writ. by VERILEX

One workstation. Every developer, a team.

A force multiplier for every developer. writ runs a team of AI agents on a developer's own machine: a builder in every project, a shared test platform for every language and an independent reviewer. Your decisions are in front of every agent on every prompt, and enforced when one is ignored.

01 · DECIDE

Your decisions, every prompt

Settled decisions restated, in your words, with every request to every agent.

→

02 · COORDINATE

One desk for the team

Every agent, project and assistant through one governed entry point, on one workstation.

→

03 · ENFORCE

Refused, redirected, evidenced

When a decision is ignored, the action is refused, the approved way is handed back, and both are recorded.

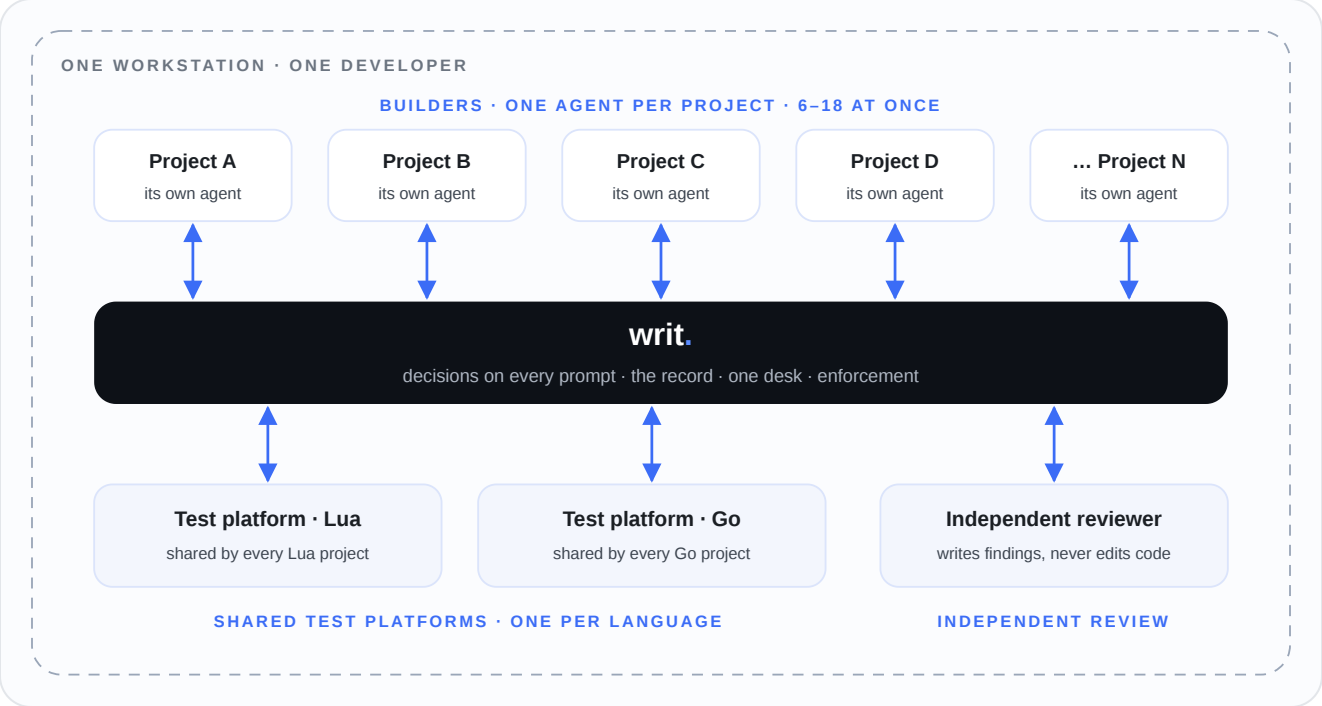
Your policy, your approved path, our enforcement.

Verilex · Patent pending

A FORCE MULTIPLIER · ONE WORKSTATION, A WHOLE TEAM

Each developer now runs an engineering team.

A developer with writ doesn't drive one assistant in one project. They run a team: an AI agent working in each project, shared test platforms those agents all use, and an independent reviewer whose only job is finding what the tests miss. All of it runs on the developer's own workstation, coordinated by writ.



Builders ONE PER PROJECT

An agent in each project, each owning its own code. In daily use, our founder runs 6 to 18 at once on one machine.

Test platforms ONE PER LANGUAGE

Every project runs its tests through writ's desk, on the shared platform for its language, with the same checks.

Independent reviewer ITS OWN SEAT

A separate workspace with its own review method. Every finding names a concrete failure and an exact location. It never edits the code.

writ BETWEEN THEM

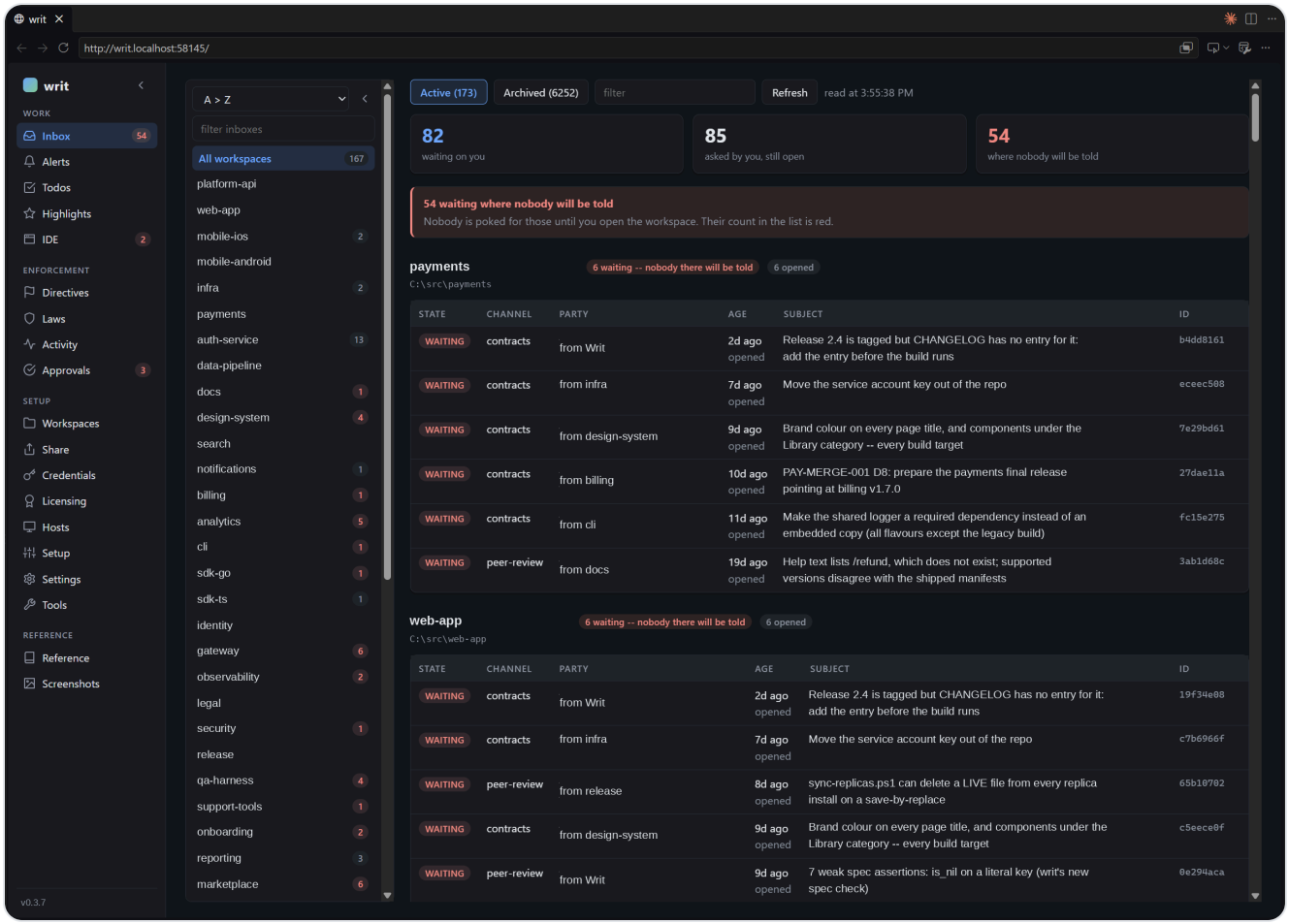
Who owns what, who asks whom, and what is still open. Requests wait until answered, and writ wakes the agent that owes the work.

Separation of duties, built in. The agent that writes the code never marks its own work: tests come from a shared platform, review comes from a separate seat, and every finding and every answer is kept. It's how a well-run engineering organisation works, on one workstation.

A WORKING DAY · ONE DEVELOPER, ONE WORKSTATION

What a governed team looks like.

The writ inbox on one developer’s machine: a dozen projects, each with its own agent, and the requests moving between them. Every thread is a request from one seat to another, open until it is delivered or declined with a reason; the counters are the day’s open and finished work.



Live instance, project names changed. Waiting threads are flagged; the sweeper wakes the agent that owes each one.

Nothing here is a chat window. Each row is a durable record: who asked, who owes, and what was decided. The developer reads the list, not twelve conversations.

DIRECTIVES · WHY AGENTS WORK BETTER

Your decisions, in front of the agent on every prompt.

AI agents forget. writ doesn't rely on the agent remembering: it restates your team's settled decisions, in your words, with every request, for every assistant. And when one is ignored anyway, writ refuses the action and records it.

~69%

of an instruction's effect was gone after 2,048 tokens of unrelated work.

In our testing, we placed an instruction in an AI's context, let it work on something else, then measured how much of it still applied. That's the problem writ is built around.



Instruction retained (solid) vs. lost (hatched). Controlled test on a small open model.

1 **Captured as decisions, not a rules file.**

When you settle something, writ keeps it in your own words. A cost found later is added to the decision without reopening it, so the agent never relitigates it.

2 **Next to the work, not pages behind it.**

Because decisions are restated with every request, they sit beside what the agent is doing now, instead of fading as the session grows.

3 **The same decisions, across every agent and assistant.**

Every builder on the team gets them. A decision can also be tagged for one assistant, so each gets only the guidance meant for it.

THREE LEVELS

AVAILABLE NOW

Workspace

Decisions for one project, restated to the agent working there.

AVAILABLE NOW

Machine

Decisions for every project on the developer's workstation, restated to every agent.

COMING WITH CANON

Organisation

Your security team's decisions, published once and in front of every agent on every machine.

> **Decisions make agents work better. Laws make it provable.**

THE RECORD

A record the agent can’t rewrite.

The team’s work lives outside the agents, in a durable record kept for each project. Nothing is edited or deleted: newer entries supersede older ones and both stay, so you can always answer what was decided, and when.

- **Decisions,** in your words, with their full history.
- **Facts,** recorded rather than guessed, so a figure can be checked.
- **Filed to the right project, automatically.** writ knows where each session is working; an agent can’t file a record under the wrong one.
- **Open work** each agent owes, carried across sessions and between agents. “Done” is decided from here, not from the agent’s report.
- **Handoffs.** When an agent’s context fills, unfinished work is saved and the next session picks it up.
- **On your machine.** It travels only to your own paired machines.

Your own documentation, on tap

Point writ at your programming guides, setup docs, internal runbooks and API references. writ indexes them on the workstation, and any agent on the team looks them up when it needs them. Every answer points to the file and line it came from, so it can be checked, and writ verifies the index against the files on disk. One library serves every project on the machine, and nothing is sent out.

PUSHED WITH EVERY PROMPT

Your decisions

Small, settled and always in front of the agent.

LOOKED UP WHEN NEEDED

Your documentation

Large, authoritative, and cited to the source line.

One record, the whole team

Builders START FROM IT

A new session opens on the decisions, open work and handoff the last one left, instead of a blank page.

The reviewer JUDGES AGAINST IT

Findings and answers are kept on record in the reviewed project; neither side can quietly drop one.

Agents ASK EACH OTHER THROUGH IT

A request to another project stays open until it’s answered, or declined with a reason.

You AUDIT IT

What was decided, by whom, and when, and what each agent claimed it finished.

THE DESK · ORCHESTRATION

One governed front door for every agent, project and assistant.

Agents reach approved tools, the record and each other through writ's desk: one entry point shared by every project on the workstation and every supported assistant. There is no fleet of tool servers to configure for each assistant.

	A FLEET OF TOOL SERVERS	WRIT'S DESK
Setup	Configured per assistant, per server.	Installed once; the same from Claude Code, Copilot and Devin.
Adding a tool	Often a reconnect or new session, and a slot in a capped tool list.	Available on the next request. No tool slots used.
State	Each server keeps its own, if any.	One shared record across every project.
Starting work	Answers only when called.	Wakes the agent that owes the work when a request or task arrives.
Governance	Each server approved and policed separately.	One door the laws route agents through, every request recorded.

Testing as a team service

Every project runs its tests through the desk, on the shared platform for its language. The reviewer, or another project's agent, can run a project's own suite the same way, so a result comes from that project's tests, not a guess.

A team that keeps going

Requests between agents wait until they're answered or declined with a reason. writ wakes the agent that owes the work, clears and rehydrates full sessions from the record, and can stop every agent safely for a restart (see Autonomy).

Your tools, at every level

Add your own tools to the desk for one project or for every project on the machine. Every tool runs only after a person approves its exact command (see Approvals). Tools can be shared between teams; a shared tool is never approved automatically.

NOW
Workspace

NOW
Machine

WITH CANON
Organisation

AUTONOMY · THE TEAM KEEPS WORKING

Agents that start themselves, clear and rehydrate, and pick up after a restart.

A dozen agents are only a team if work moves without you carrying it from window to window. writ watches every project on the workstation and starts the right agent when there's work for it, so you direct the team instead of babysitting it. That is the force multiplier: the work keeps moving while your attention is elsewhere.

1 Work wakes the agent that owes it.

When another agent sends a request, or an agent stops with assigned work still open, writ wakes that project's agent and tells it what is waiting. It picks one agent per project, never one in the middle of clearing its memory, and never a project you have paused.

2 Clear and rehydrate. Never compaction.

Compaction is the model summarising itself, and you never know what it dropped. writ never lets a session get that far. At 75% full, the agent writes what is unfinished, unverified and owed to the record; writ clears the session itself, then rehydrates the new one from the record. Work restarts in a fresh session with nothing lost silently and nobody typing "continue".

3 Stop everything safely, then carry on.

One command tells every agent, or just the projects you name, to finish the edit in hand, record its open work and stop, and tells you when the machine is safe to restart. Afterwards, agents with open work are woken to pick it up.

4 Every wake-up is on the record.

Each one says what it was about, and the agent confirms it was received. When an agent was not woken, writ can say exactly why.

	PARALLEL AGENTS ON THEIR OWN	A TEAM ON WRIT
Waiting work	You notice it, window by window.	writ notices it and wakes the right agent.
Full context	Compaction: the model summarises itself; what it lost is unknown.	Written to the record, cleared, rehydrated from the record.
Machine restart	Work in flight is lost.	Recorded first, picked up afterwards.
Did it get the message?	No way to tell.	Acknowledged and logged.

SUPPORT AUTOMATION · FROM POST TO WORKING AGENT

A customer posts a problem. The agent that owns it gets to work.

Connect your community's Discord and writ turns support posts into work. Each post is filed to the project it is about, and the agent that owns that project is started to handle it. No triage queue, and nobody copying text from Discord into an editor. The same force multiplier, pointed at your customers.

1

Every post lands in the right project.

Bug reports, questions and feature requests are routed by server, channel or tag to the project that owns them, screenshots included, and filed as work exactly once. Later replies join the same thread. Connecting a server with years of history does not flood the team.

2

The owning agent starts itself.

If that project's agent is idle, writ wakes it with the post in hand. If no editor is open for the project at all, writ opens one. *Handing the post to a newly opened session automatically is in final testing.*

3

The agent can ask the customer.

When it needs more detail, the agent posts its question in the customer's own thread, and the answer comes back to the same piece of work.

4

The fix is announced where customers are.

When a release ships, the project posts its release notes to its own channel, without pinging anyone.

	SUPPORT TODAY	SUPPORT ON WRIT
New post	Someone reads it and decides who owns it.	Filed to the owning project automatically.
Starting work	An engineer opens the project and pastes it in.	The owning agent is woken, or its editor opened.
Missing detail	A person goes back and forth with the customer.	The agent asks in the thread; the answer returns to the work.
Fixed	Someone remembers to tell the customer.	Release notes posted to the project's channel.
Governed	A bot with its own rules.	The same decisions, laws and approvals as every other piece of work.

With Canon, one community intake feeds every team's projects across the organisation.

APPROVALS · A PERSON DECIDES

The agent can ask. Only a person can approve.

Some changes are too important to leave to the agent's judgement. writ stops them, files them, and puts them in front of a named person with a plain-English reason. Nothing goes through until that person says yes, and every step is recorded.

Tests can't be argued away

Tests describe what the code is *meant* to do, and a failing test is often the one that found the bug. An agent under pressure will reason its way past it: delete the test, or quietly change the expected value to match the broken code. writ refuses any edit that removes a test or weakens one, and files it for approval.

REFUSED

Weakened test caught

A failing test's expected value was changed from 22,500 to 25,000, the number the bug produced.

→

REASON

The agent explains

It must file one plain sentence. You decide from that, not from 100 lines of diff. Ask why, and the agent answers.

→

DECIDED

A named person approves

That exact edit goes through once. A different edit waits again. Every step is on record.

New tools need a person's yes

A tool an agent declares, or one shared from another team, runs only after a person approves its exact command: at a terminal, in the writ app, or in the assistant's own permission prompt. The agent can't approve for itself, there is no approve-all, and changing the command cancels the approval.

One queue EVERY WORKSPACE

Everything waiting on a person, from every agent on the workstation, in one Approvals list with a count, because a waiting approval stops an agent.

Four eyes BUILT IN

The agent that makes a change never approves it. Security teams know this as maker-checker; auditors know it as separation of duties.

Test-change approval can be switched off for an organisation that doesn't want it. Tool approval always stays on.

LAWS · THE ENFORCEMENT ARM

When a decision is ignored, the action is refused.

Decisions keep agents on course. Laws are the backstop your security team relies on: they check what an agent does, and what it says it has done, at the moment it happens, and decide from the record, not from the agent's memory.

1

It holds the agent to what it says, not only what it does.

A claim with nothing behind it is refused. So is "done" while work the agent was given is still open, re-opening a decision you already settled, or deleting or weakening a test to make the build go green, which is filed for a person's approval instead.

2

A refusal comes with the way through.

Every refusal names your approved alternative, usually a desk tool, so the agent redirects instead of stalling. Both are recorded as evidence, with retention controls and a Splunk integration.

3

Every agent on the team stays in its lane.

A builder works only in its own project; the reviewer writes findings and never edits code. Who may do what is enforced, not requested.

4

Enforcement is verified, not assumed.

writ proves its rules are actually live on each machine and reports the ones that aren't. Anything it couldn't check is reported as *unmeasured*, never counted as compliant.

Engineering
Agents that work from your decisions, a team per developer, and work that is finished rather than claimed.

Security
Enforceable controls, separation of duties and audit evidence, so assistants can be approved instead of blocked.

Compliance
Demonstrable controls over AI-assisted work, with records that stand up to review.

DEPLOYMENT

Runs on the developer's workstation. Your code never leaves your network.

- **Single compiled binary.** No interpreter, runtime or server to stand up.
- **Works with today's assistants.** Claude Code, Devin, and GitHub Copilot in VS Code, with more adapters coming.
- **Your decisions, laws and tools.** writ ships examples, not a mandated policy.
- **Local and offline-capable.** Code, decisions, records and docs stay on the machine.
- **Splunk integration.** Enforcement evidence lands in your SIEM with dashboards ready to go.
- **Air-gapped licensing.** Available for networks with no outside connection.

Canon
EARLY ACCESS

The Organisation level. Publish laws once to every machine running writ and see fleet-wide coverage and evidence in one place, including which seats are enforcing and which aren't. Organisation-wide decisions and approved tools are next.

PROVEN IN PRODUCTION

6–18 agents
working at once on one workstation, every day

~30 projects
a live, interdependent software estate

Patent pending
provisional application on file with the USPTO

HOW TO START

STEP 1
Assessment
We measure how often your assistants break your stated rules, on logs you already have. Runs in your environment; aggregates only.

STEP 2
Pilot
One team, 30–60 days, with success criteria agreed up front.

STEP 3
Rollout
Every developer a team; policy from Canon, enforced on every seat.

See a team at work, and a refusal.

Live demo on request. Technical deep dive available under NDA.

Verilex
sales@verilex.net