B-Instruct) to compute CQL and TS. We manually reviewed and validate the outputs produced by the LLM to ensure annotation quality. Finally, we compute Mann–Whitney U tests [17] to assess whether differences between human and AI Agents (as one group) were statistically significant, and used Cliff’s Delta (δ) to quantify effect size.

To answer **RQ2**, we evaluate the extent to which SATD introduced by AI Agents are eventually resolved, we conduct a longitudinal tracking analysis of every SATD instance in our dataset. We collected a total of 478 SATD instances introduced by AI Agents. For each new introduced SATD instance, we record its *introduction commit*, *author*, and *fixed commit* by reconstructing the commit graph. To compare remediation effort of each agent, we compute quantified remediation effort for each SATD size metrics, using five metrics: *Resolution Time (RT)*, *Files Changed (FC)*, *LOC Added (LA)*, *LOC Deleted (DL)*, and *Commits Touching File (CTF)* until the fix. To assess whether remediation-effort metrics differ across agents, we use the non-parametric Kruskal–Wallis H test, appropriate for skewed distributions with outliers and unbalanced sample sizes.

To answer **RQ3**, we analyzed 525 SATD instances authored by AI agents using inductive thematic analysis [15]. We classified SATD comments based on their content [14]. The analysis proceeded iteratively: during open coding, the first author identified initial concepts across the SATDs, which were synergistically refined with co-authors into broader categories representing major types of AI-generated debt. After stabilizing the top-level categories, subcategories were derived using the same iterative coding process. This bottom-up process led to the emergence of AI-specific SATD types that extend or diverge from traditional SATD classifications. To ensure inter-rater reliability, the authors conducted multiple calibration sessions on representative samples, resolving all disagreements through negotiated consensus.

3 STUDY RESULTS

3.1 RQ1: To what extent do AI Agents and human-authored SATD differ in prevalence and quality?

3.1.1 SATD prevalence. SATD is less prevalent in AI Agent than human.

Table 1 shows that humans exhibit a higher SATD prevalence across commits (4.25%) and comments (1.60%) compared to AI agents ($\approx 2.6\%$ per commit and $\approx 0.95\%$ per comment), suggesting that human acknowledge technical debt more frequently. Among the agents, Cursor has the highest SATD ratio per commit (2.8%) and per file (0.72%), indicating that its generated code more often

Table 1: Prevalence of SATD: AI agents and Human

Author	#Commits	#Files	#Comments	SATDs	%Commits	%Files	%Comments
Copilot	13,976	75,667	40,325	405	2.90	0.54	1.00
Devin AI	16,507	63,686	28,204	313	1.90	0.49	1.11
Cursor	3,534	13,860	15,013	100	2.83	0.72	0.67
Claude	111	457	589	2	1.80	0.44	0.34
Human	46,847	519,497	124,204	1,991	4.25	0.38	1.60

Table 2: Comparison of SATD quality metrics across AI Agents, with p-values for AI (Combined) vs. Human

Author	FRE	GFI	TS	CQL
Copilot	37.96	12.63	0.3043	1.5783
Cursor Agent	40.62	11.49	0.2583	1.4458
Devin AI	43.71	11.77	0.2700	1.5060
Human	39.77	11.07	0.2564	1.3896
p-value	0.129	0.055	0.001	0.013
Size effect δ	Negligible (0.08)	Negligible (0.10)	Small (0.16)	Negligible (0.11)

contains technical debt. Copilot follows closely with a similar SATD ratio per commit (2.9%), suggesting comparable patterns of SATD introduction. In contrast, Devin shows the highest SATD density per comment (1.1%), meaning that its comments are more likely to contain explicit technical debt.

Humans write more SATD than AI agents because they possess project context, accountability, and intentionality regarding design trade-offs. Conversely, The prompt-driven nature of AI agents can prevent common code-quality-related debt, such as incomplete logic or low-quality code, through direct generation, reducing the need for explicit SATD.

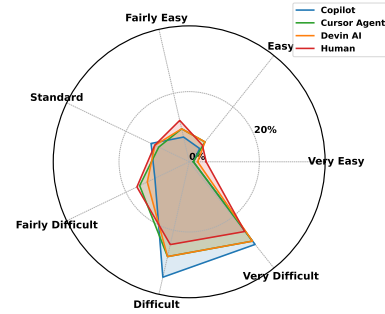
3.1.2 SATDs Quality. Both humans and AI agents produce SATD comments that mostly identify a problem but provide limited or no guidance on how to fix it.

As shown in Table 2, SATDs written by both AI agents and human developers exhibit comparable levels of linguistic complexity. Readability remains uniformly low across all authors (FRE $\approx$ 38–44), reflecting the dense and technical nature of SATD. The GFI ($\approx$ 11–12) suggests that SATD comments assume familiarity with software engineering terminology, making them challenging to interpret without prior domain expertise. For instance, the quality score of the SATD “*TODO: Implement cache management*” [8], have a TS score of 0.2, and 1 in terms of CQL. TS and CQL are lower because the SATD omits details about the cache type, which component should manage caching, and the intended implementation. Overall, comparison between Human and AI agents shows negligible differences in readability, comprehensibility, and comment quality ($\delta < 0.11$), whereas technical specificity exhibits a small effect ($\delta = 0.16$), indicating a modest yet consistent improvement in the detail of AI-generated SATD. This aligns with recent studies showing that LLMs tend to produce more elaborated, and precise comments compared to human authors [19, 20].

We argue that AI agents are trained on more structured programming text [22], helping them to generate comments that follow clearer syntactic patterns, include more contextual cues, and better articulate intent [22]. Human developers often write SATD under time pressure as self-reminders, leading to more fragmented phrasing and lower specificity, such as “*TODO: Add Authz*” [7] which has QCL = 1. While readability and comprehensibility remain similar across groups due to the technical nature of SATD, AI-generated comments achieve marginally better structural organization.

The readability distribution in Figure 1 shows that AI agents and human developers produce SATD that clusters predominantly

in the Difficult ($\approx$ 24%–33%) and Very Difficult categories ($\approx$ 25%–30%), and minor Easy SATDs ($>$ 6%). These two bands represent the largest areas across all curves, confirming that SATD comments are generally dense, technical, and written under strong contextual assumptions. This reflects the inherent nature of SATD rather than a limitation of any specific author: SATDs tend to omit explanatory detail, which naturally increases their linguistic difficulty.

**Figure 1: Readability Category Distribution per Agent**

We observe that AI Agents exhibit substantial proportions in the Difficult and Very Difficult zones, with Copilot showing the strongest presence in the Difficult region and Devin AI extending more toward the Very Difficult end of the spectrum. For example, a Devin-generated SATD such as “*TODO: write makeWhereClause equivalent for kysely*” [6] receives a high GFI score (=22.4), illustrating highly technical phrasings often push Devin’s comments into the harder readability categories. Cursor and Human both follow a distribution dominated by the difficult bands, with only small amounts of SATD falling under Easy or Fairly Easy. For instance, a Cursor Agent comment such as “*FIXME: Do not use useEffect to set local state paneWidth. eslint-disable-next-line react-hooks/exhaustive-deps*” is rated as difficult (GFI = 14.8) due to its high density of React-specific terminology, even though its CGL score is 2 because it describes the issue without offering actionable repair guidance. Human exhibits a readability profile that closely mirrors the AI agents, reinforcing that low readability is a structural characteristic of SATD itself. This observation is consistent with the CQL patterns in Table 3, where both humans and AI agents produce mostly Level 1 and Level 2 SATD comments that identify a problem but provide limited or no guidance on how to fix it.

As shown in Table 3, most SATDs fall into Level 1 (45%–63%), with humans producing the low-quality comments ($p = 0.012$, with negligible effect sizes: $\delta = 0.11$). Between 34% and 51% reach Level 2, where Copilot performs best. Only 2.4–3.6% attain Level 3, though Devin AI and Copilot generate more actionable SATDs (e.g., comment from commit #a7179b0 [12]). These proportions reinforce the earlier findings: SATD, regardless of whether it is written by humans or AI agents, tends to be difficult to read, and generally lacking in actionable detail. Only about 2–4% of SATD provides actionable guidance for maintenance. This slight similarity between human and AI SATD quality is largely due to the nature of SATD, the low actionability of human-written SATD, and the fact that LLMs replicate existing patterns of technical debt annotations seen in their training data, which confirms the finding of O’Brien, et al. [30].

Table 3: Comparison of CQLs Across AI Agents, with p-values for AI (Combined) vs. Human

Author	Level 1 (%)	Level 2 (%)	Level 3 (%)
Copilot	45.78	50.60	3.61
Cursor	57.83	39.76	2.41
Devin AI	53.01	43.37	3.61
Human	63.45	34.14	2.41
p-value	0.012	0.019	0.594
Effect size (δ)	Negligible (0.111)	Negligible (0.103)	Negligible (0.008)

3.2 RQ2: How are AI agent authored SATDs fixed?

Differences in SATD-fixing behavior across AI agents emerge from prompting and developer-defined workflows.

As shown in the Table 4, the human effort required to prompt each agent to a successful fix SATD varies significantly (H-stat: 38.8–82.7). Copilot consistently produces the smallest and simplest fixes. For example, its median FC is 55, compared with 5 for Cursor and 7 for Devin. Similarly, Copilot shows extremely large variation in change size (LA: mean = 2,976), indicating occasional massive auto-generated patches, like observed on the SATD [13]. Cursor Agent shows moderate, stable fix sizes (TR median 2.02h) versus Copilot (0.13h) and Devin (2.40h). Devin AI yields the most extensive changes, averaging 243.9h—more than 3× Cursor and almost 4× Copilot.

AI agents often make large code changes when resolving SATD, as they tend to rewrite entire files rather than apply localized fixes—a behavior tied to full-code regeneration and refactoring [23]. Some extreme cases result from repository operations; for example, Copilot’s largest deletion (797,880 LOC [4]) removed *vendored dependencies*. Overall, SATD resolution effectiveness appears driven more by prompting context and task scope than by the agent itself.

Table 4: Comparison of fix-effort metrics across Agents.

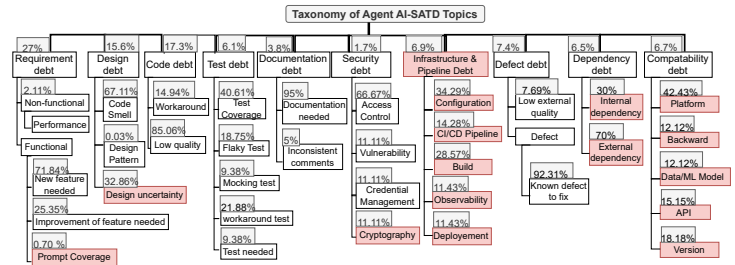
Metric	Copilot		Cursor Agent		Devin AI		H-Stat
	Median	Mean	Median	Mean	Median	Mean	
TR(H)	0.13	63.99	2.02	81.63	2.41	243.86	38.78
FC	55	119.71	5	15.55	7	39.49	52.71
LA	2	2976.52	183	762.40	31	865.71	67.30
LD	8983	82979.15	81	217.30	54	498.61	82.66
CTF	1	1.23	1	2.60	2	4.18	45.51

3.3 RQ3: What types of SATD are produced by AI Agents?

Our thematic analysis revealed 10 topics for which AI Agents write SATD, encompassing 34 sub-topics.

As shown in Figure 2, we expand on prior SATD topic taxonomies [28, 29, 31] by introducing new categories (colored boxes), including *Infrastructure and Pipeline*, and subcategories such as *Prompt coverage* and *Design uncertainty* (e.g., “*TODO: Do we need to do this...?*”), where AI agents express doubts about current design decisions [29]. Most AI-generated SATD relates to **requirements** (27.0%), followed by **code** (17.3%) and **design debt** (15.6%), suggesting that agents often flag structural or refactoring needs during code generation. We argue that these results indicate: (1) AI agents focus on requirement-related gaps, highlighting difficulties in inferring implicit or incomplete specifications; and (2) the combined share of compatibility and dependency debt (13.2%) suggests that AI agents often struggle with integration concerns.

◆ **(T1) Requirement:** emerges as the most dominant SATD category across AI agents. For instance, within this category, AI agents

**Figure 2: Taxonomy of SATD Topics authored by AI Agents.**

focused on *new feature needed* ($\approx 71.84\%$), such as “*TODO: Implement code-request workflow*” [11]. This indicates missing specifications, suggesting that AI Agents frequently surface ambiguity in system behavior as technical debt.

❖ **(T2) Design:** These comments target unresolved architectural or modular issues, likely reflecting the complexity of large projects and the limitations of the LLM context window. Design SATD is dominated by *code smell* debt ($\approx 67.11\%$) and *design uncertainty* ($\approx 32.86\%$).

▲ **(T3) Code:** This debt is dominated by low-quality code ($\approx 85\%$), indicating frequent generation of code that adversely affects readability and maintainability, such as “*TODO: Replace this hack with something better*” [10].

✓ **(T4) Test:** Test-related SATD is moderate across AI agents (9.5%). These comments highlight missing validation steps, insufficient testing coverage ($\approx 41\%$), or flaky test ($\approx 19\%$), such as “*TODO: Fix this test for mobile as it’s flaky*” [9].

🔒 **(T5) Documentation & Security:** Documentation and security topics remain consistently low across all AI agents, showing that AI rarely flags documentation debt or potential security risks. Likely due to limited contextual and threat-aware reasoning in prompts and LLM capabilities.

➡ **(T6) Infrastructure & Pipeline:** These comments typically relate to CI/CD Pipelines, configuration, deployment constraints, or observability (e.g., “*TODO: Add detailed health checks for all services*” [3]) that agents cannot fully address. For instance, observability debt occurs when agents identify missing health checks but defer fixes because of environment uncertainty.

✗ **(T7) Defect:** AI agents may document defect-related SATD instead of fixing it when the available context, specifications, or task scope are insufficient to apply a safe and correct remediation.

◆ **(T8) Dependency & Compatibility:** This SATD arises from challenges with internal module dependencies, external libraries, or framework version mismatches, reflecting integration and ecosystem alignment issues faced by AI agents in large projects (e.g., the mlfow project [2]).

4 Conclusions & Takeaways

This study provides the first empirical investigation of SATD authored by AI agents. By analyzing 856 repositories, we compared SATD written by AI agents and human in terms of prevalence and quality. Furthermore, we quantify the remediation effort, and manually classify SATD into 10 topics and 34 sub-topics. We observe that the most frequently occurring reasons behind Agent SATD is to document design debt and adding new features, as well as dependency

and compatibility issues. Our findings show that AI agents generate SATD with more technical details but exhibit similar readability.

However, the underlying mechanisms that lead AI agents to produce SATD, particularly comments related to low-quality code or lacking actionable guidance, remain insufficiently understood. These observations point to potential influences of training data, prompt formulation, and agent reasoning strategies, and raise open questions about whether AI-authored SATD reflects deliberate design rationale or artifacts of prompt-driven uncertainty. Understanding why AI agents rarely provide takeaway guidance, and whether they can be considered true authors of SATD, remains an open area for future investigation. Another direction is to study SATD across AI agent, LLM constraints, and project-scale perspectives to better understand AI-generated SATDs.

References

- [1] [n. d.]. *comment parser library*. <https://pypi.org/project/comment-parser/> Accessed: October 20, 2025.
- [2] [n. d.]. *SATD in Mlflow project*. <https://github.com/mlflow/mlflow/blob/c4e5d97/requirements%2Fcore-requirements.yaml> Accessed: October 20, 2025.
- [3] [n. d.]. *SATD in databuddy-analytics/Databuddy*. <https://perma.cc/L59K-JDKE> Accessed: October 20, 2025.
- [4] 2025. *Commit*. <https://perma.cc/JRF2-6SS7> Accessed: Dec 10, 2025.
- [5] 2025. *The replication package for our study*. https://figshare.com/articles/software/SATD_MSR-2026/32305275
- [6] 2025. *SATD in cal.com*. <https://perma.cc/T98Y-GBGU> Accessed: Dec 10, 2025.
- [7] 2025. *SATD in care*. <https://perma.cc/3334-BRD8> Accessed: Dec 10, 2025.
- [8] 2025. *SATD in langfuse*. <https://perma.cc/SY7A-DACZ> Accessed: Dec 10, 2025.
- [9] 2025. *SATD in liam-hq/liam*. <https://perma.cc/67S9-JHBL> Accessed: Dec 23, 2025.
- [10] 2025. *SATD in lostintangent/gistpad*. <https://github.com/lostintangent/gistpad/blob/a816d74/out/repos/store/actions.js#L267> Accessed: Dec 23, 2025.
- [11] 2025. *SATD in mattpocock/total-typescript-monorepo*. <https://perma.cc/K7RM-3Q75> Accessed: Dec 20, 2025.
- [12] 2025. *SATD in MS msquic*. <https://perma.cc/B34A-Z4SK> Accessed: Dec 10, 2025.
- [13] 2025. *SATD in Typespec*. <https://perma.cc/KH7N-JJFY> Accessed: Dec 10, 2025.
- [14] Gabriele Bavota and Barbara Russo. 2016. A large-scale empirical study on self-admitted technical debt. In *Proceedings of the 13th international conference on mining software repositories*. 315–326.
- [15] Virginia Braun and Victoria Clarke. 2006. Using thematic analysis in psychology. *Qualitative Research in Psychology* 3, 2 (2006), 77–101. The standard framework for inductive thematic analysis in software engineering.
- [16] 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 *Proc. 21st Int. Conf. Mining Softw. Repos. (MSR)*. 212–216.
- [17] William Jay Conover. 1999. *Practical nonparametric statistics*. John Wiley & Sons.
- [18] Maarten Duijn, Adam Kucera, and Alberto Bacchelli. 2015. Quality questions need quality code: Classifying code fragments on stack overflow. In *2015 IEEE/ACM 12th Working Conference on Mining Software Repositories*. IEEE, 410–413.
- [19] Aysa Xuemo Fan, Arun Balajee Lekshmi Narayanan, Mohammad Hassany, and Jiaye Ke. 2024. Evaluating the Quality of Code Comments Generated by Large Language Models for Novice Programmers. *arXiv:2409.14368* (2024).
- [20] Mingyang Geng, Shangwen Wang, Dezun Dong, Haotian Wang, Ge Li, Zhi Jin, Xiaoguang Mao, and Xiangke Liao. 2024. Large language models are few-shot summarizers: Multi-intent comment generation via in-context learning. In *Proc. of the 46th IEEE/ACM International Conference on Software Engineering*. 1–13.
- [21] Robert Gunning. 1952. The technique of clear writing. (*No Title*) (1952).
- [22] Qi Guo, Junming Cao, Xiaofei Xie, Shangqing Liu, Xiaohong Li, Bihuan Chen, and Xin Peng. 2024. Exploring the potential of chatgpt in automated code refinement: An empirical study. In *Proc. of the 46th IEEE/ACM International Conference on Software Engineering*. 1–13.
- [23] Kosei Horikawa, Hao Li, Yutaro Kashiwa, Bram Adams, Hajimu Iida, and Ahmed E Hassan. 2025. Agentic Refactoring: An Empirical Study of AI Coding Agents. *arXiv:2511.04824* (2025).
- [24] Qiao Huang, Weijun Liu, Zixuan Dai, Ye Wang, and Chengyi Lin. 2025. An Empirical Study on the Short-Term Self-Admitted Technical Debt. In *2025 25th International Conference on Software Quality, Reliability and Security (QRS)*. IEEE, 440–451.
- [25] Ninus Khamis, René Witte, and Juergen Rilling. 2010. Automatic quality assessment of source code comments: the JavadocMiner. In *International Conference on Application of Natural Language to Information Systems*. Springer, 68–79.
- [26] Hao Li, Haoxiang Zhang, and Ahmed E. Hassan. 2025. The Rise of AI Team-mates in Software Engineering (SE) 3.0: How Autonomous Coding Agents Are Reshaping Software Engineering. *arXiv:2507.15003* (2025).
- [27] Junyi Lu, Lei Yu, Xiaojia Li, Li Yang, and Chun Zuo. 2023. Llama-reviewer: Advancing code review automation with large language models through parameter-efficient fine-tuning. In *Proc. 34th ISSRE*. IEEE, 647–658.
- [28] Biruk Asmare Muse, Csaba Nagy, Anthony Cleve, Foutse Khomh, and Giuliano Antoniol. 2022. FIXME: synchronize with database! An empirical study of data access self-admitted technical debt. *Empirical Software Engineering* 27, 6 (2022), 130.
- [29] David OBrien, 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 *Proc. ESEC/FSE*. 734–746.
- [30] David OBrien, Sumon Biswas, Sayem Mohammad Imtiaz, Rabe Abdalkareem, Emad Shihab, and Hridesh Rajan. 2024. Are prompt engineering and todo comments friends or foes? an evaluation on github copilot. In *Proc. 46th ICSE*. 1–13.
- [31] Aniket Potdar and Emad Shihab. 2014. An exploratory study on self-admitted technical debt. In *2014 IEEE International Conference on Software Maintenance and Evolution*. IEEE, 91–100.
- [32] Xiangru Tang, Qiao Jin, Kunlun Zhu, Tongxin Yuan, Yichi Zhang, Wangchunshu Zhou, Meng Qu, Yilun Zhao, Jian Tang, Zhuosheng Zhang, et al. 2024. Prioritizing safeguarding over autonomy: Risks of llm agents for science. In *ICLR 2024 Workshop on LLM Agents*.
- [33] Haoye Wang, Zhipeng Gao, Tingting Bi, John Grundy, Xinyu Wang, Minghui Wu, and Xiaohu Yang. 2024. What makes a good todo comment? *ACM Transactions on Software Engineering and Methodology* 33, 6 (2024), 1–30.
- [34] Yue Wang, Weishi Wang, Shafiq Joty, and Steven CH Hoi. 2021. CodeT5: Identifier-aware Unified Pre-trained Encoder-Decoder Models for Code Understanding and Generation. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*. 8696–8708.
- [35] Tao Xiao, Dong Wang, Shane McIntosh, Hideaki Hata, Raula Gaikovina Kula, Takashi Ishio, and Kenichi Matsumoto. 2021. Characterizing and mitigating self-admitted technical debt in build systems. *IEEE Transactions on Software Engineering* 48, 10 (2021), 4214–4228.