

FOR CTOS AND ENGINEERING LEADERS

Verification Engineering

Writing code got cheap. Knowing it works did not. How teams prove every change is safe to ship when agents write the code.

INSIDE

- What verification engineering is
- Why the bottleneck moved
- The five parts of a verification system
- Verification vs end-to-end testing
- Verification autonomy levels
- Who owns what
- Verification in the outer loop

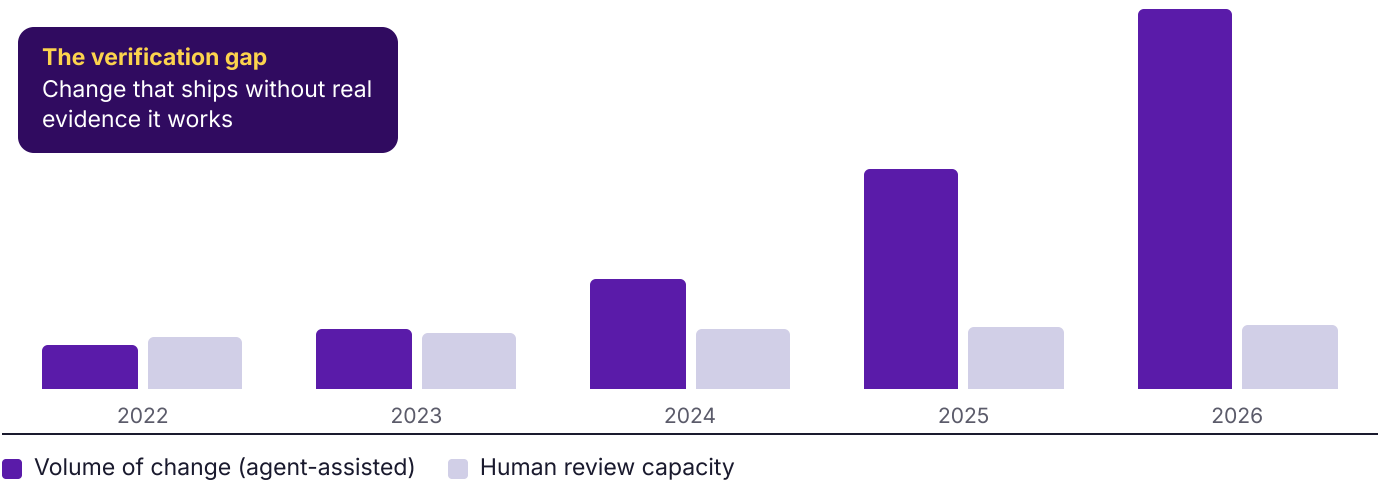
PR #4182

opened by a coding agent

✓	Map the journeys	01
✓	Check every change	02
✓	Verify in production	03
✓	Attach evidence	04
✓	Loop to the fix	05
Safe to ship		Merge →

Writing code got cheap. Knowing it works did not.

A coding agent can open a pull request in minutes. Deciding whether that pull request is safe to merge still takes a person reading the diff, running the app, and hoping the tests cover what matters. As agents write a growing share of the code, the amount of change outgrows what people can review by reading it.



Illustrative. Review capacity grows with headcount; change volume grows with agents.

Verification engineering is the work of closing that gap. It decides what “works” means for a product, checks it automatically on every change, keeps checking after release, and sends what breaks back to whoever, or whatever, writes the fix.

OPTION A

Slow the agents down

Cap output at the speed of human review. You pay for agents and get human velocity.

OPTION B

Merge without knowing

Ship at agent speed and let users find the regressions.

OPTION C

Verify at agent speed

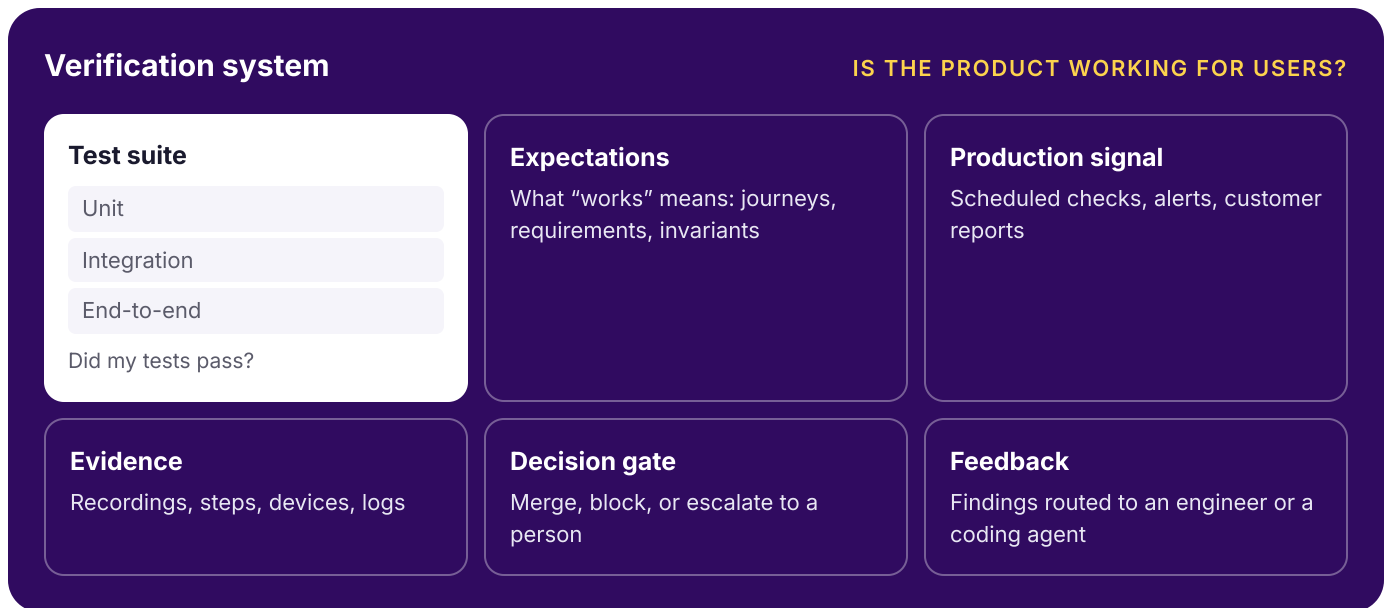
Build a system that produces evidence as fast as agents produce change.

THE THESIS

Teams that can verify a change quickly can ship it quickly. Verification, not code generation, now sets the speed limit.

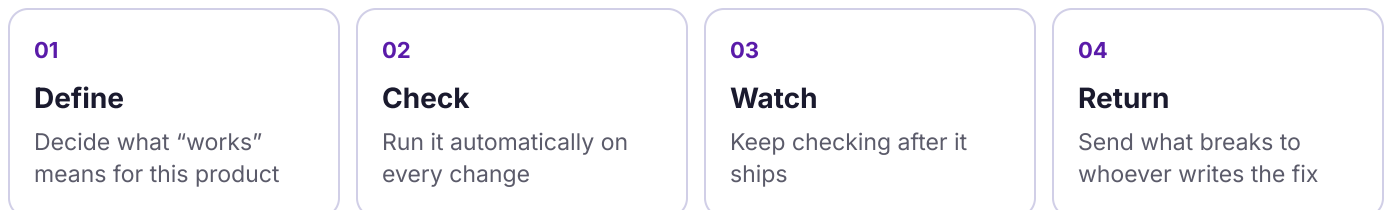
A test suite is one artifact. Verification is the whole system.

Verification engineering is the practice of proving that software does what it should, on every change, before and after it ships. It is broader than testing in the usual sense. Tests are one input. The system around them is what tells you, with evidence, whether a change is safe and whether what is already live still works.



Tests sit inside the system. The system owns the question that matters.

Four jobs, every change

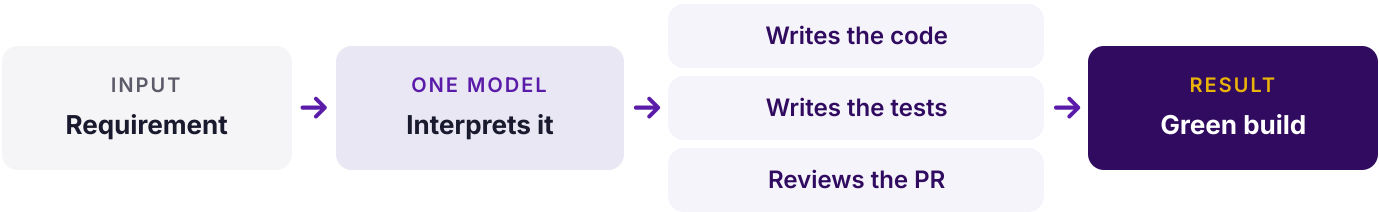


NOT THE SAME AS QA

QA is often a stage at the end of a release. Verification engineering builds the checks into every change, so most of that work is automated and happens before merge.

When one model does everything, a green build proves very little.

When engineers wrote every line, code review and a handful of tests kept pace. Two independent readings of the requirement met in CI: one in the code, one in the tests. With agents, the same model often interprets the ticket, writes the code, writes the tests, and reviews the PR. If it misreads the requirement, everything passes.



The self-confirming loop. A green build confirms the model’s reading, not the product requirement.

180% → 30%

A 2026 study of more than 100,000 GitHub developers found agents raised commit activity by 180%, but only 30% at actual releases.

12,000+

Unit tests in one fully agent-written codebase. The suite stayed green while real defects reached users.

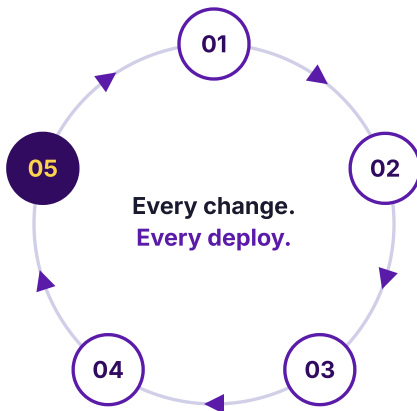
What changes for the outer loop

	BEFORE AGENTS	WITH AGENTS
Bottleneck	Writing the code	Proving the code works
Independent check	A second engineer	Must be designed in
Review capacity	Matched the rate of change	Outpaced by it
Cost of a missed defect	One bad PR	Repeated across every agent run

Source: “Beyond Code Generation: Reliability, Verification, and Cost Economics in the Agentic Software Development Lifecycle,” arXiv 2609.04681.

Five parts that turn change into evidence.

A verification system is a loop, not a stage. It starts from what users actually do, checks every change against it, keeps checking in production, and hands back findings precise enough to fix without a meeting.



The loop runs before merge and after deploy.

01 Map what users do

The journeys that matter, such as signing up, checking out, or inviting a teammate, on the devices real users have.

02 Check every change

Run each pull request against those journeys and compare with production, so a regression is caught before merge.

03 Verify in production

Re-run the same journeys on a schedule, so a bad deploy or broken dependency is found by the team, not its users.

04 Attach evidence

The recording, the steps, and the device, so a failure can be understood without reproducing it by hand.

05 Loop back to the fix

Findings precise enough to hand to an engineer, or to a coding agent as a prompt.

Where each part runs

Before merge

Preview build of every PR

Parts 01 · 02 · 04

After deploy

Production, on a schedule

Parts 03 · 04

Always

Findings into the fix loop

Parts 05

EVIDENCE, NOT A VERDICT

A red check tells you something broke. A recording, the steps, and the device tell you what broke and where, so nobody has to reproduce it by hand.

Not “did my tests pass?” but “does the product still work?”

Hand-written end-to-end tests cover what someone thought to write down. They break when the interface changes and fall behind the product. Plain-English test tools make each test easier to write, but someone still has to describe every one. Verification engineering starts from the journeys users actually take, not only the ones someone listed.

HAND-WRITTEN E2E COVERAGE

Only the journeys someone wrote down, and only until the UI changes.

DISCOVERED JOURNEY COVERAGE

Journeys mapped from how the product is actually used, kept current as it changes.

	END-TO-END TESTING	VERIFICATION ENGINEERING
Core question	Did my tests pass?	Does the product work for its users?
Coverage comes from	Whatever someone wrote down	Journeys discovered from the product
When the UI changes	Tests break and need repair	Journeys are re-mapped by agents
Runs	In CI, before merge	Before merge and in production
Output	Pass or fail	Verdict plus recording, steps, device, fix prompt

CODE VERIFICATION

Does the system match developer intent?

PRODUCT VERIFICATION

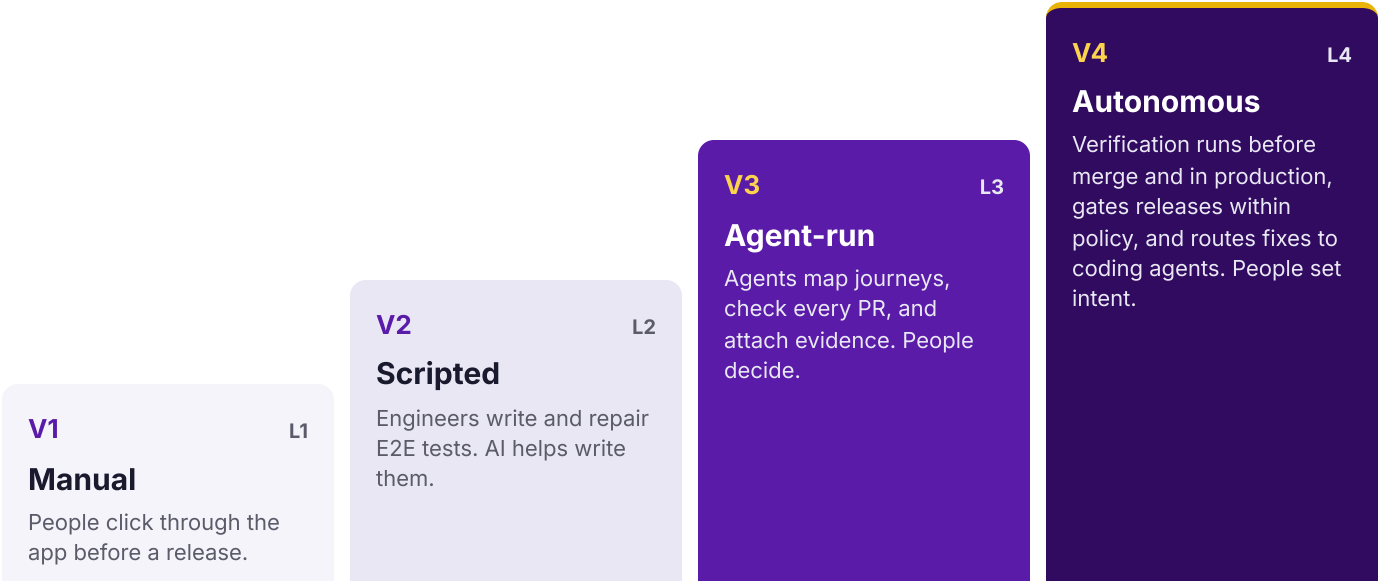
Does it deliver the right capabilities?

UNIT TESTS STAY

Unit tests check code in isolation and still matter wherever correctness is fixed by rules: cryptography, pricing and tax, compliance formulas, permission matrices. Most teams keep both.

Verification climbs the same four levels as the rest of the company.

Our AI transformation model describes four levels, from AI as a thought partner to an AI Native Operational System. Verification follows the same path. A team cannot run agents at Level 4 while its verification is still at Level 2; the slower layer sets the pace for everything above it.



V-levels for verification, mapped to the L-levels of the AI transformation model.

LEVEL	WHO CHECKS	WHEN	WHAT A PERSON DOES
V1 · L1	People	Before a release	Everything
V2 · L2	Scripts written by people	In CI	Write, repair, and triage tests
V3 · L3	Agents, with evidence	Every PR, before merge	Review findings and approve
V4 · L4	Agents, within policy	Every PR and in production	Set intent, policies, and expected outcomes

WHERE MOST TEAMS ARE Most teams sit at V2: AI helps write tests, but people still maintain them and read every diff. Moving to V3 is what lets agent output actually reach production.

Agents do the checking. People own what “correct” means.

Agents are good at the mechanics of verification and should do as much of it as possible. What they should not do is decide what the right answer is. If the agent that wrote the code can also rewrite the expectations, verification turns back into self-confirmation.

- 01

Source expectations from outside the loop
Derive journeys and acceptance criteria from PRDs, user research, support tickets, and domain experts, before the agent starts coding.
- 02

Let agents maintain the mechanics
Selectors, fixtures, devices, environments, and execution plumbing are exactly the upkeep agents now handle in minutes.
- 03

Gate expectation changes behind people
An agent should never rewrite an expected outcome to turn a check green. Changing what “correct” means is a product decision.

Who owns what

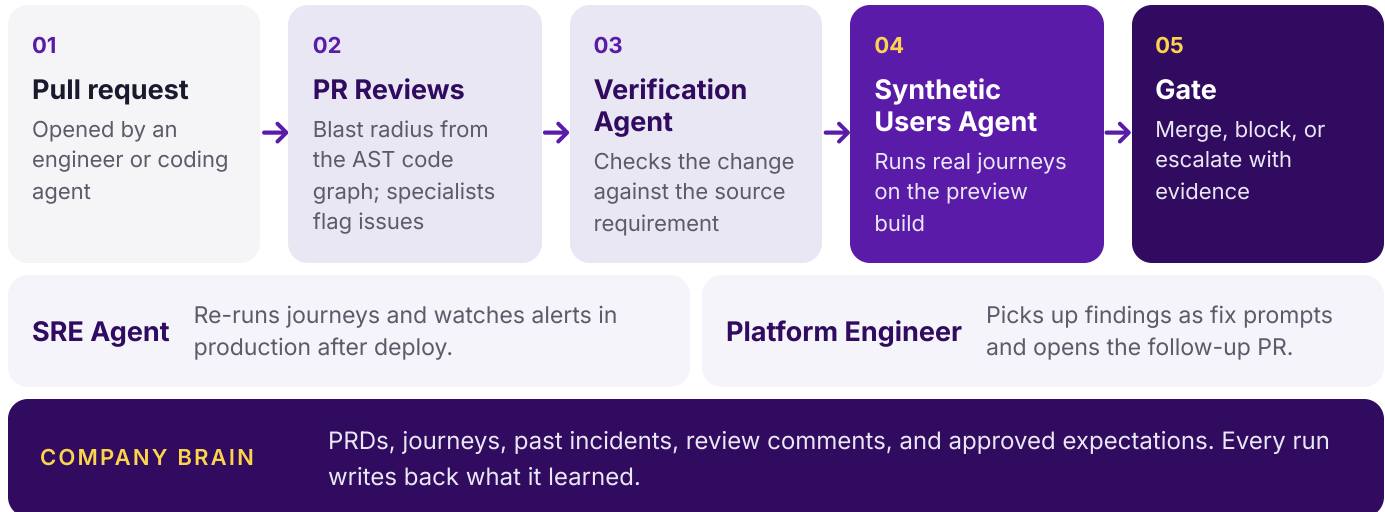
AGENTS MAY CHANGE FREELY Selectors and page objects Test data and fixtures Devices, retries, and timing Environments and CI wiring	PEOPLE MUST APPROVE Expected outcomes and assertions Which journeys are covered Removing or skipping a check Release acceptance criteria
---	---

NO DEDICATED TEAM REQUIRED

Verification can be owned by the engineers who ship the code. Automating it is what lets it keep pace with change, whoever or whatever wrote it.

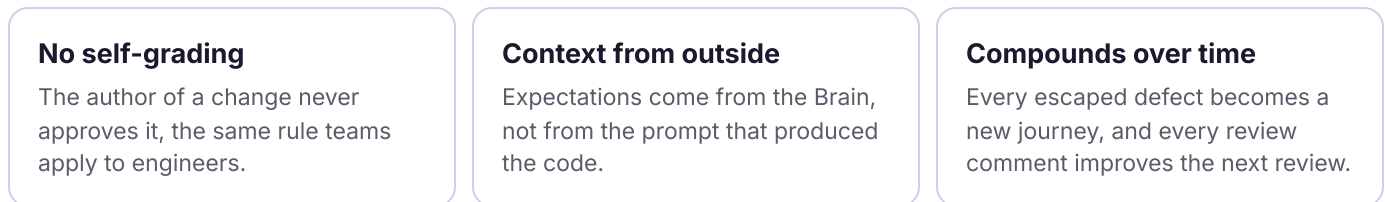
How XHawk runs verification engineering.

XHawk's AI employees run each part of the verification system as a separate role, so the agent that wrote the change is never the one that approves it. All of them read from and write to the same Company Brain.



Each role has its own objective and acceptance criteria. Humans sit at the gate and own expectations.

Why separate roles matter



START WITH ONE LOOP

Turn on PR Reviews or the Verification Agent on one repository. Add production checks once the first loop is producing evidence you trust.

QUESTIONS FOR THE CTO

- 01 How long does it take your team to know a PR is safe to merge?
- 02 Is the agent that writes your code also the one that approves it?
- 03 Which user journeys would break today without anyone noticing for a day?
- 04 Can a finding in production reach a coding agent as a fix prompt without a meeting?

**Code generation sets
how fast you can write.**
**Verification sets how
fast you can ship.**

**Ready to verify at agent speed? Talk to the
XHawk team.**

AI Employees for Product, Engineering & Analytics.

xhawk.ai →