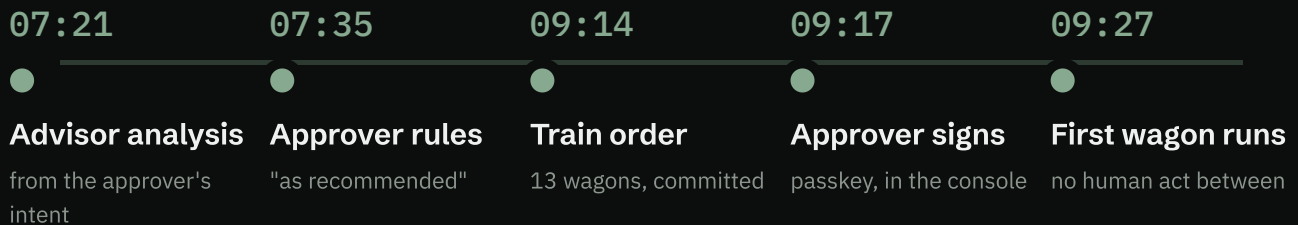


# Two sentences, two hours, a signed train.

A case study from the record: how AI agents planned a new Saphan Studio feature under one person's signed approval.



30 October 2026 · times in UTC · from file timestamps, commits and the console

**13**

wagons, each with a definition of done a test must prove

**1**

ruling and 1 passkey signature from the approver

**10 min**

from the signature to the first wagon running

# Contents

Executive summary

---

1. The story, in order

---

2. What the train builds, in plain words

---

3. For the product manager: who does what, and where the human comes in

---

4. For the architect: the moving parts

---

5. For the QA engineer: how "done" is defined

---

6. For the reviewer and the tester

---

7. Security and governance, from the same week

---

8. Evidence you can check yourself

---

9. Honest limits

---

10. How this case study was written

---

Appendix A. Excerpts from the train order

---

Appendix B. The master at work, screen by screen

---

Appendix C. Glossary

---

Written for five readers: the executive, the product manager, the architect, the QA engineer, and the reviewer or tester. Each has a chapter; the executive summary stands on its own.

# Executive summary

This case study comes from Saphan Studio running Saphan's own engineering. From here on, it speaks in roles.

On 30 October 2026, the approver, the engineering lead accountable for this work, wrote a few sentences of intent into a chat with the advisor agent:

"Since we are building mocks, Tests Studio and skills, let's do deterministic integrations with the other standard tools used in software development and CI/CD: a skill to run the tests, or to run Scout for security analysis, the OWASP tools. Let's go wild. Now it will be super easy."

About two hours after the advisor's first analysis, the approver read and signed a complete plan: thirteen work packages ("wagons"), each with a definition of done that a test must prove. Ten minutes after the signature, the first wagon was running on a build machine. On the way, the approver made one ruling ("as recommended") and pressed one passkey.

**Who approves, what enforces, what the auditor gets.** One accountable person, the approver, signs the plan once and approves every landing on the main line. Saphan Studio enforces the boundary in code: the budget, the list of work packages, the repositories, outbound network and landing. The auditor gets the record: every order, review, decision and signature, each one a commit.

This document follows that one feature from idea to running work, using only what the record holds: the orders, the messages between agents, the console screens and the commit history. It is written for five readers, and each has a chapter: the executive, the product manager, the architect, the QA engineer, and the reviewer or tester.

## What the case shows

| What                                                            | Evidence in this case                                                                                                                                                               |
|-----------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Speed without giving up control</b>                          | Analysis to signed plan in about two hours. Signature to first running work in ten minutes. The approver's own time: a few sentences, one ruling, a read-through and one signature. |
| <b>The agents work alone inside a boundary</b>                  | After the signature, the master agent signs its own work orders, places them on machines, sends fix rounds after reviews, and re-replaces work when a machine dies.                 |
| <b>The boundary is enforced by code, not by good intentions</b> | The budget, the list of wagons, the repositories, outbound network and landing on the main line are all checked by the system. An agent cannot talk its way past them.              |
| <b>Review is never the author's own</b>                         | Every sensitive wagon is reviewed by a different agent, and where the rules ask for it, by a different model family. Review strength follows the risk of the work, not its size.    |
| <b>Everything is in the record</b>                              | Every order, review, decision and signature is a commit in a git repository that anyone with access can read later.                                                                 |

| What                         | Evidence in this case                                                                                                                                                                                                                                                 |
|------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Bring your own agents</b> | Claude Code, Codex, Cursor's agent, Qwen Code and Droid run under it today, next to Saphan's own agent loop, which runs NVIDIA Nemotron. The coding runs on the agents a team already uses. Orchestration, governance, security and the record stay in Saphan Studio. |

**What the feature itself is.** The train in this case adds a library of the standard tools that teams already run in CI/CD (test runners, dependency and secret scanners, OWASP tools, SBOM and signing tools) as ready-to-use, pre-configured skills. An agent will no longer improvise a command for Trivy or pytest. It will ask Saphan Studio for the exact, pinned, tested way to run that tool on the repository in hand, run it, and get a normalised result. Adding a new tool to the library becomes one order to one agent, and a code gate decides whether the result is good enough to enter.

**Evidence you can check yourself.** Agents working under Saphan Studio built Proven, an open-source code reviewer, in 139 hours. The repository is public, and so is a report that compares the build with a conventional team. Chapter 8 tells you where to start if you want to judge the code yourself.

# 1. The story, in order

All times are UTC, taken from file timestamps, commits and the console.

| Time (UTC)   | Who      | What happened                                                                                                                                                                                                                   |
|--------------|----------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| ~06:15       | Approver | Rules on a queue of 14 open decisions from all masters in one sentence: "as recommended. Everything is now set up the way the customer will work." This becomes the overriding rule for every train.                            |
| Before 07:21 | Approver | Writes the idea (quoted above) to the advisor.                                                                                                                                                                                  |
| 07:21        | Advisor  | Writes an analysis: a first-cut catalogue of about 50 tools in 12 classes, a tiered design (deterministic first, an optional local model, the frontier model last), a proposed train and four decisions for the approver.       |
| 07:25        | Approver | Corrects it: "But the model is only an option. A deterministic invocation, so the agent knows how to run it. As a knowledge base in ASK."                                                                                       |
| 07:28        | Approver | Names the end state: "Once we put all this together and build a proper, tested wiring, adding something new will be launching an agent: 'analyse tool X and put it into the ASK library'."                                      |
| 07:28        | Advisor  | Writes the kickoff for a new master agent, with the approver's words, the binding rulings, what to read first and the first three acts.                                                                                         |
| ~07:35       | Approver | Accepts all four decisions: "as recommended".                                                                                                                                                                                   |
| ~09:09       | Master   | Starts. Declares its track, checks the current state of three repositories, reads the two neighbouring trains' orders.                                                                                                          |
| ~09:10       | Master   | Sends interface questions to two other master agents (the ASK knowledge base and Tests Studio), with a reply required. Gets both answers.                                                                                       |
| 09:14        | Master   | Commits the train order: 13 wagons, readings to refute, order of work, routing, definition of done, questions for the advisor. Checks the order's grammar and rehearses the approver's signature. Writes the approver's script. |
| 09:16        | Master   | Reports: "The train now waits on one script run by you and on three answers from other heads."                                                                                                                                  |
| 09:17:10     | Approver | Opens the order in the console, reads it, and signs with a passkey.                                                                                                                                                             |
| 09:27        | Master   | Verifies the signature again at placement and places the first wagon (W0, the conformance gate) on a Linux build machine with an admitted Claude Code seat.                                                                     |
| 09:29        | Master   | Sets a watcher on W0's run and waits for its return.                                                                                                                                                                            |

The master worked on the plan for about seven minutes of its own time. Most of the two hours was the gap between the kickoff and the moment the new master was seated and started work.

## 2. What the train builds, in plain words

---

**The problem.** Today, when an AI agent needs to run a test suite, a dependency scan or a secret scan, it composes the command from general knowledge, on whatever machine it is on, and then reads the raw log. Sometimes the command is wrong. Sometimes the tool reaches the network in a way nobody declared. Sometimes the agent spends a frontier model's attention on hundreds of lines of scanner output that did not change since yesterday.

**The idea.** Saphan Studio's rule for agents is "chores in the tools, thinking in the model". So the train turns the standard tools of software delivery into **deterministic skills**:

1. **An invocation catalogue, served by Saphan's knowledge base (ASK).** For each tool and ecosystem it holds the exact, pinned way to run it: the command template, the version and image digest, the inputs, the outbound network it needs (often none), the meaning of each exit code, and the output format. An agent asks `saphan ask tool trivy` and gets that record, with a receipt. An unknown tool is refused by name, with the list of tools that exist.
2. **A deterministic runner.** It runs the recorded command, normalises the result to SARIF (findings), JUnit XML (tests) or CycloneDX (software bill of materials), compares it with the last run, honours waivers, and reports the run to Tests Studio. The model sees only what changed.
3. **A conformance gate.** A record can enter the catalogue only by passing a gate written in code. The gate pulls the pinned tool, runs it on a small fixture repository with a known outcome, checks that the output is valid against the format's schema, checks the exit code, and checks that the only network the tool reached is the network the record declared.
4. **One order, one agent, one new tool.** Once the wiring is tested, adding a tool is a single order: "analyse tool X and add it to the ASK library". The agent writes the record and its skill file, and the gate decides.

**Why it matters for a customer.** Your teams already have their tools: their test frameworks, their scanners, their CI. This feature means Saphan's agents arrive with that knowledge built in and pre-configured. They run your tools the way your pipeline runs them, in your own CI and forge (GitHub Actions, GitLab CI or Gitea Actions emitters are part of the train), against mocks rather than production, and every run leaves a result you can compare and audit. It is not a new product. It is one more use case on the same Saphan Studio core: the same orders, the same boundaries, the same record.

**The first catalogue.** Tests (go test, pytest, jest/vitest, Maven/Gradle, dotnet test, cargo test); security (OSV-Scanner, Trivy, gitleaks, Semgrep CE); OWASP (ZAP against a Mocks Studio mock, Dependency-Check); supply chain (Syft, OpenSSF Scorecard, cosign); the first paid pack (Docker Scout). Open-source tools are free in the catalogue. Commercial tools come as paid packs, under the same rule as Mocks Studio.

**W0a — the invocation catalogue is an ASK source, and `saphan ask tool` answers from it (built with the saphan-ask master)**

- **lands:**
  - The tool-invocation record schema, versioned ( `tool-invocation/v1` ). One record per (tool, ecosystem):
    - `id`, `version` and `pin` (image digest or asset sha256), plus a `licence` class ( `osi` | `paid-pack` );
    - `command`, a template over TYPED inputs only ( `repo_root`, `config_path`, `output_path`, declared extras). No free text and no shell metacharacters; the template is refused on load otherwise;
    - `inputs` (paths read), `egress` (a host list, or `none`), `exit_map` (code → `clean` | `findings` | `tool-error` );
    - `output` ( `sarif-2.1.0` | `junit-xml` | `cyclonedx-1.5` | `json` ) and its path;
    - `admitted_by`, the conformance gate run that admitted it: fixture repository digest, output digest, egress observed.
  - The ingest path and the source contract, as the saphan-ask master answers R3. Its shape is fixed in the wagon order, never guessed here.
  - `saphan ask tool <id> [--ecosystem <e>]` compiles to the same typed request `ask resolve` takes, so there is one resolver. The answer prints the record and a receipt ref. An unknown id is refused, and the refusal prints the catalogue's closed list.
- **dod:**
  - (1) **Fails before:** on a fixture store holding one admitted `gitleaks` record, `saphan ask tool gitleaks` prints the pinned command, the egress `none` and the output path. On the base it answers `unknown command "tool"`.
  - (2) Negative control: a record whose `admitted_by` is empty is refused at ingest, and the field is named. A record whose command carries a shell metacharacter is refused, and the character is named.
  - (3) Unplug: drop the source registration and (1) reddens.
- **review:** named (a new source, schema and projection in ASK)

**W0 — the conformance gate: a record enters the catalogue only by passing it (engine)**

- **lands:**
  - `saphan tool admit <record>`. It pulls the pin and refuses on a digest mismatch, then runs the recorded command on the record's fixture repository with the egress wall scoped to the record's declared `egress`. It validates the output against the format's schema (SARIF 2.1.0 JSON Schema, the JUnit XML XSD, CycloneDX 1.5) and checks the exit code against `exit_map` for the fixture's known outcome. Last, it compares the hosts the wall observed with the declared list.
  - A pass writes `admitted_by` and hands the record to W0a's ingest. Any fail refuses with what was measured and names the field (law #55).
  - The gate run is itself reportable to Tests Studio as one run (the shape follows the tests-studio master's answer to message `dfa53c3c2828f2c6` ).

W0a and W0 in the signed order: the tool-invocation record schema, the definition of done with its negative controls, and the conformance gate that decides what enters the catalogue.

### 3. For the product manager: who does what, and where the human comes in

#### The roles

| Role            | Who                         | What it does in this case                                                                                                                                                |
|-----------------|-----------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Approver</b> | A person                    | Has the idea, rules on decisions, signs the train, approves every landing.                                                                                               |
| <b>Advisor</b>  | An AI agent                 | Turns the idea into an analysis with options and a recommendation, appoints the master, collects everything that waits on the approver into one list.                    |
| <b>Master</b>   | An AI agent, one per train  | Writes the train order, negotiates interfaces with other masters, signs the work orders inside its delegation, places them on machines, reads reviews, sends fix rounds. |
| <b>Executor</b> | AI agents on build machines | Do the work of one wagon, in its own worktree, and return with evidence.                                                                                                 |
| <b>Reviewer</b> | A different AI agent        | Reviews a wagon it did not write. For security work, from a different model family.                                                                                      |
| <b>Tester</b>   | An AI agent                 | One tester over the whole train at the end.                                                                                                                              |

#### How an idea becomes a plan

The approver does not write specifications. The approver writes what is wanted in plain words, and those words travel with the work. The advisor's analysis quotes them. The kickoff quotes them. The train order opens with them, verbatim and in the requester's language, in its first section, so the approver signs a document that starts with the approver's own sentences.

The advisor proposes and the approver corrects. In this case the approver corrected the advisor twice in seven minutes: first to make the local model optional, then to name the end state. Both corrections went straight into the kickoff.

#### Questions come with an answer attached

Saphan Studio has a policy for every question an agent asks: it must never be "A or B?". Every question to the approver has four parts:

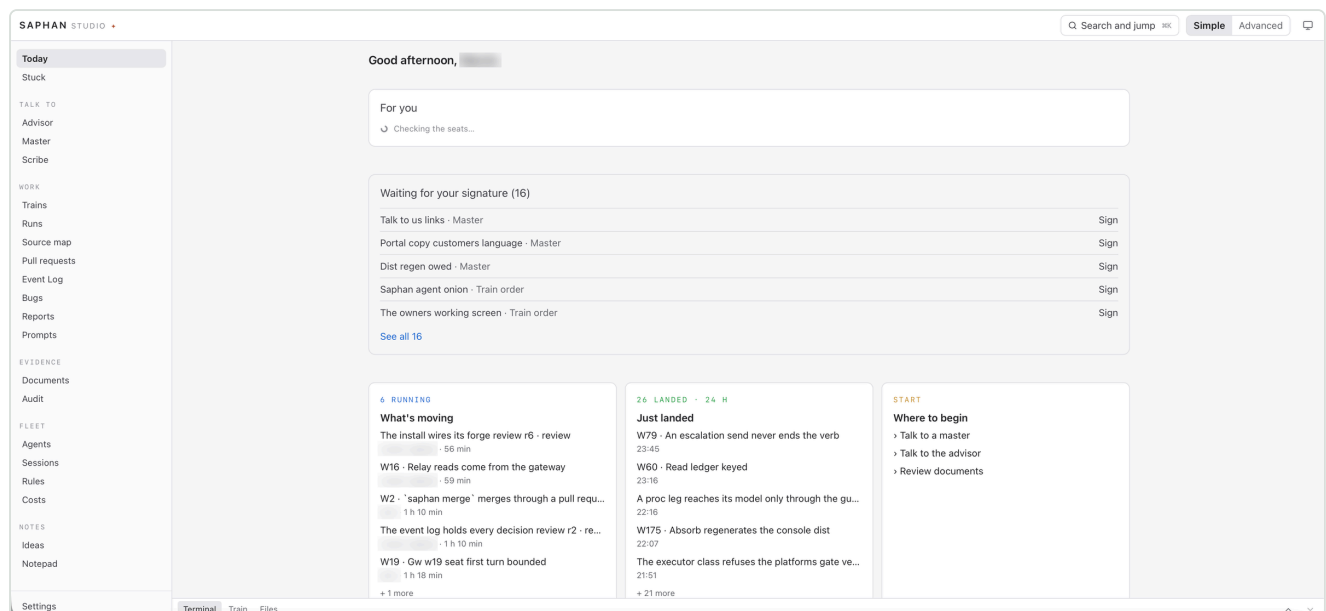
1. the problem, with its dated measurement;
2. each option, with what it costs and what it risks;
3. a named recommendation, and why;
4. what it costs if the recommendation is wrong.

That is why one sentence, "as recommended", can settle fourteen decisions at once without being a rubber stamp: each one arrived complete. A real one from that morning, shortened, with the product and the dates taken out:

*[a product train] — BLOCKED ON THE APPROVER since 13:18Z the day before. Decision: the scope of the first release. Measured: this train has had about one usable machine at a time; two wagons verified, the rest not started. Options: (a) keep the full scope, not reachable at this rate; (b) keep the full scope and move the date by three days; (c) release the core and the first examples on the date, the rest three days later. Recommendation: (c). Cost if wrong: the first customer sees fewer features.*

## What the approver signs

The approver starts from the console's Today screen. Everything that waits for a signature is in one list, next to what the agents are running and what landed in the last day. Nothing on it moves forward until the approver opens an item and signs it.



The console's Today screen the next day, 31 October: sixteen items waiting for the approver's signature (orders from masters and train orders, each with a "Sign" action), six runs in progress and twenty-six landings in the last 24 hours. The approver's name and machine names are masked.

## Sign orders

One touch, in your name.



### TRAIN ORDER — standard-tools-run-as-deterministic-skills

Train order

standard-tools-run-as-deterministic-skills

Round 1

sha256 matches

Details

Hide the order

This is the signed text — every byte, including link definitions and link addresses.

Text Source

A formatted reading aid, not the signed text: it leaves out link definitions and shows a link as its words only.

#### TRAIN ORDER — standard-tools-run-as-deterministic-skills

- master: MASTER, claude-code seat [REDACTED] (appointed by the ADVISOR's KICKOFF of 2026-10-30, on the owner's order)
- status: FOR THE OWNER'S SIGNATURE — written 2026-10-30; nothing dispatched
- track: standard-tools-run-as-deterministic-skills
- repo: saphan-studio-engine (W0a, W0, W0c, W4) · tests-studio (W0b, W1–W3, W5–W7 public runners and skills)
- base: saphan-studio-engine `develop @ 7e9159b36` · tests-studio `main @ 7dffb3` (the local refs are the fresh ones at home; `origin/develop` 4b933f808 and `origin/main` b166e3d are stale by construction; measured 2026-10-30)
- author: MASTER

#### §0 THE OWNER, VERBATIM

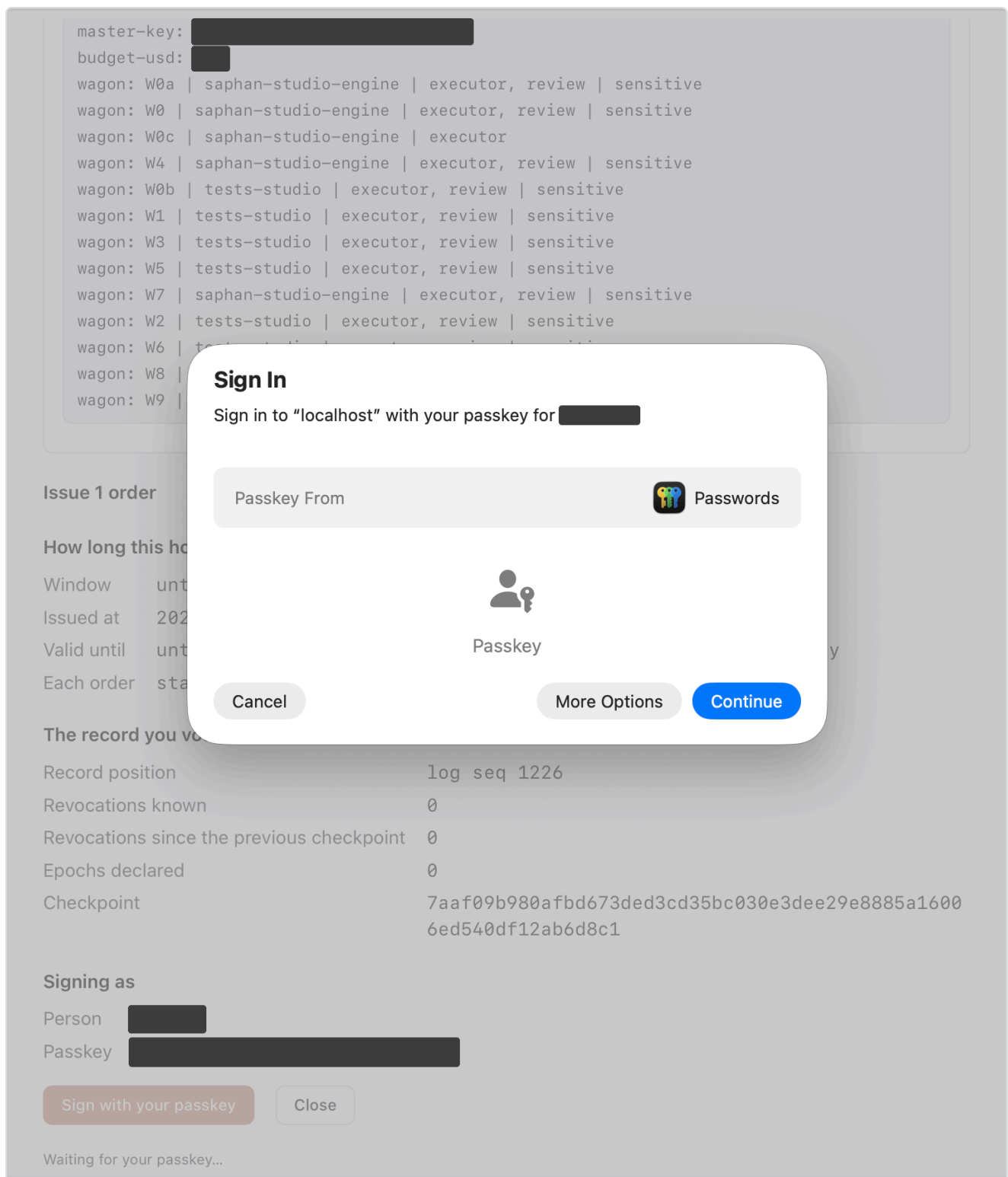
2026-10-30, morning (advisor analysis `ADVISOR_ANALYSIS_standard-tools-as-deterministic-skills_2026-10-30.md`):

„Jak robimy mocki i test studio i skille. To może zrobimy integracje deterministyczne z innymi narzędziami standardowymi w software development, które są używane w CI/CD? skill do uruchomienia testów, albo do uruchomienia scout do analizy bezpieczeństwa, narzędzia owasp. Poszalejmy. Teraz to już będzie super easy."

„ale ten model to jako opcja. Deterministyczne wywołanie żeby agent wiedział jak odpalić. Jako baza wiedzy w ask"

The console's "Sign orders" screen, "One touch, in your name", with the train order opened: the checksum matches, the signed text is "every byte, including link definitions and link addresses", and the order opens with the master, the status (nothing dispatched before the signature), the repositories, the base commits, and §0 with the approver's own words, verbatim, in the requester's language. The order and the console call the approver "owner". The approver then presses "Review what you will sign" before anything is signed.

The console separates what is signed from what is shown. The rendered order is labelled "a formatted reading aid, not the signed text". The signed text is every byte of the source, "including link definitions and link addresses". The approver signs with a passkey on the approver's own device.



The passkey prompt. The delegation block above it lists every wagon, its repository, its roles and whether it is sensitive. The key identifier and the budget are masked.

## What happens after the signature, without the approver

From the advisor agent's written answer (the full list is in chapter 6):

- The master writes and signs each wagon's work order, every fix round after a review, re-placements after a run died, and the review orders. The approver does not see these one by one.

- It chooses where each piece runs and moves work when a machine dies. That same morning, masters re-placed their work within an hour after a build host froze.
- It reports in the shared record. "The record is a git repository, so every step is a commit that anyone can read later."
- It cannot put the result on the main line. Landing always comes back to the approver.

## When the master comes back

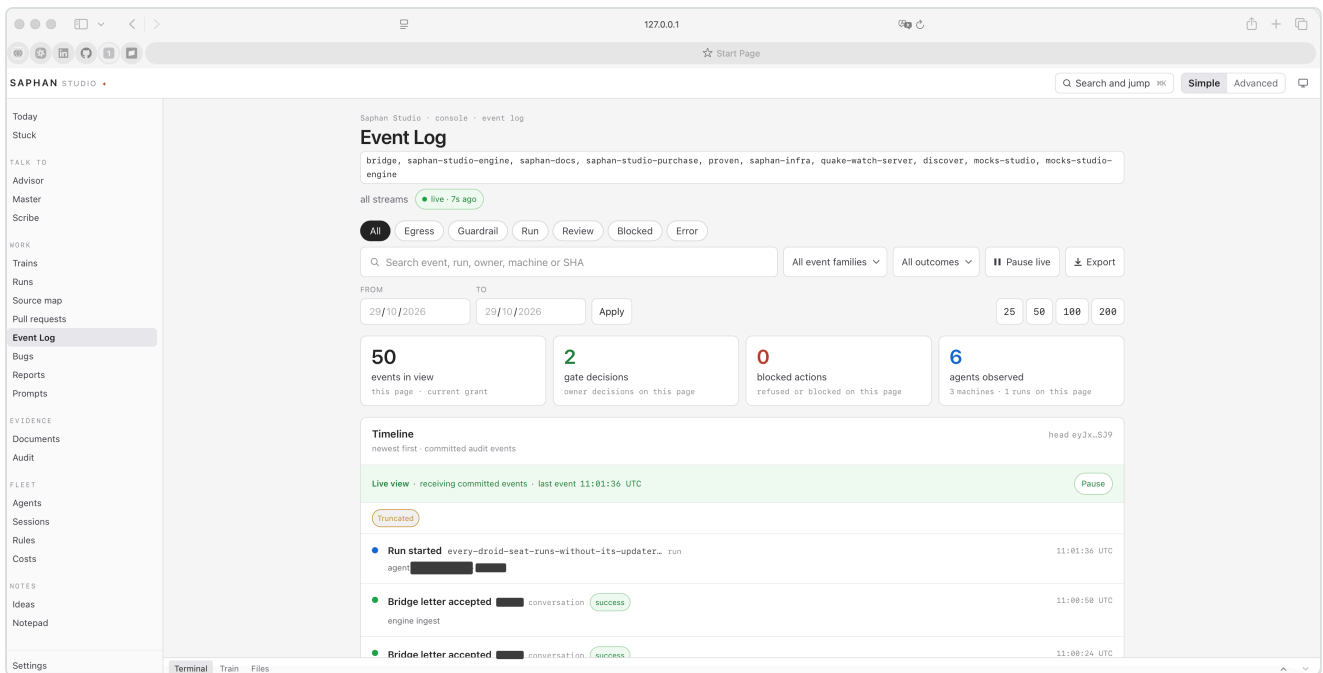
The list of situations, and whether the system or the master's own rule stops it, is in chapter 6. For a product manager the short version is: the master comes back when it needs **more money, more scope, another team's territory, more network, a licence decision, a landing, anything in production**, or when the work proves the plan wrong in a way that needs one of those.

A real return, from the same day. The `saphan-message` train had been given a small budget. By early morning, three orders signed by its master (two reviews and a fix round) were refused by the system with "budget exceeded". The master did not look for a way around. It sent one rehearsed script that raised the budget and re-signed the train, and stated plainly that the figure it had asked for earlier was wrong, and why. The block started at about 05:15. The approver ran the script at 07:00:29, and it finished at 07:00:41. The act itself took 12 seconds. The refused orders went out again, and the train kept moving.

## 4. For the architect: the moving parts

### The components in this case

| Component                    | What it is                                                                                                                      | Where you see it here                                                                                                                                        |
|------------------------------|---------------------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>saphan CLI and engine</b> | One binary on the approver's workstation and on the build machines. Places orders, starts runs on runners, enforces boundaries. | <code>saphan dispatch place</code> , <code>saphan train sign</code> , <code>saphan message send</code> , <code>saphan dry-run ...</code> in the screenshots. |
| <b>The record (bridge)</b>   | A git repository with orders, returns, reviews, decisions, correspondence and the agents' laws. The source of truth.            | Every order in this case is a file there, committed.                                                                                                         |
| <b>The console</b>           | The web interface, served from the same binary. The approver reads and signs here.                                              | Sign orders, the event log.                                                                                                                                  |
| <b>ASK</b>                   | Saphan's knowledge base: capabilities, sources, receipts and audit.                                                             | Hosts the invocation catalogue (W0a).                                                                                                                        |
| <b>Tests Studio</b>          | The results store and runner contract for test runs.                                                                            | Receives every tool run as a recorded result (W0b, W1).                                                                                                      |
| <b>Mocks Studio</b>          | Mocks of the systems around your product.                                                                                       | The only allowed target for DAST (W3).                                                                                                                       |
| <b>The egress wall</b>       | Outbound network control, closed by default, per task.                                                                          | Refused GitHub to the agents the same morning, until the approver ruled. Scoped to each record's declared egress in the gate.                                |
| <b>The permission layer</b>  | Which verbs an agent seat may run at all.                                                                                       | Refused one master the right to mint a signing key, so it handed the approver a script instead.                                                              |
| <b>Identity</b>              | Machines enrolled with keys, people with passkeys, masters with per-train keys.                                                 | The train's key is enrolled and granted for this train only, valid 30 days.                                                                                  |



The console's event log, filtered by stream and outcome: egress, guardrail, run, review, blocked, error. Every event is committed to the audit record.

## Trains, wagons, runs

A **train** is one coherent change, owned by one master. It is made of **wagons**, each one a reviewable unit of work with its own repository and definition of done. A wagon is executed by one or more **runs**: an agent working on a **runner** (a build machine), in its own worktree. The order of work is explicit:

- W0a, W0 and W0b run in parallel.
- W0c waits for W0 and W0a. It drives both.
- W1, W2, W3 and W5 wait for W0c. Each is N W0c orders, one tool per run, and they run in parallel.
- W6 waits for W1. W8 waits for W0c and one landed tool wagon. W4 and W7 are cut last, each on its approver condition.

## The delegation block

The approver signs the train once. The delegation block inside the signed order is the boundary the master works in:

**\$6 QUESTIONS TO THE ADVISOR (law #60: the readings named; nothing below is decided by this order)**

- **Q1 — licences of tools RUN, not linked.** The allow-list (MIT, Apache, BSD, ISC, 0BSD) governs code we ship and depend on. (a) A tool run as a pinned external image is outside it, so Semgrep CE (LGPL-2.1 engine, non-OSI rules licence), trufflehog and k6 (AGPL-3.0) and hadolint (GPL-3.0) may be catalogued. (b) The allow-list binds catalogued tools too, and those four are dropped or replaced: Semgrep CE by gosec/Bandit/ESLint security, trufflehog by gitleaks. The first cut (W1–W3, W5) touches only Semgrep CE.
- **Q2 — routing.** The kickoff sends executors to runner-cursor and runner-saphan. Law #9 class (b) sends W0 and W0a to claude-code or codex. This order follows law #9 for those two wagons and the kickoff for every per-tool leg. Is that the owner's reading?
- **Q3 — paid-pack mechanics (W7).** Is the mocks-studio paid-pack mechanism the one Scout rides, or does this train wait for it to exist?

## Delegation

```
master-key: [REDACTED]
budget-usd: [REDACTED]
wagon: W0a | saphan-studio-engine | executor, review | sensitive
wagon: W0 | saphan-studio-engine | executor, review | sensitive
wagon: W0c | saphan-studio-engine | executor
wagon: W4 | saphan-studio-engine | executor, review | sensitive
wagon: W0b | tests-studio | executor, review | sensitive
wagon: W1 | tests-studio | executor, review | sensitive
wagon: W3 | tests-studio | executor, review | sensitive
wagon: W5 | tests-studio | executor, review | sensitive
wagon: W7 | saphan-studio-engine | executor, review | sensitive
wagon: W2 | tests-studio | executor, review | sensitive
wagon: W6 | tests-studio | executor, review | sensitive
wagon: W8 | tests-studio | executor
wagon: W9 | saphan-studio-engine | executor
```

### Issue 1 order

### How long this holds

|             |                                                              |
|-------------|--------------------------------------------------------------|
| Window      | until withdrawn                                              |
| Issued at   | 2026-10-30T09:17:10Z                                         |
| Valid until | until you sign this train order again or revoke this passkey |
| Each order  | starts one run, once                                         |

The delegation block and the master's open questions, as the approver saw them before signing. Key and budget masked.

Each line names a wagon, the repository it may touch, the roles it may run (executor, review) and whether it is sensitive. The system refuses a master-signed order for a wagon that is not in this list, for a repository that is not named for it, or above the budget. The approver's signature is checked again at every placement.

```
Ran 3 shell commands

• The .sig file remains untracked since the write-set doesn't cover it, and placement verifies the signature directly. I'm targeting █████ (Linux, has admitted claude-code seats, matches engine-leg memory pointers), and will first refresh its mirror to pull develop 7e9159b36.

Ran 2 shell commands

• The █████ mirror already holds 7e9159b36. Placing W0 on █████

• Place the W0 order on █████ · 1m 24s
  └─ $ saphan dispatch place --order /Users/Sources/saphan/saphan-studio/bridge/development/backlog/standard-tools-run-as-deterministic-skills/orders/ORDER_tool-records-enter-only-by-the-gate.md --role executor --host █████ --workspace /Users/Sources/saphan/saphan-studio 2>&1 | grep -v '^remark' | tail ... (1m 24s)
    (ctrl+b to run in background)

★ Schlepping... (6m 57s · ↓ 17.7k tokens)
  └─ Tip: Use /btw to ask a quick side question without interrupting Claude's current work
```

The master places W0 after the placement step verified the signature directly. It chose a Linux machine with admitted Claude Code seats, because W0 is security work.

## Routing by risk

The master does not send work to the cheapest machine by default. It routes by the class of the work, under policy #9:

- **Security work** (W0, the gate that runs third-party code under confinement and decides what enters a signed catalogue; W0a, the authority in ASK) goes only to Claude Code or Codex seats. Never to Saphan's own agent loop, never to Qwen.
- **Network work** (W3, the live ZAP run) goes to Claude Code.
- **Contained development verified by a code gate** (the per-tool runs of W1, W2 and W5) goes to the cheaper seats: Cursor's agent running Grok, and Saphan's agent loop running NVIDIA Nemotron.
- **Reviews** always go to Claude Code or Codex.

Where the kickoff and the policy disagreed, the master followed the policy and asked the advisor to confirm (question Q2 in the order). It did not quietly choose.

## Bring your own agents

Saphan Studio does not ask a team to replace the coding agents it already uses. In this case, today:

| Backend                 | What runs on it                                                                                                           |
|-------------------------|---------------------------------------------------------------------------------------------------------------------------|
| Claude Code             | Anthropic's models; masters, security work and reviews                                                                    |
| Codex                   | OpenAI's models; security work and reviews                                                                                |
| Cursor's agent          | Grok; contained development runs                                                                                          |
| Saphan's own agent loop | NVIDIA Nemotron, through NVIDIA's API; development runs. NVIDIA's Nemotron safety model also runs locally as a guardrail. |
| Qwen Code               | Qwen models                                                                                                               |
| Droid                   | Factory's agent                                                                                                           |

Each of them is an executor under the same rules. The coding can run on whatever the customer already pays for. What stays in Saphan Studio is everything around it:

- **Orchestration:** the train, the wagons, the order of work, the placement.
- **Governance:** who signed, under which boundary, what came back, what landed.
- **Security:** grants per task, outbound network closed by default, the permission layer.
- **Documents:** orders, returns, reviews and decisions, in the record.

A Droid or Cursor seat in this model is an executor like any other: it gets an order, a worktree, a declared network, and its work is reviewed by a different agent before anything lands.

## Self-hosted

Saphan Studio runs on the organization's own infrastructure. In this case that is a workstation, a machine that runs NVIDIA's safety model locally, several Linux build machines and test hosts. No Saphan cloud sits in the path to the code.

## 5. For the QA engineer: how "done" is defined

### Every definition of done has three parts

The train order states it once, for every wagon:

"Each wagon's first DoD item is a test that **fails on the base and passes on the tip**, and the return pastes both runs. Unplugging the mechanism reddens that same test (law #41)."

So every wagon must show:

1. **Fails before:** a test that fails on the code as it was and passes after the change, with both outputs pasted into the return.
2. **Negative controls:** deliberate wrong inputs, each mutating exactly one field (policy #27), that the mechanism must refuse, naming the field.
3. **Unplug:** switch the new mechanism off, and the test from step 1 must go red again. This proves the test tests the mechanism, not something else.

Here is W0, the conformance gate, as written:

```
### W0 — the conformance gate: a record enters the catalogue only by passing it (engine)
- dod:
  - (1) Fails before: `saphan tool admit` on a gitleaks record against a fixture repository with one planted test secret admits it.
    On the base: `unknown command "tool"`.
  - (2) Negative controls, each mutating exactly one field (law #27):
    - `command` with a wrong flag: the tool exits `tool-error` and the gate refuses, naming `command`;
    - `pin` with a wrong digest: refused at pull, naming `pin`;
    - `egress: none` on a tool that dials out (the fixture's network probe): refused, naming `egress` and the host observed;
    - `output: sarif-2.1.0` on a tool that writes plain JSON: refused, naming `output` and the first schema violation.
  - (3) Unplug: disable the egress comparison and the third control in (2) reddens.
```

### The plan is written to be refuted

Before any wagon starts, the master writes down what it believes about the code, with a date and how it measured it. Every executing agent must re-take those readings at its own base before changing anything. If a reading is wrong, the agent writes the measurement into a field called `master-was-wrong:`.

### §1A THE MASTER'S READINGS — CLAIMS, DATED, FOR EVERY LEG TO REFUTE (law #44)

Taken by MASTER on 2026-10-30. Every wagon's first step re-takes the readings it rests on, at its own base, before it changes anything. A leg that finds one wrong writes the measurement into its return's `master-was-wrong:` field.

| #  | reading                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        | how taken                                    |
|----|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------------------------------------------|
| R1 | Tests Studio has <b>no JUnit and no SARIF ingest</b> . <code>git grep -ciE 'junit sarif'</code> matches zero files at <code>tests-studio main @ 7dffbb3</code> and <code>tests-studio-engine main @ 2ba64dc</code> . The only ingest is a live runner-v1 adapter ( <code>spec/TEST_RUNNER_V1.md v1.1: spec → open/attach → step start/finish → evidence → close → tests.run-result/v1</code> ). The spec has no findings model.                                                                                | git grep, both repos                         |
| R2 | The engine holds an egress wall with recorded posture: <code>internal/egresswall/</code> (rules, resolver, scope, shim), <code>internal/agent/egresswall.go</code> , and an egress record ( <code>internal/egress/record.go</code> ). <code>egress-resolve-denied</code> is the refusal a leg met on github.com on 2026-10-30. Whether the wall can run an ARBITRARY pinned tool process (not an agent) and hand back the list of hosts it reached is <b>unmeasured</b> .                                      | git grep at <code>develop @ 7e9159b36</code> |
| R3 | <code>saphan ask</code> has <code>resolve</code> , <code>receipt</code> and <code>audit</code> . The saphan-ask train's W1 (capability sweep) and W2 (source checkpoint) stand at master-verify r3. Its W3 (projection) and W5 (receipts) have not landed. Its §1A reading 2 makes a capability slot a PRESENCE claim (owner DECISION 2026-09-18), while a catalogue record is a structured VALUE. Which ASK shape holds it is <b>asked of the saphan-ask master</b> (message <code>2df74b669d22c8cf</code> ). | ask train order, returns dir                 |
|    | The record holds <code>docker-oci</code> capability rows for <code>runner-saphan-and-builder-omd64-1-only</code>                                                                                                                                                                                                                                                                                                                                                                                               |                                              |

§1A: the master's readings, dated, which every run must try to refute. R1 says Tests Studio has no JUnit or SARIF ingest, and shows the command that proved it.

This is how the master found, within minutes, two places where the advisor's own analysis did not match the code: Tests Studio had no JUnit or SARIF ingest yet, and the ASK knowledge base stored capabilities as present-or-absent facts while a catalogue record is a full recipe. It said so in its summary to the approver, under "Findings that change the advisor's analysis".

## Fixtures with known outcomes

W0b ships small fixture repositories that a stranger can run:

- ``tools/fixtures/<ecosystem>/'`, small fixture repositories with known outcomes: one failing and one passing test per test ecosystem; one planted, obviously fake secret; one dependency with a known published advisory; one misconfigured Dockerfile.
- A conformance runner, ``tools/conformance/'`, that a stranger runs against a record. It is the same check as W0's gate, without the egress wall, and it says so in a ``remark:'` line (law #52).

## Results you can compare

Every tool run reports to Tests Studio as a run: JUnit for tests, SARIF for findings, CycloneDX for SBOMs. A second run on unchanged code produces an empty delta. A new planted finding produces exactly one entry. A waived finding produces none until its waiver expires, and a waiver past its expiry is reported, not hidden.

## 6. For the reviewer and the tester

### Review follows the risk, not the size

Every wagon in the order states its review class and the reason, in brackets:

| Review class                                                | Wagons         | Reason, as written in the order                                                         |
|-------------------------------------------------------------|----------------|-----------------------------------------------------------------------------------------|
| <b>Named, security class (b):</b> Claude Code or Codex only | W0, W0a, W2    | "it runs third-party code under confinement and decides what enters a signed catalogue" |
| <b>Named, network class (a):</b> Claude Code                | W3             | the live ZAP run reaches the network                                                    |
| <b>Named, public interface</b>                              | W0b, W6        | what a customer will see and depend on                                                  |
| <b>Named</b>                                                | W1, W4, W5, W7 | new code in the product                                                                 |
| <b>Master verification only</b>                             | W0c, W8        | "a document over W0/W0a's seam"; "it adds no mechanism; it walks W0–W0c"                |
| <b>One tester over the whole train</b>                      | W9             | run at the end, in a worktree cut at the commit that carries every wagon                |

On long wagons, intermediate rounds carry master verification and the named review goes on the last round only (policy #29), so review effort is spent where it decides.

### The reviewer is never the author

From the advisor's answer:

"Any wagon marked sensitive in the signed train gets an independent review by a different agent (a different model family where the rules ask for it) before it can land. The master reads the review and sends the fix round itself."

### What the reviews found before

These numbers come from the public Proven report (chapter 8), the last full build run under Saphan Studio:

- 154 independent reviews returned 307 verdicts. 113 of them were rejections, about 37%.
- 53 architecture decisions were recorded, each with its reasons.

A review that never rejects is not a review. Here, rejections are normal and recorded.

## Where the line between code and rule sits

The advisor's answer separates what the system enforces from what the master's own rules enforce. For a reviewer this is the most useful table in the document: the "code" rows cannot be argued with, and the "rule" rows are what you check in the record.

| #  | The master meets...                                           | Who stops it                                                                                                                                                         |
|----|---------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 1  | the budget the approver granted is used up                    | <b>Code:</b> the next order is refused, "budget exceeded"                                                                                                            |
| 2  | a new wagon, or a change of scope or definition of done       | <b>Code</b> for a new wagon (only listed wagons can be signed); <b>rule</b> for scope inside a wagon (policy #60)                                                    |
| 3  | work in another track or another repository                   | <b>Code</b> for the repository (named in the signed block); <b>rule</b> for the track (policy #59)                                                                   |
| 4  | a change of security class, or egress beyond what is declared | <b>Code</b> for egress (the wall refused GitHub on 30 October until the approver ruled) and for the sensitive mark; <b>rule:</b> never re-classify to avoid a review |
| 5  | a licence or paid-pack decision                               | <b>Rule</b> only                                                                                                                                                     |
| 6  | landing on the main line                                      | <b>Code:</b> the approver types a one-time confirmation on the approver's own device; an agent cannot type it                                                        |
| 7  | anything touching production or the approver's workstation    | <b>Mostly code:</b> the permission layer denies those verbs. <b>Rule:</b> a denied act stays denied in every spelling (policy #57)                                   |
| 8  | the work proves the plan wrong                                | <b>Rule:</b> the master re-plans inside the boundary, or comes back if the fix needs rows 1–2                                                                        |
| 9  | the delegation expires or is revoked                          | <b>Code</b>                                                                                                                                                          |
| 10 | a failed run used up an approver-signed order                 | <b>Code:</b> approver-signed orders start one run, once                                                                                                              |

While it waits, a master is not idle. Saphan Studio's policy is that a blocked master measures the next problems and prepares exact, ready-to-sign orders, so that the approver's next touch unlocks work that is already written.

## 7. Security and governance, from the same week

---

Three things that happened around this train show the same pattern: the boundary holds, and the agent comes back with what it found.

**Outbound network.** On 30 October, agents tried to reach github.com and the egress wall refused them ( `egress-resolve-denied` ). Nothing was opened by default. The approver ruled on a narrow, recorded GitHub read grant as the first trial of a new grant-levels policy. Until it exists, the master reads public repositories from its own seat and pins what it read, and the executors develop from that. In this train's own gate (W0), the wall is scoped to each tool's declared egress and compares it with what the tool actually reached.

**A refusal is not worked around.** The same morning, the permission layer refused another master the right to mint its train's signing key. It did not ask another agent to do it, and did not look for a different command. It wrote one script for the approver. Policy #57 says a denied act stays denied in every spelling.

**A supply-chain alert, checked in hours.** On 30 October a trojanised build of a popular AI coding extension was reported in an editor marketplace. Within hours the agents checked ten of the organization's machines against every indicator in the public analysis. They matched the attacker's key by fingerprint rather than by its comment, which anyone can edit. Zero hits. And one lasting finding: the organization runs the vendors' command-line agents, not editor extensions, so that delivery path never reached it.

The agents' report on that check, summarised:

```
SUPPLY-CHAIN CHECK · trojanised AI coding extension, version 2.1.313
exposure window (vendor notice): 29 Oct, 09:27-20:35 UTC

scope    10 hosts: 8 Linux runners and 2 Macs
method   every indicator in the public analysis, not only the quick three;
         the attacker's key matched by fingerprint, because a key's
         comment can be changed in any editor

indicator                                     result
attacker's Ed25519 key, by fingerprint        0 hits on 10 hosts
key comment "svc-telemetry"                  0
persistence files (~/.config/.smt, .smtk)     0
fake macOS LaunchAgent                       0
cron @reboot entry                           0
temporary payload files                       0
reverse-tunnel processes (autossh, ssh -R)    0
connections to the attacker's server          0
the extension itself                          installed nowhere

authorised keys  each host holds exactly the operator's own keys,
                  and no foreign key
finding          the organization runs the vendors' command-line agents, not
                  editor extensions; the runner that hosts one of those
                  agents has no extension directory at all, so this
                  delivery path never reached it
not checked      two machines, named in the report as not checked
```

**What the record gives an auditor.** For any change, the record answers: who asked for it (the requester's words, verbatim), who planned it, under which signed boundary, which agent wrote it, which different agent reviewed it, what the review rejected, what was fixed, and who landed it. The answers are commits, not a dashboard someone has to trust.

## 8. Evidence you can check yourself

---

This case covers the planning of one train. To judge the quality of the work agents produce under Saphan Studio, look at the last full build: **Proven**, an open-source pull-request reviewer that reports only what it can prove. The repository and the report are public.

From the report, *"Proven effort report: six days with agents, 14 to 20 months for people"*:

- 139 hours from the first build commit to the submission.
- 42,036 lines of production Go and 58,956 lines of Go tests (1,828 test functions).
- 53 architecture decision records; 307 review verdicts from 154 independent reviews, 113 of them rejections.
- Test coverage 80.9%, "above the 75% of level 4 in Google's voluntary coverage levels", measured on packages holding 64% of production code.
- 96.1% of the public API documented (1,490 of 1,550 exported Go identifiers).
- 88.9% of functions at or below McCabe's complexity limit of 10.
- Four estimation methods (COCOMO II, QSM's project database, McConnell's productivity ranges and a bottom-up count) put the same scope at 9 to 11 people for 14 to 20 months.

The report also says what it does not prove: "Lines of code measure size, not value." "Equal quality is not proven." "Six days is not the whole effort."

**How to check it.** Clone the repository, check out the tag `submission`, and run your own tools on it: your coverage, your static analysis, your complexity metrics. Start with "Honest limits" in the README, then the record of every change in the repository.

- Repository: [github.com/saphan-ai/proven](https://github.com/saphan-ai/proven)
- Report: [saphan.ai/research/proven-effort](https://saphan.ai/research/proven-effort)
- Documentation: [docs.saphan.ai](https://docs.saphan.ai)

## 9. Honest limits

---

We would rather you read these here than find them later.

- **Some facts in the plan were from memory.** The tool licences in the order are marked "from memory and unverified". The order makes each wagon read the licence file at the pinned version before any work is placed.
- **This train has not finished.** This case covers the planning, the signature and the start. The build, its reviews and its tester are still ahead, and their results will be in the record.

## 10. How this case study was written

---

This document was written by an AI agent working on Saphan's marketing, not by the people or agents who did the work. It was written from the record, not from memory.

- **Screens** are screenshots of the terminal and the console, taken while the work happened. We masked key identifiers, passkey identifiers, amounts, personal names and machine names. One report, written in a chat in the requester's language, is summarised in English as text instead of shown.
- **Order excerpts** are copied from the committed files.
- **Times** come from file timestamps, commits and the console.
- **The delegation chapter** comes from a written request to the advisor agent, sent through the record, and its written answer. The advisor answered from the governing documents, the train order, the status files of approver acts and the masters' messages, and marked every section as public, internal or not for publication.

The same discipline caught an error in an earlier draft of related material. We had a time for an incident, "29 Oct 23:53", from a screenshot of an agent's summary. The advisor checked the record and answered: that time "appears nowhere in the infra state, the brief, the finding or the bridge correspondence ... so it is not a measured time". The correct time, from the machine's own journal, was 28 Oct 08:13:20Z. We use the record, even when it corrects us.

# Appendix A. Excerpts from the train order

## §1 What this train is

Today a head that wants to run a test suite, a dependency scan or a secrets scan composes the command from general knowledge, on whatever host it is on, and reads the raw log. When this train is done:

- A head asks `'saphan ask tool <id>'` and gets the exact pinned invocation for the repository in hand: command, inputs, declared egress, exit-code map, output format and path. It comes with a receipt, or a refusal by name.
- A deterministic runner executes that invocation, normalises the output to SARIF 2.1.0 (findings), JUnit XML (tests) or CycloneDX (SBOM), diffs it against a baseline with waivers honoured, and reports the run to Tests Studio. The model sees only the delta.
- Adding a tool is one order to one run: "analyse tool X and add it to the ASK library". A code gate makes that safe.

## W0a: the record schema

- The tool-invocation record schema, versioned (`'tool-invocation/v1'`). One record per (tool, ecosystem):
  - `'id'`, `'version'` and `'pin'` (image digest or asset sha256), plus a `'licence'` class (`'osi'` | `'paid-pack'`);
  - `'command'`, a template over TYPED inputs only (`'repo_root'`, `'config_path'`, `'output_path'`, declared extras). No free text and no shell metacharacters; the template is refused on load otherwise;
  - `'inputs'` (paths read), `'egress'` (a host list, or `'none'`), `'exit_map'` (code → `'clean'` | `'findings'` | `'tool-error'`);
  - `'output'` (`'sarif-2.1.0'` | `'junit-xml'` | `'cyclonedx-1.5'` | `'json'`) and its path;
  - `'admitted_by'`, the conformance gate run that admitted it: fixture repository digest, output digest, egress observed.

## W3: DAST only against mocks

- lands: ZAP records whose target is a Mocks Studio mock opened for the run, and never a production URL. A record naming a non-mock target is refused by the gate.
- dod: ZAP baseline against the fixture mock yields SARIF with the planted header finding. The negative control points it at an undeclared host and is refused by the egress comparison.
- review: named (network class (a) → claude-code)

## W8: the acceptance test of the whole train

- lands: one W0c order, filled with a tool the catalogue does not hold, placed on an NVIDIA seat with no human act other than the landing. Then a head that did not build anything asks `saphan ask tool <it>`, runs the runner, and the run appears in Tests Studio.
- dod: the run's return, the gate's admit row, the fresh head's transcript and the Tests Studio run id. Negative control: the same order with one wrong flag planted by the master is refused by the gate.

## §5 Out of scope (selected)

- Running any scanner against a production target. DAST runs against Mocks Studio mocks only.
- Commercial tools other than W7, until the approver rules on each pack.

## §6 Questions to the advisor (nothing below is decided by this order)

- Q1 – licences of tools RUN, not linked. The allow-list (MIT, Apache, BSD, ISC, 0BSD) governs code we ship and depend on. (a) A tool run as a pinned external image is outside it ... (b) The allow-list binds catalogued tools too, and those four are dropped or replaced ...
- Q2 – routing. ... This order follows law #9 for those two wagons and the kickoff for every per-tool run. Is that the [approver]'s reading?
- Q3 – paid-pack mechanics (W7). Is the mocks-studio paid-pack mechanism the one Scout rides, or does this train wait for it to exist?

# Appendix B. The master at work, screen by screen

```
Ran 3 shell commands

• Act 1 complete and the watcher is armed, with fresh trunks confirmed on engine, tests-studio, and tests-studio-engine (origin lagging on all three as expected). Next, I'll read the two partner train orders.

Made 2 scratchpad edits +78, read 4 files, ran 4 shell commands

• I've drafted both interface messages (to saphan-ask and tests-studio). Sending them now; next comes TRAIN_ORDER.md.

• Sending interface questions to the ask and tests-studio masters · 29s
  └─ $ cd ~/.code-fleet-2-1 && S=/private/tmp/claude-501/-Users-Sources-saphan-saphan-studio/427b938b-2266-40a4-9820-454a905b4d07/scratchpad && saphan message send --workspace /Users/Sources/saphan/saphan-studio --track standard-tools-run-as-deterministic-skills --to track:saphan-ask --reply required --... (29s · 17 lines) (ctrl+b to run in background)

+ Photosynthesizing... (2m 17s · + 9.6k tokens)
  └─ Tip: Use /btw to ask a quick side question without interrupting Claude's current work
```

1. The master's first acts: it confirms the current state of three repositories and sends interface questions to the two neighbouring masters.

```
Made 2 scratchpad edits +78, read 4 files, ran 4 shell commands

• I've drafted both interface messages (to saphan-ask and tests-studio). Sending them now; next comes TRAIN_ORDER.md.

Ran 1 shell command

• Both messages are sent (2df74b669d22c8cf to ask, dfa53c3c2828f2c6 to tests-studio). Act 3 next: TRAIN_ORDER.md from the template.

Ran 2 shell commands

• I have what I need. Writing TRAIN_ORDER.md now.

* Photosynthesizing... (4m 8s · + 17.4k tokens)
  └─ Tip: Use /btw to ask a quick side question without interrupting Claude's current work
```

2. Both messages sent, both answered. "I have what I need. Writing TRAIN\_ORDER.md now."

```
... +252 lines

• Now the delegation block: I mint this train's master key and write the delegation into the order.

• Mint this train's master order-signing key
  └─ $ saphan train delegation --order
    /Users/Sources/saphan/saphan-studio/bridge/development/backlog/standard-tools-run-as-deterministic-skills/TRAIN_ORDER.md --master-key
    [REDACTED] --budget-usd [REDACTED] --write --workspace /Users/Sources/saphan/saphan-studio 2>&1; tail -20 /Users/Sources...

• Photosynthesizing... (4m 53s · + 23.1k tokens)
  └─ Tip: Use /btw to ask a quick side question without interrupting Claude's current work
```

3. The delegation block: the master mints this train's own order-signing key and writes the delegation into the order. Key identifier and budget masked.

```

● Checking train-order grammar and rehearse the owner's signature
└─ $ cd /Users/Sources/saphan/saphan-studio &&
   F=/Users/Sources/saphan/saphan-studio/bridge/development/backlog/standard-tools-run-as-deterministic-skills/TRAIN_ORDER.md; saphan doc check
   $F --family train-order --require-checked 2>&1 | tail -15; echo ===; saphan dry-run order sign $F --workspace /Us...

● Bash(cd /Users/Sources/saphan/saphan-studio &&
   F=/Users/Sources/saphan/saphan-studio/bridge/development/backlog/standard-tools-run-as-deterministic-skills/TRAIN_ORDE...)
└─ workspace: /Users/Sources/saphan/saphan-studio (discovered)
   found via: /Users/Sources/saphan/saphan-studio/.saphan
   doc check: /Users/Sources/saphan/saphan-studio/bridge/development/backlog/standard-tools-run-as-deterministic-skills/TRAIN_ORDER.md - f
   ... +15 lines
└─ 1 file changed (binary, mode only or too large to show)
└─ (files that changed while this command ran, by the command or by another process; a convenience view, not a review or audit of the
   command)
└─ Allowed by auto mode

* Photosynthesizing... (5m 16s · ↓ 24.7k tokens)
└─ Tip: Use /btw to ask a quick side question without interrupting Claude's current work

```

4. Before handing anything to the approver, the master checks the order's grammar and rehearses the approver's signature.

```

... +15 lines
└─ 1 file changed (binary, mode only or too large to show)
└─ (files that changed while this command ran, by the command or by another process; a convenience view, not a review or audit of the
   command)
└─ Allowed by auto mode

Listed 1 directory, ran 2 shell commands, searched memories

● The train order is committed (504c1bc64). Now I'm writing the owner's script: key enrolment, delegation grant, and the train signature, each
rehearsed.

● Rehearse the key enrolment and delegation grant
└─ $ ls tmp/saphan-message-2026-10-12/tools/act-status.sh tmp/saphan-message-2026-10-12/tools/reason.sh; saphan dry-run identity enroll --kind
   actor --subject master-standard-tools-run-as-deterministic-skills --key-id [REDACTED] --reason 'the order-signing key of
   train standard-tools...

* Photosynthesizing... (5m 52s · ↓ 26.5k tokens · thinking with high effort)
└─ Tip: Use /btw to ask a quick side question without interrupting Claude's current work

```

5. The train order is committed. The master writes the approver's script: key enrolment, delegation grant and train signature, with the signature rehearsed first.

```

# OWNER ACT – enrol the train master's order-signing key, grant it order-signing for this train, and sign the train order
# once with an unlimited window (MASTER.md §3 "every train runs delegated"; kickoff 2026-10-30, rulings accepted ~07:35Z).
# Written by MASTER from governance/templates/OWNER_SCRIPT.md. Run it ALONE.

"TRACK: standard tools as deterministic skills (standard-tools-run-as-deterministic-skills) – key enrolment, deleg
ation, train signature"
set

... +27 lines
└─ Allowed by auto mode

● Adding memory index pointer
└─ $ echo '- [Standard-tools track appointment](standard-tools-track-appointment.md) – 2026-10-30 master of
   standard-tools-run-as-deterministic-skills; key, script, three open asks' >>
   [REDACTED]

* Photosynthesizing... (6m 58s · ↓ 31.4k tokens · thinking with high effort)
└─ Tip: Use /btw to ask a quick side question without interrupting Claude's current work

```

6. The approver's script, from the Saphan Studio template: what it does, under which rule, and "Run it ALONE". The master also writes itself a pointer into its own memory, so a later session does not start from zero.

# Appendix C. Glossary

| Term                             | Meaning                                                                                                                                   |
|----------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Approver</b>                  | The accountable person who signs plans and approves every landing. Rules on decisions. The console and the orders call this role "owner". |
| <b>Requester</b>                 | The person whose words of intent open the order, verbatim. In this case, the same person as the approver.                                 |
| <b>Advisor</b>                   | An AI agent that plans with the approver, appoints masters and collects what waits on the approver.                                       |
| <b>Master</b>                    | An AI agent that owns one train: writes its order, signs work orders inside its delegation, places and verifies them.                     |
| <b>Train</b>                     | One coherent change, made of wagons, signed once by the approver.                                                                         |
| <b>Wagon</b>                     | One reviewable unit of work, with its own repository, definition of done and review class.                                                |
| <b>Run</b>                       | One agent working on one runner, in its own worktree, executing an order.                                                                 |
| <b>Runner</b>                    | A build machine where runs execute.                                                                                                       |
| <b>Delegation</b>                | The boundary inside a signed train: listed wagons, repositories, roles, sensitivity, budget and validity.                                 |
| <b>Record (bridge)</b>           | The git repository that holds every order, return, review, decision and message.                                                          |
| <b>ASK</b>                       | Saphan's knowledge base, which will serve the tool-invocation catalogue.                                                                  |
| <b>Egress wall</b>               | Outbound network control, closed by default, scoped per task.                                                                             |
| <b>SARIF / JUnit / CycloneDX</b> | Standard formats for findings, test results and software bills of materials.                                                              |
| <b>Policy #n</b>                 | A numbered policy enforced in Saphan Studio, each born from a recorded incident. The excerpts and screens call it "law #n".               |
| <b>Head</b>                      | In the excerpts: one agent session.                                                                                                       |