Who Tells the Documentation Team? How AI Agents Disrupt Documentation Feedback Loops

Avinash Bhat
McGill University
Montréal, Canada
avinash.bhat@mail.mcgill.ca

Jin L.C. Guo
McGill University
Montréal, Canada
jguo@cs.mcgill.ca

Abstract

Software documentation teams rely on developer activity to identify and correct problems in their content. As AI coding assistants reshape how developers seek information, their impact on software documentation feedback channels has gone largely unnoticed. When developers reduce reading documentation and stop posting questions, documentation teams lose the signals they need to find and fix problems. We use systems thinking to trace how agent-mediated information seeking disrupts the *balancing loops* that currently contribute to documentation quality and how this disruption would create *reinforcing loops* that degrade documentation and code quality over time. We identify *leverage points* where researchers can develop quality metrics and self-correcting documentation systems for agent-mediated use, and system designers can surface agent consumption patterns to documentation teams. Through this position paper, we call on the research community to investigate how agent-mediated documentation consumption reshapes documentation and its quality.

CCS Concepts

• **Software and its engineering** → **Documentation**; • **Human-centered computing** → *Computer supported cooperative work*; • **Computing methodologies** → *Intelligent agents.*

Keywords

Coding Assistants, Software Documentation, Feedback Loops, Agentic Programming, Documentation Quality

ACM Reference Format:
Avinash Bhat and Jin L.C. Guo. 2026. Who Tells the Documentation Team? How AI Agents Disrupt Documentation Feedback Loops. In *34th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (FSE Companion '26), July 05–09, 2026, Montreal, QC, Canada.* ACM, New York, NY, USA, 2 pages. https://doi.org/10.1145/3803437.3806708

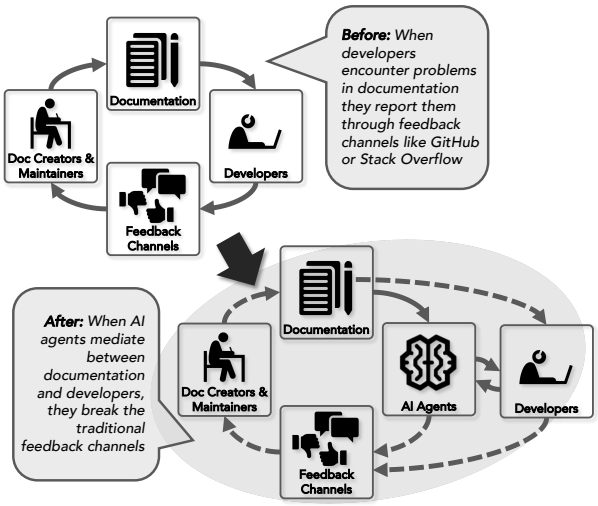

Figure 1: Documentation feedback loops before and after adopting agents. Before, developers consult documentation and report problems through feedback channels, creating a *balancing loop*. After, AI agents mediate this interaction, weakening the feedback paths (shown as dashed lines).

1 Introduction

AI coding assistants have quickly become a central component of software development workflows. Controlled experiments and practitioner surveys show that these tools significantly improve productivity and are increasingly integrated across the development lifecycle [8, 9]. However, the impact of this shift on the teams that create and maintain developer documentation, such as API documentation, has been largely overlooked. Documentation teams depend on signals from developer interactions with documentation, like questions or votes on forums like GitHub, to identify what needs improvement and are captured through feedback channels such as issue trackers and forums. Agents consuming documentation do not produce these signals, and these channels go silent. What does it mean for documentation quality? To answer this question, we examine how these feedback channels worked before agent adoption, and how agents disrupt them.

From a systems thinking perspective [6], we identify two *balancing loops* originating from developers that the documentation teams and illustrate with API documentation as an example. The first is a direct feedback loop: when a developer finds the API documentation incomplete or incorrect, they report the issue through channels such as issue trackers or pull requests maintained by the developing organization. The second, indirect loop emerges from developer activity across platforms. When developers encounter problems with API documentation in the course of other activities, their concerns surface on whichever platform hosts those activities [1]: content related concerns appear in issue trackers and pull requests, process related concerns on mailing lists, and tooling related questions on

FSE Companion '26, Montreal, QC, Canada

ACM ISBN 979-8-4007-2636-1/2026/07
https://doi.org/10.1145/3803437.3806708

Stack Overflow. Some developers also create their own resources, such as blog posts or video tutorials [2], to complement the official documentation and catalog missing information.

2 How Agents Affect Feedback Loops

As developers seek information through AI agents, they stop consulting documentation sources directly, posting questions on forums, or creating resources. Barke et al. [3] found that programmers often accepted Copilot-suggested code without consulting any documentation source; to many participants, Copilot replaced what would previously have been a Stack Overflow search. Subsequent studies have similarly found that developers use language models as faster alternatives to reading documentation [5] and external forums [9]. When agents absorb these behaviors, the questions, votes, and issue reports that would have reached documentation teams are never created.

In systems thinking, a *reinforcing loop* is one where each cycle amplifies the output of the previous one [6]. Such loops can emerge when balancing loops lose their corrective feedback signal, leading to problems that would otherwise be corrected now persisting in the system. This is the risk for documentation consumed by agents. When an agent retrieves ambiguous or incomplete documentation, those problems propagate into the code it generates. If future models are trained on this code, the same documentation problems can lead to the same errors in the next generation of models. This constitutes a reinforcing loop that deteriorates code and documentation quality.

The same dynamic produces a reinforcing loop at the organizational level. When agents absorb the friction that previously generated complaints from developers, documentation teams see fewer incoming issues. Reduced complaint volume can appear to indicate adequate quality, which justifies cutting investment in documentation maintenance. Less investment results in poorer documentation, which agents continue to consume without correction.

3 Implications

Meadows and Wright [6, Chapter 6] argue that not all interventions in a system are equally effective, and proposed a list of twelve places to intervene, which they term *leverage points*. The list contains several factors, including numbers (their term for metrics and standards), information flows, and goals that govern how the system behaves. Some leverage points, like goals, are more powerful because they change the conditions that produce problems, rather than addressing symptoms. The reinforcing loops described in Section 2 can be addressed through interventions at several levels. However, these interventions carry a tension: improving documentation for agent consumption and improving it for human readers can pull in different directions. The interventions below should be evaluated against both audiences, with a bias toward human readability, because agents can be adapted to work with human-readable content more readily than humans can reconstruct meaning from content optimized for machine retrieval.

For (Agentic) System Designers. A higher-order leverage point is the structure of *information flows* in the system. Coding assistants like Copilot already observe which documentation pages agents retrieve and whether developers accept or reject the generated output. If developers frequently reject outputs generated from a particular page, that pattern suggests the page contains ambiguities or inaccuracies. Surfacing these patterns to documentation maintainers would create a new feedback channel. Prior work has shown that surfacing consumption traces to creators leads to improvements in documentation [7]. Practitioners have already begun developing ad hoc heuristics for agent-friendly documentation [4], but systematic feedback from tool usage can support it empirically. However, this depends on cooperation from agentic tool providers, who may view consumption and rejection data as proprietary. Surfacing these patterns also raises privacy concerns, since rejection signals are tied to individual coding sessions that developers may not expect to be shared broadly; requiring clear agreement on what gets reported, to whom, and at what granularity. Designing this feedback channel, therefore, requires aligning the interests of tool providers, developers, and documentation maintainers, none of whom currently have incentives or mechanisms to negotiate.

For Researchers. Existing documentation quality frameworks measure properties like readability or completeness, which assume a human reader who constructs meaning from the text [10]. Evaluating documentation for model consumption requires a different set of quality metrics; it must account for how agents misinterpret documentation and the downstream errors they produce. This *measurement* infrastructure would support the interventions described earlier. A longer-term vision is to develop documentation systems that can detect their own degradation through agent consumption patterns and trigger corrections autonomously. Such systems would represent a move toward *self-organization*, in which the documentation itself participates in the balancing loop rather than relying on external actors to close it.

References

[1] Emad Aghajani, Csaba Nagy, Olga Lucero Vega-Márquez, Mario Linares-Vásquez, Laura Moreno, Gabriele Bavota, and Michele Lanza. 2019. Software Documentation Issues Unveiled. In *2019 IEEE/ACM 41st International Conference on Software Engineering*. doi:10.1109/ICSE.2019.00122

[2] Deeksha M. Arya, Jin L. C. Guo, and Martin P. Robillard. 2024. Why People Contribute Software Documentation. In *Proceedings of the 2024 IEEE/ACM 17th International Conference on Cooperative and Human Aspects of Software Engineering*. doi:10.1145/3641822.3641881

[3] Shraddha Barke, Michael B. James, and Nadia Polikarpova. 2023. Grounded Copilot: How Programmers Interact with Code-Generating Models. *Proceedings of the ACM on Programming Languages* 7, OOPSLA1 (2023). doi:10.1145/3586030

[4] Rebecca M. Deprey. 2025. How to Create LLM-Ready Documentation. https://web.archive.org/web/20250613231213/https://rebeccamdeprey.com/blog/how-to-create-llm-ready-documentation [Accessed 02-04-2026].

[5] Ranim Khojah, Mazen Mohamad, Philipp Leitner, and Francisco Gomes de Oliveira Neto. 2024. Beyond Code Generation: An Observational Study of ChatGPT Usage in Software Engineering Practice. *Proc. ACM Softw. Eng.* 1 (2024). doi:10.1145/3660788

[6] Donella H. Meadows and Diana Wright. 2008. *Thinking in Systems: A Primer*. Chelsea Green Publishing, White River Junction, Vermont.

[7] Alok Mysore and Philip J. Guo. 2018. Porta: Profiling Software Tutorials Using Operating-System-Wide Activity Tracing. In *Proceedings of the 31st Annual ACM Symposium on User Interface Software and Technology*. doi:10.1145/3242587.3242633

[8] Sida Peng, Eirini Kalliamvakou, Peter Cihon, and Mert Demirer. 2023. The Impact of AI on Developer Productivity: Evidence from GitHub Copilot. doi:10.48550/arXiv.2302.06590

[9] Italo Santos, Cleyton Magalhaes, and Ronnie De Souza Santos. 2025. Model-Assisted and Human-Guided: Perceptions and Practices of Software Professionals Using LLMs for Coding. In *2025 2nd IEEE/ACM International Conference on AI-powered Software*. doi:10.1109/AIware69974.2025.00019

[10] Christoph Treude, Justin Middleton, and Thushari Atapattu. 2020. Beyond accuracy: assessing software documentation quality. In *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. doi:10.1145/3368089.3417045