EFFORT REPORT · PROVEN

Six days with agents, 14 to 20 months for people

An effort analysis of Proven, built by AI coding agents under one person's approval, compared with what the same software would take a conventional team. Every number in this report can be checked in the public repository.

PUBLIC EDITION

139 h	14–20 mo	80.9%	307
agent build, first commit to submission	estimated for a team of about ten people	statement coverage, measured packages	review verdicts, 37% of them rejections

REPOSITORY

github.com/saphan-ai/proven

tag submission · commit d864718 · Apache-2.0

PUBLISHED

29 October 2026

Saphan Studio · saphan.ai/research/proven-effort สะพาน

CONTENTS

01 Summary
02 What Proven is
03 How it was built, and the agents role by role
04 What was delivered, measured
05 Code quality, measured against published baselines
06 The record of the work
07 What the agent build cost
08 How human developers work: what the research says
09 The human-team estimate
10 What this comparison does not claim
11 Check it yourself
– Sources

HOW TO READ THIS REPORT

Sections 1 to 7 measure what was built and how. Section 8 summarises the published research on how human developers work. Section 9 turns that research into an estimate for a human team, and section 10 lists what the comparison does not prove. Section 11 gives the commands to check every number yourself.

This is the public edition. It leaves out money figures for the agent build and some internal details; every measurement and estimate is the same as in the full report.

SECTION 01

Summary

Proven is an open-source AI code reviewer that reports a problem only when it can show evidence: a test that fails on the change, or a data query and the rows it returned. It was built between 22 and 27 October 2026 at the Open Agent Hackathon 2026. One accountable person, the approver, decided and landed every change and wrote no code. AI coding agents, run by Saphan Studio, planned, built, tested and reviewed. Every order, report, review and decision is in the repository next to the code.

We measured what was delivered. We then estimated, with four published methods, what a conventional human team would need to build the same thing.

In one sentence: what the agents built in six days would take a conventional team of about ten people 14 to 20 months. The two ends of that range come from two methods:

- **14 to 15 months:** the bottom-up estimate (section 9.2), or the COCOMO II model at its maximum schedule compression.
- **19 to 20 months:** the COCOMO II model with nominal settings.

Measured against 139 hours of agent work, that is roughly 70 to 100 times longer in calendar time.

	BUILT BY AGENTS UNDER ONE APPROVER	ESTIMATED FOR A HUMAN TEAM
Calendar time	**139 hours** (5.8 days), first build commit to submission	**14 to 20 months**
People	1 approver (decides and merges, writes no code)	**9 to 11** full-time people in 8 roles
Effort	not comparable in person-months (see section 10)	**100 to 225 person-months**, central estimate about **135 to 180**
Labour cost	not published (see section 7)	**$1.6M to $3.6M** at the U.S. median developer wage, fully loaded
Production code	42,036 lines of Go, plus 9,246 lines of shell, TypeScript, Python and SQL	same scope
Test code	58,956 lines of Go in 456 test files, 1,828 test functions	same scope
Design record	53 architecture decision records	same scope
Review record	307 review verdicts in 154 independent reviews; 113 of them were rejections	same scope

Code quality, against published baselines (section 5):

MEASURE	PROVEN	PUBLISHED BASELINE
Test coverage	**80.9%** of statements, in the packages that hold 64% of the production code	75% is level 4 of Google's 5 voluntary coverage levels

MEASURE	PROVEN	PUBLISHED BASELINE
Documented public API	**96.1%** of exported Go identifiers have a doc comment	Go: “Every exported (capitalized) name should have a doc comment.”
Comment density	**18.7%**	18.67%, the mean of more than 5,000 open-source projects
Complexity	**88.9%** of functions at or below 10; one known outlier	NIST / McCabe: a limit of 10 per function
Documentation	53 ADRs, 22 product documents, a generated settings reference checked by tests	93% of open-source users meet incomplete or outdated docs (GitHub survey)

The comparison is about calendar time and scope, not about whether agents are “better programmers”. Section 10 lists what it does not prove.

SECTION 02

What Proven is

Proven reviews a pull request with six agent roles:

- a **Planner** names the risks;
- an **Investigator** gathers facts;
- a **Prover** tries to prove each problem with a test;
- a **Skeptic**, run on another model family, tries to refute it;
- a **Judge** keeps only findings that have evidence, by fixed rules;
- a **Writer** produces the report.

What a buyer would recognise as scope:

- **Evidence-only findings.** A finding needs one of two proofs:
  - a test that fails on the change and passes on the code before it, or
  - a query and its rows, from OSV, CISA KEV or FIRST EPSS, through Zetaris.
- **A locked-down runner.** The change is executed in a container with no network and no secrets. The command is built by Proven and is never taken from a model.
- **Signed evidence packs.** `proven verify` re-derives every verdict offline. A forgery bench tries to break the packs.
- **Ecosystem adapters.** Go, npm, PyPI, PHP (Composer), Rust (Cargo) and Maven/Gradle, with their own version-comparison rules.
- **Delivery.** A CLI, a GitHub Action and Gitea Actions. Output as a pull-request comment, JSON, SARIF and text.
- **Deployment.** Air-gapped deployment, a gateway, OpenTelemetry telemetry, and a Cloudflare deployment for the hosted judge.
- **A local lab.** `make up` starts Gitea, sample pull requests with planted defects, and Grafana dashboards.
- **A test corpus and benchmarks.**
  - 21 cases (PV-001 to PV-021), a held-out set, and conformance suites for four engines.

  - A comparison against GitHub Dependabot on the repository itself.
- **Documentation.** 22 product documents and a documentation site.

The README has a section called **Honest limits**, which lists what Proven does not do yet, with dates. We recommend reading it before anything else.

SECTION 03

How it was built

People. One accountable person, the approver. The approver decides, rules on disputes, and executed every merge into main; no agent ever pushed. The repository's `metadata.json` records this role under the name it uses, "owner: decides and merges; writes no code".

Who the commits are attributed to. On 23 October the agents' git identity in this repository was mapped to the approver's, so `git shortlog` lists almost every commit under the approver's name. Who built each change is recorded in the `Saphan-*` commit trailers, in `hackathon/fleet/LEDGER.md` and in `BUILD_LOG.md`.

Agents. Each change is reviewed by a model family other than the one that wrote it.

- **Executors that write the product code:** Cursor agents, two to four at a time.
- **Planning, tests and review:** Claude agents.
- **Further reviewers:** agents on other model families, including GPT, Qwen/GLM and NVIDIA Nemotron.

Process. Every change passes through four public documents:

1. **Order.** A lead agent measures the code first, then names exactly what to change, the acceptance criteria and a budget.
2. **Status.** The executor reports what it built and what it ran, with the commands and their output.
3. **Review.** An agent that did not write the change tries to break it. For smaller orders (67 of 116 are marked `master-verify-only`), the lead agent checks the work instead of a separate reviewer. A review returns one of three verdicts: `accept`, `accept-with-conditions` or `reject`. Each condition is weighted **BLOCKING**, **OWED** or **NOTE**.
4. **Decision.** The approver decides, with the reason in the approver's own words.

A rejected change goes back for another round. The ledger (`hackathon/fleet/LEDGER.md`) records every change: the model that built it, its rounds, its BLOCKING, OWED and NOTE conditions, and its outcome.

Timeline.

DATE (2026)	WHAT HAPPENED	CODE
16 to 21 October	The topic is chosen by asking the agents, and 25 design decisions are recorded	none
22 October, 00:03 (UTC)	First build commit (tag `start`)	first Go file
end of day 1 (tag `day-1`)		8,594 lines of product Go, 8,387 lines of test Go
end of day 2 (tag `day-2`)		17,634 product / 17,799 test

DATE (2026)	WHAT HAPPENED	CODE
end of day 3 (tag `day-3`)		29,168 product / 29,912 test
27 October, 19:06 (UTC)	Submission (tag `submission`)	55,295 product / 69,789 test

The line counts in this table are physical lines, including blanks and comments. Section 4 counts only code lines.

The tag `prior-art` marks the state of the repository before the build window. It contains **no Go code at all**: the planning documents existed, the software did not.

Commits landed in **110 of the 140 clock hours** of the build window. The longest pause was 7.6 hours. In the same six days, the same approver and agents also kept working on Saphan Studio itself and on other projects.

3.1 The agents, role by role

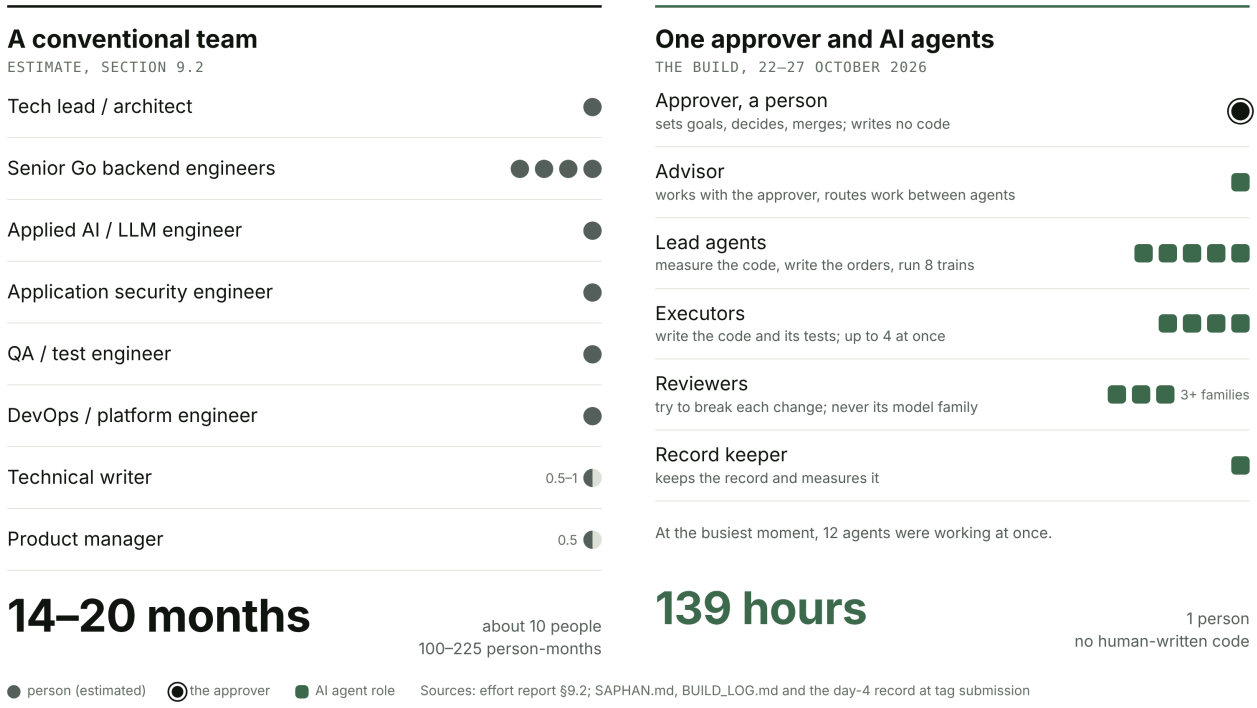


Figure 1. The same scope, two teams. Left: the conventional team estimated in section 9.2. Right: who built Proven. A dot is a person; a square is an AI agent role. Sources: section 9.2; `hackathon/SAPHAN.md`, `BUILD_LOG.md` *and the day-4 record at the tag* `submission`.

Each role a conventional team needs has a counterpart among the agents, and every counterpart leaves a written trace. Two product terms appear below: a *lead agent* measures the code and writes the orders for one area, and a *train* is a group of related changes planned and approved as one.

ROLE IN A CONVENTIONAL TEAM	PEOPLE	WHO DID IT WITH SAPHAN STUDIO	TRACE IN THE RECORD
Product manager	0.5	The approver, a person: goals, decisions, merges	87 decisions, each with its reason

ROLE IN A CONVENTIONAL TEAM	PEOPLE	WHO DID IT WITH SAPHAN STUDIO	TRACE IN THE RECORD
Tech lead / architect	1	The approver and 5 lead agents: each order measures the code before a change	116 orders, 53 ADRs
Senior Go backend engineers	4	Executors, up to 4 coding at once	192 status reports, with commands and output
Applied AI / LLM engineer	1	Executors and the lead agent of the reviewer-core train	train `proven-reviewer-core`
Application security engineer	1	The security-review train: each of 9 pieces reviewed by two model families	train `proven-security-review`
QA / test engineer	1	Executors and reviewers; tests shown red with the code broken on purpose	58,956 lines of tests; “negative control” 340 times in the record
DevOps / platform engineer	1	The lead agent and executors of the infrastructure train	lab, container images, telemetry, docs site
Technical writer	0.5 to 1	Executors and lead agents	22 product documents, about 179,000 words
Code review (about 6 person-months in section 9.2)		Reviewers, never from the model family that wrote the change	154 reviews, 307 verdicts
(no usual counterpart)		The record keeper: keeps the record and measures it	about 638,000 words of build record
(no usual counterpart)		The advisor: works with the approver, routes work between agents	50 of the 165 agent messages on day four

The last two rows have no counterpart in most teams: a complete written record, and agents that measure their own work.

SECTION 04

What was delivered, measured

The counts were taken at the tag `submission`. “Code lines” means non-blank lines that are not comment-only lines. The commands to reproduce every number are in section 11.

WHAT	FILES	PHYSICAL LINES	CODE LINES (OR WORDS, FOR DOCS)
Go, production	303	55,295	**42,036**
Go, tests	456	69,789	**58,956**
Shell (lab, Action, scripts)	97	9,078	6,856

WHAT	FILES	PHYSICAL LINES	CODE LINES (OR WORDS, FOR DOCS)
TypeScript / JavaScript (docs site, tooling)	31	2,015	1,658
Python and SQL	24	957	732
Build and configuration (YAML, TOML, Docker, Make)	18	1,205	900
Product documentation (README, `docs/`, ADRs, test reports)	170	13,511	about 179,000 words
Build record (`hackathon/`: orders, reports, reviews, decisions, conversations)	621	55,997	about 638,000 words

Structure

WHAT	COUNT
Go packages	142
Command-line binaries in `cmd/`	9
Test functions (`func Test…`)	1,828
Architecture decision records (`docs/adr/`)	53
Test corpus cases	21 (PV-001 to PV-021), plus a held-out set
Direct Go module dependencies	9 (mostly OpenTelemetry). Version parsing, OSV handling, signing and the pipeline are written in the repository.
Lines added in that range	216,456 (616 deleted)

Two observations matter for the estimate:

- **The test code is larger than the production code** (58,956 lines against 42,036, a ratio of 1.4). This is a property of how the agents were ordered to work: an acceptance item needs a test that was run, and many reviews mutate the code to check that the tests catch it.
- **The design record is unusually large.** It has 53 ADRs and about 179,000 words of product documentation. In a human team this work is real effort, and it is usually the first thing to be cut.

SECTION 05

Code quality, measured against published baselines

Size says how much was built. It does not say whether the code is any good. Below are five measures of code quality that anyone can reproduce. Each is set next to a published baseline from industry or research. We

report weak spots too.

5.1 Test coverage

WHAT	VALUE
Statement coverage (`go test -cover`)	**80.9%**
Packages measured	51 with statements to cover, all tests passing
Production code in the measured packages	25,307 of 39,778 code lines in Go packages (64%)
Packages at or above 75%	40 of 51
Lowest-covered packages	`internal/service` 23%, `cmd/meterless-conform` 22%, `internal/agent/planner` 36%

The six agent roles are the core of the product. Each has its own package, with this coverage:

AGENT PACKAGE	STATEMENT COVERAGE
Writer	95.8%
Skeptic	91.2%
Prover	79.4%
Judge	77.9%
Investigator	77.8%
Planner	36.2%

How it was measured.

- Proven requires Go 1.26, and our measuring environment could not download it. We ran Go 1.24.7, the newest toolchain available there.
- The environment also could not fetch the OpenTelemetry modules. Every package that imports them was left out, including `internal/review`, `internal/pipeline` and `internal/cli` (about 9,800 code lines).
- So the figure covers 64% of the production code, not all of it. With Go 1.26, `go test -cover ./...` gives the full figure in one command.

The baseline.

- *Source:* “Code Coverage at Google” (Ivanković et al., ESEC/FSE 2019).
- *Finding:* Google enforces no coverage threshold, but many of its projects adopt voluntary levels:
  - **60%** project coverage is level 3;
  - **75%** is level 4;
  - **90%** is level 5.
- *Usefulness:* in the same study, 40% of 512 Google developers said coverage is “often” or “very often” useful when reviewing a change.

The comparison. Proven’s measured 80.9% is above the 75% of level 4. Its 58,956 lines of test code are 1.4 times its production code.

5.2 Tests that are checked to fail

Coverage shows that code ran under a test, not that the test would catch a defect. Mutation testing checks that: inject a defect, and see whether a test fails.

- *Source:* Google's mutation testing system (Petrović and Ivanković, ICSE 2018), used by 6,000 engineers in code review.
- *How Proven compares:* the agents' process asks for a manual form of the same check, called a **negative control**. The executor or reviewer breaks the code on purpose, shows that a named test turns red, and restores it.
- *How often:* the phrase "negative control" appears **340 times** in the status reports and reviews. 254 of those files mention a negative control or a mutation.
- *Example:* a status report records a mutated field in a test fixture, the test that went red with its message, and the restore.

5.3 Comments and API documentation

WHAT	PROVEN	BASELINE
Comment density, CL / (CL + SLOC)	**18.7%** (9,682 comment lines, 42,036 code lines)	18.67% mean, 16.74% median, in more than 5,000 open-source projects
Exported Go identifiers with a doc comment	**96.1%** (1,490 of 1,550)	Go's guidance: "Every exported (capitalized) name should have a doc comment."
Go files formatted by `gofmt`	758 of 759	`gofmt` formatting is the Go community norm

- **Comment density.** The baseline is Arafat and Riehle, "The Comment Density of Open Source Software Code" (ICSE 2009). Their definition is comment lines divided by comment lines plus source lines, which is the formula we used.
- **What the numbers do not say.** A comment can be accurate or stale, and no count can tell which. The doc-comment figure shows that the public API is described; it does not grade the descriptions.

5.4 Complexity

We measured McCabe cyclomatic complexity for every one of the 2,075 production Go functions.

WHAT	VALUE
Mean complexity	5.1
Median complexity	3
90th percentile	11
Functions above 10	230 (11.1%)
Functions above 15	93 (4.5%)
Median function length	11 lines
Functions longer than 100 lines	18

The baseline.

- *Source:* NIST Special Publication 500-235 (Watson and McCabe, 1996).
- *Guidance:* a limit of **10** per function, as McCabe proposed, "has significant supporting evidence". Limits up to **15** have been used successfully, but should be reserved for projects with experienced staff, formal design, code walkthroughs and a comprehensive test plan.

The comparison.

- 88.9% of Proven's functions are within McCabe's limit of 10, and 95.5% within 15.
- The outlier is known and public. The function `pipeline.review` has a complexity of 134 and is 545 lines long. Proven's README lists it under **Honest limits** as "accepted debt" to be split after 6 November.
- The next most complex functions are a query executor (58), a configuration parser (56) and a review loader (51).

5.5 Documentation

WHAT	VALUE
Architecture decision records, each with context, decision and consequences	53
Product documents in `docs/`: install, configuration, architecture, adapters, output formats, air-gapped mode, evidence pack, lab, and more	22
Words of product documentation, including ADRs and test reports	about 179,000
Generated settings reference, built from the code so it cannot drift	yes (`cmd/settings-doc`)
Honest limits section, dated and updated with every push	yes

The baselines.

- *GitHub Open Source Survey 2017* (5,500 randomly sampled respondents from more than 3,800 projects): **93%** had seen incomplete or outdated documentation, which the survey calls a pervasive problem.
- *DORA's research on documentation quality*: documentation quality drives the implementation of every technical practice DORA studied. Teams with above-average documentation gain far more from those practices. For example, continuous integration was associated with gains of 750% for teams with above-average documentation, against 34% for teams below average.

The comparison. In a six-day build, documentation is normally the first thing cut. Here it was built and tested with the code. The settings reference is generated from the code. A Go test (`lab/docs_generated_test.go`) runs the documentation site's generator and fails when a committed page differs from what the generator writes, so a stale page turns the test suite red.

5.6 What these numbers mean together

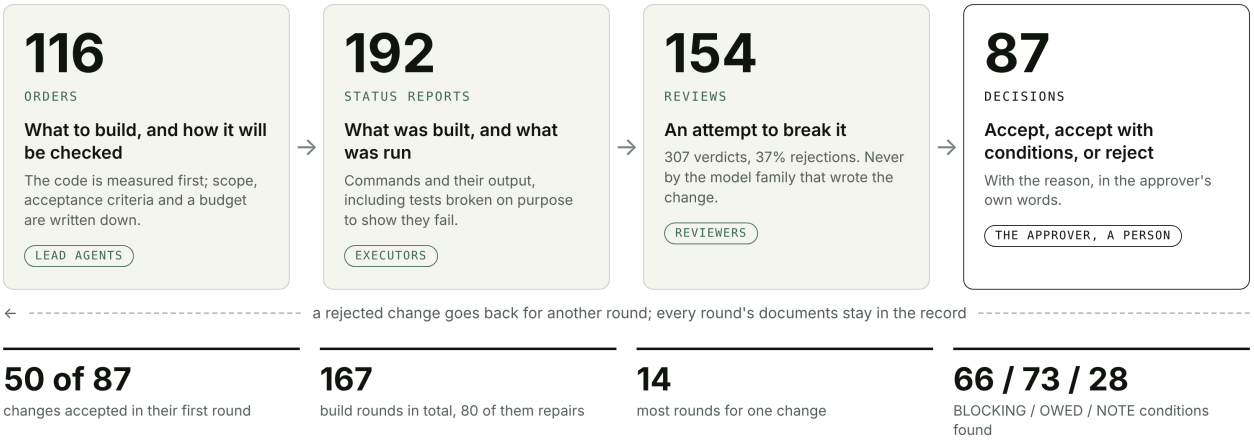
- On the measures where published baselines exist, Proven is **at or above them**:
  - coverage, on the measured part, is above the 75% of Google's level 4;
  - comment density matches the open-source mean;
  - nine in ten functions are within McCabe's limit.

- On one measure, **API documentation (96%)**, it is close to what Go's guidance asks for.
- Its weak spots are visible and named: one very large function, and a few packages with low coverage. That is what a buyer should expect to find, and the README lists it.

SECTION 06

The record of the work

The build record is in `hackathon/fleet/`. This is the part a reader cannot get from any other project of this size: every step of the work, in writing.



Sources: hackathon/fleet/ (orders, status, reviews, decisions) and LEDGER.md at tag submission; 67 of the 116 orders were verified by their lead agent instead of a separate reviewer (section 3).

Figure 2. Every change went through four public documents, each written by a different author. Sources: `hackathon/fleet/` and `LEDGER.md` at the tag `submission`.

DOCUMENT TYPE	FILES
Orders (`ORDER_*`), each with a budget and a cap	116
Train orders (groups of related changes)	9
Executor status reports	192
Reviews (on 45 changes; the rest were verified by their lead agent)	154
Approver decisions	87

Review verdicts. The review files record 307 verdicts:

VERDICT	COUNT	SHARE
accept	24	8%
accept-with-conditions	170	55%
reject	113	37%

The ledger, one row per change. It covers 87 changes plus one hand merge.

WHAT	VALUE
Changes accepted in their first round	50 of 87 (57%)
Build rounds in total	167
Repair rounds (a round after a review or decision sent the change back)	80
Most rounds for one change	14
BLOCKING conditions found	66
OWED conditions found	73
NOTE conditions found	28

The ledger also shows which model built each change. The orders are public, so a reader can check whether a weak result came from the model or from a badly written order.

The scale of the review problem. The README reports a further measurement, taken across all the agents' work on 25 October: 240 independent reviews raised 201 blocking problems in work an agent had called finished, and 116 were stopped before they reached the main branch. This is the problem Proven is built to solve, measured on the project that built it.

SECTION 07

What the agent build cost

Every one of the 116 orders carried a budget and a cap, set before the work started. Executor run records carry their token counts. This public edition does not publish money figures for the agent build: the comparison in this report is about calendar time and scope, and the approver's own working hours are not measured in the repository.

SECTION 08

How human developers work: what the research says

We used published, widely cited sources. Where a figure is a range, we used the range.

How much of a day is coding.

- *Source:* Microsoft Research, "Today was a Good Day: The Daily Life of Software Developers" (Meyer et al., IEEE TSE 2019), with 5,971 responses from professional developers.
- *Finding:* on a typical workday of about 9.1 hours, a developer spends about **92 minutes reading or writing code and tests**, about 17% of the day. Meetings take about 82 minutes.

How code review works at scale.

- *Source:* Google, “Modern Code Review: A Case Study at Google” (Sadowski et al., ICSE-SEIP 2018), based on about 9 million reviews.
- *Findings:*
  - the median change is about **24 lines**;
  - the median time to approval is **under 4 hours**;
  - most changes have one reviewer;
  - developers spend about **3 hours a week** reviewing.
- *Source:* the SmartBear / Cisco study of peer review.
- *Findings:* review **no more than 200 to 400 lines at a time**, at **under 500 lines an hour**. Defect detection falls at faster rates.

What that means for this repository.

- One careful human pass over Proven’s 101,000 lines of Go, at 500 lines an hour, takes about **200 reviewer-hours**, or about five weeks of one person’s full time.
- That covers a single pass. The record has 307 verdicts and 167 build rounds.
- At Google’s three hours of review a week per developer, the same reading would take a ten-person team about seven weeks.

How productivity scales with project size.

- *COCOMO II.2000*, Barry Boehm’s model (University of Southern California):
  - Effort = 2.94 × KSLOC^E person-months, where E = 0.91 + 0.01 × (sum of five scale factors).
  - With all scale factors nominal, the sum is 18.97 and E = 1.0997.
  - Schedule = 3.67 × Effort^F months, with F = 0.28 + 0.2 × (E − 0.91) = 0.318.
  - One person-month is 152 hours.
- *QSM* database of completed projects:
  - the median project has a team of 4.9 full-time people, takes 25.8 person-months over 5.1 months, and is about 11,000 logical lines;
  - productivity is **414 lines per person-month**.
- *Steve McConnell*, as reproduced by Jeff Atwood: productivity falls as projects grow. For a project of about 100,000 lines, the observed range is **1,000 to 20,000 lines per staff-year**.

What AI tools change, so far.

- *METR* randomised controlled trial (July 2025): 16 experienced open-source developers worked 246 real tasks. With AI tools allowed, they took **19% longer**, while believing afterwards that AI had made them 20% faster.
- *METR follow-up* (February 2026): 57 developers and more than 800 tasks. It suggests a speed-up with late-2025 tools, but METR calls the data “very weak evidence”, partly because developers refused to work without AI.
- *DORA 2025* State of AI-assisted Software Development, with nearly 5,000 respondents:
  - about **90% use AI** at work;
  - AI adoption is now associated with **higher throughput** but continued **higher delivery instability**;
  - the Stack Overflow 2025 survey, cited alongside it, found only 3% of developers “highly trust” AI output.

Read together, these sources make one point. An individual developer with an AI assistant gets a modest, uncertain gain. Proven was built differently: no human wrote code, many agents worked in parallel, and every change went through an independent review recorded in writing. Its calendar time cannot be explained by “a developer typing faster”.

SECTION 09

The human-team estimate

9.1 Four methods

Size inputs

- *Narrow:* 42,036 lines of production Go only.
- *Wide:* 51,282 lines, which adds the shell, TypeScript, Python and SQL that ship with the product.

Both inputs leave out the 58,956 lines of tests, the 53 ADRs and the documentation, so they understate the work.

METHOD	EFFORT (PERSON-MONTHS)	CALENDAR	TEAM
A. COCOMO II, all factors nominal	179 (narrow) to 223 (wide)	19.1 to 20.5 months	9.4 to 10.9
B. QSM median productivity (414 lines / person-month)	102 to 124	(not modelled)	
C. McConnell size range (1,000 to 20,000 lines / staff-year)	25 to 615. The geometric middle (about 4,500 lines / year) gives **112 to 137**	(not modelled)	
D. Bottom-up, by work package (table 9.2)	**134**	14 to 15 months with about 9.5 people	9 to 11

Notes on each method

- **A. COCOMO II.** “Nominal” means an average team with average process maturity. The product is security-critical, which COCOMO would rate as *high reliability* and *high complexity*. That raises effort, so nominal is the cautious choice. Compressing the schedule to COCOMO’s minimum of 75% of nominal gives about 14 to 15 months, and raises effort by 43%.
- **B. QSM.** This is a floor. QSM’s median project is about one-fifth of Proven’s size, and productivity falls as size grows.
- **C. McConnell.** The range is too wide to use on its own. Its middle agrees with methods B and D.
- **D. Bottom-up.** Implied productivity is 313 production lines per person-month, between QSM’s median and COCOMO.

Reconciled estimate: about **135 to 180 person-months**, over **14 to 20 months**, with a team of **9 to 11 people**. The outer range is 100 to 225 person-months.

9.2 Bottom-up: work packages and the roles they need

WORK PACKAGE	WHAT IT CONTAINS (MEASURED IN THE REPO)	PERSON-MONTHS
Product and architecture	53 ADRs, design documents, threat model, settings model	8
Review pipeline and six agent roles	`internal/review`, `pipeline`, `agent/*`, `generator`, `guardrail`, `scout`, `markov`, `world`, `hmem`: about 16,000 production and 22,000 test lines	30
Evidence layer	OSV / KEV / EPSS, query templates, Zetaris, evidence store, signed evidence pack, `proven verify`, forgery bench, custody	16
Locked-down Prover runner and fixer	`testrun`, `network`, `pathguard`, `agent/fixer`, container lockdown	9
Ecosystem adapters	`manifest`, `pep440`, `pep503`, `semver`, `mavenver`, `composerver`, `cargover`, `osv`	8
CLI, output formats, GitHub / Gitea Action, settings generator	`internal/cli`, `cmd/*`, `action/`, JSON, SARIF, text, PR comment	10
Gateway, telemetry, deployment	`gateway`, `telemetry`, judges gateway, Docker, Cloudflare, air-gapped mode	7
Lab	Gitea, sample pull requests with planted defects, Grafana dashboards, about 6,900 lines of shell	6
Test corpus, conformance, benchmarks	21 corpus cases, held-out set, four engine conformance suites, Dependabot comparison	10
Documentation and docs site	22 product documents and the site, about 179,000 words	8
Independent security review of the product's own code		4
Human code review of about 101,000 lines of Go, with re-reviews	at under 500 lines an hour	6
Coordination and product management (about 10%)		12
Total		**134**

The team this needs, working in parallel:

ROLE	PEOPLE	MAIN WORK PACKAGES
Tech lead / architect	1	architecture, ADRs, pipeline design, code review

ROLE	PEOPLE	MAIN WORK PACKAGES
Senior Go backend engineers	4	pipeline, evidence layer, adapters, CLI and outputs
Applied AI / LLM engineer	1	agent roles, model routing, guardrails, multi-family review
Application security engineer	1	locked-down runner, signed evidence packs, threat model, security review
QA / test engineer	1	corpus, held-out set, conformance, benchmarks
DevOps / platform engineer	1	lab, containers, CI, Cloudflare, Grafana
Technical writer	0.5 to 1	documentation and docs site
Product manager	0.5	scope, priorities, acceptance
Total	**about 10**	

Adding people does not shorten the schedule in proportion. Coordination grows with team size (Brooks), and COCOMO's schedule equation captures this: even at maximum compression, the model does not go below about 14 months.

9.3 Labour cost of the human team

- **Wage.** The U.S. Bureau of Labor Statistics gives a median annual wage for software developers of **$135,980** (May 2025).
- **Loaded rate.** Wages are 70.0% of employer compensation costs in private industry (BLS, June 2026). The fully loaded monthly cost is therefore about **$16,200** per person.
- **Applied to the estimates:**

EFFORT	LABOUR COST
102 person-months (QSM floor)	$1.6M
134 person-months (bottom-up)	$2.2M
179 person-months (COCOMO, narrow)	$2.9M
223 person-months (COCOMO, wide)	$3.6M

These figures are for a U.S. team at the median wage. Senior and security roles cost more. In lower-cost regions the figures scale down with the local rate. They exclude recruiting, tooling and office costs.

9.4 The comparison

	AGENTS UNDER ONE APPROVER	HUMAN TEAM (ESTIMATE)
Calendar time	139 hours	14 to 20 months (about 425 to 610 days)
Ratio		**roughly 70 to 100 times longer**

	AGENTS UNDER ONE APPROVER	HUMAN TEAM (ESTIMATE)
People who decide	1	1 product manager and 1 tech lead, plus 8 or more others
Written record of every decision and review	complete and public	usually partial: pull-request comments and tickets

SECTION 10

What this comparison does not claim

We would rather a reader find the limits here than in the code.

- **Lines of code are a measure of size, not of value.**
  - Agents may write more verbose code than a team would. Section 4's measurement can be audited, but this report does not judge whether a human team would need fewer lines.
  - COCOMO and QSM were calibrated on human projects, so applying them to an agent-written codebase assumes that comparable scope means comparable size.
- **Equal quality is not proven.**
  - Proven's own README lists open limits, for example that the review spine is one function of about 500 lines, and that two engines' benefits are not measured yet.
  - A human team might have shipped fewer features with fewer known limits.
- **Six days is not the whole effort.**
  - The topic and 25 design decisions came from planning on 16 to 21 October.
  - The agents run on Saphan Studio, a platform built before the hackathon and not part of this repository.
  - The approver's experience in infrastructure and security is an input this report does not measure.
- **The approver's working hours are not measured.** The approver decided and merged every change; the repository does not record how long that took.
- **Cost is not fully measured** (section 7).
- **Test coverage is measured on part of the code, with an older toolchain.** Section 5 explains why. The full figure needs Go 1.26, and the command is in section 11.

What the comparison does show is narrower and checkable: a codebase of this size and test depth, with this written record, was produced in 139 hours under one person's authority. The published research on how human teams work puts the same scope at well over a year.

SECTION 11

Check it yourself

```
git clone https://github.com/saphan-ai/proven.git && cd proven
git checkout submission

# Build window and commits
```

```
git log -1 --format=%aI start; git log -1 --format=%aI submission
git ls-tree -r --name-only prior-art | grep -c '\.go$'   # Go files before the build: 0

# Go size: physical lines, then code lines (non-blank, not comment-only)
find . -name '*.go' ! -name '*_test.go' -print0 | xargs -0 cat | wc -l
find . -name '*.go' ! -name '*_test.go' -print0 | xargs -0 cat | grep -vE '^\s*$' | grep -
vE '^\s*//' | wc -l
find . -name '*_test.go' -print0 | xargs -0 cat | grep -vE '^\s*$' | grep -vE '^\s*//' | wc
-l
grep -rhE '^func Test' --include='*_test.go' . | wc -l          # test functions
ls docs/adr | grep -c '^ADR-'                                   # ADRs

# The record
ls hackathon/fleet/orders    | grep -c '^ORDER_'
ls hackathon/fleet/status    | grep -c '^STATUS_'
ls hackathon/fleet/reviews   | grep -c '^REVIEW_'
ls hackathon/fleet/decisions | grep -c '^DECISION_'
grep -h '^- verdict:' hackathon/fleet/reviews/REVIEW_*.md | sort | uniq -c

# The tests and their coverage (Go 1.26)
go test ./...
go test -coverprofile=cover.out ./... && go tool cover -func=cover.out | tail -1

# Comment density: comment lines / (comment lines + code lines), production Go
git ls-files '*.go' | grep -v _test.go | xargs cat | awk '{s=$0; gsub(/^[ \t]+/,"",s)}
  s=="" {next} s ~ /^\/\// {c++; next} {k++} END {print c/(c+k)}'

# Formatting, and complexity (gocyclo is a third-party tool)
gofmt -l $(git ls-files '*.go')
go run github.com/fzipp/gocyclo/cmd/gocyclo@latest -over 10 . | wc -l
go run github.com/fzipp/gocyclo/cmd/gocyclo@latest -top 5 .
```

The complexity figures in section 5.4 come from our own counter, built on Go's `go/ast` package. It follows the same rules as `gocyclo`: one, plus one for each `if`, `for`, `case`, `&&` and `||`. Small differences between the two tools are possible. The doc-comment figure counts every exported function, method, type, constant and variable declaration in production Go files.

Read next:

- `hackathon/fleet/LEDGER.md`: every change, its model, its rounds and its conditions.
- Any order in `hackathon/fleet/orders/`, followed by the status, review and decision with the same name.
- `README.md` → *Honest limits*.
- `BUILD_LOG.md`: the build day by day.

REFERENCES

Sources

1. Proven repository: **https://github.com/saphan-ai/proven**, tag `submission`.
2. Boehm et al., *COCOMO II Model Definition Manual*, 2000.0, University of Southern California: **https://student.cs.uwaterloo.ca/~cs430/CII_modelman2000.0.pdf**
3. QSM, "What does a 'typical' software project look like?" (2025): **https://www.qsm.com/blog/2025/what-does-"typical"-software-project-look**

4. J. Atwood, “Diseconomies of Scale and Lines of Code” (reproducing S. McConnell’s productivity ranges): **https://blog.codinghorror.com/diseconomies-of-scale-and-lines-of-code/**
5. Meyer, Barr, Bird, Zimmermann, “Today was a Good Day: The Daily Life of Software Developers”, IEEE TSE 2019: **https://discovery.ucl.ac.uk/id/eprint/10090507/**
6. Sadowski et al., “Modern Code Review: A Case Study at Google”, ICSE-SEIP 2018: **https://research.google/pubs/modern-code-review-a-case-study-at-google/**
7. SmartBear, “Best Practices for Peer Code Review” (Cisco study): **https://smartbear.com/learn/code-review/best-practices-for-peer-code-review/**
8. METR, “Measuring the Impact of Early-2025 AI on Experienced Open-Source Developer Productivity” (July 2025): **https://metr.org/blog/2025-07-10-early-2025-ai-experienced-os-dev-study/**
9. METR, “We are Changing our Developer Productivity Experiment Design” (February 2026): **https://metr.org/blog/2026-02-24-uplift-update/**
10. DORA / Google Cloud, *2025 State of AI-assisted Software Development*, as reported by InfoQ: **https://www.infoq.com/news/2025/09/dora-state-of-ai-in-dev-2025**
11. U.S. Bureau of Labor Statistics, Occupational Outlook Handbook, Software Developers: **https://www.bls.gov/ooh/computer-and-information-technology/software-developers.htm**
12. U.S. Bureau of Labor Statistics, compensation costs for private industry workers, June 2026: **https://www.bls.gov/opub/ted/2026/compensation-costs-for-private-industry-workers-averaged-46-89-per-hour-worked-in-june-2026.htm**
13. F. P. Brooks, *The Mythical Man-Month*, Addison-Wesley, 1975 (anniversary edition 1995).
14. Ivanković, Petrović, Just, Fraser, “Code Coverage at Google”, ESEC/FSE 2019: **https://homes.cs.washington.edu/~rjust/publ/google_coverage_fse_2019.pdf**
15. Petrović, Ivanković, “State of Mutation Testing at Google”, ICSE-SEIP 2018: **https://research.google/pubs/state-of-mutation-testing-at-google/**
16. Arafat, Riehle, “The Comment Density of Open Source Software Code”, ICSE 2009 (companion): **https://dirkriehle.com/2009/02/04/the-comment-density-of-open-source-software-code/**
17. The Go Project, “Go Doc Comments”: **https://go.dev/doc/comment**
18. Watson, McCabe, *Structured Testing: A Testing Methodology Using the Cyclomatic Complexity Metric*, NIST Special Publication 500-235, 1996: **https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication500-235.pdf**
19. GitHub, Open Source Survey 2017: **https://opensourcesurvey.org/2017/**
20. DORA, “Documentation quality” capability (2022 research): **https://dora.dev/capabilities/documentation-quality/**

Saphan Studio lets organisations run AI coding and DevOps agents under human approval and keeps the record of their work: ***saphan.ai · docs.saphan.ai***