Documented System Abstractions for AI-Mediated End-User Programming

Stefano Casafranca
scasafrancal01@gmail.com

***Abstract*—Prompt-based AI environments have lowered barriers to software creation but rely on linear, conversational interaction models that fragment program intent over time. This paper proposes *documented system abstractions*, reframing editable documents as executable systems and persistent sources of truth. Computation emerges from structured document edits—prompts, constraints, and refinements—that remain readable, editable, and inspectable as a whole. We ground this framing through an exploratory qualitative evaluation with 12 individual sessions (90 minutes each) involving light and professional technical users working with AI-assisted development tools. Findings show that consolidating prompts into a single document mitigates common breakdowns observed in chat-based environments, improving clarity, control, and preference. We argue that treating documents as systems enables a robust foundation for AI-mediated programming beyond conversational interfaces.**

I. Motivation

AI-assisted programming tools increasingly rely on prompt-based interaction to generate executable artifacts. While effective for rapid prototyping, these environments typically structure interaction as a linear dialogue. As sessions progress, earlier prompts, constraints, and design decisions are buried within chat histories, making revision and reasoning increasingly difficult.

Prior work in end-user programming demonstrated the value of embedding computation within familiar representations. Spreadsheets [1] and computational notebooks [2] lowered barriers by colocating data, logic, and output. Collaborative editors based on Operational Transformation (OT) and Conflict-Free Replicated Data Types (CRDTs) support concurrent authoring [3], [4] but treat documents primarily as linear text streams, offering limited support for persistent computational semantics.

From a human–computer interaction perspective, documents function as external cognitive artifacts that support inspection, reflection, and shared reasoning [1]. When program intent is distributed across ephemeral interaction histories, this cognitive role breaks down. This paper argues that AI-mediated programming benefits from treating the document itself as the executable system.

II. Conceptual Framing

We adopt a *document-as-system* abstraction in which system behavior emerges from transformations of a structured document rather than from replaying a temporal interaction log. Prompts, constraints, and refinements are treated as persistent document elements that collectively define program behavior.

TABLE I
Comparison of editor classes by representation and execution model

Property	OT Editors	CRDT Editors	Notebooks	Document Systems
Primary abstraction	Text operations	Replicated state	Cells + outputs	Structured document
Primary role	Concurrency control	Concurrency control	Interactive execution	Intent-centric execution
Persistent intent	Low	Low	Medium	High
Executable text	No	No	Partial	Yes
History inspectable	Medium	Medium	Low	High
Single source of truth	No	No	Partial	Yes
Execution semantics	None	None	Cell replay	Whole-document recomputation
AI as execution substrate	No	No	Optional	Yes
Multi-user real-time edits	Yes	Yes	Limited	Optional

This framing builds on work in computational documents [5], [6], live programming [7], and provenance-aware systems [8]. By representing both authored intent and derived behavior within the same artifact, documented systems preserve readability while enabling execution.

a) Vibe Coding.: We define *Vibe Coding* as an AI-mediated programming paradigm in which system behavior is shaped through iterative refinement of a shared document rather than explicit source code. In this paradigm, the document functions simultaneously as the source of truth and an executable specification, while AI systems act as collaborative interpreters that translate user intent into behavior, prioritizing legibility, flow, and incremental change over strict syntactic control [9].

III. System Properties

Documented systems differ from chat-based tools in how change is expressed and propagated. Prompt-based environments bind meaning to interaction order: modifying an early prompt often requires restoring a prior checkpoint or manually reconstructing subsequent prompts, fragmenting intent and increasing cognitive overhead. In contrast, documented systems bind meaning to document structure, allowing edits at any point without invalidating later content. System behavior is recomputed from the revised document rather than replayed from a linear history.

Table I situates documented systems relative to established editor classes.

IV. Exploratory Evaluation

We conducted an exploratory qualitative evaluation consisting of 12 individual sessions, each lasting approximately 90 minutes. Participants were light to professional technical users familiar with basic programming concepts but not necessarily experienced with AI-assisted development environments.

A. Method

Participants were introduced to *Lovable AI*, a prompt-based system for generating functional application prototypes. Each participant was asked to design and build an application of their choosing. After producing an initial working version, participants were encouraged to iteratively refine functionality and behavior through additional prompts.

As a comparison point, selected prompts were demonstrated using Cursor / Google Anti-Gravity, illustrating a code-centric AI-assisted workflow. Observations focused on how participants reasoned about system behavior, managed prior prompts, and responded to iterative breakdowns.

B. Observed Interaction Frictions

Across sessions, participants frequently encountered difficulty recovering earlier constraints and intentions once prompt histories grew. Iterative refinement often led to rephrasing previously stated requirements or entering cycles of repeated prompting without clear progress.

Table II summarizes recurring interaction frictions observed across tools.

TABLE II
OBSERVED INTERACTION FRICTIONS ACROSS AI-ASSISTED TOOLS

Friction	Chat-based	Documented
Prompt loss in history	High	Low
Constraint visibility	Low	High
Intent recovery	Low	High
Iterative control	Medium	High

C. Document Consolidation Intervention

For each participant, all prompts issued to Lovable AI were consolidated into a single document with intermediate retries removed. Participants were shown this document and asked how they would prefer to modify system behavior from that representation.

Subjective ratings collected immediately after this intervention are summarized in Table III.

TABLE III
SUBJECTIVE RATINGS AFTER DOCUMENT CONSOLIDATION (1–5 LIKERT SCALE)

Measure	Mean	Min	Max
Clarity	4.4	4	5
Sense of control	4.3	3	5
Preference	4.5	4	5

Participants frequently described the consolidated document as "reading the system as a whole," contrasting it with scrolling through fragmented chat histories.

V. DISCUSSION

The findings suggest that many breakdowns in AI-mediated programming stem from representation rather than model capability. Chat-based interfaces obscure accumulated intent, while documented system abstractions preserve it as a manipulable artifact. Vibe Coding leverages this persistence to support reasoning, refinement, and trust without requiring explicit source code.

VI. LIMITATIONS

This study is exploratory and qualitative. The participant pool was limited, and comparisons were demonstrative rather than controlled. Results should be interpreted as indicative rather than conclusive.

VII. CONCLUSION

Documented system abstractions offer a principled alternative to chat-centric AI programming interfaces. By treating documents as both runtime and representation, they support clarity, control, and iterative refinement in AI-mediated development workflows. Vibe Coding emerges as a viable mode of end-user programming grounded in persistent, readable system artifacts.

REFERENCES

[1] B. A. Nardi and J. R. Miller, "The spreadsheet interface: A basis for end user programming," in *Proceedings of the IFIP TC13 Third International Conference on Human-Computer Interaction (INTERACT '90)*, 1990, pp. 977–983.

[2] J. Jakubovic, J. Edwards, and T. Petricek, "Technical dimensions of programming systems," *The Art, Science, and Engineering of Programming*, vol. 7, no. 13, pp. 1–59, 2023.

[3] M. Kleppmann, A. Wiggins, P. V. Hardenberg, and M. McGranaghan, "Local-first software: You own your data, in spite of the cloud," in *Proceedings of the 2019 ACM SIGPLAN International Symposium on New Ideas, New Paradigms, and Reflections on Programming and Software*, 2019, pp. 154–178.

[4] G. Litt, S. Lim, M. Kleppmann, and P. van Hardenberg, "Peritext: A crdt for collaborative rich text editing," *Proceedings of the ACM on Human-Computer Interaction*, vol. 6, no. CSCW2, 2022.

[5] T. Petricek, "Foundations of a live data exploration environment," in *The Art, Science, and Engineering of Programming*, vol. 4, no. 8, 2020, pp. 1–37.

[6] J. Edwards, "Subtext: Uncovering the simplicity of programming," in *Proceedings of the 20th Annual ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications*, 2005, pp. 505–518.

[7] S. McDirmid, "Usable live programming," in *Proceedings of the 2013 ACM International Symposium on New Ideas, New Paradigms, and Reflections on Programming & Software (Onward! 2013)*, 2013, pp. 53–62.

[8] J. Cheney, S. Chong, N. Foster, M. Seltzer, and S. Vansummeren, "Provenance: A future history," in *Proceedings of the 24th ACM SIGPLAN Conference Companion on Object Oriented Programming Systems Languages and Applications*, 2009, pp. 957–964.

[9] A. Sarkar and I. Drosos, "Vibe coding: Programming through conversation with artificial intelligence," 2025. [Online]. Available: https://arxiv.org/abs/2506.23253