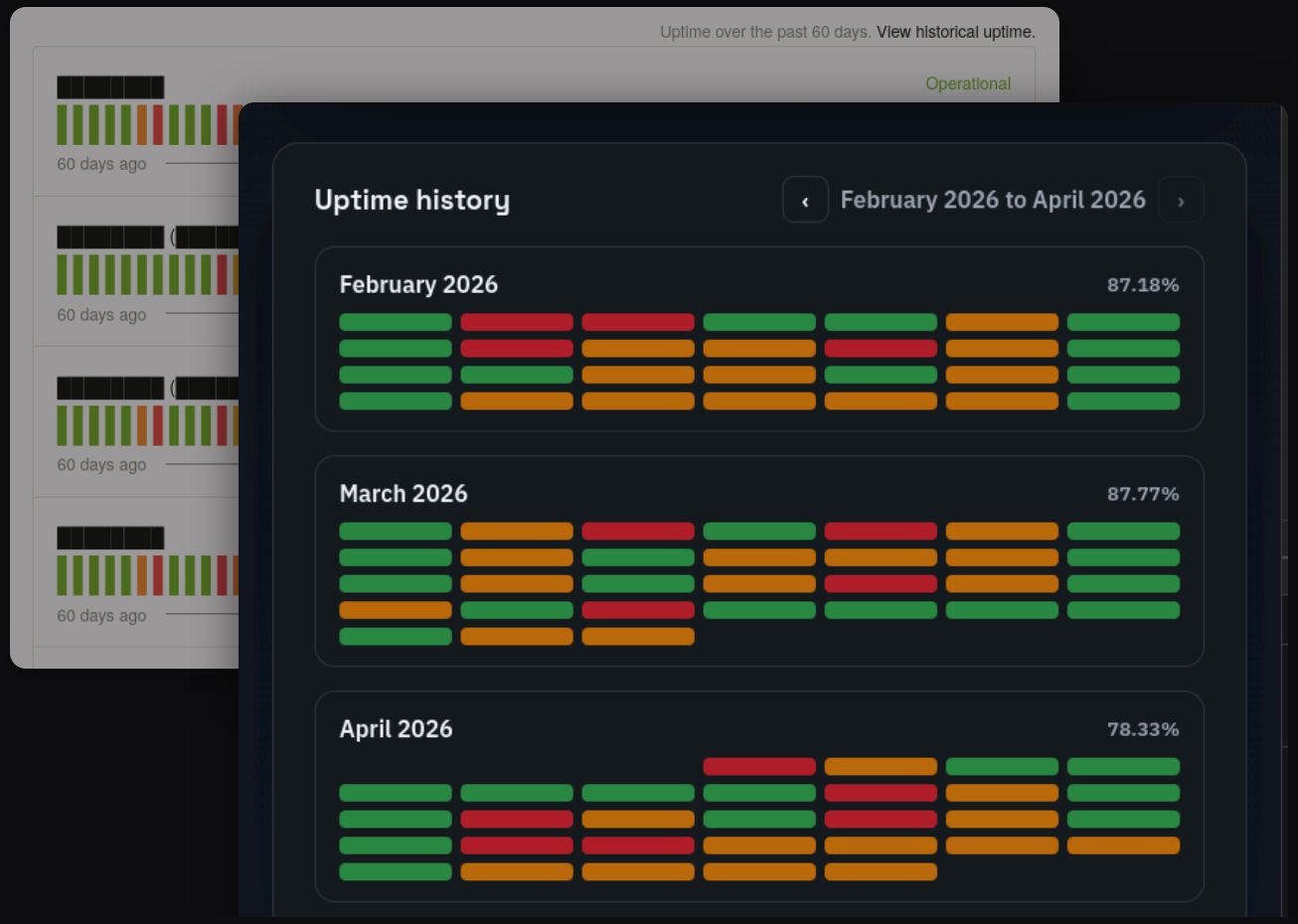
Your Agent Is Fast. Your Verification Loop Isn't.

Verification debt & agentic development

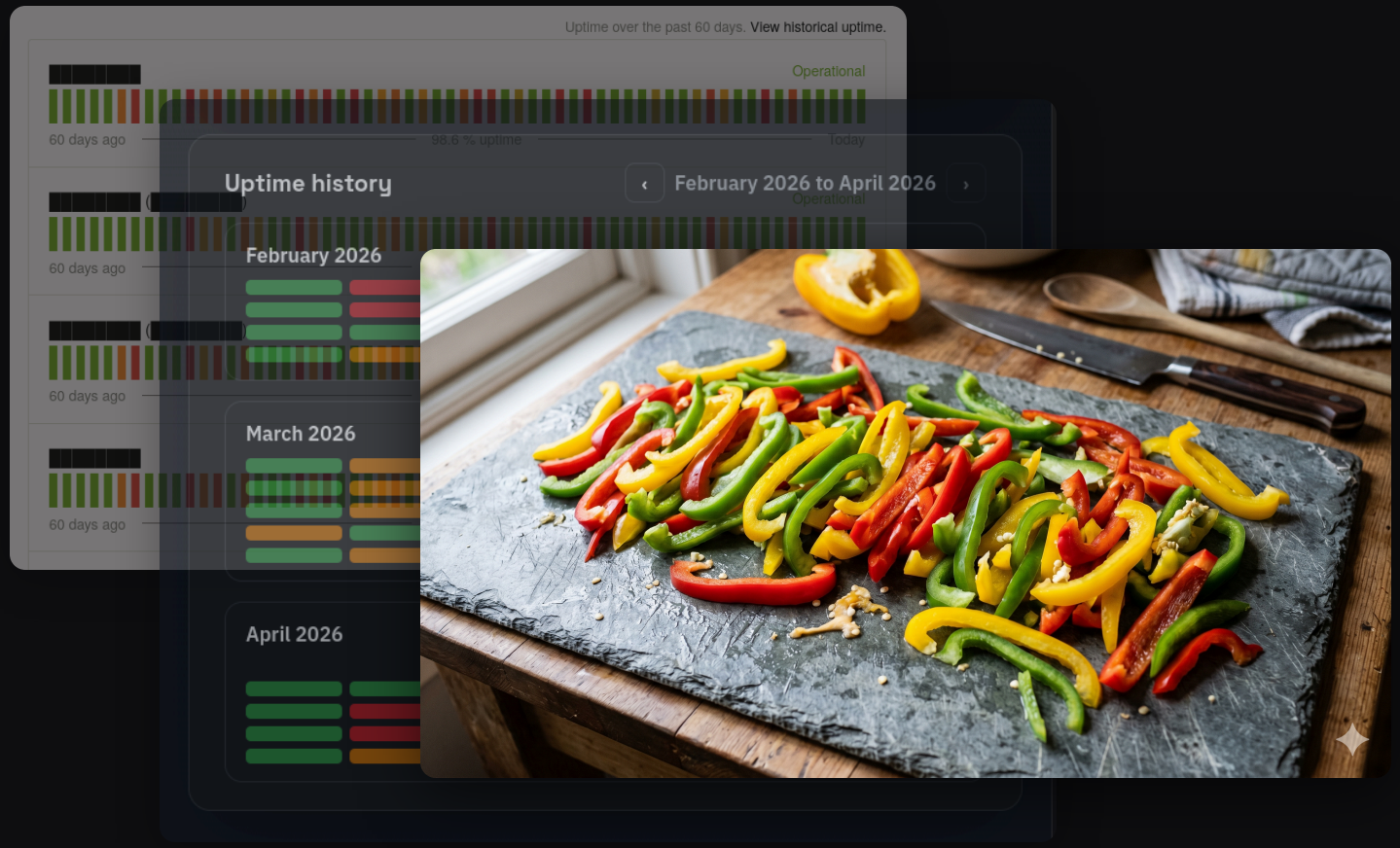
> Cliff Ressel · CTO, Nori · Agentics NYC · 2026-05-06

A trend in our industry

A trend in our industry



A trend in our industry



A bug we shipped

THE ASYMMETRY

You spotted that bug in five seconds.

Every model or harness looking at this code *couldn't diagnose or reproduce* it.

Unit tests. Integration tests. User input about the architecture. Access to the protocol docs and the underlying agent adapter codebase. Nothing was enough.

Reproducing it required *being there* — pressing a key mid-stream, in the right state, watching a TUI render in real time.

LIVE DEMO

Let's kick off a session while we talk.

> **Remember this prompt.** We'll come back to it.

Quick intro

> **Cliff Ressel** — CTO, Nori
> We build dev tools for agentic engineering, and we daily-drive three production codebases primarily with agents.

OUR WORK

Nori Sessions — Off-the-shelf runtimes for your background agents.

OPEN SOURCE

`github.com/tilework-tech/`**`nori-cli`** multi-provider agent TUI

`github.com/tilework-tech/`**`nori-skillsets`** organization context mgmt

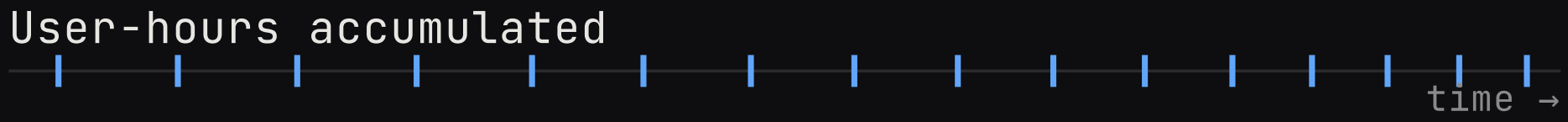
THE THESIS

Think hard about *how you verify your product.*

Your agent can verify the **code** it produces.
Your agent cannot verify the **product** the code produces —
until you enable that.

Traditional pace of software

> pre-agents baseline

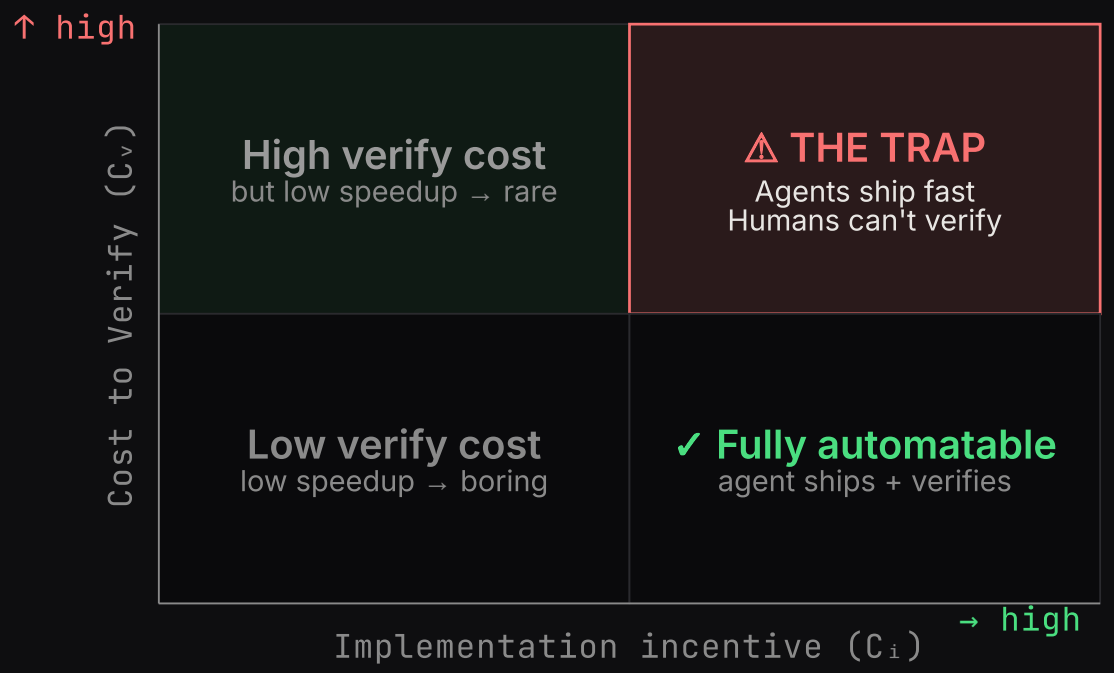


After agents arrived

> one curve compressed. The other didn't.

Where humans can't beat agents

> Cost to Implement (C_i) ↓ · Cost to Verify (C_v) flat



VERIFICATION DEBT

Tech debt buys speed today for the price of **optionality** in the future.

Verification debt buys speed today for the price of **quality** in the future.

If you're not investing in verification, you're shipping a larger and larger backlog of **unverified changes**.

The simple solution

`> close the loop — write verification recipes next to the code`

Give your agents concrete instructions on how to use and verify your product.
Not vague guidance — **runnable recipes** that end with an observable signal.

Critically, these instructions must live **next to the code itself** — in your repo's `AGENTS.md` or `CLAUDE.md`. This is the last section in all of ours.

> **The pattern:** After every change, the agent must build → launch → drive the product through a real interaction → assert a specific external observable (a glyph on screen, an HTTP 200, a Playwright selector). If the assertion fails, the agent keeps iterating. The verification signal is a real artifact, not the model claiming it works.

What does our practice look like?

This is not just:

1. **E2E testing.**
2. **Test-driven development.**
3. **Faster delivery.**

Just E2E testing with token costs?

E2E testing is how you do this, but letting the agent use the product is more general purpose.

Imagine if your engineers couldn't use the product. After every code change and each PR, there's a Slack message to the product owner:

> "Code is done. Can you try this for the first time?"

Engineers direct agents through most code changes, sometimes in parallel.
That's the team you have today.

So I have to do agent TDD now?

> **No.** TDD is a sequenced workstyle.
> This is just the **leverage point for velocity**.

The question to ask before any agentic task: **what shape would verification take here?**

If you can't answer that — that's the signal to slow down. Go human-in-the-loop on the implementation, execute your verification at the end, and invest a little time in the verification capability itself, so the **next** task in this area can run hands-off.

> **The verification capability compounds.** One slow-and-careful implementation doesn't.

Is this just faster delivery?

Some jaded engineers think coding agents are just a faster way to write the same code.
Why would I push features faster through the same broken process?

ASKING THE AGENT TO *USE* THE PRODUCT, THEN ASKING:

- **"How did it feel to use that new form?"**
- **"How convenient was it to click through this layout?"**
- **"Was the error message useful?"**

> ...is a **completely different capability**, not a difference of speed.
> No automation framework does this. Only the agent or human can.

Verification debt compounds

> the speedup turns into a slowdown

Every un-verified PR raises the verification cost of the next one. Bad patterns get committed, cargo-culted, embedded as "how this codebase does things."

Higher C_v → higher future C_i → the agentic *speedup* becomes an agentic *slowdown*.

METR (Jul 2025): experienced OSS devs measured **19% slower** with AI tools than without.

metr.org/blog/2025-07-10-early-2025-ai-experienced-os-dev-study/

If this is going to turn into a measurable speedup instead of a slowdown, there needs to be some amount of verification effort built in.

The harness

```
> give the agent a terminal
```

The convenience here is kicking off work like this asynchronously. With a background runtime, the agent has the capabilities to figure out any messy parts of using this product.

NORI SESSION

- Ephemeral cloud dev VM
- Coding agent inside
- Real TTY, real tmux
- Real screen capture

TUI PUPPETEERING

- tmux-backed repro
- Send keystrokes mid-stream
- Capture frames at moment X
- Diff text + image output

CLOSE THE LOOP

Remember the escape-cancel bug?

Watch what happens when we follow careful instructions for how to verify behaviors within the product itself.

What the session did

> see the product like a user

1. **Checkout** a branch without the escape-cancel fix.
2. **Drive** the TUI puppeteering harness — tmux-backed repro of the bug.
3. **Capture** obvious references of the failure moment.
4. **Summarize** the failure + relevant log excerpts.

[LIVE SESSION OUTPUT]

failure capture · agent's one-paragraph summary · log excerpt

backup →

RESULTS?

In under ten minutes, it found our stubborn bug —

... which no model or agent could pinpoint previously through code alone.

If this problem sounds familiar

```
> try this out tomorrow
```

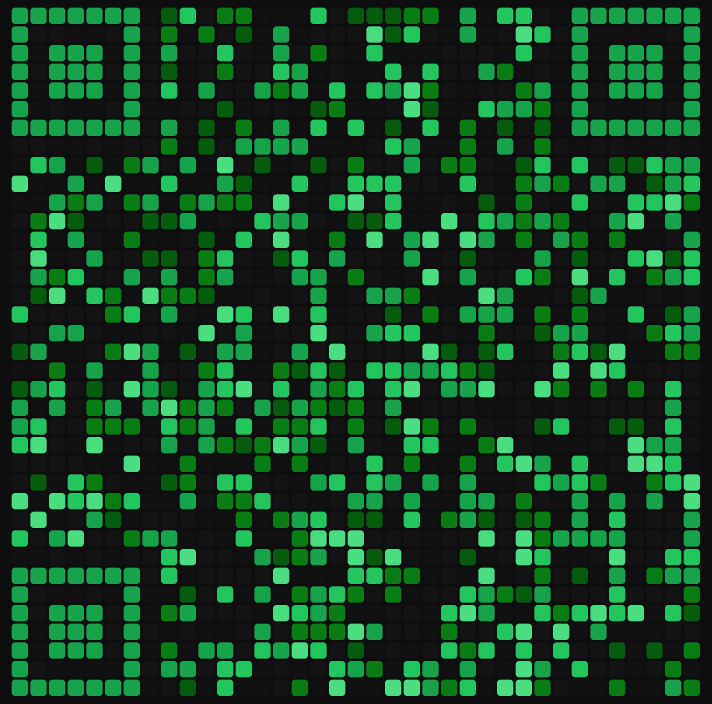
And don't overcomplicate it!

- Think about how you use your product
- Provide the agent with tools it was trained on
- Make all outputs legible or plain text

> No company or service or library can sell you:
> **The proper way to think about how customers use your product.**

One thing to try tomorrow

> resources: slides · skills · open-source examples



TALK NOTES

noriagentic.com/talks/2026-05-06-verification.html

- Slide deck (these slides)
- Skill for webapps: `webapp-testing`
- Skill for TUIs: `tui-puppeteering-with-tmux`
- "Close the loop" example: `nori-cli/AGENTS.md`
- Related reading (OpenAI, Datadog, Ladybird)

AND LASTLY

Longer-horizon work happens best with an asynchronous agent — one that keeps working while you're away from your terminal.

norisessions.com

We're building agent runtimes that help your agent use your tools to finish your work.

Think hard about how you verify your product.

The teams that do so will build faster.

> thanks · clifford@noriagentic.com