Do Generative AI Tools Change How Developers Comment and Document Code?

Oluwatito Ebiwonjumi [1], Love Allen Chijioke Ahakonye [2], Dong-Seong Kim [3], [4]
[1] Independent Researcher, Nashville, *Tennessee*, USA
[2] ICT-Convergence Research Center, *Kumoh National Institute of Technology*, Gumi, South Korea
[3] IT Convergence Engineering, *Kumoh National Institute of Technology*, Gumi, South Korea
[4] NSLab Co. Ltd., Gumi, South Korea, *Kumoh National Institute of Technology*, Gumi, South Korea
titoebiwonjumi@gmail.com,(loveahakonye, dskim)@kumoh.ac.kr

***Abstract*—Code comments and documentation are essential for software maintainability, facilitating code understanding, modification, and debugging. While generative artificial intelligence (AI) coding tools have reshaped development practices since 2022, their impact on commenting and documentation remains uncharted territory. This paper presents an empirical study examining documentation patterns in major open-source repositories before (2018-2021) and after (2022-2025) the rise of AI-assisted programming. We analyze commit histories from six repositories (pandas, scikit-learn, TensorFlow, Django, React, Node.js) across Python and JavaScript ecosystems, measuring documentation commit ratios, commit message quality, and documentation scope. Our findings reveal an 8.3% decrease in documentation-focused commits and a 53.4% increase in commit message detail between eras. We observe that commit messages in the post-AI era contain 56.8% more words on average (34.8 vs 22.2 words), with statistical significance (p ¡ 0.0001). Documentation commits also show an 84.9% reduction in scope (260.7 vs 1,721.1 lines changed), though this change is not statistically significant. These patterns suggest that AI tools may be influencing how developers approach code documentation, with potential implications for long-term maintainability. This work provides empirical evidence to guide best practices for documentation in AI-assisted development and highlights the need for tooling that encourages comprehensive commenting across code-generation methods.**

***Index Terms*—Generative AI, code documentation, commit messages, empirical software engineering, software maintainability,**

I. Introduction

Code documentation serves as the bridge between implementation and understanding. Well-documented code enables maintenance, facilitates onboarding, prevents bugs through clarity, and supports collaborative development [1]. Despite its importance, documentation often receives insufficient attention, with developers prioritizing feature delivery over comprehensive commenting [2]. The advent of generative AI tools like GitHub Copilot and ChatGPT introduces a new variable into this equation. Understanding the relationship between AI adoption and documentation practices has significant implications for software quality and maintainability.

From a theoretical perspective, AI-assisted development may alter documentation patterns through several mechanisms. Cognitive load theory suggests that when AI tools reduce the mental effort required for code generation, developers may reallocate cognitive resources toward other activities, potentially including documentation [3]. Alternatively, the perceived self-explanatory nature of AI-generated code may reduce documentation motivation, a phenomenon related to the illusion of explanatory depth [4]. Additionally, dual-process theory in human cognition suggests that AI assistance may shift development from deliberative to more automatic processing, potentially affecting the reflective practices that produce quality documentation [5].

Two competing hypotheses emerge from these theoretical considerations. First, AI tools might improve documentation by automatically generating boilerplate comments or freeing developer time for explanatory writing. Alternatively, AI-generated code might arrive with minimal context, and developers may assume AI-produced code is self-explanatory, reducing documentation effort. Understanding which pattern dominates requires empirical investigation of real-world development practices.

This study investigates documentation and commenting patterns across the pre-AI (2018–2021) and post-AI (2022–2025) eras through four research questions:

- **RQ1:** Has the proportion of documentation-focused commits changed between pre-AI and post-AI eras?
- **RQ2:** Has commit message quality (length, detail) changed between these periods?
- **RQ3:** What types of documentation changes (major documentation updates, minor docstring edits, comment additions) show the most substantial shifts?
- **RQ4:** Has the scope of documentation commits (measured by lines changed) evolved?

We focus on the same six repositories as our companion study on code review patterns, enabling comparison across multiple dimensions of software quality and providing a comprehensive picture of AI's impact on development practices.

II. Related Work

This section highlights studies on the critical role of documentation in software quality and maintainability. We examine theoretical foundations of documentation practices, recent empirical work on AI-assisted development, and commit message quality research. While AI-driven code generation and commit messaging are evolving rapidly, their impact on documentation

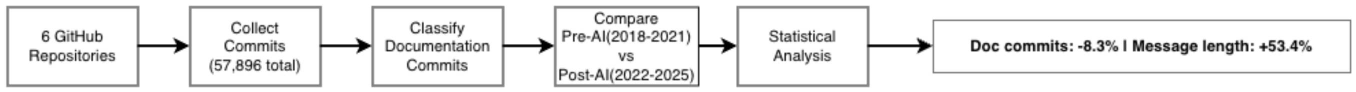
6 GitHub Repositories
Collect Commits (57,896 total)
Classify Documentation Commits
Compare Pre-AI(2018-2021) vs Post-AI(2022-2025)
Statistical Analysis
Doc commits: -8.3% | Message length: +53.4%

Fig. 1. Software Engineering Practices Analysis: Pre-AI vs Post-AI

practices in real-world projects remains underexplored from both theoretical and empirical perspectives.

A. Theoretical Foundations of Documentation

Software documentation research has established theoretical frameworks for understanding documentation's role in development. The cognitive dimensions framework identifies documentation as essential for supporting the progressive evaluation and viscosity dimensions of programming systems [6]. Information foraging theory explains how developers use documentation as information scent when navigating codebases [7]. Recent theoretical work on technical debt identifies inadequate documentation as a form of documentation debt that increases maintenance costs over time [8].

Studies examining code quality perceptions found that readability, structure, and documentation consistently rank among the defining properties of high-quality code [2]. Research on software maintainability in embedded systems identified documentation, along with coding rules and conventions, as essential practices [9]. Empirical investigations demonstrate that well-documented systems reduce maintenance costs and enable faster issue identification [1]. However, documentation adequacy remains a persistent challenge, with studies showing many projects have insufficient or outdated comments.

B. AI Code Generation and Documentation

Recent work has examined the characteristics of AI-generated code and its documentation from both theoretical and practical perspectives. Research evaluating large language model (LLM)-based documentation generation found that professionals and students often lack prompt engineering skills, resulting in documentation perceived as less readable and concise when using ad-hoc prompts compared to prepared prompts [10]. This finding aligns with theories of tool appropriation, where users must develop mental models of AI capabilities to leverage them [11] effectively.

Studies of automated commit message generation using LLMs demonstrate that these models can generate high-quality messages [12], [13], though questions remain about their impact on broader documentation practices. From a sociotechnical perspective, the introduction of AI tools may shift team norms around documentation, as developers adjust their practices based on perceived tool capabilities [14]. Empirical analysis of how AI adoption affects real-world documentation practices at scale remains limited in the literature.

C. Commit Message Quality

Commit messages serve as project-level documentation. Recent research demonstrates that commit message quality impacts software defect proneness, though overall message quality decreases over time despite developers believing they write better messages [15]. Large-scale empirical studies found that LLMs can generate commit messages that outperform traditional automatic generation methods by orders of magnitude [12], [13]. Research on in-context learning for commit message generation shows that LLMs can produce high-quality messages with only a few demonstrations [16]. However, systematic examination of how generative AI adoption affects documentation practices in real open-source software (OSS) over time remains absent.

D. Research Gap

Despite extensive theoretical and empirical work on documentation and recent advances in AI-assisted development, no prior work has systematically examined how generative AI adoption affects documentation practices in real OSS projects over time from a dual theoretical-empirical perspective. Our longitudinal analysis addresses this gap by combining empirical measurement with theoretical interpretation grounded in cognitive load theory, dual-process theory, and sociotechnical systems perspectives.

III. Methodology

A. Research Design

We employ a longitudinal comparative approach with two time periods: Pre-AI (2018-01-01 to 2021-12-31) and Post-AI (2022-01-01 to 2025-11-11), using GitHub Copilot's June 2022 public release as the temporal boundary. This natural experiment design, illustrated in Figure 1, allows us to examine changes in documentation patterns before and after the widespread availability of AI coding assistants. While we cannot establish definitive causation, the temporal division provides a meaningful framework for investigating correlations between AI tool availability and documentation practices.

B. Repository Selection

We analyze the same six repositories as our companion study: **pandas**, **scikit-learn**, **TensorFlow**, **Django** (Python ecosystem), **React**, and **Node.js** (JavaScript ecosystem). These projects span diverse application domains, have active development in both eras, and have substantial commit histories for robust analysis.

C. Data Collection

Using GitHub's application programming interfaces (APIs), we collected all commits from each repository within both time windows. For each commit, we extracted: commit message, commit date, author, files changed, lines added/deleted, and commit type classification. **Commit Classification**: We classified commits using keyword matching:

- **Documentation commits**: Messages containing “documentation”, “readme”, “tutorial”, “guide”, “doc:”, “docstring”, “comment”
- **Bug-fix commits**: Messages containing “fix”, “bug”, “issue”, “regression”
- **Feature commits**: Messages containing “add”, “implement”, “new”

We further subdivided documentation-related commits into:

- **Major documentation**: “documentation”, “readme”, “tutorial”, “guide”
- **Minor documentation**: “docstring”, “comment”, “doc”
- **Documentation typos**: “typo”/“spelling”/“grammar” + “doc”/“comment”/“readme”

D. Metrics

The **documentation commitment metrics** are quantified using the ratio $R_{doc} = \frac{N_{doc}}{N_{total}}$, which represents the proportion of commits focused on documentation in relation to the total number of commits. Additionally, the proportions of major and minor documentation commits are denoted by R_{major} and R_{minor}, respectively. The **commit message quality metrics** include the character count L_{chars} and the word count L_{words} of each commit message, which serve as indicators of the brevity and clarity of the documentation within the commit messages. The **documentation scope metrics** are represented by $S_{doc} = L_{add} + L_{del}$, which measures the average number of lines added and deleted in documentation-related commits, providing insight into the extent of changes made to the documentation.

E. Analysis Methods

We use chi-square tests for categorical comparisons (documentation commit ratios), as in Equation 1.

$$\chi^2 = \sum \frac{(O_i - E_i)^2}{E_i} \tag{1}$$

Equation 2 is the Welch’s t-test for continuous metrics,

$$t = \frac{\bar{X}_1 - \bar{X}_2}{\sqrt{\frac{s_1^2}{n_1} + \frac{s_2^2}{n_2}}}, \tag{2}$$

and Mann-Whitney U tests for non-parametric validation. Significance threshold is $\alpha = 0.05$.

F. Threats to Validity

Internal Validity: Keyword-based commit classification may misclassify some commits [15]. Commit messages are proxies for code-level documentation changes and may not fully capture inline comment density. We cannot definitively attribute changes to AI tools versus other factors, like team composition shifts.

External Validity: Results may not generalize beyond the six studied repositories or beyond Python/JavaScript ecosystems. Private repositories may show different patterns.

Construct Validity: Commit message length does not necessarily indicate quality; verbose messages are not always better. The documentation commit ratio does not measure documentation adequacy; it only measures the frequency of documentation commits. We mitigate through multiple complementary metrics, pre-repository analysis for pattern consistency, and transparent acknowledgment of limitations.

IV. Results

A. Dataset Overview

Our dataset comprises 30,000 pre-AI-era commits and 27,896 post-AI-era commits across six repositories. Table I presents changes in the distribution and documentation commit ratio.

TABLE I
Documentation Commit Ratio Changes by Repository

Repository	**Pre-AI Doc Ratio**	**Post-AI Doc Ratio**
pandas	0.080	0.146
scikit-learn	0.317	0.246
React	0.016	0.015
Node.js	0.165	0.117
TensorFlow	0.021	0.015
Django	0.081	0.088
Overall	**0.113**	**0.104**

B. Documentation Commit Ratio (RQ1)

Documentation-focused commits decreased from 11.3% (3,397 commits) to 10.4% (2,896 commits), an 8.3% relative decline ($\chi^2 = 13.14$, $p = 0.0003$). The decrease is statistically significant, suggesting reduced documentation commitment in the post-AI era. 2 displays this comparison. Per-repository analysis reveals substantial heterogeneity. Notably, *pandas* increased dramatically (8.0% $\rightarrow$ 14.6%), while *scikit-learn* decreased substantially (31.7% $\rightarrow$ 24.6%). The JavaScript projects (React, Node.js) both showed decreases, with Node.js declining from 16.5% to 11.7%. TensorFlow showed minimal documentation commitment in both eras (2.1% $\rightarrow$ 1.5%).

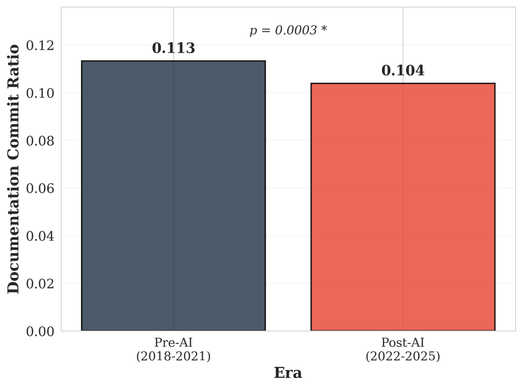

Fig. 2. Documentation Commit Ratio: Pre-AI vs Post-AI

C. Commit Message Quality (RQ2)

Commit message length increased substantially from 191.7 to 294.0 characters (53.4% increase, $t = -25.39$, $p < 0.0001$). Word count showed similar patterns: 22.2 to 34.8 words (56.8% increase). The median message length increased from 101 to 149 characters.

The Mann-Whitney U test confirmed significance ($U = 333{,}931{,}762$, $p < 0.0001$), indicating this is not merely a mean shift but a distributional change toward longer, more detailed commit messages. Figure 3 illustrates the character length comparison, while Figure 4 shows the word count comparison. Both metrics demonstrate the substantial increase in commit message detail. This finding is particularly striking: while dedicated documentation commits have declined slightly, developers are writing substantially more detailed commit messages. This suggests a shift in where documentation effort is being invested, from in-code documentation to commit-level explanations.

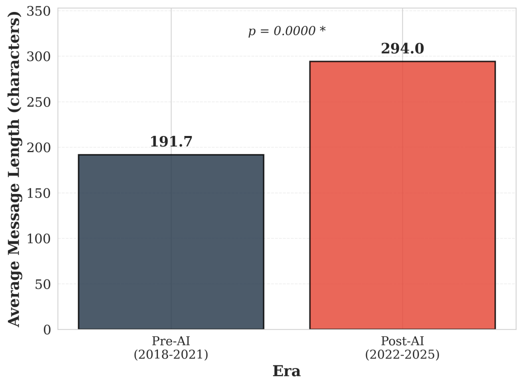

Fig. 3. Commit Message Length: Pre-AI vs Post-AI

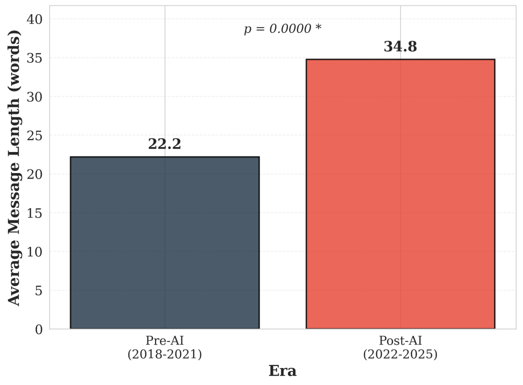

Fig. 4. Commit Message Detail: Pre-AI vs Post-AI

D. Documentation Detail Level (RQ3)

The analysis of documentation commits by type reveals evolving priorities before and after the introduction of AI. **Pre-AI**, major documentation commits accounted for 7.74% of the total, with 2,322 commits, while minor documentation represented 9.26% with 2,778 commits. Documentation-related typo corrections comprised 0.45% of the total, amounting to 134 commits. **Post-AI**, the proportion of major documentation commits increased to 9.44%, with 2,633 commits, while minor documentation decreased to 7.74% with 2,160 commits. Documentation typo corrections also saw a reduction, making up only 0.32% of the total with 88 commits. The pattern shows a shift toward major documentation updates (7.74% $\rightarrow$ 9.44%) and away from minor documentation work (9.26% $\rightarrow$ 7.74%). This suggests that, in the post-AI era, developers who commit to documentation focus on comprehensive updates rather than incremental improvements.

E. Documentation Commit Scope (RQ4)

Documentation commits in the pre-AI era changed an average of 1,721 lines, compared to 261 lines in the post-AI era, an 84.9% reduction. However, this change is not statistically significant ($t = 0.91$, $p = 0.3637$), likely due to high variance in both distributions. The substantial point estimate difference suggests documentation commits may be becoming more focused and targeted, touching fewer lines per commit. Combined with the shift toward major documentation commits, this may indicate a pattern of more frequent, smaller-scope documentation updates rather than significant, comprehensive documentation overhauls.

F. Ecosystem Comparison

Python projects showed mixed patterns (pandas +82%, scikit-learn -22%, TensorFlow -29%, Django +9%), while JavaScript projects consistently decreased (React -6%, Node.js -29%). However, the small sample size (4 Python, 2 JavaScript projects) limits the strength of ecosystem-level conclusions. Commit message length increased substantially in both ecosystems, suggesting this is a universal trend rather than language-specific behavior. Table II summarizes all measured metrics with statistical test results.

G. Discussion

1) Findings: Our analysis reveals a nuanced and somewhat paradoxical shift in documentation practices. While documentation-focused commits decreased by 8.3% ($p = 0.0003$), commit message detail increased dramatically to 53.4% in character length and 56.8% in word count ($p < 0.0001$). This suggests not a decline in documentation effort, but rather a *redistribution* of that effort. One interpretation is that AI tools reduce the need for certain types of in-code documentation. Suppose AI-generated code is clearer or more self-documenting. In that case, developers may invest less in inline comments while compensating with richer commit messages that explain the *why* rather than the *what*. Alternatively, developers may be using AI tools to generate more comprehensive commit messages, leveraging AI's language-generation capabilities for documentation [12], [16]. The shift from minor documentation commits (9.26% $\rightarrow$ 7.74%) to major documentation commits (7.74% $\rightarrow$ 9.44%) suggests a strategic reallocation: when developers do document, they focus on comprehensive updates rather than incremental tweaks.

TABLE II
SUMMARY OF DOCUMENTATION METRICS ACROSS PRE-AI AND POST-AI ERAS

Metric	**Pre-AI (2018-2021)**	**Post-AI (2022-2025)**	**Change**	**Test Stat**	**p-value**	**Sig?**
Doc Commit Ratio	0.113	0.104	−8.3%	$\chi^2 = 13.14$	0.0003	Yes
Avg Message Length (chars)	191.7	294.0	+53.4%	$t = -25.39$	<0.0001	Yes
Avg Message Length (words)	22.2	34.8	+56.8%	$t = -25.39$	<0.0001	Yes
Avg Doc Commit Size (lines)	1721.1	260.7	−84.9%	$t = 0.91$	0.3637	No

This could reflect efficiency gains from AI assistance, enabling more substantial documentation work in less time. The 84.9% reduction in documentation commit scope (though not statistically significant) may indicate more granular, focused documentation updates rather than large-scale overhauls. This aligns with modern development practices that favor smaller, more frequent commits.

2) Implications for Software Maintainability: The findings have mixed implications for maintainability. On the one hand, richer commit messages (56.8% more words) provide better historical context, aiding future developers in understanding the rationale for changes [1], [2]. This is valuable for long-term maintenance and onboarding. On the other hand, the 8.3% decrease in documentation commits raises concerns. If this reflects actual reductions in inline code documentation (docstrings, comments), it could harm code comprehension. Future work should analyze source code directly to determine whether inline documentation density has indeed decreased. The relationship between documentation patterns and bug-fix rates (from our companion study) is noteworthy. Our first paper showed bug-fix rates increased 6.1% in the post-AI era. The modest decrease in documentation commits may contribute to this pattern if reduced documentation leads to misunderstandings and errors.

3) Implications for Stakeholders: For developers, these findings highlight the importance of maintaining balanced documentation practices. While the data indicates a positive trend in commit message quality, ensuring long-term maintainability requires attention to both commit-level and code-level documentation. Developers should recognize that AI tools may unconsciously shift documentation habits and actively monitor their practices to ensure comprehensive documentation across all levels. Organizations should actively monitor and enforce standards for both commit message quality and the density of in-code documentation. This can be accomplished through several mechanisms: implementing guidelines and code review checklists to ensure adequate inline documentation, especially for complex logic; measuring documentation metrics as part of quality assurance processes; and fostering a culture that values documentation as integral to code quality rather than as an afterthought. Tool developers should focus on enhancing AI-driven solutions to not only generate code but also provide comprehensive documentation at multiple levels. Current advancements in AI tools could facilitate the automatic generation of commit messages based on code changes [12], [13], create docstrings for functions [10], and prompt developers to document non-obvious logic. Integrating such capabilities with documentation generation tools could further improve documentation quality while preserving the cognitive benefits of AI-assisted development. Researchers should investigate the underlying mechanisms driving the observed trends. Several research directions emerge from our findings: What cognitive and social factors drive the increasing detail in commit messages? Are developers utilizing AI for message generation, and if so, what are the qualitative differences? Does the shift away from inline documentation represent a problematic gap or a rational adaptation to clearer AI-generated code? What is the relationship between documentation patterns and downstream quality outcomes such as bug introduction, maintenance costs, and developer onboarding time?

H. Limitations and Future Work

Our keyword-based classification may miss nuanced documentation work that uses non-standard terminology or implicit documentation practices. Commit-level analysis does not capture inline comment density within files, a critical dimension of code documentation that may be changing independently of commit patterns. We cannot directly measure AI tool usage or definitively attribute changes solely to AI adoption, rather than confounding factors such as team composition changes, evolving project maturity, or shifts in development methodology.

Future work should pursue several directions:

(1) Analyze source code directly for comment density and quality using static analysis tools, correlating changes with commit patterns to provide a complete picture of documentation practices.

(2) Correlate documentation patterns with downstream maintenance costs, bug severity, time-to-fix, and developer onboarding time to establish whether the observed shifts have measurable impacts on software quality outcomes.

(3) Conduct developer surveys and interviews to understand conscious versus unconscious changes in documentation practices, the role of AI tools in generating documentation, and developer perceptions of documentation adequacy in AI-assisted development.

(4) Perform controlled experiments comparing documentation quality and completeness with and without AI tools, isolating the causal impact of AI assistance from other temporal factors.

(5) Investigate whether AI-generated commit messages differ qualitatively from human-written ones in terms of infor-

mativeness, accuracy, and utility for archaeological debugging [13], [17].

V. Conclusion

This study provides empirical evidence of how generative AI adoption correlates with code documentation practices in open-source software. Our findings indicate a complex shift: documentation commit frequency decreased modestly (8.3%), yet commit message detail increased substantially (56.8% more words). This suggests not a decline in documentation effort but a redistribution toward commit-level explanations rather than in-code documentation. While the improvement in commit message quality is encouraging, the decrease in documentation commits warrants attention. Maintainability requires comprehensive documentation at multiple levels: inline comments explaining complex logic, docstrings describing APIs, and commit messages explaining change rationale [1], [15]. Organizations and developers should ensure that AI adoption doesn't inadvertently reduce in-code documentation quality while improving commit messages.

We call for: (1) explicit documentation guidelines for AI-assisted development emphasizing both inline and commit-level documentation, (2) tool features supporting comprehensive commenting at all levels, (3) continued empirical research on documentation quality and its relationship to maintainability outcomes, and (4) community awareness that documentation remains critical regardless of code generation method. Together with our companion study on code review and bug patterns, this work reveals a nuanced picture of AI's impact: potential productivity gains, shifts in where developers invest effort, but also quality concerns that require thoughtful mitigation strategies. The path forward requires balancing AI's capabilities with sustained commitment to software engineering best practices.

Acknowledgment

This work was partly supported by Innovative Human Resource Development for Local Intellectualization program through the Institute of Information & Communications Technology Planning & Evaluation (IITP) grant funded by the Korean government (MSIT) (IITP-2026-RS-2020-II201612, 40%), the Priority Research Centers Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Education, Science and Technology (2018R1A6A1A03024003, 30%), by the Institute of Information & Communications Technology Planning & Evaluation (IITP)-ITRC(Information Technology Research Center) grant funded by the Korea government(Ministry of Science and ICT)(IITP-2026-RS-2024-00438430 30%)

References

[1] E. Aghajani, C. Nagy, O. L. Vega-Márquez, M. Linares-Vásquez, L. Moreno, G. Bavota, and M. Lanza, "Software documentation: the practitioners' perspective," *IEEE Transactions on Software Engineering*, vol. 46, no. 12, pp. 1341–1364, 2020.

[2] J. Börstler, K. E. Bennin, S. Hooshangi, J. Jeuring, H. Keuning, C. Kleiner, B. MacKellar, R. Duran, H. Störrle, D. Toll *et al.*, "Developers talking about code quality," *Empirical Software Engineering*, vol. 28, no. 6, p. 128, 2023.

[3] J. Sweller, J. J. van Merriënboer, and F. Paas, "Cognitive architecture and instructional design: 20 years later," *Educational Psychology Review*, vol. 31, no. 2, pp. 261–292, 2020.

[4] L. Rozenblit and F. Keil, "The misunderstood limits of folk science: An illusion of explanatory depth," *Cognitive Science*, vol. 26, no. 5, pp. 521–562, 2002.

[5] D. Kahneman, *Thinking, Fast and Slow*. New York: Farrar, Straus and Giroux, 2011.

[6] T. R. G. Green and M. Petre, "Usability analysis of visual programming environments: A 'cognitive dimensions' framework," *Journal of Visual Languages & Computing*, vol. 7, no. 2, pp. 131–174, 1996.

[7] P. Pirolli and S. Card, "The sensemaking process and leverage points for analyst technology as identified through cognitive task analysis," in *Proceedings of International Conference on Intelligence Analysis*, vol. 5, 2005, pp. 2–4.

[8] N. S. Alves, T. S. Mendes, M. G. de Mendonça, R. O. Spínola, F. Shull, and C. Seaman, "Identification and management of technical debt: A systematic mapping study," *Information and Software Technology*, vol. 70, pp. 100–121, 2016.

[9] S. Motogna, A. Vescan, C. Serban, and G. Czibula, "Empirical investigation in embedded systems: Quality attributes in general, maintainability in particular," *Information and Software Technology*, vol. 157, p. 107155, 2023.

[10] H. Tang and S. Nadi, "Evaluating software documentation quality," in *2023 IEEE/ACM 20th International Conference on Mining Software Repositories (MSR)*. IEEE, 2023, pp. 542–554.

[11] D. Wang, Q. Yang, A. Abdul, and B. Y. Lim, "Designing theory-driven user-centric explainable ai," in *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems*. ACM, 2021, pp. 1–15.

[12] P. Xue, L. Wu, Z. Yu, Z. Jin, Z. Yang, X. Li, Z. Yang, and Y. Tan, "Automated commit message generation with large language models: An empirical study and beyond," *IEEE Transactions on Software Engineering*, vol. 50, no. 12, pp. 3208–3224, 2024.

[13] C. V. Lopes, V. Klotzman, I. Ma, and I. Ahmed, "Commit messages in the age of large language models," *arXiv preprint arXiv:2401.17622*, 2024.

[14] S. Barke, M. B. James, and N. Polikarpova, "Grounded copilot: How programmers interact with code-generating models," *Proceedings of the ACM on Programming Languages*, vol. 7, no. OOPSLA1, pp. 85–111, 2023.

[15] J. Li and I. Ahmed, "Commit message matters: Investigating impact and evolution of commit message quality," in *Proceedings of the 45th International Conference on Software Engineering (ICSE)*. IEEE/ACM, 2023, pp. 806–817.

[16] Y. Wu, Y. Wang, Y. Li, W. Tao, S. Yu, Y. Yang, W. Jiang, P. Li, L. Briand, and T. Lethbridge, "An empirical study on commit message generation using llms via in-context learning," *arXiv preprint arXiv:2502.18904*, 2025.

[17] Y. Zhang, Z. Qiu, K.-J. Stol, W. Zhu, J. Zhu, Y. Tian, and H. Liu, "Automatic commit message generation: A critical review and directions for future work," *IEEE Transactions on Software Engineering*, vol. 50, pp. 816–835, 2024.