Markdown Mayhem : Taming the Agentic Documentation Explosion

Harsha Kokel
harsha.kokel@ibm.com
IBM
San Jose, California, USA

Abstract

As software engineering enters an era of agentic development—where large language model (LLM)-based agents increasingly participate as autonomous collaborators—the role and form of documentation is undergoing a profound transformation. Traditional human-centered artifacts such as READMEs and developer guides are being augmented, and in many cases supplanted, by agent-facing instruction files including AGENTS.md, .cursorrules, CLAUDE.md, SKILLS.md, and other tool-specific manifests. While these artifacts promise more precise agent behavior and faster onboarding for automated contributors, their unchecked proliferation has introduced a systemic documentation failure mode: Markdown Mayhem. This position paper highlights an emerging documentation crisis in agentic software engineering, where too many agents continually update too many markdown files, creating ambiguity, redundancy, and erosion of authoritative truth. We argue that without principled governance, agent-oriented documentation risks undermining both human comprehension and agent reliability, and we propose foundational directions for restoring coherence.

1 Introduction

Documentation has traditionally served as a socio-technical contract among human developers. It encodes intent, constraints, and shared understanding across time, teams, and organizational boundaries. In classical software engineering, documentation failures were largely human failures: outdated `README`s, contradictory design documents, or critical context lost to informal communication channels.

Agentic software engineering fundamentally alters this relationship. Large language model (LLM)–based agents now participate as semi-autonomous or autonomous contributors that read, interpret, generate, and modify documentation as part of their operational workflow [1–5]. Documentation is no longer written solely *for* humans, nor updated solely *by* humans. Instead, it increasingly functions as both *input* and *output* of automated systems operating at scale.

This shift changes the failure modes of documentation. Traditional concerns such as staleness or incompleteness are no longer sufficient to characterize the problem. Instead, agentic environments introduce a new class of risk: documentation overproduction without clear authority. When multiple agents continuously externalize implicit knowledge, preferences, and constraints into persistent artifacts, repositories accumulate instructions faster than they humans can be reconcile.

Crucially, this transformation is not accidental. It is driven by legitimate technical pressures: agents lack shared situational awareness, operate across heterogeneous tools, and cannot reliably infer project-specific conventions from code alone. To compensate, development practices increasingly externalize operational knowledge into explicit, agent-readable artifacts.

In this position paper, we argue that the current trajectory of documentation practices in agentic software engineering is unsustainable. While agent-oriented documentation artifacts are individually well-motivated responses to real technical needs, their uncoordinated proliferation has exposed a structural gap: documentation systems were designed for human readers, but are now being repurposed—without sufficient differentiation or governance—for autonomous agents. **Documentation is no longer merely a collection of static files; it is evolving into an agentic memory system that actively shapes and is shaped by automated contributors.**

Position. We take the position that addressing this fundamental shift requires an explicit separation between human-facing and agent-facing documentation, continuous semantic consistency checking across all documentation artifacts, establishing clear ownership and change control, and a shift away from treating Markdown as a universal substrate for all forms of instruction. Without this reframing, documentation risks becoming an active source of system inconsistency rather than a stabilizing socio-technical contract.

2 From Human-Centered Documentation to Agent-Oriented Artifacts

The rapid emergence of agentic software development has driven a corresponding shift in documentation practices. As large language model (LLM)–based agents increasingly participate as semi-autonomous or autonomous contributors, repositories are required to expose information that was previously implicit, informally communicated, or embedded in developer experience.

This proliferation of agent-oriented artifacts reflects several concrete technical needs:

- **Explicit behavioral specification.** Agents require precise, declarative instructions to ensure predictable behavior, particularly in the absence of shared situational context or long-lived memory.
- **Tool-specific alignment.** The ecosystem of agentic tooling varies widely in capabilities, assumptions, and execution models, necessitating localized instruction surfaces tailored to individual tools.
- **Fine-grained control.** Constraints governing coding style, architectural boundaries, safety considerations, and workflow conventions are difficult to infer reliably from source code alone and must be externalized.

As a consequence, modern software repositories increasingly contain a growing collection of agent-facing documentation artifacts, including but not limited to:

- **Behavioral contracts**, such as `AGENTS.md`, `CLAUDE.md`, and editor- or tool-specific rule files (e.g., `.cursorrules`), which encode operational expectations for automated contributors.
- **Tool-scoped policy files**, defining permissions, execution constraints, or preferred interaction patterns specific to a given agent or development environment.
- **Capability and skill manifests**, such as `SKILLS.md`, intended to advertise or constrain the functional scope of an agent.
- **Hierarchically scoped meta-instructions**, distributed across directory structures to capture context-sensitive behavior in monorepos or multi-package systems.

Individually, each of these artifacts serves a legitimate and well-motivated purpose. Collectively, however, they introduce a fragmented and weakly governed instruction landscape, in which overlapping concerns, inconsistent phrasing, and unclear authority relationships complicate both human comprehension and agent interpretation.

3 The Emergence of Markdown Mayhem

We define Markdown Mayhem as a systemic condition characterized by:

Unbounded Growth. Every new agent or tool introduces yet another markdown file, compounding the documentation explosion

Lost Coherence. Conflicting and fragmented instructions across agents lead to globally incoherent system.

Instructional Redundancy. The same rule or constraint appears in multiple files with slightly different wording, often tailored to specific agents or tools.

Conflicting Authority. It becomes unclear which document represents the "source of truth"—especially when agents update documentation independently.

Self-Amplifying Drift. Agents trained or instructed on inconsistent documentation may generate further inconsistent updates, accelerating entropy.

Unlike traditional documentation debt, Markdown Mayhem is actively produced by the system itself. Markdown Mayhem is not merely a tooling annoyance; it exposes a deeper mismatch between static documentation models and dynamic agent ecosystems. Traditional best practices (e.g., "keep docs up to date") fail to address these structural tensions. Key mismatches include:

Human legibility vs. machine operability. Documents optimized for agents may degrade human understanding, while human-friendly prose can introduce ambiguity for machines.

Local optimization vs. global coherence. Agents optimize for their immediate task or tool, not for ecosystem-wide consistency.

Version control vs. semantic consistency. Git captures change, but not intent and instruction alignment across multiple documents.

4 Position: Documentation Must Become a Differentiated System with Special Governance

We argue that documentation in agentic environments must evolve from an undifferentiated collection of markdown files into a governed, role-aware system with explicit distinctions between audiences, purposes, and lifecycles. Treating all documentation as interchangeable Markdown has become a core contributor to Markdown Mayhem.

4.1 Human vs. Agent Documentation Are Fundamentally Different

A central failure mode in current practice is the assumption that a single documentation format—typically Markdown—can simultaneously serve human understanding and sensemaking, and precise, executable instructions for autonomous agents. These goals are often in direct tension. Human-oriented documentation optimizes for explanation, rationale, examples, and narrative flow. Ambiguity is often acceptable, and even useful. Agent-oriented documentation must optimize for determinism, unambiguous interpretation, explicit constraints, and tool-aligned structure. Conflating these audiences results in documents that are too loose and vague for agents, too rigid and mechanical for humans, and ultimately authoritative for neither.

A foundational requirement for agentic software engineering is therefore **explicit separation of human-facing and agent-facing documentation**, even when they describe the same underlying system intent.

4.2 Markdown Is Overloaded Beyond Its Design Limits

Markdown's success as a lightweight, human-friendly format has made it the default container for everything: READMEs, policies, instructions, prompts, capabilities, constraints, and tool configuration. In agentic systems, this has led to Markdown being used as a narrative medium, a policy language, a behavioral control surface, and a machine-interpreted configuration format. This overloading is a primary source of confusion. Markdown lacks formal semantics, enforceable schemas, machine-verifiable constraints, and clear affordances for authority or precedence.

As a result, meaning is increasingly inferred rather than specified—by both humans and agents—making documentation brittle and inconsistent under automation. The problem is not Markdown per se, but its unchecked expansion into roles it was never designed to play.

4.3 Documentation Requires Explicit Authority and Audience Boundaries

To restore coherence, documentation systems must make several boundaries explicit.

- **Audience boundary:** Is this document intended for humans, agents, or translators between the two?
- **Authority boundary:** Is this document normative, advisory or illustrative?

- **Lifecycle boundary:** Is this document current, transitional, archival, or superseded—and under what conditions does its status change?
- **Mutation boundary:** Which agents (if any) are permitted to modify this artifact?

Without these boundaries, agents default to treating any readable Markdown as actionable truth, further amplifying drift when documents conflict.

4.4 From Files to Memory Systems

Ultimately, documentation in agentic environments must be treated as a memory system, not a folder of files. This system should support differentiated representations, clear ownership and authority, controlled mutability, and semantic validation across artifacts. Without this shift, adding more markdown files—even with good intentions—will continue to accelerate Markdown Mayhem rather than resolve it.

5 Conclusion

Agentic software engineering is forcing a reckoning with long-standing assumptions about documentation. The problem is no longer whether documentation exists, but whether it can remain coherent under autonomous, high-velocity modification. Markdown Mayhem is an early warning sign: a symptom of documentation treated as an unstructured, infinitely extensible medium in a world that demands governance, structure, and intent preservation. The future of documentation is not fewer markdown files—but better systems for telling agents and humans what truly matters, and why. If left unaddressed, Markdown Mayhem risks reduced trust in agent behavior, increased onboarding friction for both humans and agents, and silent divergence between intended and actual system behavior. Conversely, addressing it opens new opportunities, documentation as a first-class coordination layer for multi-agent systems, shared ontologies and schemas for agent instructions, and automated validation of documentation consistency.

References

[1] Custom instructions with AGENTS.md – Codex | OpenAI Developers — developers.openai.com. https://developers.openai.com/codex/guides/agents-md. [Accessed 28-04-2026].
[2] How Claude remembers your project - Claude Code Docs — code.claude.com. https://code.claude.com/docs/en/memory. [Accessed 28-04-2026].
[3] Rules | Cursor Docs — cursor.com. https://cursor.com/docs/rules. [Accessed 28-04-2026].
[4] Agentic AI Foundation. Agents.md: A simple, open format for guiding coding agents, 2025. Accessed April 28, 2026.
[5] J. L. Lulla, S. Mohsenimofidi, M. Galster, J. M. Zhang, S. Baltes, and C. Treude. On the impact of agents. md files on the efficiency of ai coding agents. *arXiv preprint arXiv:2601.20404*, 2026.