Claude Design and the Novel T

By Austen Tucker · 2026

Claude Design and the Novel T

Why I'm writing my own word processor, because of course I am.

I have made a terrible discovery.

If you give me agents, a CMS, a publishing pipeline, and one free evening, I will eventually try to write my own word processor.

Apparently I cannot be trusted with tools.

The thing I am building is called Novel T. It is a word processor for the agentic writing era — which is a phrase I will defend in a minute, because I know how it sounds. First, the why.

Why existing word processors feel wrong

Most writing tools treat a novel like a long document. One scrollable river of text. If you're smart, you learn to navigate by \<h1\>s and a repeatable pattern. In the apps, it's their proprietary workflow. In "fuck it, we roll open source" land? it's markdown and yaml all the way down.

But a novel is not only a long document. A novel is:

- scenes
- beats
- arcs
- continuity threads
- revision states \<this paragraph is locked, this one is volatile, this one is a placeholder I keep promising to come back to\>
- comments from past me
- comments from future me, which read as more accusatory than I would like
- the ghost of a better idea I had on the train and did not write down

A linear document model flattens all of that into prose plus margin notes. Which is fine for a memo. It is not fine for a 90,000-word object that needs to be internally consistent across nine months of drafting.

The tools that do know about scenes — Scrivener, Ulysses, a few others — get the structure right but were built before agents existed as a daily collaborator. They were built for a writer alone in a room.

I am no longer alone in the room. There is a polite robot next to me asking if I noticed that the protagonist's sister has had three different first names.

What Novel T is trying to be

Not "AI writes the book." Hard no. I have a strong opinion about this and the opinion is no.

What I want is more like:

- Scene-aware drafting. The unit of work is the scene, not the chapter or the document. Scenes know their own metadata: POV, time, location, beats they are responsible for.
- Structured revision passes. "Do a continuity pass." "Do a tension pass." "Do a dialogue pass." Each pass is a different agent persona with different reading instructions, and their suggestions land in a structured review panel rather than as inline comments that destroy the manuscript layout.
- Accept / reject suggestions, surgically. The way you handle a code review, but for prose. Each suggestion is a discrete object you can take or leave.
- Style anchors. A small set of paragraphs you have explicitly marked as "this is what the voice sounds like." Agents read those before they comment on anything else.
- Continuity maps. A live, queryable model of who knows what, where they are, what color the kitchen is, when the moon is full. Less "ask me about chapter 4" and more "tell me what would break if I changed this."
- Export to Payload. Because we just built a printing press. Once a sequence of scenes is locked, it leaves Novel T as a clean post (or a clean batch of scheduled posts) and enters the pipeline I already built.

The connective tissue is that the writing tool, the review process, and the publishing pipeline are all aware of each other. Which has not really been true before.

A scene, in this model, is roughly:

\\\ts\ type Scene = \{\ id: string;\ title: string;\ pov: CharacterId;\ location: LocationId;\ time: Narrative-Timestamp; // not wall-clock — story-clock\ beats: BeatId\[\]; // what this scene is responsible for\ body: RichText;\ state: "stub" | "draft" | "locked";\ styleAnchors?: ParagraphId\[\]; // optional voice samples to weight\ \};\ \\\

Once a scene is structured, every other useful behavior — revision passes, continuity checks, agent comments, export — has something to grab onto.

The Claude Design angle

The interesting design questions here are not "can the model write a scene." The model can write a scene. That is uninteresting.

The interesting questions are:

- What should the editor surface, and what should it hide? (A revision pass that surfaces every comma is a revision pass that gets ignored.)
- Where does an agent comment live? (Inline annotations destroy reading flow. A side panel risks being ignored. A modal is a war crime.)
- How do you make a revision pass feel like a cockpit instead of a punishment?
- What does it look like when an agent disagrees with another agent about the same paragraph, and the writer has to adjudicate?
- What is the minimum amount of structure a scene needs before it stops being prose and starts being queryable?

These are design problems, not modeling problems. Claude is useful here as a design partner — the thing I argue with about whether the suggestions panel should group by pass type or by location in the manuscript. (Both. The answer is both, with a toggle. We argued for a while.)

Where this fits

I built the printing press first, because it solved an immediate problem: I had writing that needed to ship. I built the publishing pipeline second, because the press needed somewhere to send the output. I am building Novel T third, because the upstream of the press — the place where the writing actually happens — is the part that is still a long scrollable river when it should be something smarter.

The arc is going somewhere. CMS becomes programmable. Publishing becomes automated. Writing tool becomes agent-native. Each layer hands a cleaner artifact to the next.

I do not want a word processor that writes for me.

I want a word processor that can hold the lantern while I go into the cave.