"TODO: Fix the Mess Gemini Created": Towards Understanding GenAI-Induced Self-Admitted Technical Debt

Abdullah Al Mujahid
Missouri University of Science and Technology
Department of Computer Science
Rolla, MO, USA
amgzc@mst.edu

Mia Mohammad Imran
Missouri University of Science and Technology
Department of Computer Science
Rolla, MO, USA
imranm@mst.edu

Abstract

As large language models (LLMs) such as ChatGPT, Copilot, Claude, and Gemini become integrated into software development workflows, developers increasingly leave traces of AI involvement in their code comments. Among these, some comments explicitly acknowledge both the use of generative AI and the presence of technical shortcomings.

Analyzing 6,540 LLM-referencing code comments from public Python and JavaScript-based GitHub repositories (November 2022–July 2025), we identified 81 that also self-admit technical debt (SATD). Developers most often describe postponed testing, incomplete adaptation, and limited understanding of AI-generated code, suggesting that AI assistance affects both when and why technical debt emerges. We term **GenAI-Induced Self-admitted Technical debt** (**GIST**) as a proposed conceptual lens to describe recurring cases where developers incorporate AI-generated code while explicitly expressing uncertainty about its behavior or correctness.

CCS Concepts

• **Software and its engineering** → **Software maintenance tools**; *Maintaining software*; • **General and reference** → **Empirical studies**; • **Human-centered computing** → *Open source software*.

Keywords

self-admitted technical debt, generative AI, large language models, AI-assisted software development, human-AI collaboration

ACM Reference Format:
Abdullah Al Mujahid and Mia Mohammad Imran. 2026. "TODO: Fix the Mess Gemini Created": Towards Understanding GenAI-Induced Self-Admitted Technical Debt. In . ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 Introduction

Large Language Models (LLMs) are rapidly transforming software development workflows. Integrated tools such as GitHub Copilot, ChatGPT, and Gemini now assist with code generation, refactoring, documentation, and testing. While these systems increase productivity, they also introduce novel forms of risk and uncertainty. Developers must evaluate, correct, and maintain code partially written by non-human agents.

```
# def generate_content_with_function_calling
(prompt: str):
# '''TODO created by Gemini'''
# print("TODO implement me")
# pass
```

Figure 1: Example self-admitted comment illustrating requirement debt

Code comments, particularly those that self-admit technical shortcomings, offer a unique window into how developers interpret and manage AI-assisted contributions. In software engineering, self-admitted technical debt (SATD) signals areas requiring future improvement. When such comments reference LLMs, they reveal not only perceived code deficiencies but also evolving conceptions of accountability between human and machine collaborators. Figure 1 shows one such example, where a developer notes that a function was generated by Gemini and remains unfinished, illustrating how AI involvement and deferred implementation converge to create a potential debt.

While prior research has established taxonomies and automated methods for identifying SATD [2, 3, 20, 25, 26], little is known about how technical debt manifests in code influenced by generative AI. Early studies on AI-assisted development highlight productivity benefits [4, 8], but also point to challenges of correctness, explainability, and trust. What remains unclear is how developers themselves recognize and document these challenges in practice, and whether AI involvement reshapes the kinds of debt that are self-admitted during development.

This study explores SATD in the context of generative AI-assisted software development. By analyzing code comments that explicitly reference both LLM use and technical debt markers, we investigate how developers identify and articulate shortcomings in AI-generated code. Specifically, we ask:

RQ1: What types of self-admitted technical debt emerge in developer comments that acknowledge both LLM involvement and technical debt?

We observed that developers most frequently report *Design Debt* (33/81), followed by *Requirement Debt* (17/81) and *Test Debt* (17/81). Compared to prior SATD distributions, design-related issues are proportionally lower, while requirement and testing debts are higher, indicating that AI-assisted development tends to shift self-admitted

Conference'17, Washington, DC, USA

ACM ISBN 978-x-xxxx-xxxx-x/YYYY/MM
https://doi.org/10.1145/nnnnnnn.nnnnnnn

debt toward later development stages, particularly requirement completion and validation activities.

RQ2: How do developers attribute the role of AI in generative-AI induced SATDs?

We found that developers often attribute AI-assisted code as a trigger for potential issues, expressing uncertainty about its completeness and correctness and noting the need for more verification or revision (34/81 cases). In several instances, developers hold AI-assisted code responsible for errors, redundant logic, or unstable behavior that must be resolved later (22/81 cases). At the same time, they also recognize AI assistance as beneficial for improving, refining, or validating existing implementations which contributes to resolving existing technical debts (19/81 cases).

Our findings show that AI-assisted development reshapes the distribution of technical debt, with fewer design-related issues but a higher prevalence of requirement and testing debts. Beyond these shifts, we identify a distinct form of debt is emerging when AI-generated code is adopted without full developer understanding or confidence. We term this **GenAI-Induced Self-admitted Technical debt (GIST)**.

To the best of our knowledge, this is the first empirical study that systematically examines self-admitted technical debt explicitly linked to generative AI usage in source code comments. Looking ahead, our work posits several directions to examine how AI involvement influences the emergence and evolution of technical debt across the software development life cycle (SDLC), and how development tools or team practices might adapt to identify, document, and manage these AI-related debts more effectively.

DATA AVAILABILITY The dataset is available at [23].

2 Related Work

Self-Admitted Technical Debt. Potdar and Shihab conducted an exploratory study on SATD by manually analyzing code comments to characterize how developers acknowledge and describe technical debt [24]. Maldonado et al. introduced a widely used taxonomy and later developed NLP-based methods for automatic SATD detection [20, 21]. Li et al. conducted a systematic mapping study on technical debt and its management, categorizing major debt types, summarizing management strategies, and identifying gaps in empirical evidence and automation that shaped subsequent research on technical debt [18]. Large-scale empirical analyses further quantified SATD prevalence and types across open-source projects [5, 14]. Subsequent work compared SATD practices in industry and open source [34], examined keyword-based detection [25, 26], and proposed refined taxonomies [7]. Other research extended detection beyond comments to multiple artifacts such as commits, issues, and pull requests [17]. In parallel, automation efforts have mapped tools for technical-debt management across the development lifecycle [6].

AI-Assisted Software Development. Recent studies have examined how generative AI is influencing software engineering practice. Sauvola et al. discussed how AI may reshape development processes and responsibilities [28], while Coutinho et al. reported productivity gains alongside reliability and trust challenges [8]. Empirical studies of GitHub Copilot and other LLM tools have analyzed developer–AI collaboration and code quality [4, 9, 13, 15]. Related work has also explored AI for identifying or managing technical debt [16, 22].

Despite these advances, little has been explored about how the use of AI itself contributes to the formation of technical debt. Our study addresses this gap by analyzing code comments that explicitly mention both AI involvement and self-admitted technical debt.

3 Methodology

3.1 Data Collection

3.1.1 Collection of LLM Referenced Comments. We collected source code comments from public GitHub repositories between November 2022 and July 2025, focusing on Python and JavaScript as these two are the most widely used languages in open-source ecosystem [10, 11, 31]. To identify comments where developers explicitly mention the use of AI/LLM tools, we built 196 structured search queries combining (A) 7 AI related terms (e.g., *LLM, AI, GPT, ChatGPT, Copilot, Gemini, Claude*), (B) 6 generative verbs (e.g., *generated, suggested, written*), and (C) 4 connector terms (*by, from, with, using*). Queries of the forms (A + B) and (B + C + A) were executed via the GitHub Code Search API [12], yielding 37,234 files. We then used AST parsing to extract matched comments [30]. Since many comments were duplicates (e.g., `"generated by ChatGPT"`, we removed them. After deduplication, we found 6,540 unique comments.

3.1.2 Debt Acknowledgment Detection. Within this LLM-referenced set, we applied a second filter using canonical *self-admitted technical debt* keywords recognized in empirical software engineering research: `TODO, FIXME, HACK, XXX` [26]. The search was performed using regular expressions with case-insensitive matching to capture variations such as `// todo` or `# FixMe`.

3.1.3 Resulting Dataset. The intersection of these two filters yielded 96 unique comments that both (a) acknowledged the use or influence of AI/LLM and (b) contained explicit indicators of technical debt. Of the collected comments, 1.47% met our criteria, which is close to the 1.86% of SATD comments reported by [21].

3.2 SATD Types

3.2.1 Taxonomy of SATD. To ensure compatibility with prior research on self-admitted technical debt, we adopted the widely used taxonomy by Maldonado et al. [20]. They proposed 5 common SATD types:

- **Design Debt**: These comments show design flaws in the code, such as misplaced logic, missing abstractions, overly long methods, or temporary workaround implementations.
- **Defect Debt**: Comments where author states that there is a defect in the code.
- **Documentation Debt**: Authors clearly mention the need for documentation of the code.
- **Requirement Debt**: Expresses incompleteness of the code or a unit of code such as: class, function or method.
- **Test Debt**: Shows the need for implementation or improvement of test.

3.2.2 Annotation. Two annotators independently labeled the 96 source code comments following the guidelines provided by Maldonado et al. [20], as detailed in their replication package [19]. Each comment was classified into one of the five SATD categories described in Section 3.2.1. Inter-annotator agreement, measured using Cohen's κ=0.896, indicated high consistency. Disagreements were resolved through in-person discussion.

During annotation, we found 10 instances where AI/LLM mention without actual usage of AI, and in 5 instances did not contain technical debt. These 15 instances were labeled as *False Positive* and removed from the dataset, yielding a final set of 81 annotated comments.

3.3 Identifying the Role of AI

To understand how developers attribute the role of AI in LLM-referenced SATD comments, we manually analyzed all 81 comments using an open coding approach, where categories and concepts were inductively derived from the data rather than predefined [29, 32]. As no prior taxonomy describes how AI contributes to technical debt, the two authors collaboratively developed through an iterative discussion. They reviewed the comments, compared interpretations, and refined emerging themes until stable and consistent categories were established. This process led to 4 types of roles:

- **Source**: Developers indicate that AI-generated code directly introduces problems such as incorrect logic, incomplete implementation, redundant code, or temporary fixes. In these cases, AI is explicitly described as causing the debt. Example: "`TODO:ChatGPT suggested super().close() and it crashed so I added the if. I don't about this.`"
- **Catalyst**: Developers express uncertainty about AI-generated contributions that function but may require later verification. AI does not cause an immediate issue but prompts awareness of potential future debt. Example: "`TODO! validators generated by copilot , should be verified :works but doesn't mean it works all the time`".
- **Mitigator**: Developers mention using AI to help address existing debt, such as by generating tests or refactoring suggestions. Example: "`TODO - Try these tests, generated by Copilot`"
- **Neutral**: AI or technical debt terms are mentioned without a clear link to the creation or resolution of technical debt.

The annotation instructions for open coding are available in our replication package [23].

4 RQ1: What types of self-admitted technical debt emerge in developer comments that acknowledge both LLM involvement and technical debt?

We begin by examining the types of self-admitted technical debt that appear in comments explicitly mentioning LLM use.

4.1 Results

Table 1 summarizes the types of SATD identified in code comments that explicitly mention generative-AI usage. The most common category is *Design Debt (33/81)*, which reflects issues such as misplaced code, poor implementation or a temporary workaround. For example, "`TODO - this is copilot generated code, needs refactoring to a kdata object`" is indicating that the implementation needs to be refactored, adding a design debt to the developers. Similarly, "`TODO: Modify this component to fit your needs. The ProfilePage component was generated with Github Copilot.`" suggests that the developers need to modify the AI-suggested component to fit project-specific needs.

Requirement Debt appears in 17/81 comments. Figure 1 illustrates a typical example where a function was initially generated by Gemini but left unfinished. Another comment, "`TODO: Add parameter to include ingredients from the gpt generated check`", demonstrates a deferred integration task where AI generated code needs to include the additional parameters.

We observed *Test Debt* in 17/81 cases, where developers defer testing or validation of the integrated AI-assisted code, e.g., "`TODO: test this Copilot generated code`". There are also comments mentioning about tests generated by generative AI, which requires further improvement, e.g., *"TODO 2023-08-23 10:41: - [ ] Skeleton of tests written by ChatGPT, write tests"*.

We further noticed *Defect Debts* in 11/81 instances, where developers acknowledging a defect/bug that needs to be fixed. For example, comments such as "`TODO fix this ChatGPT created code.`" and "`TODO: Does not work. It's just generated from ChatGPT`" illustrate those. A very small portion were *Documentation Debt (3/81)*, e.g., "`TODO (USERNAME): This comment is generated by ChatGPT, which may not be accurate.`"

4.2 Discussion.

Design Debt appears most frequently, but its proportion is notably lower than in prior studies, while *Requirement* and *Test* Debts are comparatively higher. This shift suggests that AI-assisted development may influence how debt is introduced: developers rely on generative models to produce initial structures but defer completion and validation. The higher share of *Test Debt* indicates that AI-generated code often enters projects without immediate verification, reflecting postponed quality assurance. Despite our smaller dataset size compared to Maldonado et al. [19, 21], our findings suggest a possible difference in the distribution of technical debt categories. In particular, we observe a lower proportion of design-related issues (40.74% vs. 71.84% in Maldonado et al.), alongside comparatively higher shares of requirement (20.98% vs. 14.24%) and testing-related concerns (20.98% vs. 2.09%). Taken together, these differences suggest a possible tendency for AI-generated code to be associated more often with implementation and evaluation activities than with upfront design decisions, although these observations remain exploratory given the dataset size, although further investigation with larger datasets is needed to substantiate this interpretation.

5 RQ2: How do developers attribute the role of AI in generative-AI induced SATDs?

While RQ1 established that AI-assisted development shifts the distribution of technical debt toward implementation and validation stages. However, these categories alone do not capture how developers perceive AI's responsibility in creating or mitigating such

Table 1: Distribution of Technical Debt Types

Debt Type	What the comment addresses	Example	Count
Design Debt	Problematic design, misplaced code, poor implementation	"`TODO - this is copilot generated code, needs refactoring to a kdata object`"	33 (40.74%)
Requirement Debt	Incomplete/partial implementation	"`TODO: Add parameter to include ingredients from the gpt generated check [...]`"	17 (20.98%)
Test Debt	Needs test/verification/validation of AI supported code	"`TODO: test this Copilot generated code'`"	17 (20.98%)
Defect Debt	Bug/defect in code	"`TODO fix this ChatGPT created code.`"	11 (13.58%)
Documentation Debt	Incomplete/improper documentation	"`TODO (USERNAME): This comment is generated by ChatGPT, which may not be accurate.`"	3 (3.70%)
Total			**81**

debt. To address this, we examine the ways developers attribute roles to AI when discussing self-admitted technical debt.

5.1 Results.

Figure 2 summarizes the roles developers attribute to AI within SATD-related comments. The most common role is that of a *Catalyst - (34/81, 41.98%)*, where AI-assisted codes prompt awareness of potential technical debt. These comments express uncertainty about correctness or completeness, leading developers to defer testing and validation. For instance, "`TODO: generated by ChatGPT, don't know how reasonable this is`" shows doubt about the reliability of a generated artifact, while, "`TODO: AI Generated, please check the fields.`" records postponed checking. Here, AI functions as a *Catalyst*, surfacing uncertainty that developers acknowledge as potential debt.

In 22/81 (27.2%) cases, developers are identifying AI as a *Source* of technical debt. These comments describe situations where AI-generated code introduces errors, redundant logic, or unstable behavior that developers must later address. For example, "`TODO: Does not work. It's just generated from ChatGPT`" signals nonfunctional output that requires fixing, while "`TODO: remove all unnecessary methods; this is an AI-generated file`" points to redundant code.

A third pattern, we observed in 19/81 cases, developers frame AI as a *Mitigator*. Here, AI contributes to reducing existing debt by generating refactors, test scaffolds, or suggesting design alternatives. For instance, "`PJB replaced with 'new' in each case. TODO: use an improved const as suggested by Claude`" here, Claude is suggesting how to mitigate a past debt Finally, In 6/81 (7.4%) of the comments are classified as neutral, where AI is mentioned but its role to technical debt cannot be clearly determined.

Distribution of AI Roles Across SATD Types. Figure 2 shows how AI roles vary across SATD types. When AI is identified as the *Source* of debt, it is most often associated with *Design Debt* (9 instances), followed by *Requirement Debt* and *Test Debt* (6 instances each), and *Documentation Debt* (1 instance).

As a *Catalyst*, AI is primarily linked to *Test Debt* (15 instances) and *Design Debt* (13 instances). When acting as a *Mitigator*, it is most commonly associated with *Requirement* Debt (9 instances) and *Design Debt* (8 instances), reflecting its use in supporting code generation and refactoring.

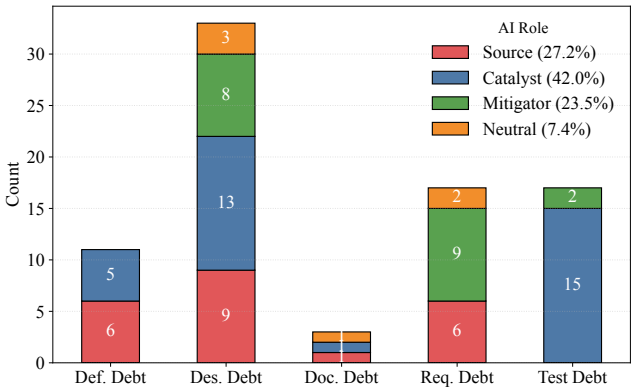

Figure 2: GenAI roles across SATD types

5.2 Discussion.

The most frequent role developers attribute to AI is that of a *Catalyst*, suggesting that AI raises awareness of potential issues. Developers often treat AI-generated code as provisional material that may require additional testing or refactoring. This reflects a human-in-the-loop workflow in which AI speeds up implementation while developers retain responsibility for quality assurance. When AI is identified as the *Source*, it is typically blamed for introducing faulty or redundant code linked to incomplete or unstable functionality. AI also appears as a *Mitigator*, assisting in debt reduction by suggesting improvements, generating tests, or supporting refactoring, particularly in *Design* and *Requirement* Debt. Developers thus engage with AI in multiple, sometimes contrasting ways, as a potential risk, an opportunity for inspection, and a mechanism for improvement.

6 GenAI-Induced Self-admitted Technical debt (GIST)

We observed a recurring pattern that developers are integrating AI-assisted code into their production codebases despite i) uncertainty about its correctness, and ii) limited understanding of its internal logic. We introduce this notion as a *conceptual interpretation grounded in observed self-admitted comments*. This pattern appears across multiple SATD types discussed in Section 3.2.1. For example,

in the comment, "`TODO: Copilot suggested this function (I have no clue what the regex is doing)`", the developer acknowledges incorporating an AI-generated function without understanding its behavior. Similarly, "`TODO: This is totally GPT generated and I'm not sure it works`" expresses uncertainty about correctness, and "`TODO: generated by ChatGPT, don't know how reasonable this is`" reflects doubt about the plausibility of the AI output. Collectively, these comments illustrate a recurring form of uncertainty-driven technical debt that we refer to as **GenAI-Induced Self-admitted Technical debt (GIST)**.

The observed pattern is consistent with prior work on automation bias, where developers rely on automated suggestions despite uncertainty about their correctness or rationale [1, 27]. Research in human-AI interaction has shown that such reliance can influence judgment, verification behavior, and trust in automated systems [1, 27, 33]. In our study, we observe how automation-related uncertainty is documented as self-admitted technical debt in source code comments.

GIST characterizes situations in which uncertainty about AI-generated code is not immediately resolved but instead deferred, creating a latent burden for future development and maintenance. This form of debt manifests through two recurring dimensions:

- **Knowledge Deficit and Deferred Quality Assurance**: Developers integrate AI-generated code without a full understanding of its underlying logic or correctness, creating a gap between the artifact and their mental model of the system. This partial comprehension often leads to postponed validation or testing, as developers defer closer examination to a later stage. Such deferral constitutes a cognitive form of technical debt: understanding is temporarily outsourced to the AI, leaving uncertainty that may resurface during maintenance or modification.
- **Lack of Trust and Delegated Responsibility**: Developers' uncertainty about the reliability or rationale of AI-generated code may erode their confidence in the system as a whole. To manage this uncertainty, responsibility for verifying or improving the AI's output is implicitly shifted to other team members or future revisions. This delegation reflects an operational dimension of **GIST**, where provisional integration of AI output allows unverified code to persist, creating ambiguity around ownership and long-term accountability.

7 Implications and Reflections

7.1 Technical Debt in AI-Assisted Development

Our findings reveal a shift in the composition of SATD. *Design* Debt has become less dominant, while *Requirement* and *Test* Debts are increasingly common. The changing distribution of debt types suggests that AI assistance is reshaping not the taxonomy of technical debt but its underlying causes. In traditional settings, debt often results from conscious trade-offs or time pressure. In contrast, AI-assisted debt tends to emerge from partial reliance on automatically generated code and deferred human oversight. This shift implies that technical debt frameworks may need to account for delegated decision-making, where responsibility for quality and completion is shared between humans and AI systems. More broadly, our observations suggest that some AI-related debt may arise unintentionally, not from deliberate shortcuts but from uncertainty about the behavior or suitability of AI-generated code.

7.2 Interpreting GIST

The notion of **GIST** highlights how limited understanding of AI-generated code can affect long-term maintainability. When code functions without a clear rationale, comprehension and accountability may become fragmented, increasing future effort to modify, validate, or extend the code. In the analyzed comments, developers explicitly acknowledge such uncertainty when integrating AI-generated artifacts, indicating awareness of potential future maintenance challenges.

Integrating principles from *explainable AI* and *responsible AI* could help mitigate these risks by promoting transparency about how models produce code, documenting decision rationales, and clarifying accountability when AI assistance is used. Such practices may help developers retain insight into AI-generated code and better recognize when limited understanding could introduce additional maintenance effort over time.

7.3 Integrating AI Practices into the SDLC

The observed roles of AI, as Source, Catalyst, and Mitigator, point to the need for clearer integration within the software development life cycle (SDLC). Differentiating these roles can guide appropriate oversight, from validation and testing to documentation and provenance tracking. Such awareness may help teams balance AI's supportive functions with the risks of introducing uncertainty or incomplete understanding into codebases.

8 Threats to Validity

This study relies on keyword-based detection of LLM mentions and self-admitted technical debt indicators, which may result in false positives and false negatives. Developers may reference AI-generated code using alternative phrasing not captured by our queries, and not all instances of `TODO` or `FIXME` necessarily indicate genuine technical debt. Although manual annotation was used to filter irrelevant cases, some subjectivity in interpretation remains.

The analysis is based on 81 AI-related SATD instances drawn from public Python and JavaScript repositories between November 2022 and July 2025. While this enables focused qualitative analysis, the limited sample size constrains quantitative generalization. The findings should therefore be interpreted as exploratory and may not generalize to proprietary projects, other languages, or future AI-assisted development practices as generative AI tools and developer workflows continue to evolve.

9 Conclusion and Future Work

This study provides an empirical examination of self-admitted technical debt that explicitly references the use of generative AI. By analyzing 81 LLM-related code comments, we observe that AI assistance is associated with a shift in how debt is self-admitted: design-related issues appear less frequently, while requirement and testing debts are more common, reflecting deferred completion and validation of AI-generated code. We propose **GenAI-Induced Self-admitted Technical debt (GIST)** as a conceptual lens to describe

recurring patterns of self-admitted technical debt in which developers explicitly express uncertainty about the behavior or rationale of AI-generated code, inviting researchers to further examine, refine, and empirically validate this notion across broader contexts.

Our study represents an initial step toward understanding AI-related technical debt and how generative AI is influencing software development life cycle practices. The analysis is limited to Python and JavaScript projects; future work should extend to additional languages and ecosystems to assess whether similar patterns hold. Longitudinal repository studies and automated analysis techniques could further illuminate how AI-related debts evolve over time and whether they are resolved or accumulate as systems mature. Future research may also explore practices that improve the transparency and verifiability of AI-generated code, supporting developers in managing uncertainty and sustaining long-term maintainability in AI-assisted software development.

References

[1] Ujué Agudo, Karlos G Liberal, Miren Arrese, and Helena Matute. 2024. The impact of AI errors in a human-in-the-loop process. *Cognitive Research: Principles and Implications* 9, 1 (2024), 1.

[2] Nicolli SR Alves, Thiago S Mendes, Manoel G De Mendonça, Rodrigo O Spínola, Forrest Shull, and Carolyn Seaman. 2016. Identification and management of technical debt: A systematic mapping study. *Information and Software Technology* 70 (2016), 100–121.

[3] Nicolli S.R. Alves, Leilane F. Ribeiro, Vivyane Caires, Thiago S. Mendes, and Rodrigo O. Spínola. 2014. Towards an Ontology of Terms on Technical Debt. In *2014 Sixth International Workshop on Managing Technical Debt*. 1–7. doi:10.1109/MTD.2014.9

[4] Stefan Barke, Reid Holmes, and Christian Bird. 2023. Grounded Copilot: How Programmers Interact with Code-Generating Models. In *Proceedings of the 45th International Conference on Software Engineering (ICSE '23)*. IEEE/ACM, 1433–1445. doi:10.1109/ICSE48619.2023.00119

[5] 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.

[6] João Paulo Biazotto, Daniel Feitosa, Paris Avgeriou, and Elisa Yumi Nakagawa. 2024. Technical debt management automation: State of the art and future perspectives. *Information and Software Technology* 167 (2024), 107375.

[7] Nathan Cassee, Fiorella Zampetti, Nicole Novielli, Alexander Serebrenik, and Massimiliano Di Penta. 2022. Self-admitted technical debt and comments' polarity: an empirical study. *Empirical Software Engineering* 27, 6 (2022), 139.

[8] Mariana Coutinho, Lorena Marques, Anderson Santos, Marcio Dahia, Cesar França, and Ronnie de Souza Santos. 2024. The role of generative ai in software development productivity: A pilot case study. In *Proceedings of the 1st ACM International Conference on AI-Powered Software*. 131–138.

[9] Xueying Du, Mingwei Liu, Kaixin Wang, Hanlin Wang, Junwei Liu, Yixuan Chen, Jiayi Feng, Chaofeng Sha, Xin Peng, and Yiling Lou. 2024. Evaluating large language models in class-level code generation. In *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering*. 1–13.

[10] GitHub. 2023. The State of Open Source and AI in 2023. https://github.blog/news-insights/research/the-state-of-open-source-and-ai/ [Available Online].

[11] GitHub. 2024. Octoverse 2024: AI leads Python to top language. https://github.blog/news-insights/octoverse/octoverse-2024/ [Available Online].

[12] GitHub, Inc. [n. d.]. GitHub Code Search API. https://docs.github.com/en/rest/code-search. [Available Online].

[13] 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 *Proceedings of the 46th IEEE/ACM International Conference on Software Engineering*. 1–13.

[14] Qiao Huang, Emad Shihab, Xin Xia, David Lo, and Shanping Li. 2018. Identifying self-admitted technical debt in open source projects using text mining. *Empirical Software Engineering* 23, 1 (2018), 418–451. doi:10.1007/s10664-017-9522-4

[15] Kailun Jin, Chung-Yu Wang, Hung Viet Pham, and Hadi Hemmati. 2024. Can chatgpt support developers? an empirical evaluation of large language models for code generation. In *Proceedings of the 21st International Conference on Mining Software Repositories*. 167–171.

[16] Jun Li, Lixian Li, Jin Liu, Xiao Yu, Xiao Liu, and Jacky Wai Keung. 2025. Large language model ChatGPT versus small deep learning models for self-admitted technical debt detection: Why not together? *Software: Practice and Experience* 55, 1 (2025), 3–28.

[17] Yikun Li, Mohamed Soliman, and Paris Avgeriou. 2023. Automatic identification of self-admitted technical debt from four different sources. *Empirical Software Engineering* 28, 6 (2023), 131. doi:10.1007/s10664-023-10297-9

[18] Zengyang Li, Paris Avgeriou, and Peng Liang. 2015. A systematic mapping study on technical debt and its management. *Journal of systems and software* 101 (2015), 193–220.

[19] E. Maldonado and E. Shihab. [n. d.]. tse.satd.data: Manually-labelled dataset of self-admitted technical debt. GitHub repository. https://github.com/maldonado/tse.satd.data

[20] Everton da S Maldonado and Emad Shihab. 2015. Detecting and quantifying different types of self-admitted technical debt. In *2015 IEEE 7Th international workshop on managing technical debt (MTD)*. IEEE, 9–15.

[21] Eduardo S. Maldonado, Emad Shihab, and Nikolaos Tsantalis. 2017. Using Natural Language Processing to Automatically Detect Self-Admitted Technical Debt. *IEEE Transactions on Software Engineering* 43, 11 (2017), 1044–1062. doi:10.1109/TSE.2017.2654243

[22] Eric L Melin and Nasir U Eisty. 2025. Exploring the advances in using machine learning to identify technical debt and self-admitted technical debt. In *2025 IEEE/ACIS 23rd International Conference on Software Engineering Research, Management and Applications (SERA)*. IEEE, 15–22.

[23] Abdullah Al Mujahid. 2026. "TODO: Fix the Mess Gemini Created": Towards Understanding GenAI-Induced Self-Admitted Technical Debt. Zenodo. doi:10.5281/zenodo.18194031

[24] Aniket Potdar and Emad Shihab. 2014. An Exploratory Study on Self-Admitted Technical Debt. In *Proceedings of the 30th IEEE International Conference on Software Maintenance and Evolution (ICSME)*. IEEE, 91–100. doi:10.1109/ICSME.2014.31

[25] Leevi Rantala, Mika Mäntylä, and Valentina Lenarduzzi. 2024. Keyword-labeled self-admitted technical debt and static code analysis have significant relationship but limited overlap. *Software Quality Journal* 32, 2 (2024), 391–429.

[26] Leevi Rantala, Mika Mäntylä, and David Lo. 2020. Prevalence, contents and automatic detection of KL-SATD. In *2020 46th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*. IEEE, 385–388.

[27] Charvi Rastogi, Yunfeng Zhang, Dennis Wei, Kush R Varshney, Amit Dhurandhar, and Richard Tomsett. 2022. Deciding fast and slow: The role of cognitive biases in ai-assisted decision-making. *Proceedings of the ACM on Human-computer Interaction* 6, CSCW1 (2022), 1–22.

[28] Jaakko Sauvola, Sasu Tarkoma, Mika Klemettinen, Jukka Riekki, and David Doermann. 2024. Future of software development with generative AI. *Automated Software Engineering* 31, 1 (2024), 26.

[29] Carolyn B. Seaman. 1999. Qualitative methods in empirical studies of software engineering. *IEEE Transactions on software engineering* 25, 4 (1999), 557–572.

[30] Tree sitter Contributors. [n. d.]. Tree-sitter: An Incremental Parsing System for Programming Tools. https://github.com/tree-sitter/tree-sitter [GitHub repository].

[31] Stack Overflow. 2024. Technology | 2024 Stack Overflow Developer Survey. https://survey.stackoverflow.co/2024/technology [Available Online].

[32] Anselm Strauss and Juliet Corbin. 1990. *Basics of qualitative research*. Sage publications.

[33] Tim Lewis Wingerter, Tim Straub, and Sascha Schweitzer. 2025. Mitigating automation bias in generative AI through nudges: a cognitive reflection test study. *Procedia computer science* 270 (2025), 2106–2114.

[34] Fiorella Zampetti, Gianmarco Fucci, Alexander Serebrenik, and Massimiliano Di Penta. 2021. Self-admitted technical debt practices: a comparison between industry and open-source. *Empirical Software Engineering* 26, 6 (2021), 131.