Malleable Interfaces for Reviewing Software Documentation

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

Abstract

Research on documentation quality has examined developers and software artifacts but overlooked technical writers, the dedicated practitioners who synthesize distributed software knowledge into documentation. In interviews with 31 technical writers, we found that documentation quality is achieved through a multi-stage review process involving practitioners with different expertise evaluating against different quality criteria. This process lacks dedicated tooling, forcing documentation teams to adopt developer-focused platforms like GitHub, which exclude non-technical reviewers who then resort to alternative tools, leaving feedback scattered across platforms. This thesis proposes malleable interfaces for collaborative documentation review, where practitioners reshape interfaces to fit their work rather than adapting to fixed tool constraints. We propose to build a review environment where practitioners describe interface needs in natural language and the system generates views that accommodate different expertise and surface information relevant to their roles. We plan two evaluation studies: examining how individuals articulate and use generated interfaces, then how practitioners with different roles collaborate using them. This work contributes toward a vision of documentation tooling that practitioners can shape to fit their collaborative review needs.

CCS Concepts

• **Human-centered computing** → **Collaborative content creation**; • **Software and its engineering** → **Documentation**.

Keywords

Software Documentation Quality, Documentation Review, Collaborative Software Engineering, Generative UI, Malleable Interfaces

ACM Reference Format:
Avinash Bhat. 2026. Malleable Interfaces for Reviewing Software Documentation. In *Companion Proceedings of the 34th ACM Symposium on the Foundations of Software Engineering (FSE '26), June 5–9, 2026, Montreal, Canada.* ACM, New York, NY, USA, 4 pages. https://doi.org/XXXXXXX.XXXXXXX

1 Introduction

> "Software developers have amazing tools because they can make their own tools. Technical writers have unmet needs because we're not software developers and we can't make tools, so we have to rely on developers to make tools for us."
>
> — *anonymous technical writer [4, p. 1765]*

Conference'17, Washington, DC, USA

Documentation captures and communicates knowledge that is distributed across an organization. This knowledge exists in artifacts like code, design documents, and project schedules, and in the tools practitioners use to track their work [3]. Some knowledge never makes it into artifacts at all; design rationale exists only in the heads of practitioners who made those decisions [12, 14]. Creating documentation requires synthesizing all of these sources.

Technical writers synthesize this distributed knowledge into documentation. They consult developers for technical accuracy, support teams for common user problems, and domain experts for subject matter context [10]. Prior research describes this synthesis work as knowledge brokering, connecting people who would not otherwise communicate and translating fragmented knowledge across team boundaries [5, 8]. Despite their central role, research on documentation quality has focused on developers and software artifacts, not technical writers. Aghajani et al. [2] mined developer discussions to build a taxonomy of issues and surveyed developers, architects, and testers to obtain their perspectives [1]. Other researchers have defined quality attributes and developed metrics to assess documentation artifacts themselves [17, 20, 22]. We know little about how technical writers themselves synthesize knowledge from distributed sources and ensure quality.

This thesis aims to address this gap. Through interviews with 31 technical writers, we investigated how they synthesize distributed knowledge into documentation and ensure quality. We found that the review process is the primary mechanism for achieving documentation quality. This process has five stages, each involving different practitioners: developers verify technical accuracy, editors assess clarity and consistency, QA teams evaluate from a user perspective. We also found challenges with tooling (or rather, the lack thereof): documentation teams adopt developer-focused platforms that exclude the non-technical reviewers essential to documentation quality. We describe these findings in detail in §3.

Why do documentation teams lack appropriate tools? Current software couples what users can do with how they can do it, implementing fixed procedures rather than flexible tools [18]. Platform designers decide which roles access which features and which views exist. Users have few opportunities to change their software [19]. GitHub, for example, provides diff views and inline commenting designed for code review. Editors reviewing documentation cannot get prose-level feedback mechanisms in this interface, so they export to PDF. Faced with such constraints, workers develop workarounds; Spinuzzi [16] observed that practitioners "avoid relying on official documentation by developing unofficial ways of sharing information." These workarounds go undocumented, so when the practitioner who created them leaves, the workaround disappears.

Malleable software offers an alternative where users manipulate the interface elements according to their needs [19]. A developer and an editor reviewing the same document could each have views

suited to their expertise. Recent advances in large language models support this by translating user intent into interface changes, so users describe what they need rather than navigate fixed menus [15]. The technical writer whom we quote in the opening describes their need and reliance on developers. With our work, we aim to address this: practitioners can prompt for review interfaces suited to their work rather than building tools or relying on others.

2 Related Work

Collaboration in Software Engineering. Whitehead [21] characterizes SE collaboration as model-based, where practitioners coordinate around shared artifacts like requirements, code, and UML diagrams. Subsequent empirical work has examined how developers collaborate through these artifacts. Constantino et al. [7] interviewed twelve open source developers and found that collaboration extends beyond coding to include issue management and documentation tasks, though their analysis centers on developer roles. Feng et al. [9] broadened the scope to cross-disciplinary teams where domain experts and software engineers co-develop software, identifying 21 frictions arising from mismatched expectations around tools, practices, and division of labor. Documentation review is a form of cross-disciplinary SE collaboration that these studies do not cover: technical communicators construct representations of their audience to evaluate content against criteria like readability and task fit [10], while developers assess accuracy against source code. These practitioners must converge on a single artifact without sharing evaluation criteria or tooling fluency.

Malleable Interfaces. When tools cannot be reshaped, practitioners who need to work differently are forced into workarounds or excluded [18, 19]. The malleable software research program addresses this by making interfaces reshapeable at the site of use: Webstrates [11] demonstrated that web pages can serve as shareable substrates whose content, computation, and interaction users personalize and compose through *transclusion*, and Mackay and Beaudouin-Lafon [13] formalized design principles for how substrates structure objects, enforce constraints, and manage dependencies to combine power with simplicity. Existing demonstrations of malleable collaboration, such as co-authoring with personalized editors in Webstrates, involve participants who share a common goal and competency. Documentation review requires malleability for practitioners with asymmetric expertise who must each reshape the same artifact's interface to surface the quality criteria only they can evaluate.

3 Documentation Review Process

Existing research on documentation quality has examined developers' perspectives, overlooking practitioners with specialized expertise in documentation. We conducted semi-structured interviews with experienced technical writers to understand how documentation quality is achieved in practice. We recruited 31 participants from Write The Docs Slack and LinkedIn technical writing communities. Interviews lasted 45-70 minutes and covered questions about quality practices and review processes. We analyzed transcripts through open coding following the process described by Charmaz [6]. This work is currently under review. In the rest of this section, we highlight the major findings from this paper.

Five Review Stages. Through open coding of interview transcripts, we identified five stages: self review, technical review, editorial review, play testing, and post-publication feedback. Each stage involves practitioners with different expertise: technical writers perform self review as they create documentation; developers conduct technical review for content accuracy; fellow writers and editors perform editorial review for presentation; QA teams conduct play testing to evaluate documentation from a user perspective; users provide post-publication feedback. We found that practitioners evaluate documentation according to their expertise, focusing on distinct quality criteria: developers test whether code examples execute correctly and verify API responses; editors check narrative flow and adherence to style guides; QA teams verify that instructions work from a user perspective. Not all participants went through all five stages; we observed variation based on organizational context and team size. In addition, stages often overlap in practice. Regardless, all participants described some variation of the review process for ensuring documentation quality, with one participant stating, "`I think [documentation] review process is more important than who actually writes the content.`"

Challenges. Although documentation review involves collaboration among practitioners with different expertise, current tools do not support this. Documentation teams lack dedicated tools (21/31) and adopt developer-focused workflows that exclude non-technical reviewers. Even when teams customize or build their own solutions, the maintenance burden falls on individual writers (16/31), creating single points of failure. Moreover, this maintenance work competes with their documentation responsibilities. These tool limitations compound a broader organizational challenge. Reviewers already deprioritize documentation in favor of their core responsibilities (17/31), and friction from unfamiliar interfaces gives them another reason to disengage.

Validation. We sent the paper to the study participants for validation and received 13 responses. Eleven participants completed the survey; ten confirmed that the analysis represents their experiences and seven confirmed their quotes were accurately used in context. Two additional participants replied via email confirming the findings are accurate.

4 Towards Malleable Review Systems

In the previous section, we discussed how documentation review, despite requiring collaboration across roles, still relies on tools designed for individual, technically-fluent authors. From the challenges practitioners described, we derive three requirements for a system that would support collaborative documentation review.

Design Requirements. The system should *accommodate participation from practitioners with different expertise.* We found that developers reviewed in GitHub while documentation editors exported to Google Docs or PDF because they were not familiar with GitHub workflows. This means the comments that documentation editors provide must be manually transferred back to the source. Moreover, the feedback becomes siloed across different platforms, so reviewers cannot see each other's concerns and technical writers must manually collect and reconcile comments from multiple sources. If the system could reshape to accommodate both technical

and editorial reviewers, all practitioners could participate without workarounds.

Next, the *system must support generating interfaces for emergent interactions that arise in the review process.* A developer could request additional technical explanation, but at the same time an editor could flag the same section as too long; we found that technical writers must mediate between these asks. These conversations happen in informal channels like Slack, and the rationale for decisions becomes disconnected from the documentation itself. If practitioners could generate interfaces that surface contested sections and record how they resolved tradeoffs, this reasoning would persist with the artifact as a byproduct of their interaction.

Finally, the *system must allow practitioners to see artifacts and information relevant to their role.* A developer reviewing technical accuracy needs to see whether code examples execute correctly, while an editor checking style needs to see readability metrics. If practitioners could generate views that surface information relevant to their roles alongside the content, they could make informed judgments during review.

The concept of malleable software proposes that users should be able to reshape their tools to fit their work, rather than adapting their work to fit the tools [19]. In our case, we envision that rather than accepting fixed interfaces that exclude some practitioners, those involved in documentation review could generate interfaces as their work demands. To realize this, we propose to build a review environment where practitioners describe what they need in natural language, and the system generates appropriate interfaces on-the-fly, accommodating different expertise, supporting emergent interactions, and surfacing information specific to roles.

System Design. We propose to build a review environment that mirrors GitHub's interface. We choose GitHub because documentation teams already use it for docs-as-code workflows, so practitioners can continue using their existing repositories and processes. In addition, GitHub APIs provide access to pull requests, comments, and file contents, so we can build on existing data infrastructure. GitHub's component library, Primer, is open source, which allows us to constrain generation to valid components. Given this data and component library, we leverage a language model to compose interfaces based on natural language requests; the model does not create new components but configures existing ones.

In practice, a practitioner opens a pull request and sees a default view similar to GitHub's interface. Clicking on any component opens a chat where they describe what they need, and the system translates this into a configuration of Primer components displaying the relevant data from the pull request. The generated view replaces the current view; if it does not match their intent, they can modify the prompt and regenerate.

5 Evaluation & Expected Contributions

The rest of my PhD investigates how practitioners use this system and whether malleable interfaces support collaborative documentation review. We ask: **How does the ability to generate tailored interfaces affect collaboration in documentation review workflows?** We plan two studies to investigate this.

5.1 Study 1: Interface Generation in Practice

Before studying collaboration with generated interfaces, we need to understand how practitioners articulate their interface needs and where they require support. We examine: **How do individual practitioners generate and use malleable interfaces for documentation review?** For instance, an editor who needs to comment on prose quality but finds GitHub's diff view disorienting might prompt the system with "`show me the full rendered page with inline commenting`". Can such practitioners articulate requests that produce interfaces matching their mental models, and do they find those interfaces useful for their review tasks?

We will conduct a lab study with 8-12 participants who are experienced in the documentation review process (we identified the relevant roles in §3), working individually. We will provide them synthetic review tasks (e.g., "review this documentation for issues relevant to your expertise") where interface customization would help. Participants will think aloud as they generate interfaces, and we will observe the prompts they write, the interfaces the system generates, and how they modify their prompts when results do not match their intent. We will follow up with semi-structured interviews about their experience and analyze transcripts using thematic analysis, coding for what needs practitioners express, how they articulate those needs, and where they struggle. This will reveal which interface types practitioners articulate readily and which require scaffolding such as defaults, templates, or examples. We will use these findings to refine the system if needed.

5.2 Study 2: Collaborative Review with Generated Interfaces

This study examines whether generated interfaces support collaboration when practitioners with different expertise review together. We investigate: **How do practitioners collaborate on documentation review using malleable interfaces?**

We will conduct a lab study with pairs of participants who hold different roles in documentation review, as identified in our interview study (§3). Documentation review requires synthesizing knowledge distributed across roles. To simulate this, we will provide each participant with briefing documents containing information only they possess. For example, a developer might receive details about API behavior while a writer receives information about common user confusions. Next, participants will collaboratively review synthetic documentation containing planted issues. We will plant issues that require technical knowledge to identify (e.g., code examples that execute but produce incorrect output), issues that require documentation expertise (e.g., technically accurate prose that is incomprehensible to the target audience), and issues that require both perspectives (e.g., undocumented edge cases that users frequently encounter).

During this study, we will collect three types of data. (1) *Observation*: screen and audio recordings of the full session, capturing what interfaces participants generate, how they coordinate, and how they identify and resolve issues. (2) *Post-study survey*: perceived ease of collaboration, usefulness of generated interfaces, and comparison to participants' usual review tools. (3) *Post-study interview*: what worked, where participants struggled, and whether they would use this approach for real work. We will analyze observation data

through thematic analysis. We are particularly interested in themes like whether generated interfaces surface feedback that practitioners would have skipped in the default view, and how practitioners negotiate competing concerns across roles. We will use survey data to compare against participants' current workflows. We will use interview data to triangulate our observations, surfacing participants' reasoning and their judgments about adopting this approach in practice.

Together, these studies address our overarching research question: whether malleable interfaces are feasible for documentation practitioners and whether they support the collaborative review processes our interview study identified as essential to documentation quality. To observe how these findings hold in real-world settings, we hope to supplement with a limited pilot deployment by recruiting one or two teams willing to use the system for a review cycle. Because proprietary content prevents direct observation, we would use diary studies where participants record what interfaces they generated, why, and what happened, followed by retrospective interviews. This would provide additional ecological validation of findings from the controlled lab setting.

5.3 Expected Contributions

This thesis makes the following contributions: (1) *Process Model:* A five-stage model of documentation review that researchers can use to analyze collaborative documentation practices and practitioners can use to identify gaps in their own workflows; (2) *System:* An open-source malleable review environment demonstrating LLM-mediated interface generation constrained to valid component configurations; (3) *Design Knowledge:* Requirements for collaborative review systems derived from practitioner challenges, and empirical insights from building and evaluating a system that addresses them.

6 Conclusion

Documentation captures knowledge distributed across an organization, and ensuring its quality requires bringing together practitioners with different expertise. Our interview study revealed that this happens through a multi-stage review process where these practitioners evaluate the same artifact against different quality criteria. However, the tools available to them assume who uses them and how, forcing practitioners into fragmented workarounds. The technical writer we quoted at the outset described precisely this dependence on developers to build their tools.

This thesis proposes that practitioners should shape their own. We propose a review environment where practitioners describe their interface needs in natural language and the system generates views suited to their expertise. Our planned studies will evaluate whether practitioners can articulate these needs effectively and whether the generated interfaces support collaborative review across roles. Our research has implications for collaborative software engineering activities beyond documentation review, where practitioners with different expertise must work together using shared tools that were not designed for all of them.

7 Acknowledgements

The author is advised by Prof. Jin Guo from the School of Computer Science at McGill University.

References

[1] Emad Aghajani, Csaba Nagy, Mario Linares-Vásquez, Laura Moreno, Gabriele Bavota, Michele Lanza, and David C. Shepherd. 2020. Software Documentation: The Practitioners' Perspective. In *2020 IEEE/ACM 42nd International Conference on Software Engineering*. https://doi.org/10.1145/3377811.3380405

[2] 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*. https://doi.org/10.1109/ICSE.2019.00122

[3] Andrew Begel, Nachiappan Nagappan, Christopher Poile, and Lucas Layman. 2009. Coordination in large-scale software teams. In *Proceedings of the 2009 ICSE Workshop on Cooperative and Human Aspects on Software Engineering*. https://doi.org/10.1109/CHASE.2009.5071401

[4] Avinash Bhat, Disha Shrivastava, and Jin L.C. Guo. 2024. Do LLMs Meet the Needs of Software Tutorial Writers? Opportunities and Design Implications. In *Proceedings of the 2024 ACM Designing Interactive Systems Conference*. https://doi.org/10.1145/3643834.3660692

[5] Alexander Boden and Gabriela Avram. 2009. Bridging knowledge distribution - The role of knowledge brokers in distributed software development teams. In *2009 ICSE Workshop on Cooperative and Human Aspects on Software Engineering*. https://doi.org/10.1109/CHASE.2009.5071402

[6] Kathy Charmaz. 2014. *Constructing Grounded Theory* (2nd ed.). SAGE Publications, London.

[7] Kattiana Constantino, Shurui Zhou, Mauricio Souza, Eduardo Figueiredo, and Christian Kästner. 2020. Understanding Collaborative Software Development: An Interview Study. In *2020 ACM/IEEE 15th International Conference on Global Software Engineering*. https://doi.org/10.1145/3372787.3390442

[8] Daniela Damian, Remko Helms, Irwin Kwan, Sabrina Marczak, and Benjamin Koelewijn. 2013. The role of domain knowledge and cross-functional communication in socio-technical coordination. In *2013 35th International Conference on Software Engineering*. https://doi.org/10.1109/ICSE.2013.6606590

[9] Zixuan Feng, Thomas Zimmermann, Lorenzo Pisani, Christopher Gooley, Jeremiah Wander, and Anita Sarma. 2025. When Domains Collide: An Activity Theory Exploration of Cross-Disciplinary Collaboration. In *2025 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*. https://doi.org/10.1109/ESEM64174.2025.00017

[10] Marjorie Rush Hovde. 2000. Tactics for Building Images of Audience in Organizational Contexts: An Ethnographic Study of Technical Communicators. *Journal of Business and Technical Communication* (2000). https://doi.org/10.1177/105065190001400401

[11] Clemens N. Klokmose, James R. Eagan, Siemen Baader, Wendy Mackay, and Michel Beaudouin-Lafon. 2015. Webstrates: Shareable Dynamic Media. In *Proceedings of the 28th Annual ACM Symposium on User Interface Software & Technology*. https://doi.org/10.1145/2807442.2807446

[12] Amy J. Ko, Robert DeLine, and Gina Venolia. 2007. Information Needs in Collocated Software Development Teams. In *29th International Conference on Software Engineering*. https://doi.org/10.1109/ICSE.2007.45

[13] Wendy E. Mackay and Michel Beaudouin-Lafon. 2025. Interaction Substrates: Combining Power and Simplicity in Interactive Systems. In *Proceedings of the 2025 CHI Conference on Human Factors in Computing Systems*. https://doi.org/10.1145/3706598.3714006

[14] Martin P. Robillard. 2021. Turnover-induced knowledge loss in practice. In *29th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. https://doi.org/10.1145/3468264.3473923

[15] Mattias Rost. 2025. Reclaiming the Computer through LLM-Mediated Computing. *Interactions* (2025). https://doi.org/10.1145/3747585

[16] Clay Spinuzzi. 2003. *Tracing genres through organizations: A sociocultural approach to information design*. MIT Press.

[17] Henry Tang and Sarah Nadi. 2023. Evaluating Software Documentation Quality. In *2023 IEEE/ACM 20th International Conference on Mining Software Repositories*. https://doi.org/10.1109/MSR59073.2023.00023

[18] Philip Tchernavskij. 2017. Decomposing Interactive Systems. In *HCI.Tools Workshop at CHI 2017*. https://hal.science/hal-01614246

[19] Philip Tchernavskij. 2019. *Designing and Programming Malleable Software*. Ph. D. Dissertation. Université Paris Saclay. https://theses.hal.science/tel-02612943

[20] 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*. https://doi.org/10.1145/3368089.3417045

[21] Jim Whitehead. 2007. Collaboration in Software Engineering: A Roadmap. In *Future of Software Engineering*. https://doi.org/10.1109/FOSE.2007.4

[22] Junji Zhi, Vahid Garousi-Yusifoğlu, Bo Sun, Golara Garousi, Shawn Shahnewaz, and Guenther Ruhe. 2015. Cost, benefits and quality of software development documentation: A systematic mapping. *Journal of Systems and Software* (2015). https://doi.org/10.1016/j.jss.2014.09.042