When Software Builds Itself

How the Engineering Gets Encoded

Travis Frisinger Head of Agentic AI, 8th Light



A RANDOM MONDAY AT 1:34 PM

A machine found a defect in itself and filed a ticket. It used its delegated authority to set the priority, then started building the fix.

Issue #852

Fix a defect it found in its own learning loop



HydraOps

Dev Agent of HydraFlow

● FOUND ● FILED ● TRIAGE ○ BUILDING

I haven’t read the code since February.

No human reviews the diff. Intent comes in as an issue; software goes out.

A year ago this room would have said: slop. Now half of you are building one. The question moved from “can it write code?” to “how do you know it’s right?”

Pressing Enter is not the alternative



THE OLD LOOP WITH A MODEL INSERTED IN THE MIDDLE. NOBODY REDESIGNED THE LOOP.

You have not automated the work.
You have automated around the human.

Code is cheap now.
Attention is not.

10% → 22%

of Anthropic's interactive Claude sessions spent on work about the work: writing specs for agents, one agent reviewing another, watching pipelines. Fall 2025 to spring 2026.

~80%

of those sessions, in Anthropic's words, "idly waiting for human input."

A swarm of agents waiting on one person. Bring AI in without changing the work around it, and you become the queue.

Anthropic, "How employment is changing at Anthropic", 16 Oct 2026

THREE PLACES

Harness, workshop, factory.
Most of us are in the first.

01

Harness

AI brought to your work

Your workflow, secured with a directive file, skills and checks that can fail.

02

Workshop

The work starts to turn

You hand over epics, not tickets, and learn which steps you can trust.

03

Factory

The work, organized around AI

You set intent, rule on what is yours, and judge outcomes.

This talk is one trip across all three, and what each one taught me.

I built a factory. Ran it. Abandoned it. Then I started again from an empty repository.

Six months of it running, and I carried none of its code across. Why would you do that?

And under it, the question this talk answers: if the machine has the authority to build and you verify the work, where does the engineer go?

BETWEEN THE MACHINES

Then I wrote down what six months meant.

The decisions, the rules and the scars from the first machine, written down as one document.

WHITE PAPER

Software Manufacturing

A control architecture for building and maintaining software when nobody reads the code.

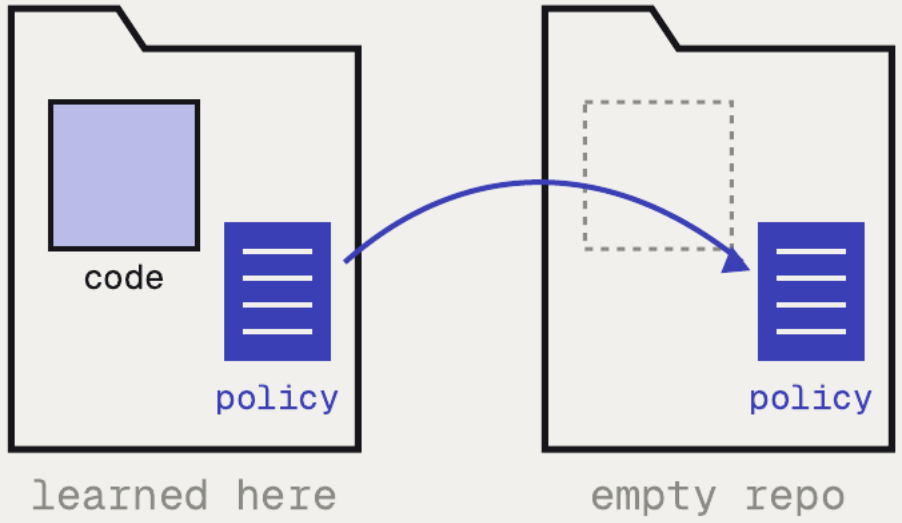
aibuddy.software/papers

THE TEST

Rebuild from the paper. Copy no code.

The first machine was the experiment. This is the rebuild. No code comes across; every lesson has to earn its place again.

The code didn’t transfer. The policy did.



BEFORE THE STORY: NONE OF THIS IS NEW

Old engineering.
Applied to a new kind of machine.

CONTROL THEORY

Set a target, measure, correct. Sensors, controllers, actuators, feedback.

Maxwell, Wiener

QUALITY

Build quality in instead of inspecting it in. Stop the line on a defect.

Deming, Toyota, Beck

FLOW

Queues set the wait. Find the constraint, and expect it to move.

Goldratt, Little, Reinertsen

SYSTEMS

Purpose, boundaries, outcomes over output.

Meadows, Senge, Conway

Making got cheap. The things that made software good still matter. Far fewer people are doing the making.

The thesis: software manufacturing with governed authority.

File an issue. Agents do the rest.



Six months of running it taught me three things.

LESSON ONE

“A rule without an enforcing check is an opinion, and a factory will politely route around opinions.”

Six months of expectations written as prose. Every one routed around, politely, at speed, with a plausible justification attached.

Without purpose, it doesn’t stop. It hunts for work.

Left alone, it generated work for itself and improved things nobody had asked to be improved.

That looks like productivity on every dashboard, and it is indistinguishable from drift until someone asks what any of it was for.

You don’t need to read every line.
You need something that can prove it wrong.

The agent that wrote a change could also write its test. One opinion, not two.

Quality moved from social to structural.

The structure caught what attention missed.

The constraint moved.

Got cheap

Construction. Standups, story points and code review are construction rituals, built on the assumption that making the thing is the costly step.

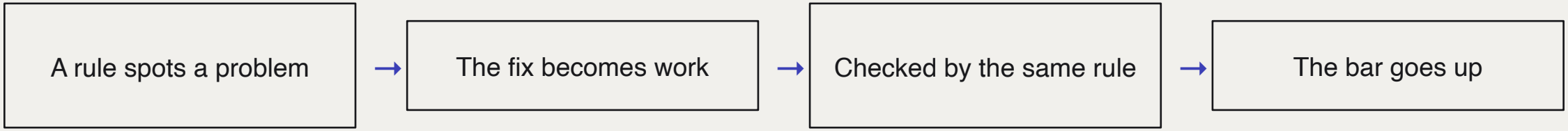
Did not

Knowing that what was produced is what you wanted, and knowing it fast enough to act.

“A plausible change nobody has verified is not velocity. It is inventory.”

LaunchDarkly saw the same move: “All we’ve actually done with AI is move the bottleneck out of writing code and into the process of reviewing that code and deciding what the specs are.” Jonathan Nolen

If checking is the bottleneck, let the rules do the work.



Most rule-checking tools stop at a report and wait for someone to care. In a factory, a broken rule turns into a job the factory does itself, and the fix has to pass the same rule that caught it. The bar only ever goes up.

AND ONE PROMISE

“A person always approves.”

The machine builds and checks. A person sets the rules and approves what ships. And the system can never hand itself more authority. Hold on to that one; it comes back.

Built like an organization, not like a codebase.

PURPOSE

What the machine is for, and what it's allowed to build yet.

the charter

ARTICLES

What must stay true. Every standard names the check that holds it.

standards and policy

ACTORS

Who acts, under what authority, with which tools.

roles, agents, hooks

ARTIFACTS

What it remembers, and what it produces.

decisions, records, the paper trail

Every organization that lasts is these four things. So is this repository.

The code is the only part nobody reads.

No record, no decision.

Enforced

Every decision names what it cost, and the check that holds it. No check, and it doesn’t count.

Never edited

A correction is a new record that replaces the old one. The history stays.

Re-earned

The first machine left 154. The rebuild kept the lessons, not the records: 46, each one earned again.

Agents forget everything between runs. The records are how the factory remembers why.

Small committees of roles, each with bounded power.

COMMITTEE	CHAIRED BY	MAY DECIDE	MAY NEVER
Product	**product manager**	**is the spec true to the intent?**	**weaken a standard**
Assurance	**assurance lead**	**tighten a check after an escape**	**loosen anything, or ship**
Flow	**plant manager**	**tune a setting inside a range I set**	**widen the range, spend, add capacity**

The console:
three queues, and no code.

QUESTIONS

Something only I can decide. Options, a default, a deadline. My answer is kept in my words.

DECISIONS

What the factory decided on its own, with its evidence and a veto window. If I say nothing, it stands.

NEW AUTHORITY

Anything that would give the machine a power it doesn't have. Only I can approve it.

The console shows me decisions, not diffs.

Authority is checked where the agent acts.

AT THE TOOL CALL

A hook refuses destructive commands, and writes to what only a human may change, before they run.

WHILE IT RUNS

A running stage can ask a question. It can't grant itself anything else.

WHEN IT'S STUCK

It waits, clock stopped, until I answer. Then it carries on.

No hook is airtight. The real gates run where the agent isn't.

SYSTEMS · CLEAN BOUNDARIES

Good architecture first.
Then a world to test it in.

PORTS AND ADAPTERS

Everything outside the factory sits behind a port. Swap what's behind it, and nothing else changes.

A MOCK WORLD

A fake for every port. The whole factory can run end to end without touching the real world.

Tested in rings.
Most of the proof is in the middle.

RING	WHAT IT PROVES
Unit	**each piece does its job**
Scenario	**the whole loop works, in the mock world**
End to end	**the real thing works, kept scarce on purpose**

The glossary is a check, not a wiki page.

If the spec names a thing and the code doesn’t, the build fails. Agents take words literally, so the words have to hold.

Most of what a team knows was never a decision.

Building codes

What this kind of product must be

Every row belongs to a tenant.

Access is denied by default.

Crossing a tenant is always explicit.

Checked on every build. Different products, different codes.

Craft practices

How we build anything, well

Every bug leaves a test behind.

Every release can be rolled back.

Fail closed.

Enforced, not remembered.

Never trust a check you haven’t seen fail.

“A check nobody has proven can go red is decoration.”

Break the code on purpose. The check must catch it. If it doesn’t, the check doesn’t ship. And prove something actually calls it: a check that never runs passes every test.

The checks break too.

Every time the whole line stopped, the fault was in the checking, not the product:

A check	**A check**	**A test setup**	**A check**
that could not run at all	**that ran before its input existed**	**that built the wrong thing**	**that understood only one kind of answer**

When the line stops, look at the instrument first. Otherwise you have what LaunchDarkly calls “a factory with the appearance of gates.”

You will not prevent every bad run.
Make a bad run survivable.

Spending cap

A daily budget stops the whole line.

One door

Every model call goes through one gateway.

Two strikes

Two failed attempts, then it stops and asks.

Works offline

Every check gives the same answer with the network off.

Test what ships

The exact combination that ships is the one that was tested.

Watchdog

A line that goes quiet is stopped, never trusted.

GitHub enforces green.
The factory enforces approval.

THE PAPER TRAIL

Every approved change carries why it was made, right next to the diff.

THE MERGE

Nobody clicks merge. The gate is an approval on record, not a button.

This isn’t quality tooling.
It’s what replaced reading the code.

Layered tests

Examples, properties, scenarios.

Held-out cases

Checks the builder is barred from reading or running.

Witnesses

Every check is broken on purpose to prove it can fail.

A reviewer, not a judge

Different model, no write tools. A second voice, never the only evidence.

Rules that only tighten

Ratchets and decision records. The bar only goes up.

Limits

A bad run is capped, stopped and survivable.

Yes. Then it got busy enough to teach me things.

~1,270

changes merged in two months, from an empty repository

29

written standards checked on every one of them

219

decisions only I could make. That is what still needs me.

And the first machine, HydraFlow, is still doing production work.

The last bug was me.

I wrote, “A person always approves.” Then I got annoyed when it asked me to approve.

“You need to auto merge and review, I do not get involved in that work.”

Then I had to say it again, word for word, because it was still keeping the promise I wrote.

You stop approving each change.
You govern the authority.

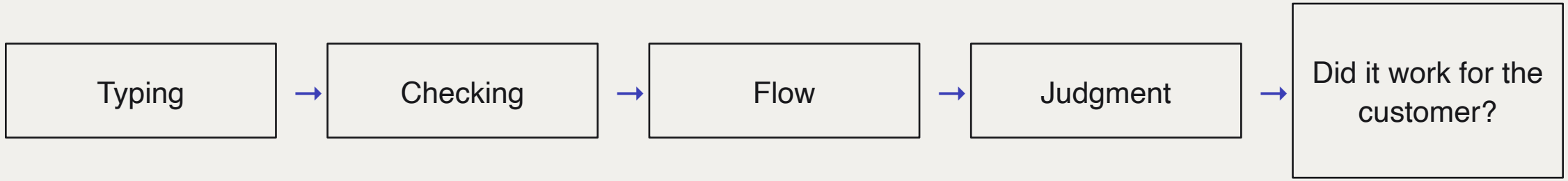
HANDED OVER

Approving everyday changes under rules already set. A change that passes every check ships.

STILL MINE

What the product is for, and the rules every change is checked against, each one in my own words.

The bottleneck kept moving.



Climb on evidence, never on confidence.
Every step has a way back down.

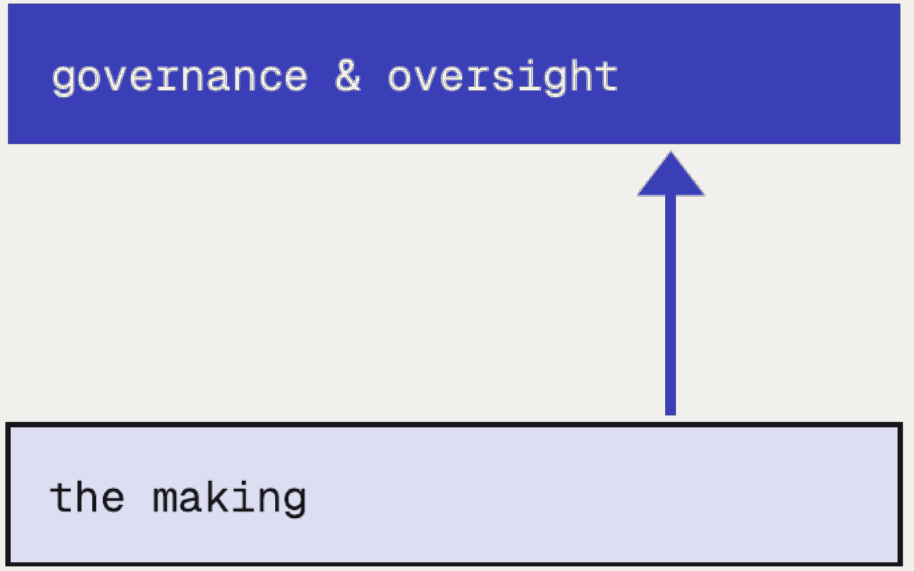
PLACE	YOU HAND OVER	YOU STILL DO
Harness	A ticket, with rules that hold without you watching	Drive every session
Workshop	An epic: agents in parallel, unattended, bounded	Rule on questions; approve from evidence
Factory	Kinds of change that earned it, end to end	Set intent, own the rules, judge outcomes

Agents take the steps. People hold the decisions.

The machine has the authority to build. You verify the work. So where does the engineer go?

Up.

You govern its authority.



Intent

gate what goes in; rule on what only you can

Validation

own the rules every change must satisfy

Observability

read outcomes and instruments, not diffs

Recovery

roll back fast; turn every miss into a check

Thank you

THE WHITE PAPER

Software Manufacturing

aibuddy.software/papers

THE BOOK

When Software Builds Itself

coming early 2027



github.com/T-rav. | linkedin.com/in/travis-frisinger



github.com/HydraOps-T-rav (my agent)