What we saw and what we changed

Internal dogfooding write-up — Atono, May 2026

What we saw

We started Atono because we were the team running into this every week. AI in everyone's workflow, things getting written faster, but more of it needing rework on the back end. Tribal knowledge isn't new – just newly expensive.

When we ran the numbers on our own AI-generated stories, 60% needed revisions. The AI wasn't obviously bad. It just didn't know our product terminology, workflows, and assumptions well enough to consistently get the details right.

Your AI doesn't know your product either.

Acceptance criteria for features we'd already shipped. Edge cases that don't exist in our system. Confident, plausible, almost right.

We assumed this was mostly an Atono problem. Then we ran a survey with Refactoring.fm of 350 engineering teams and saw the same pattern at scale: 64% store critical product knowledge in people's heads, and only 9% use AI for requirements work at all. Teams are using AI most where it has the least context.

The hardest part isn't the wrongness. Wrong gets caught quickly – almost right gets shipped. AI doesn't fail by being obviously bad; it fails by being plausibly, confidently wrong. For PMs reviewing AI-assisted specs, that means scanning every line for subtle product mistakes. Every line gets the same scrutiny because any line could be the one. Exhausting work. And the ones you miss surface in the demo, or with a customer.

What we changed

We stopped trying to make AI smarter and started giving it the product context it was missing.

We built Product Knowledge, starting with a Glossary that generates structured product terminology from existing documentation – docs, help centers, internal references, and other sources teams already maintain. It identifies concepts, definitions, synonyms, and related terminology that define how a product actually works.

Now when we generate stories or specs with AI, we can ground that work in a shared glossary instead of relying on the model to guess what our terms mean. For example, "Everything" in our product is a specific workspace view, not a generic word. The same goes for the rest of the product's language.

For our engineering work, we built AI Context. AI sessions in code typically lose context across days, IDEs, and developers – every session starts cold. Our MCP server updates AI Context directly on backlog items as work happens, capturing design decisions, investigation notes, and implementation summaries. The next AI session (or the next developer) picks up where the last one left off. No more reconstructing context from chat history.

A few things we cared about while building this:

- **Refreshable from source documentation.** As docs evolve, teams can refresh the Glossary to identify new concepts, update terminology, and remove outdated definitions without rebuilding everything from scratch.
- **Human-reviewed and controllable.** Teams can manually refine concepts, track changes over time, restore previous versions, and control who can manage glossary updates.

(We also kept it token-light – the structure feeds AI efficiently without burning the context window or slowing down your existing AI workflow.)

The Glossary and AI Context solve different sides of the same problem: AI systems struggle when product context is fragmented, inconsistent, or trapped in people's heads.

A way to try it

The Glossary takes minutes to set up. Paste a link to your docs and we'll build it: atono.io/lp/get-your-glossary-built

If you want to talk through what you're seeing on your team first, reply and we'll set something up.

- Troy

Co-founder & CEO, Atono

If you want to go deeper on the research: The Context Gap report.