Stay Left on the AI Fail Chain for a Faster, Cheaper and Better UX

Improvement across our jSwarm cycles of smoke tests, focused repairs and prompt review: failed build steps in our test runs went from more than one in three to about one in a hundred, at lower modeled cost and with a shorter measured typical wait.



Mike Idengren · PROFESSIONAL Claude Certified Architect

Synopsis

"It has to be reliable, but it also has to be fast and cheap."

A hard fail leaves a user without the requested model; a retry or escalation can rescue the work but adds another attempt. We count all three as failures in development and focus on getting the first attempt right. In bench smoke tests, structural failures fell from more than one in three to about one in a hundred; typical wait fell 12.9% (measured), and product runtime cost per build fell 26.8% (modeled). JarviSIM supplies the examples; jSwarm is the method.

Audience: AI Solution Product Managers · CIOs and CTOs · AI Developers

The business story

JarviSIM turns a user's intent and source material into a process model. When a build fails, the user may need to try again and wait again. We wanted to know whether that was becoming less likely, and whether the improvement came with a better wait and cost.

COST · PRODUCT RUNTIME — Before: $44,076 a month. In this scenario, 10,000 users upload material and click "build"; a mid-tier model builds a BPMN process 20 times per user per month (modeled).

✓ **AFTER · $32,249 a month.** That is 26.8% lower per build and $141,925 less a year (modeled).

SPEED — Before: 58.7 seconds. That is the typical wait across the same actions in our test runs (measured).

✓ **AFTER · 51.2 seconds.** The typical wait is 12.9% lower (measured). A document's first build roughly halved, from 116.0 to 55.6 seconds.

© 2026 Lean Scaled Architects LLC | https://lsadigital.com

QUALITY · IN FOCUS — Before: more than one in three failed. From a user experience perspective, it's annoying to wait for a retry, it takes even longer to wait for a separate model escalation, and the worst case is a dead-stop failure. In our test runs, about 37% of build steps had a dead-stop failure (measured).

✓ **AFTER · almost no failures, fewer retries and escalations.** Failures dropped from 36.9% to 1.2%, retries fell from 11.3% to 3.4%, and escalations fell to almost none (measured).

Terms used in this article

Term	What it means here
Build step	One tested action that builds or changes a process model from a source.
Hard fail	A build does not deliver the requested result; failed structural checks are the test proxy used here.
Fail chain	The progression from a first attempt through recovery to a possible hard fail.

More terms

Term	What it means here
Right first time	The initial model attempt succeeds without a correction or escalation.
Retry	Another attempt; the measured retry counter also includes transport retries.
Escalation	An attempt using a stronger model when an eligible route is available. The test runs count it; its cost and wait impacts are modeled.
Primary wait	Time from a user's action to the model and revision panel appearing, excluding background work.
jSwarm · JarviSIM	jSwarm is the method: its cycles, practices, skills and scripts. JarviSIM is the product supplying the example.
Smoke test	A repeatable check that exercises product paths and records whether their steps work.
Intent	What the outcome should be, written and approved by a person.
Agent Acceptance Testing (AAT)	A script drives a product walk; an agent judges each step against written intent.
User Acceptance Testing (UAT)	A person walks the product, gives feedback and accepts or rejects the outcome.
`/jSmoke`	The jSwarm smoke cycle that exercises builds and identifies the first fault.
`/jFix`	The jSwarm repair cycle that takes a fault from diagnosis to a tested change.
`/jDebug --prompt`	The jSwarm prompt check that examines the assembled request for conflicting instructions.
`/jTest`	The jSwarm testing procedure, including the human acceptance cycle.

© 2026 Lean Scaled Architects LLC | https://lsadigital.com

Introduction

> “It has to be reliable, but it also has to be fast and cheap.”
>
> — Mike Idengren

This article uses real JarviSIM development examples to illustrate jSwarm principles, practices, skills and scripts. The results come from bench smoke tests of builds under development, not a customer deployment. At scale, small per-build costs add up, so reliability, wait and product runtime spend belong in the same discussion.

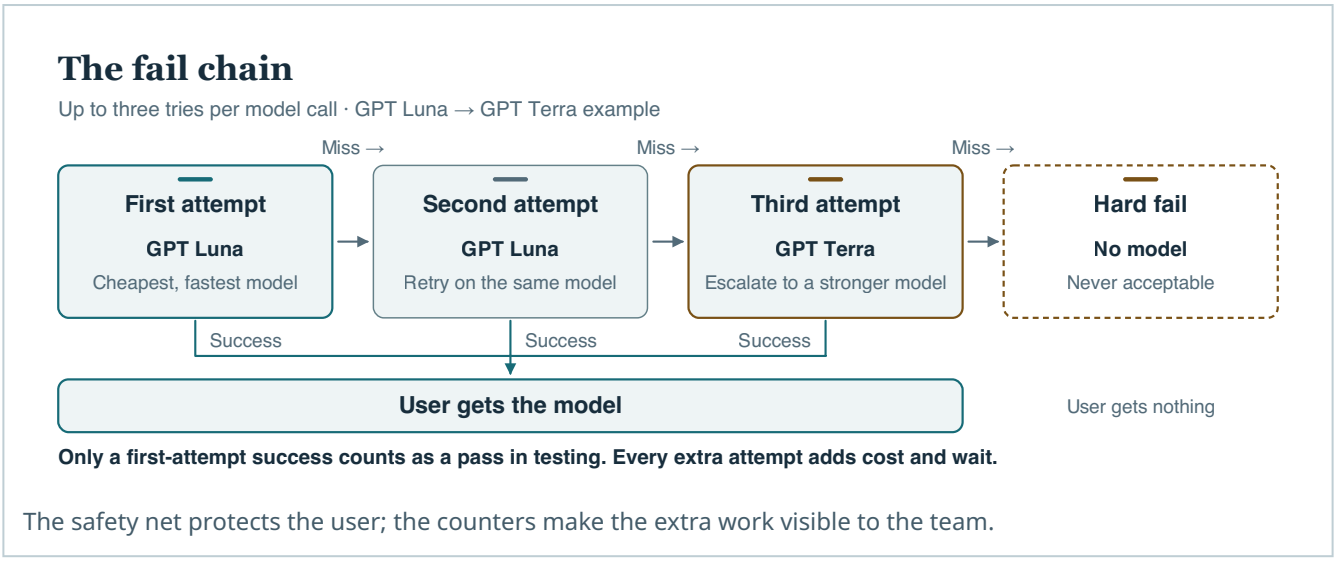
Diagram — the story from left to right: I · The fail chain → II · Before → III · How we improve → IV · After.

What we mean by intent, and how it is bound to every test and fix, is in Article 02.

Part I · The fail chain

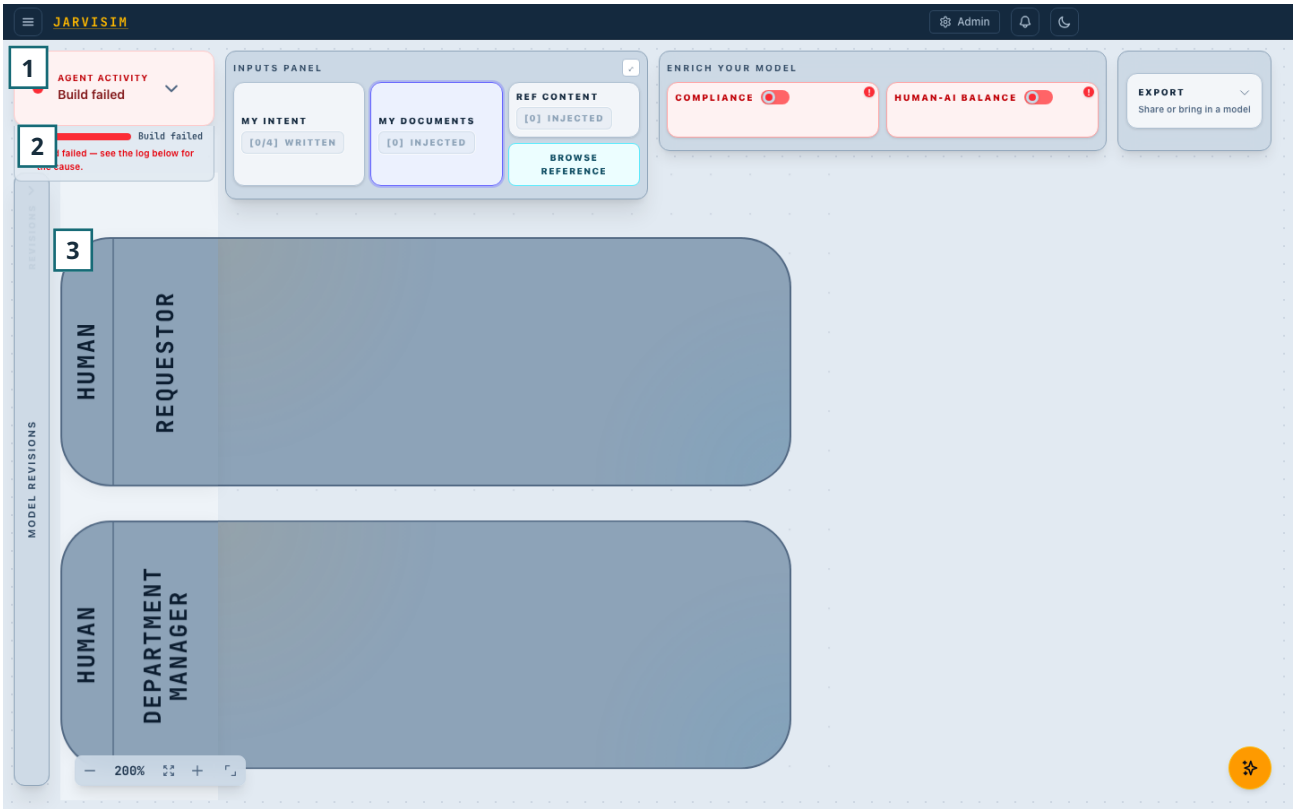
A user can start a build with their own intent, a document or a reference model.

A hard fail is never acceptable, so the product recovers in steps. The fail chain starts with the cheapest, fastest model, then a retry on the same model, then, when available, escalation to a stronger model; if recovery does not deliver the result, the build can fail. Extra model calls can add cost and, if they block the result, user wait. We count a hard fail, a retry and an escalation as failures in development, even when recovery succeeds. Fixing the first attempt can be painful during development, but at scale it avoids paying for corrections: quality, speed and cost belong together.



The safety net protects the user; the counters make the extra work visible to the team.

© 2026 Lean Scaled Architects LLC | https://lsadigital.com



The worst failure mode. The build failed and the user has no requested model.

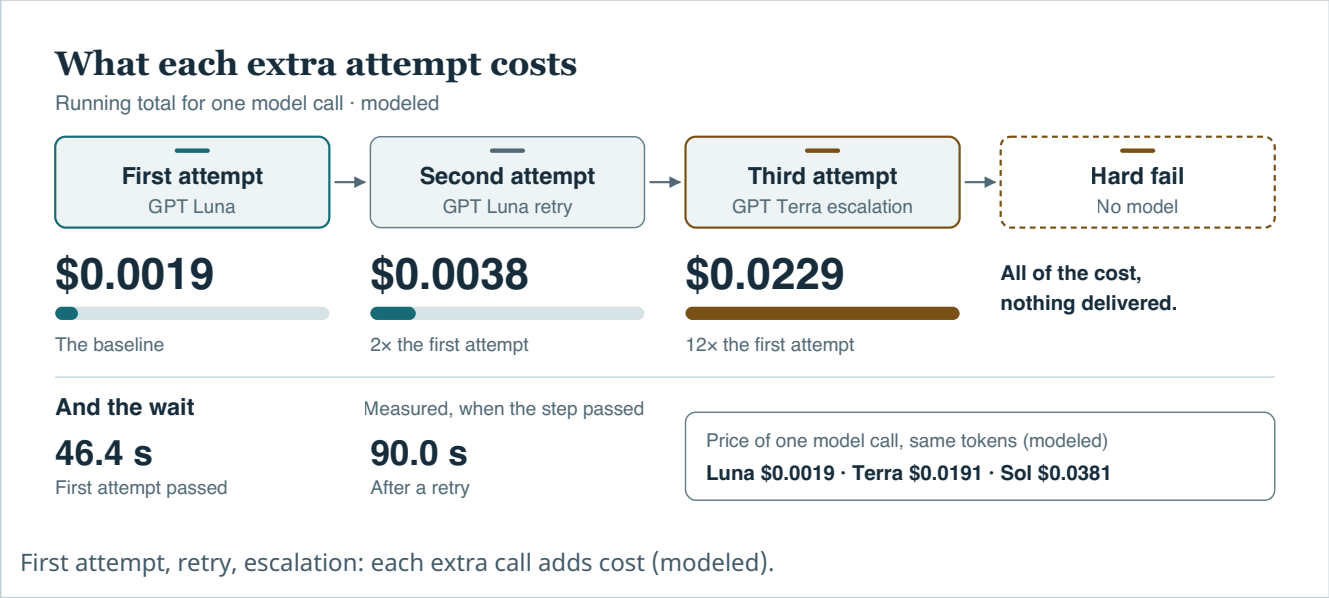
This picture is the last link in the chain: every attempt missed and the user has nothing to show for the wait.

What to look for:

1. **Agent Activity.** The build is marked failed.
2. **The message.** The user is told to see the log for the cause.
3. **The empty lanes.** The user has no model.

For an eligible model call, recovery allows one same-model correction and then at most one escalation to a stronger model when one is available; a build step can contain several calls.

© 2026 Lean Scaled Architects LLC | https://lsadigital.com



First attempt, retry, escalation: each extra call adds cost (modeled).

For one call with equal tokens, the cumulative cost is $0.0019 for a GPT Luna first attempt, $0.0038 after a Luna retry, and $0.0229 after escalation to GPT Terra (modeled). Luna costs less than Terra, and Terra less than Sol. Successful steps with no recorded recovery had a median wait of 46.4 seconds, compared with 90.0 seconds for steps with a retry (measured; different actions).

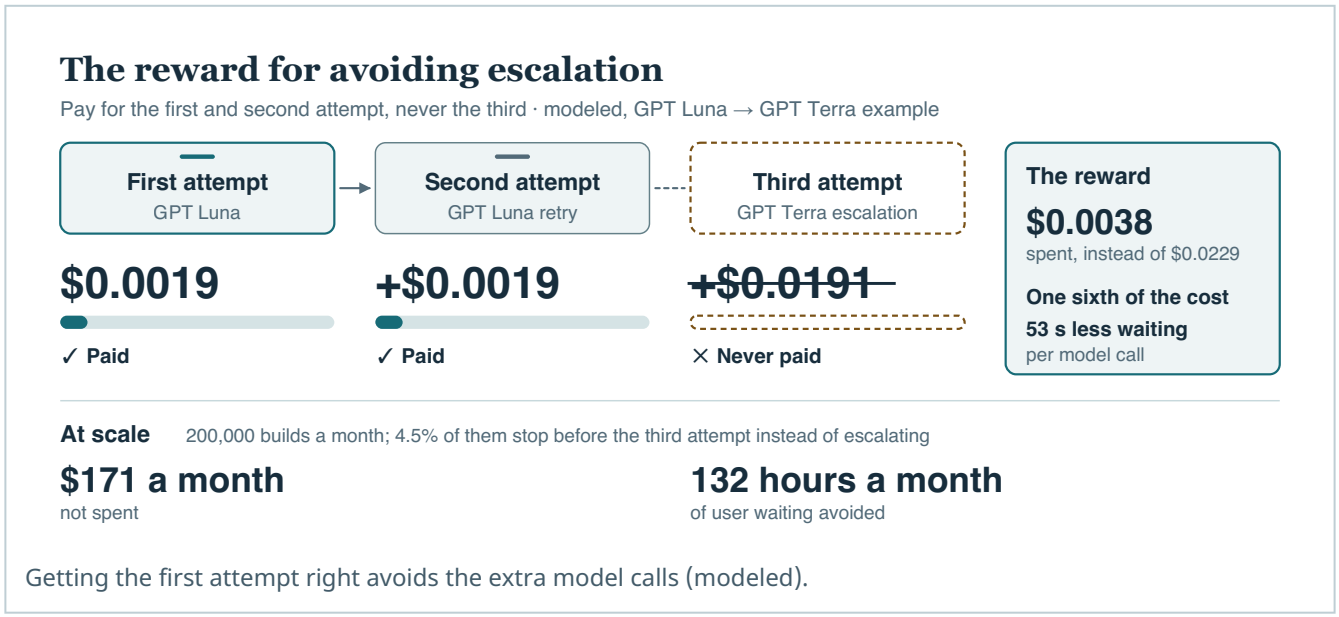
We also have fallback models from a different AI provider in case the primary provider fails. Fallback analysis is outside this case study.

Some causes were fixed before counting; with few recorded escalations, the impact is modeled.

The reward for avoiding escalation

The first and second attempts run on the cheapest model, so we pay for them at the cheap rate: $0.0019 each. The third attempt is the expensive one: escalating to GPT Terra adds $0.0191, five times the first two attempts combined, and about 53 seconds of wait. When a build succeeds by the second attempt we never pay that. At 200,000 builds a month, if 4.5% of builds stop before the third attempt instead of escalating, that avoids about $171 and 132 hours of combined wait a month; on the GPT Terra → GPT Sol tier pair, about $343 and 256 hours (modeled). The modeled scenario is not added to the separate build-cost saving below.

© 2026 Lean Scaled Architects LLC | https://lsadigital.com



At the same modeled volume, the overall build-cost comparison across selectable model tiers is:

Model	Per build before / today	Monthly before / today	Yearly saving
GPT Luna	$0.0222 / $0.0163	$4,444 / $3,252	$14,310
GPT Terra	$0.2204 / $0.1612	$44,076 / $32,249	$141,925
GPT Sol	$0.4408 / $0.3225	$88,152 / $64,498	$283,850

Modeled build costs; these savings mostly reflect assumed user resubmissions after failures, not measured escalation savings.

© 2026 Lean Scaled Architects LLC | https://lsadigital.com

Part II · Before — High failures and a bad UX



Before: the fail chain's quality, measured wait and modeled cost.

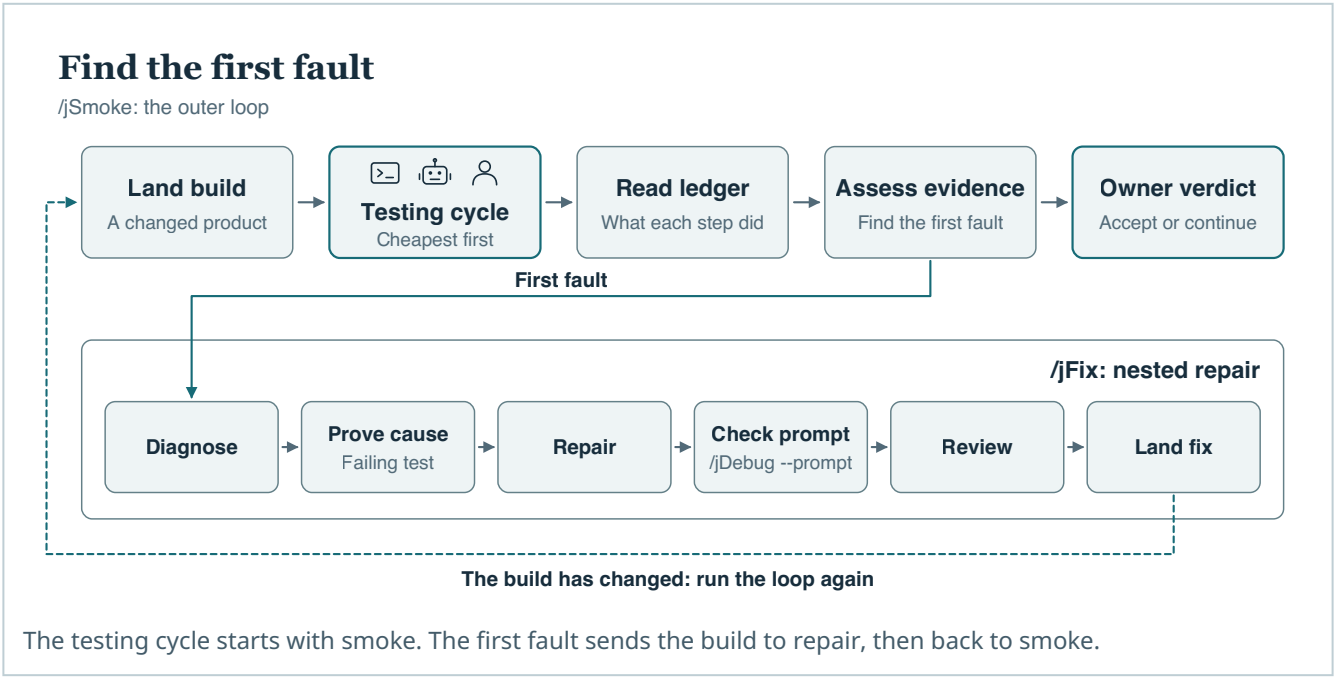
In our test runs, 36.9% of 122 build steps failed their structural check in the first measured period (measured). The earlier failed-build picture shows the worst experience such a failure can represent.

Part III · From Before → After with JSWARM

An Agentic Harness Framework

Our objective was to drive down the failures, waiting and cost in the before picture. We use cycles to find the first fault, repair it and test the result, then repeat until the outcome is acceptable.

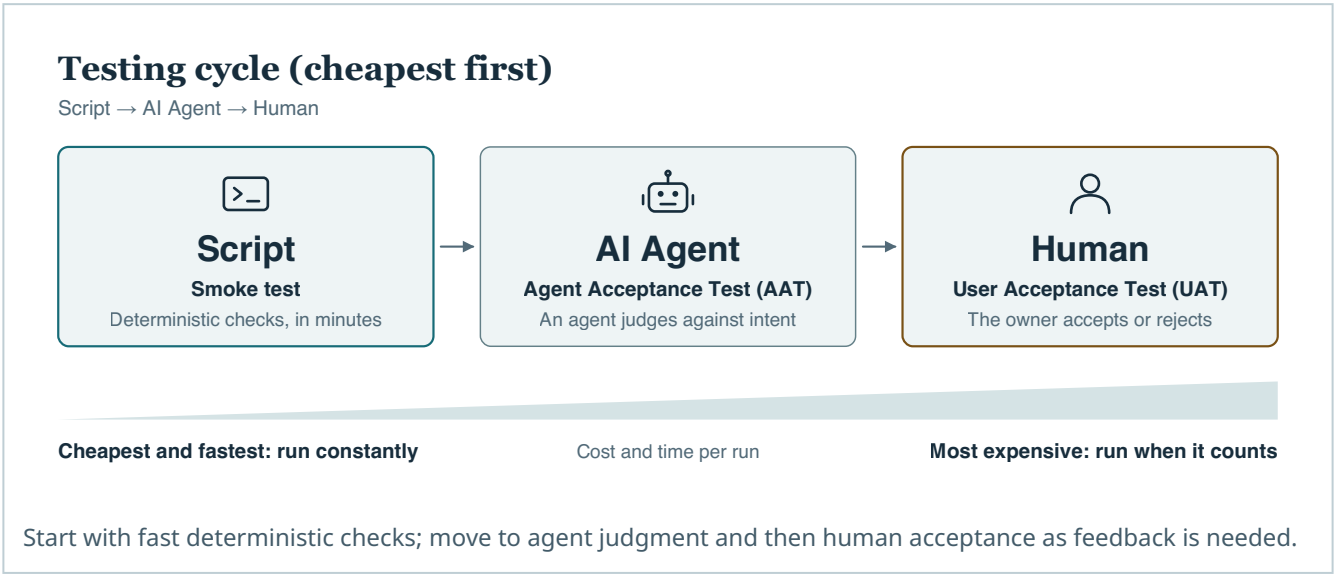
© 2026 Lean Scaled Architects LLC | https://lsadigital.com



The testing cycle starts with smoke. The first fault sends the build to repair, then back to smoke.

Testing cycle (cheapest first)

Script → AI Agent → Human: each check adds a different kind of judgment.



Start with fast deterministic checks; move to agent judgment and then human acceptance as feedback is needed.

We iterate rapidly with the cheapest, fastest smoke tests first, fix defects, then use Agent Acceptance Testing (AAT) when model judgment is needed and User Acceptance Testing (UAT) for human feedback and acceptance. Each catches something different.

Three checks, three catches

Check	Who judges	What it caught
Smoke test	Automated script	Four document-add steps escalated and still failed their structural checks (measured).

© 2026 Lean Scaled Architects LLC | https://lsadigital.com

Check	Who judges	What it caught
Agent walk	An agent against written intent	Compliance did not preview its whole effect before consent, despite a green smoke check.
Owner review	The human owner	A polite refusal passed a machinery-focused check, but was not the requested outcome.

A script drives AAT and a model judges each step against written intent. In UAT the owner walks the product, gives feedback and accepts or rejects the outcome. `/jTest` carries the cycle; feedback returns to intent.

Find the first fault

`/jSmoke` exercises the build paths and reads the ledger. When it finds the first fault, `/jFix` takes that fault into a nested repair. Once the fix lands, the build has changed, so the outer loop starts again. The owner decides when the outcome is acceptable.

The ledger gives the team a starting point, not just a score. A failure on one source directs the next repair toward the first fault found on that path. The failing test makes the diagnosis actionable: it must demonstrate the cause before the repair can be checked against it. Once the repair is in the build, another smoke run shows what happens on the changed product. The owner's verdict remains part of the loop rather than an automatic consequence of a green counter.

Rule one: meaning to the model, mechanics to code

We ask the model for judgment, not for a mechanical fact code can derive. Identities, counts, ordering and wiring can be handled by code. That leaves the model to do the work that needs interpretation. Article 01 explains this split: model for judgment and planning, deterministic tools for execution.

In one JarviSIM example, identical commitments needed normalization. Previously the product sent them back to the model for a repair. We changed the code to normalize them directly instead. In our test runs, alongside this and other changes, the retry rate among steps where retries were counted fell from 11.3% to 4.6% in the next period (measured). This is an observed association, not a claim that this change alone produced the drop.

The practical difference is where the product spends its judgment. A model can help understand a user's meaning. It need not be asked to repeat a mechanical operation the product can perform consistently. That distinction also makes a retry more meaningful: it is no longer the first choice for a task code can resolve.

More: why the split matters

A process model contains both judgment and structure. The judgment is deciding what the user's words mean for the process. The structure includes the identities and connections that make the result usable. The two belong together in the result, but they do not have to be produced in the same way.

With identical commitments, the product already had the information needed to recognize and normalize them. Asking the model to repair them introduced another attempt where code could do the job. Moving that operation to code did not take the meaning away from the model; it stopped asking for a mechanical repair. The retry counter then shows what happened to retries across the test steps, rather than presenting that one change as a standalone experiment.

© 2026 Lean Scaled Architects LLC | https://lsadigital.com

Rule two: fix the first call

A long document was failing because the first call's output limit was too small for the work requested. A bigger model or an extra attempt would not fix that mismatch. We sized the initial output budget to the document instead of simply adding retries.

In our test runs, alongside this and other changes, Document failures fell from 34.5% to 11.3% to 0.9% across successive periods (measured). Across all sources, we counted 4 escalations in 89 early covered steps and 0 in 385 latest covered steps (measured). The numbers show what happened alongside the work, not the isolated effect of this repair.

Retry and escalation remain the user's safety net. Counting recovery as a development failure gives the team a reason to investigate, while preserving another chance to deliver the outcome.

jDebug: A tool for getting it right the first time

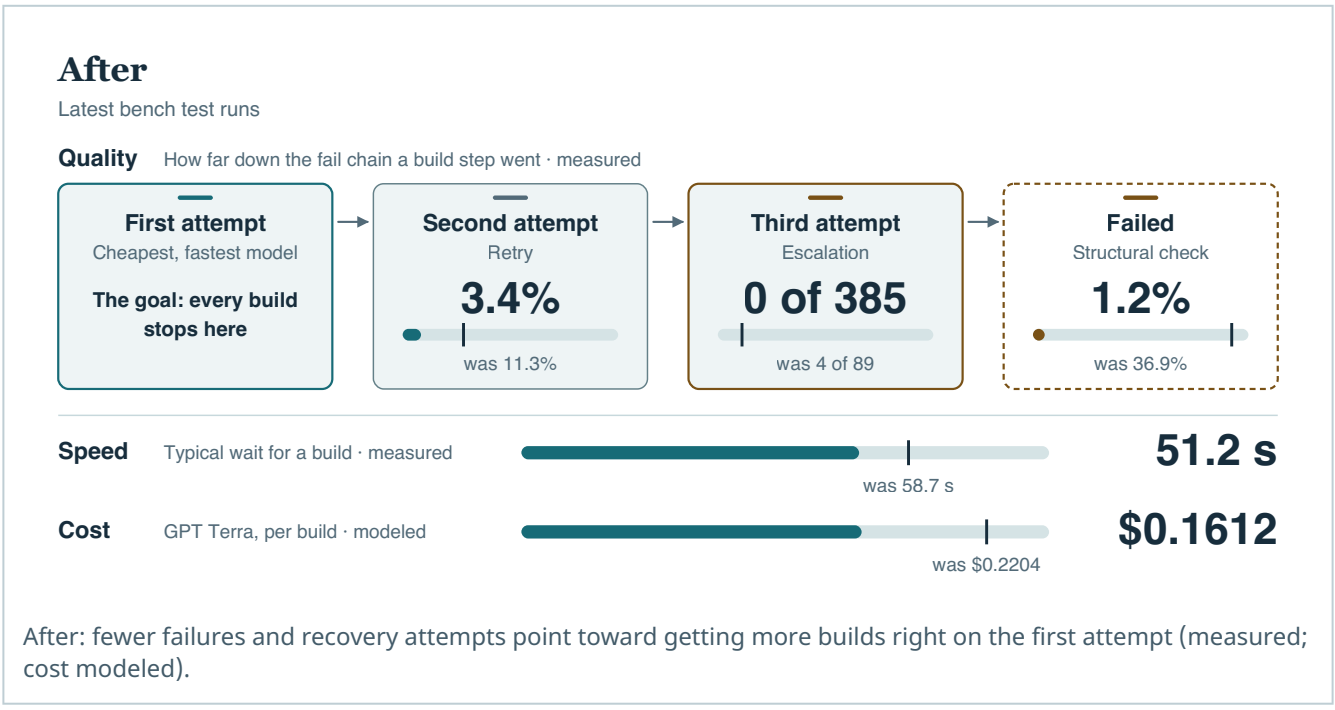
`/jDebug --prompt` checks the assembled request for conflicting instructions before a prompt or schema change is reviewed. A promptAudit applies that assessment across prompt families in risk order, helping the team fix the request before relying on another attempt.

After the highest-risk prompt fixes landed alongside other changes, the failure rate in our test runs went from 13.0% to 2.1% to 0% on successive days (measured). The six-smoke run passed first time with no repairs; the changes were not isolated.

Part IV · After — From Fail Chain to First Attempt

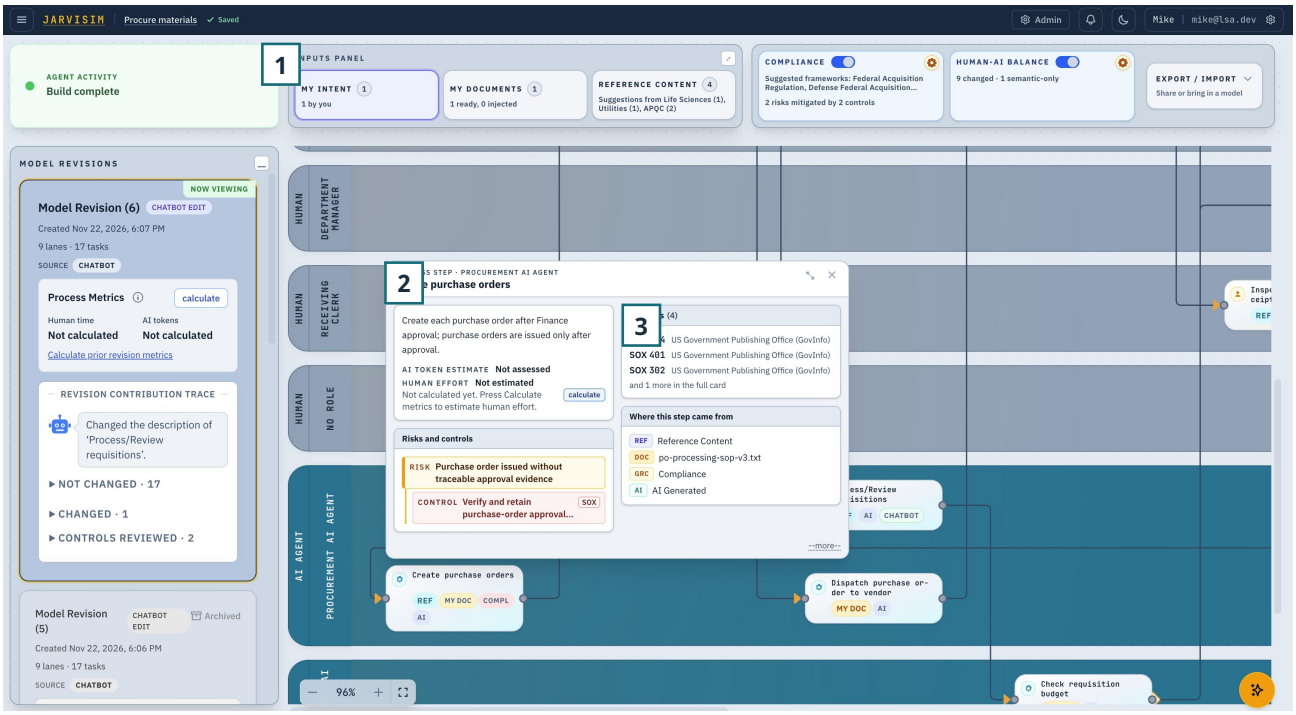
We improved the first request, moved mechanical fixes into code and repeated the smoke, repair and testing cycles. Fewer structural checks failed and fewer covered steps recorded retries or escalations, moving the product toward getting the first attempt right.

© 2026 Lean Scaled Architects LLC | https://lsadigital.com



After: fewer failures and recovery attempts point toward getting more builds right on the first attempt (measured; cost modeled).

What a successful model build looks like



The model after Compliance and Human-AI. A completed build shows the process and its supporting details.

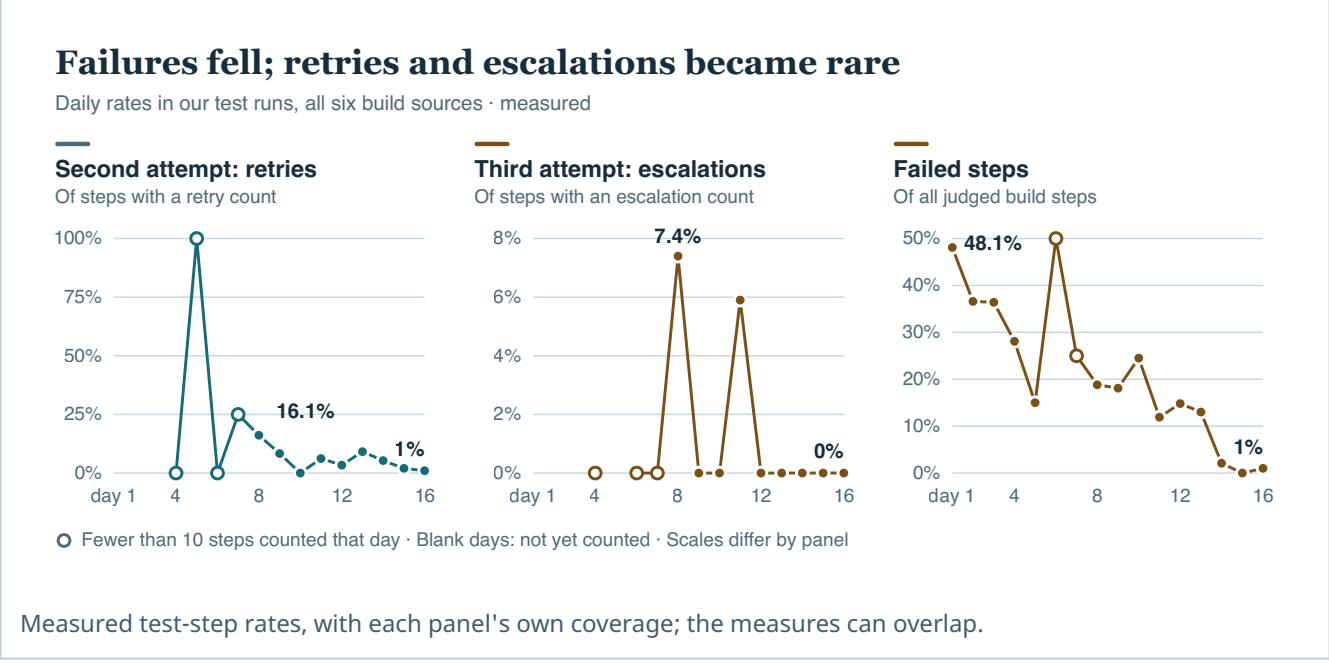
What to look for:

1. **Build complete.** Agent Activity confirms completion and the revision panel shows the current model.
2. **Process and task.** Human and procurement AI agent lanes contain the process; the open purchase-order task shows its risk and control.

© 2026 Lean Scaled Architects LLC | https://lsadigital.com

3. **Evidence.** The task card lists citations and sources, while the top panels show Compliance and Human-AI Balance.

In our test runs, structural failure rates moved down across cycles, though not every day. The retry and escalation panels show the same days where those counters were available; gaps mean not measured, not zero. Of the steps where we counted retries, the retry rate fell from 11.3% of 97 to 3.4% of 386; we recorded 4 escalations in 89 early covered steps and 0 in 385 latest covered steps (measured). Recovery counters cover smaller samples and started later than failure counts.



Measured test-step rates, with each panel's own coverage; the measures can overlap.

More (raw data): daily fail chain

The failure rows below cover all judged steps; retry and escalation counts use separately covered steps, with dashes for unavailable measurements.

Test day in order	Failed / judged steps in our test runs (measured)	Failure rate (measured)
1	13 / 27	48.1%
2	15 / 41	36.6%
3	8 / 22	36.4%
4	9 / 32	28.1%
5	3 / 20	15.0%
6	4 / 8	50.0%
7	1 / 4	25.0%
8	26 / 138	18.8%
9	13 / 72	18.1%
10	25 / 102	24.5%

© 2026 Lean Scaled Architects LLC | https://lsadigital.com

Test day in order	Failed / judged steps in our test runs (measured)	Failure rate (measured)
11	16 / 134	11.9%
12	39 / 264	14.8%
13	17 / 131	13.0%
14	4 / 191	2.1%
15	0 / 125	0.0%
16	1 / 99	1.0%

Test day in order	Retry-positive / measured	Escalation-positive / measured
1–3	—	—
4	0 / 8	0 / 8
5	1 / 1	—
6	0 / 8	0 / 8
7	1 / 4	0 / 4
8	10 / 62	4 / 54
9	1 / 12	0 / 12
10	0 / 23	0 / 23
11	1 / 16	1 / 17
12	8 / 232	0 / 229
13	5 / 55	0 / 55
14	10 / 188	0 / 187
15	2 / 99	0 / 99
16	1 / 99	0 / 99

All counts measured in bench test runs.

A shorter typical wait, with different work by source

The wait clock starts when the user acts and ends when the model and revision panel appear. Across the same 14 actions, typical wait fell from 58.7 to 51.2 seconds, a 12.9% reduction (measured). The document first builds drove the gain: waits fell from 116.0 to 55.6 seconds on one action and from 107.2 to 51.7 seconds on another.

© 2026 Lean Scaled Architects LLC | https://lsadigital.com



Document processing improved most; the recent Compliance and Human-AI waits rose alongside added work.

More (raw data): wait by source

Source	Before, typical wait	Latest, typical wait	Reading
Document	129.2 s	96.0 s	Faster
My Intent	28.4 s	29.7 s	No clear change
Reference	36.8 s	37.1 s	No clear change
Compliance	74.9 s	84.7 s	Slower, doing more work
Human-AI	66.1 s	75.2 s	Slower, doing more work
Chatbot	3.8 s	3.8 s	No clear change

Wait figures measured in our test runs.

Each comparison uses the same actions and inputs; the Compliance and Human-AI rows use the recent comparison. Each action gets equal weight. The work itself also changed.

Two sources now take a few seconds longer while each action produces more. The added work helps explain the rise, and the owner decides whether it is worth the wait.

Compliance. A suggestion now attaches each control with the exact framework sections it cites and a reason it applies, so a reviewer can check it rather than just trust it. The screen above shows what that looks like. One action adds 3 controls instead of 1 (measured). Time per 1,000 tokens written stayed about the same: 12.1 seconds before and 11.8 after (estimated).

Human-AI. The review now reads a model nearly twice as large because it carries the compliance controls and citations. In one action the input grew from 45 to 100 KB; across the source, it wrote about 21% more guidance (measured). Time per 1,000 tokens written was 10.6 seconds before and 9.9 after (estimated).

© 2026 Lean Scaled Architects LLC | https://lsadigital.com

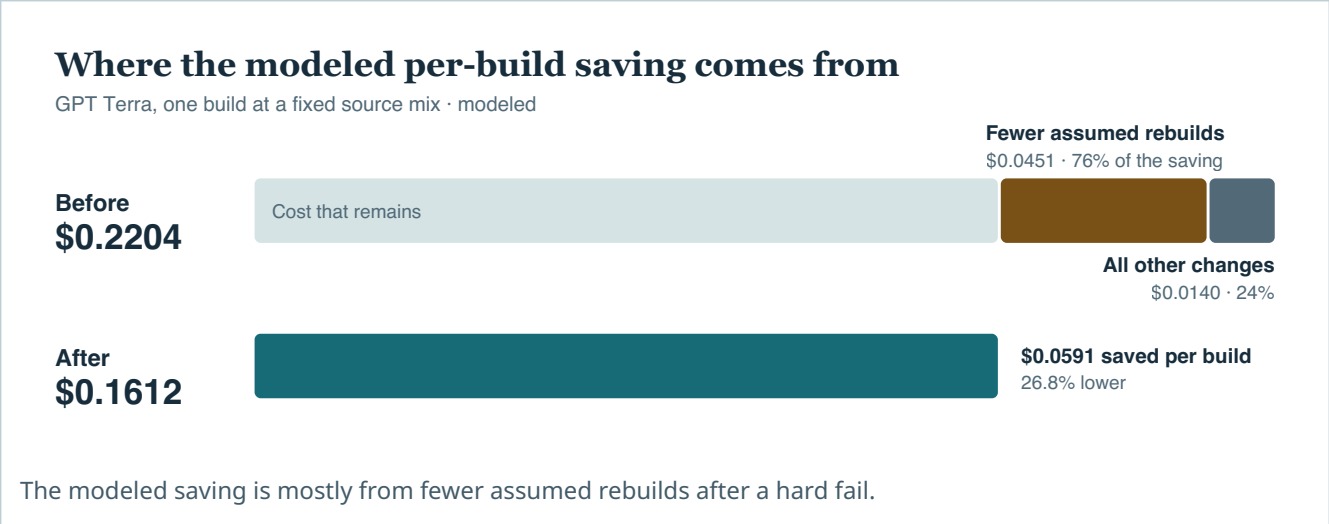
Document. Recent builds also took a few seconds longer alongside added work: every stated step in a procedure becomes a task, and each task keeps its supporting passage from the document.

At our modeled volume: what the improvement saves

We assume 10,000 users each make 20 builds a month: 200,000 builds a month, or 2.4 million a year (modeled). The user has chosen GPT Terra. The measured starting point is the fully priced median build cost on GPT Luna, mixed equally across the sources; the Terra figures below are modeled from that base.

Product runtime comparison	Before	Today	Difference
Per build	$0.2204	$0.1612	26.8% lower
200,000 builds per month	$44,076	$32,249	$11,827 saved per month
2.4 million builds per year	—	—	$141,925 saved per year

All figures modeled.



Of the modeled $0.0591 saving per build, about three quarters ($0.0451) comes from fewer hard fails under the assumption that a user builds again after a hard fail; $0.0140 is everything else. On that assumption the hard fail is the most expensive link in this modeled comparison, not a measured customer bill. The escalation impact is modeled separately in Part I; it is not added to this saving.

Human judgment is redirected, not replaced. The owner ruled what counts as a failure and decides whether added work is worth added wait.

The design lesson

> When the model misses, fix the prompt before you buy a bigger model.

© 2026 Lean Scaled Architects LLC | https://lsadigital.com

Count the failure, preserve the safety net and make the first call more capable of succeeding. Give judgment to the model and derivable mechanics to code. Then check the product with scripts, agent judgment and a human owner, each with a different job.

Region map

Visual	Region	What to look for
Earlier failed build	1 · 2 · 3	Agent Activity, failure message and empty lanes.
Modeled escalation reward	Whole chart	First attempt and third attempt; modeled extra recovery cost.
Successful build	1 · 2 · 3	Completion and revision, process and task, citations and sources.

For Claude Certified Architect–Professional readers

Anthropic's guidance on developing tests (https://platform.claude.com/docs/en/test-and-evaluate/develop-tests) connects evaluation with the outcomes a system should achieve. We apply that idea to a real product: count structural failures, inspect the first fault, and complement repeatable smoke checks with judged walks and human acceptance.

Article 04 · Quality · Product runtime. Count hard fails, retries and escalations, then improve the cheapest model's first attempt. Across jSwarm cycles, bench tests showed fewer structural failures and a shorter measured typical wait, alongside lower modeled runtime cost. With few recorded escalations, their impact is modeled separately.

© 2026 Lean Scaled Architects LLC | https://lsadigital.com