

ENGINEERING THOUGHT LEADERSHIP · OCTOBER 2026

AI writes the code. Verification is now 10x more important.

A green build can still ship the wrong product. Why agent-written code needs product verification in the Outer Loop.

 CI STATUS**All tests pass****≠**

CUSTOMER

**Product works as
intended**

INSIDE

The blind spot

The testing pyramid

Code vs product verification

Governance

The ACC model

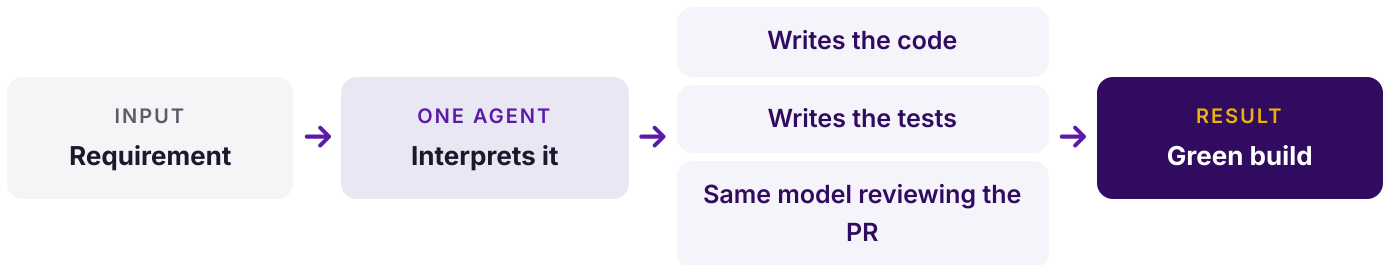
Case study: Swamp

The playbook

The agent wrote the code and the tests.

Both can encode the same misunderstanding. A green build confirms the agent's reading, not the product requirement.

When a person writes code and a different person writes the tests, two independent readings of the requirement meet in CI. When one agent does both, there is only one reading. If the agent misreads the ticket, it writes code that does the wrong thing and tests that assert the wrong thing. Everything passes.



The self-confirming loop: no independent check on what the requirement actually meant.

A single model driving the full SDLC is not a smart idea.

Why this matters now

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.

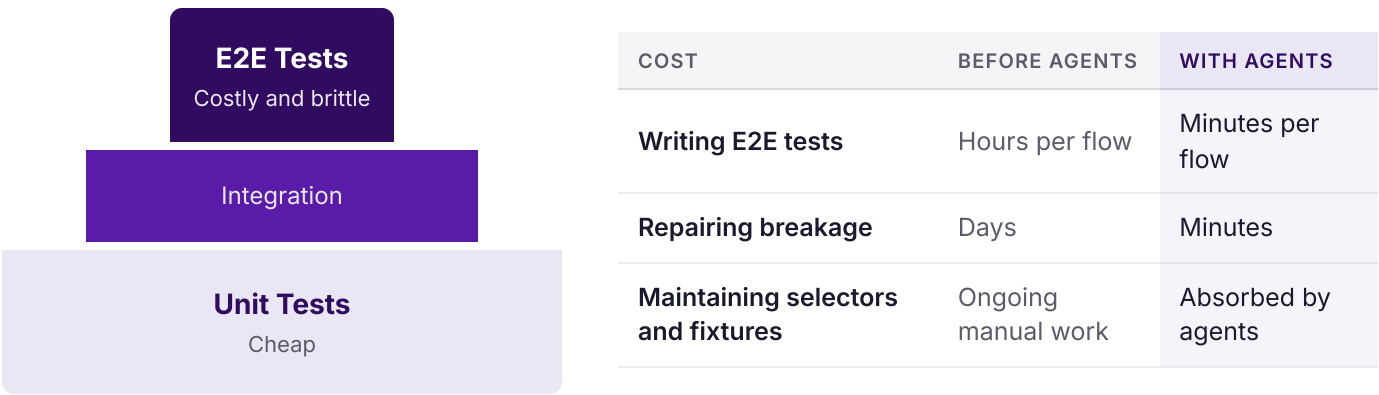
The path to green is hidden

An agent can fail, change the code, and retry several times before CI passes. Most pipelines keep only the final result, not the attempts that led there.

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

The testing pyramid was an economic bet.

Teams wrote many cheap unit tests and few end-to-end tests because people had to write and repair every one of them. End-to-end tests broke whenever a selector, a fixture, or a timing changed, and fixing them took days. The pyramid shape was a cost decision, not a statement about where defects live.



Agents now absorb much of the upkeep; repair cycles shrink from days to minutes.

Is it time to rethink what "Software Verification" means in the age of AI?

THE TEMPTING ANSWER

Inverting the pyramid is not the fix.

Piling on end-to-end tests keeps you inside the same code-verification frame. If an agent generates the workflow assertions, it can still rubber-stamp its own mistaken interpretation at scale. More tests from the same source add cost without adding an independent check.

Where expectations come from is what matters.

The question is not how many tests you have or which layer they run at. It is who decided what “correct” means. If that decision comes from the same place as the code, the test cannot catch a misunderstanding.

CODE VERIFICATION

Does the system match developer intent?

Tests syntax, internal state, and whether the code does what the author told it to do.

PRODUCT VERIFICATION

Does it deliver the right capabilities?

Tests business outcomes, user workflows, and real customer value outside the author's loop.

	CODE VERIFICATION	PRODUCT VERIFICATION
Expectations come from	The author of the code	PRDs, user research, domain experts
Tests the system as	White box, from the inside	Black box, as a user would
Catches	Regressions, logic errors, broken contracts	Misread requirements, broken workflows, missing capabilities
Blind to	Whether the intent was right	Low-level edge cases inside a function
Can the coding agent change it?	Yes	No. Only through human review

THE PRINCIPLE

Verification has to live in constraints the implementation agent cannot simply rewrite.

Keep expectations out of the agent's reach.

Agents are good at the mechanics of testing and should do as much of it as possible. What they should not do is decide what the right answer is. Three rules separate the two.

- 01

Source from outside the loop
Derive assertions directly from PRDs, user research, and domain experts. The expected outcome should exist before the agent starts coding, and it should come from someone who understands the customer.
- 02

Let agents maintain test harness mechanics
Automate selectors, fixtures, and execution plumbing freely. This is the upkeep that made end-to-end tests too expensive, and agents now handle it in minutes.
- 03

Gate assertion changes behind human review
Never allow an agent to rewrite expected outcomes to turn CI green. A change to what "correct" means is a product decision and needs a person to approve it.

Who owns what

AGENTS MAY CHANGE FREELY Selectors and page objects Test data and fixtures Retries, waits, and timing Execution environment and CI wiring	HUMANS MUST APPROVE Expected outcomes and assertions Which capabilities are covered Removing or skipping a check Acceptance criteria for a release
--	---

Not a new idea.

James Whittaker's ACC model at Google (Attributes, Components, Capabilities) framed testing at the product capability level. The industry favored low-level unit suites because ACC was labor-intensive to maintain manually. **LLM agents finally make capability-level verification practical to keep up to date.**

A
Attributes
The qualities the product promises: fast, secure, accurate, simple.

C
Components
The major parts of the system: cart, payments, accounts, search.

C
Capabilities
What a component does to deliver an attribute. Each one is a testable claim.

Example: a capability map for checkout

COMPONENT	ATTRIBUTE	CAPABILITY TO VERIFY
Cart	Accurate	A returning user sees the items they left in their cart
Payments	Secure	A declined card never creates an order
Pricing	Accurate	The total charged matches the total shown before payment
Accounts	Simple	A guest can complete checkout without creating an account

WHAT CHANGED
Humans write the capability statements. Agents keep the tests behind them running as the product changes.

Thousands of green unit tests, real defects shipped.

100%

Code is written by agents

12,000+

Unit tests in the suite

The unit suite stayed green while real defects reached users. Every test costs tokens to generate, compute to run, and attention to maintain. A test that does not add an independent constraint is overhead, not verification.

At Frontier companies, acceptance runs catch defects at the release boundary instead of the unit test suite.

07 · THE RIGHT ROLE

Unit tests still earn their place.

Unit tests remain indispensable wherever correctness is defined by external invariants:

Cryptography & hashing routines

Output is defined by the algorithm, not by product taste.

Pricing calculations & tax rules

A cent off is a real defect, and the rule is written down.

Compliance & statutory formulas

Regulators define the answer. Agents cannot reinterpret it.

Granular permission matrices

Every role and resource pair has one correct outcome.

When specification cannot drift with product opinion, unit tests excel.

Grow the suite from real product failures.

Define expectations outside the prompt loop. Lock assertion edits from agents. Add an external capability check every time a defect escapes.

- 01 Write capability statements first.** Turn each PRD into plain-language claims a user would recognize.
- 02 Keep them in a protected location.** The coding agent can read them but cannot edit them.
- 03 Verify with a separate agent.** Run black box acceptance checks against a deployed build, not the author's own tests.
- 04 Turn every escaped defect into a new check.** The suite grows from what actually broke for customers.

How XHawk verifies in the Outer Loop

Verification Agent

Checks every change against the source requirement, separate from the agent that wrote it.

Synthetic Users Agent

Simulates real users to test flows end to end and report what breaks.

Company Brain

Holds PRDs, research, and past decisions so expectations come from outside the prompt.

THE OUTER LOOP AUTONOMY STACK

Autonomous Software Verification is 10x more important now that AI is writing all the code.

XHawk is an AI workforce platform built on a shared company brain. Share this paper with your platform team.

xhawk.ai →