WORKED EXAMPLE 02 · STATIC PRINT EDITION

What we mean by “intent” (and why every test and fix starts there)

Written, owner-approved outcomes — Given/When/Then scenarios, non-functional requirements and numbered rulings — bound to every feedback, fix and test, so people decide what “right” means and AI proves it.



Mike Idengren



Synopsis

A buyer asks, “Did the AI do the right thing?” We can only answer because “right” was written down first. In the jSwarm method that written “right” is called intent: the journey a user needs, Given/When/Then (GWT) scenarios, non-functional requirements, semantic scenarios, and the owner’s numbered decisions. Every test, every fix and every acceptance walk points back at it. jSwarm is the method; JarviSIM is the product we build with it. This article uses real JarviSIM development examples to illustrate jSwarm principles, practices, skills and scripts. It follows one feature from the owner’s words to a judged result, and names the agent responsible at each step. Article 01 showed the AI changing the right thing; this is where “right” comes from.

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

The business story

COST

Pay once, not twice

Writing intent is human time well spent. The alternative is paying for the wrong thing and then paying again for the fix.

SPEED

Fewer rework cycles

A clear target means fewer fix cycles that solved the wrong problem.

QUALITY · IN FOCUS · PRODUCT AND DEVELOPMENT

Quality is only measurable against a written target

Here the target is a GWT scenario a person approved. The Agent Acceptance Test (AAT) walk reports every clause of it separately.

✓ **INTENT BINDING · CHECKED**

Every identifier in this article resolves in the current catalogs and the decision register, in both directions: scenario to requirement and journeys; defect to scenario; test to scenario; decision to scenario.

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

Terms used in this article

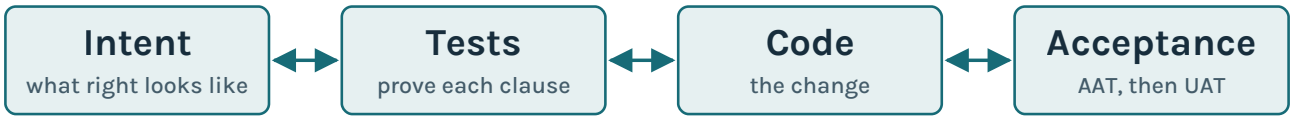
Term	In plain English
Intent	What “right” looks like, written down and approved by a person before the work is done.
Given/When/Then (GWT) scenario	An acceptance test in words: the starting state (Given), the action (When) and every result that must hold (Then). Each of those lines is a GWT clause.
AAT and UAT	Agent Acceptance Test (AAT): an agent walks the product step by step and reports, clause by clause, what was met. User Acceptance Test (UAT): a person walks the product and accepts or rejects it. UAT is the final authority.
jSwarm · JarviSIM	jSwarm is our method: the loop, principles, practices, skills and scripts. JarviSIM is our product. It supplies the real development examples in this article.
Non-functional requirement (NFR)	A quality the product must keep whatever the feature: cost, speed, durability — with a written pass/fail rule.
Journey	The outcome a user needs end to end, such as “build a process from a document and improve it”.
Semantic scenario	Any combination of a structural outcome and a judged quality of the model’s response. Example 1: a chatbot reply plus the model change it makes. Example 2: a user writes their intent in their own terms, and the output must be a valid process model in BPMN (Business Process Model and Notation) that carries the user’s terminology, scope, industry and outcomes.
Owner decision	A numbered question with options and trade-offs, closed by the owner in their own words.
jTestEngineer	The agent that drafts scenarios and writes tests, and links each test to the intent it proves.
jArchitect / jOracle	The agent that writes the fix contract: in plain English, what a correct fix must achieve.
jCoder	The agent that changes the code.
jQATester	The agent that runs the AAT walk and judges each GWT clause of the scenario.
jQAScorer	The agent that grades the model’s responses in semantic scenarios against a frozen rubric.
`/jFeedback` · `/jFix` · `/jTest`	jSwarm procedures: `/jFeedback` records feedback and checks it against intent; `/jFix` takes a fix from intent check to proof; `/jTest` runs the tests and the walks.

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

AI speeds up complex software engineering, but without intent alignment, it will make assumptions and deliver the wrong thing - causing time-consuming, expensive rework.

Intent, tests, code and acceptance line up

Four things must agree. Intent says what right looks like. Tests prove each GWT clause of it. Code is the change under test. Acceptance is the judgment that the clause is met — first by an agent, then by a person.



Each one points at the next and back again. A test without intent proves nothing; a fix without a test is a guess.

Diagram 1. Intent, tests, code and acceptance point at each other. Below, each of the three outer boxes breaks down into its kinds.

Kinds of intent

- **Journey** — the outcome a user needs end to end.
- **Given/When/Then scenario** — the acceptance test in words.
- **Non-functional requirement** — a quality to keep, with a pass/fail rule.
- **Semantic scenario** — a structural outcome plus a judged quality of the model’s response.
- **Owner decision** — a numbered question closed in the owner’s words.

JarviSIM examples: what they look like

These examples come from JarviSIM, the product we build with the jSwarm method. They show what each kind of intent looks like once it is written down and kept as a record.

Journeys and scenarios are catalogs. The current intent index for this product area lists 33 journeys, 160 GWT scenarios and 74 semantic scenarios (measured, counted in the index). It is regenerated on every change to the backlog. jTestEngineer drafts them; the owner approves them.

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

A semantic scenario, SS-47. The user accepts a chatbot recommendation with *Use this recommendation*, then presses *Apply*. The expected outcome: the product finishes the change and says truthfully whether it did, and nothing changes the model before the user presses Apply. SS-47 links to two GWT scenarios and to three defects with their states: one fixed, one deferred, one registered. It is the conversation behind Article 01.

An owner decision, D282. The chatbot was asked to make several recommendations in one answer. The design that worked end to end needed a second step. That step pushed the first answer past its 55-second limit (measured). The alternative, one call on a stronger model, also broke the limit. The owner’s verdict was “c”: adopt the two-step design now and keep the one-call route as a measured fallback. The decision register holds 282 such decisions (measured, counted in the register), each with the question, the options, the trade-offs and the owner’s verbatim words.

Kinds of tests

- **Unit** — one piece of code, checked on its own.
- **Integration** — pieces working together through the real stack.
- **Semantic scenario, structural** — did the product reach the right state? Deterministic checks.
- **Semantic scenario, semantic** — was the model’s response right? Graded against a rubric.

JarviSIM examples: intent-to-tests

In JarviSIM the links run both ways: a test names the scenario or requirement it proves, and a scenario lists the tests that prove it. A checker verifies both directions. A test with no intent, or a GWT clause with no test, is a review gap, not a pass.

A GWT scenario and its test. The step-through test for the scenario in this article opens by naming the scenario and the GWT clause each step exercises: build, check the honest not-yet-calculated state, press calculate, reload and read back.

A semantic scenario and its grade. The structural facts — did the model change, did the product reach a truthful terminal — travel with the model’s response, so the grader cannot contradict them. A good grade from jQAScorer never overrides a failed structural check.

Kinds of acceptance

- **Agent Acceptance Test (AAT)** — jQATester walks the product and reports per clause: met, not met or not exercised.
- **User Acceptance Test (UAT)** — a person walks the product, gives feedback, and accepts or rejects. UAT is the final authority.

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

JarviSIM examples: How AAT and UAT work together

Agent Acceptance Testing is agent-judged: a script drives the product step by step and a model judges each step against a written outcome brief, so dozens of walks run between human sessions for cents a walk. User Acceptance Testing is human: the owner walks the product, gives UI/UX feedback, and accepts or rejects. They work together in one loop. Intent is written and approved by people. AAT walks it at volume and reports per clause what was met, not met or not exercised. Fixes follow. UAT accepts. Feedback goes back into intent first.

An AAT walk. In the walk this article follows, jQATester walked the calculate scenario on an earlier build and judged each GWT clause: three met, three not exercised, so the walk is recorded as partial, not as a pass (measured, judged walk record). In another walk, on the scenario that says every task built from a document must show which document it came from, all seven clauses were judged met: jQATester opened each of the nine document-built task pop-ups, reloaded the page, and checked again (measured, walk receipt).

AAT collects its own metrics. Each walk leaves a verdict for every GWT clause (quality) and the time each step took (speed); the AI proxy’s records show what the walk cost. In one JarviSIM test series a typical walk cost about 19 cents (measured). That series launched 68 walks in about 66 hours, up to seven at once, and made 769 clause judgments on 336 steps (measured). You can watch a walk as it runs, and walks run side by side in separate browsers.

A UAT round. In one round the owner walked the current build and wrote their feedback through `/jFeedback`. The record split it into 11 items and checked each against intent: three were defects against existing GWT clauses, most of the rest were requests no clause covered yet, and one was a question (measured, round record). The round’s verdict was FAIL. The feedback went back into intent first: the owner approved patches to seven existing GWT scenarios — for example, the revision-card scenario now asks for separate ADDED LANES and ADDED TASKS groups, and the reference-content scenario now says what its circled suggestion count means — and three new records were registered, each linked to the scenario it serves. Only then did fixes follow.

How this article walks through it

The loop above is jSwarm’s. To show it at work, we follow one real JarviSIM development example: should the product estimate human effort and AI tokens for every task automatically, or only when the user asks?

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

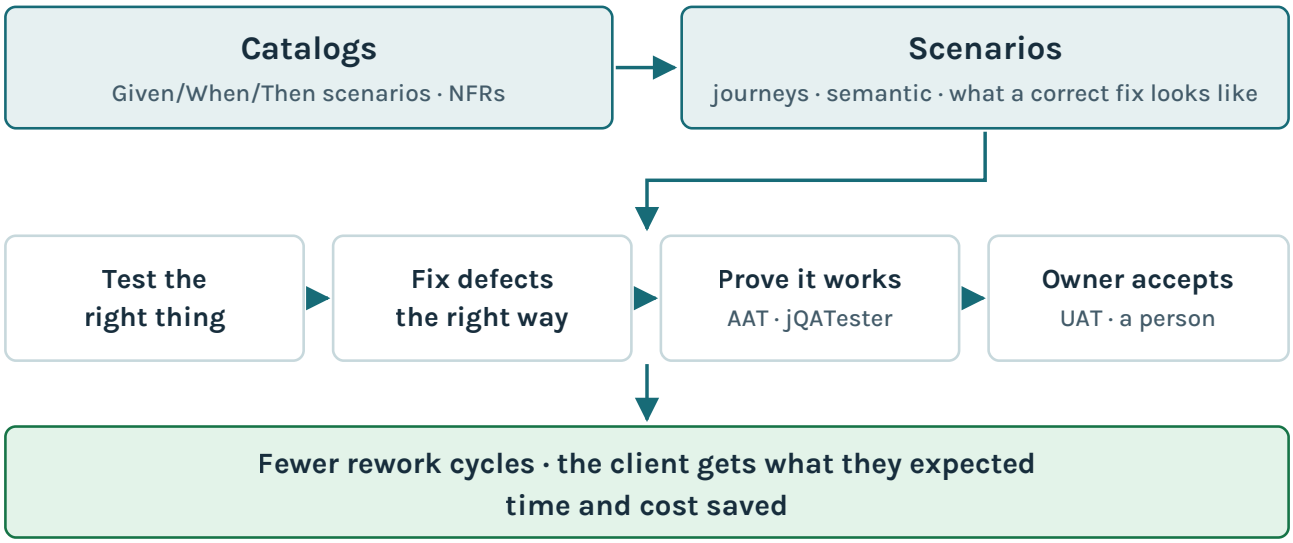


Diagram 2. The catalogs guide the scenarios and say what a correct fix looks like. So we test the right thing, fix defects the right way, prove it works (AAT) and get owner acceptance (UAT). The payoff is fewer rework cycles, a client who gets what they expected, and time and cost saved.

Real product example: from the owner’s words to a judged result

These are real JarviSIM development examples, illustrating jSwarm principles, practices, skills and scripts. Every link in the chain is real. The identifiers are the product’s own; they show the records exist. The chain has four stages.

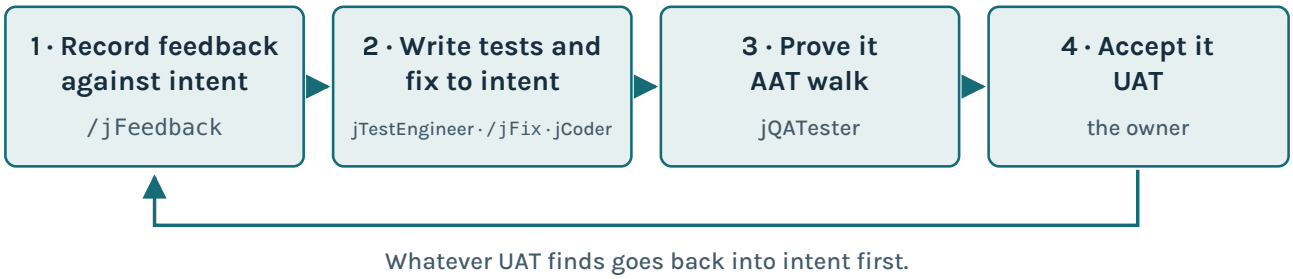
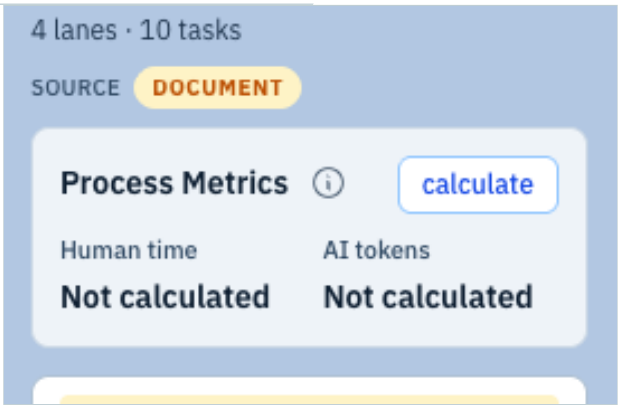
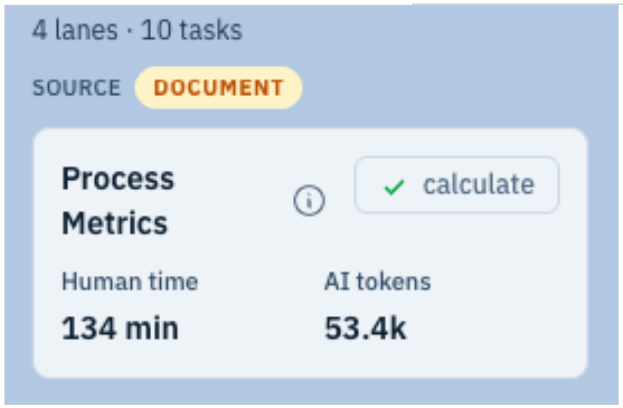


Diagram 3. The four stages of the jSwarm loop that the example follows. Each stage below names the jSwarm procedure or agent that acts, followed by its detail.

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



BEFORE CALCULATE



AFTER ONE PRESS

What the user sees. The requirement said “not calculated until asked”. Before calculate, Process Metrics reads **Not calculated** for human time and AI tokens. After one press it reads **Human time 134 min · AI tokens 53.4k**, with a check mark beside calculate. Both are displayed product estimates. Current build; cropped from the full captures in Part IV.

1 Record the feedback against intent (`/jFeedback`)

The owner’s words: “We are taking a multi-prong approach to solving the problem: LLM cache optimization techniques, as well as UX trade-off — creating calculation button instead of doing it automatically, incurring the AI cost when the user wants the information. LLM cache optimization is important, but cannot solve everything.”

The estimates were running after every build and every applied change. Each run cost paid AI calls. The owner decided to pay for the information only when a user asks for it. Why the cost was higher than it should have been is the story of Article 03.

`/jFeedback` records the feedback and its evidence first. Then it checks each item against existing intent: is there a Given/When/Then (GWT) clause that already covers it, or is this a gap? A gap is written into intent before anyone fixes code. That order matters. A surprise no GWT clause covers can easily lead to gold-plating or over-assurance: extra complexity, cost and maintenance nobody asked for.

Here the gap became two pieces of written intent. A non-functional requirement, `NFR-652-13`: when the product assesses the same tasks again with nothing changed, its requests to the model must not vary for no reason. And a GWT scenario, `UAT-652-CALCULATE-METRICS-ON-DEMAND`: the assessments run only when the user presses **calculate**, and until then each affected task says honestly that its metrics are not yet calculated. **jTestEngineer** drafted both; the owner approved them.

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

What the intent check does

For every layer that applies — acceptance criteria, non-functional requirement, GWT scenario, semantic scenario, journey — `/jFeedback` quotes the covering GWT clause with its identifier, or names the missing or contradictory expectation and the patch it needs. A behavior that breaks an existing clause is a defect against existing intent. A surprise no clause covers is a gap in intent.

The requirement’s pass/fail rule

`NFR–652–13` reads “Human-effort and AI-token assessments reuse the LLM cache unless there is a good reason not to.” Its rule: every cache miss for unchanged inputs must have a stated reason. Variation defeats the cache and costs money and time. New requirements are drafted into a proposal, previewed through the real renderer, and committed only after a person approves the preview.

The scenario in its own words (abridged)

> **Given** a fresh session with an accepted model built through the real document-upload and Build pipeline, with tasks in AI and human lanes.
>
> **When** the model is built, or a chatbot change that adds or changes a task is applied, and the user then presses the “calculate” button.
>
> **Then** before Calculate metrics is pressed, the Human-effort and AI-token assessments are not calculated automatically by the build or the Apply. Each task they affect shows an honest not-yet-calculated state (HUMAN EFFORT — Not estimated; AI TOKEN ESTIMATE — Not assessed) on the MODEL REVISIONS card and its node popup. Never “Assessing…”, a stale value from before the change, or an invented number.
>
> **Then** pressing Calculate metrics does all such metrics calculation and updates the card and the task attributes. The card and the node popup agree, and a reload shows the same values.
>
> **Then** (fail variants) an assessment still running automatically after a build or Apply, a press that updates only the card or only the task attributes, or a stale value shown as current, is a FAIL.

The scenario names its non-functional requirement (`NFR–652–13`) and the five journeys it serves. The owner’s words are embedded in it as a callout, so a reader can see where each clause came from.

2 Write the tests and fix to intent (jTestEngineer, `/jFix`, jCoder)

The owner’s feedback became five records in the defect backlog: four cost defects and one enhancement. Each one points at the intent it serves.

- `D1404` · Task and lane ids changed with every session, so the same request looked new to the cache.
- `D1405` · The document’s internal id changed with every session; same effect.
- `D1406` · The build job’s id went into the request; same effect.

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

- `D1407` · A random first-draw value went into the request; same effect.
- `D1408` · Replace the automatic assessment with a calculate action, so the AI cost is paid only when the user asks.

`/jFix` starts with the same intent check. **jArchitect** or **jOracle** then writes the fix contract: in plain English, what a correct fix must achieve, clause by clause. **jCoder** changes the code. **jTestEngineer** writes the test steps against the scenario’s GWT clauses, and the links run both ways.

How a fix stays bound to intent

The intent check comes before diagnosis or dispatch. Every implementation return states “Catalog rows touched/patched” with the scenario identifiers, or “none, with reason”. A fix cannot silently leave intent stale. The enhancement record `D1408` links back to the scenario, the non-functional requirement and the estimate conversation (`SS–67`), and carries its repair and the proof that goes with it. A defect that does not point at intent cannot be judged fixed.

The test steps, and the links both ways

The step-through test, `calculate–metrics.steps.ts`, follows the scenario’s clauses: build; check the honest not-yet-calculated state and the visible button; press calculate; check that the card, the pop-up and the stored task attributes agree; reload and read back. It opens by naming the scenario and the clause each step exercises. In the other direction, a scenario lists the tests that prove it, and a checker verifies both directions. A test without intent, or a GWT clause without a test, is a review gap, not a pass. A functional result that breaks an applicable acceptance criterion or non-functional requirement is incomplete, however green the test.

3 Prove it: the Agent Acceptance Test (AAT) walk (jQATester)

jQATester walked those steps on an earlier build and judged each GWT clause. Three clauses were met. No metrics ran automatically after the build. The calculate button worked on the current revision. After the press, the card, the pop-up and the stored values agreed for all 10 tasks and survived a reload. Parts II and III below show the walk’s own screenshots.

The clauses the walk did not reach

Three clauses were not exercised: what happens after a later change to a task, what happens when a calculation fails, and an explicit estimate request through the chatbot. The walk is recorded as partial, not as a pass. That is the point of per-clause reporting: we can say exactly which clauses are proven, instead of “the test passed”.

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

When the grader is wrong, people decide

A later walk graded a renamed task’s neighbors as having “lost” their estimates. Was that the product or the rubric? The owner decided. `D240` kept the product behavior and corrected the rubric. `D243` adopted the corrected wording. `D241` refused a wholesale re-calibration of the grader every time intent changes, and asked for a targeted second judge instead. **jQAScorer** grades; people decide what the grader may say.

4 Accept it: User Acceptance Testing (UAT) by the owner

The last stage is human. In UAT the owner walks the product, gives feedback, and accepts or rejects. UAT is the final authority; an agent walk never stands in for it. Whatever the owner finds goes back to stage 1, into intent first.

What a UAT round looks like

In one UAT round the owner walked the current build and wrote their feedback through `/jFeedback`. The record split it into 11 items and checked each against intent: three were defects against existing GWT clauses, most of the rest were requests no clause covered yet, and one was a question (measured, round record). The round’s verdict was FAIL. The feedback went back into intent first: the owner approved patches to seven existing GWT scenarios, and three new records were registered, each linked to the scenario it serves. Only then did fixes follow. That is UAT doing its job: finding the intent nobody had written down yet.

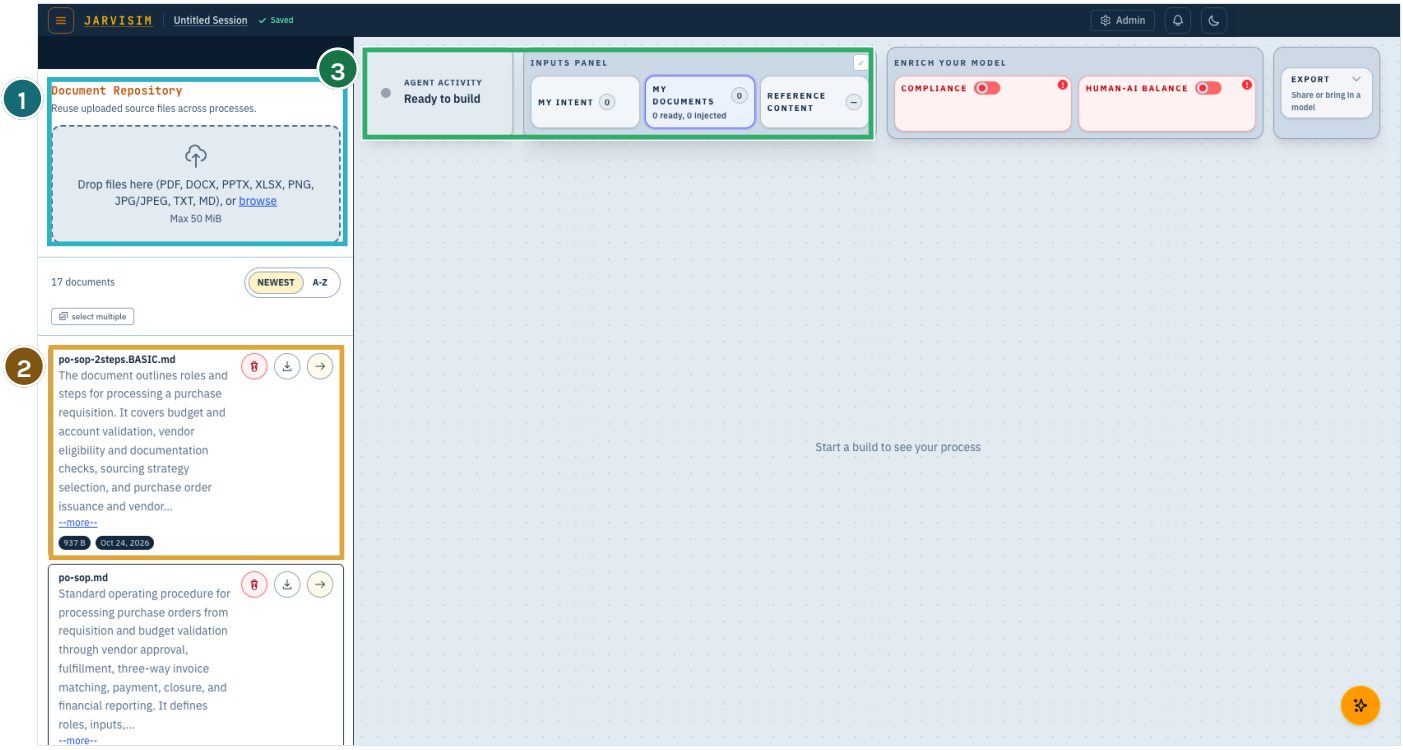
Where the people are in this chain. People wrote and approved the GWT scenario and the non-functional requirement. People decided what the grading rubric may say. People accept the result in UAT. People feed the next surprise back into intent before anyone touches code. The agents did the walking, the reading-back and the first-pass judging. That is where human judgment goes when AI does the routine work.

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

PART I

The journey starts with a document.

The Given clause asks for a model built from a real uploaded document. This is that screen on the current build: the document library, a Use control for each document, and an empty canvas waiting for a build. Nothing is estimated yet because nothing is built yet.



Given: a document to build from. Current build; account identity masked at top right. Each numbered note below matches a numbered marker on the screenshot.

1 The document repository

Given · A real source document

The scenario starts from a model built from a real document. This library holds the uploaded documents, ready for reuse.

2 The document and its Use control

Given · Bring the document in

Each document has its own Use control. Selecting it is the first step of the user’s journey.

3 Ready to build

Given · Before any build

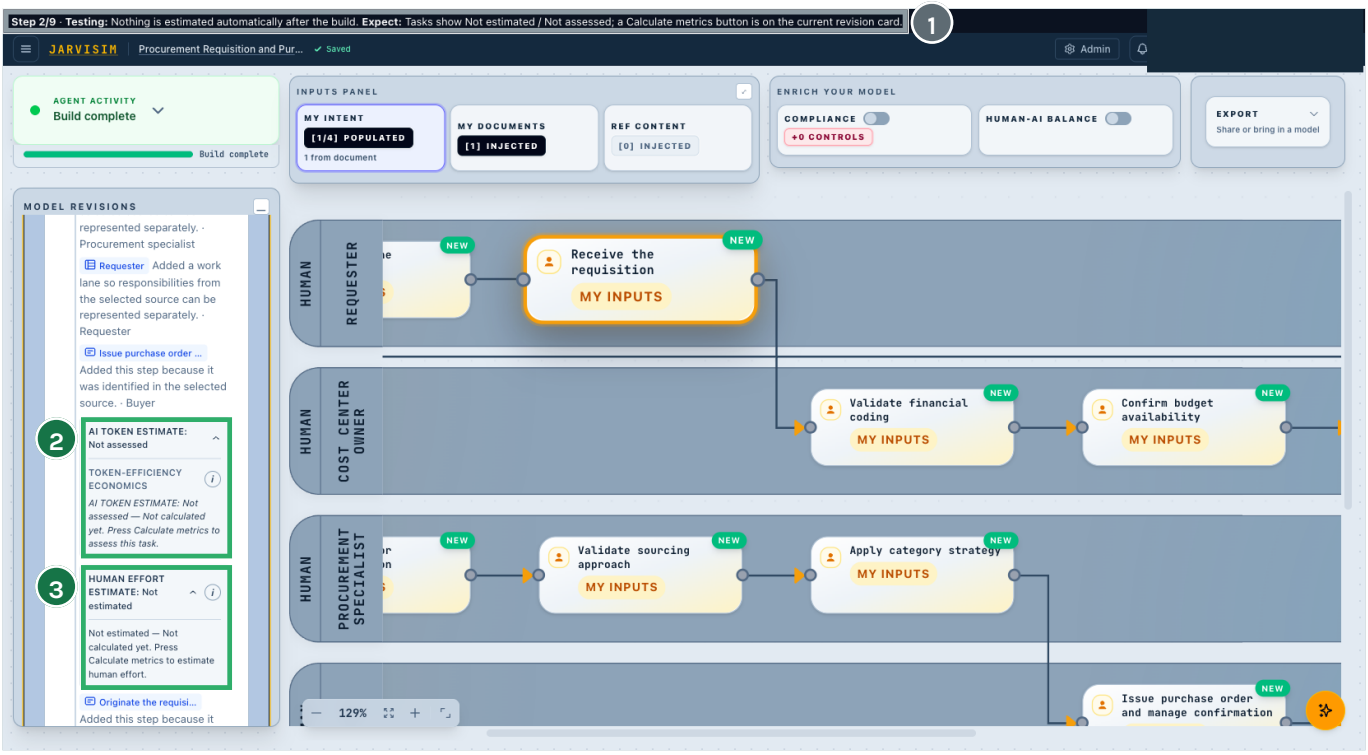
Nothing is built yet, so nothing is estimated yet. That is the starting state the scenario asks for.

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

PART II

Before calculate: honest “not yet”.

The first Then clause: before calculate is pressed, each affected task shows **AI TOKEN ESTIMATE: Not assessed** and **HUMAN EFFORT ESTIMATE: Not estimated**. Never a stale number. This is jQATester’s second step, on an earlier build. The dark bar at the top is the test harness’s own step line, not a JarviSIM feature. It names the clause being checked.



Then, before calculate. An earlier build; the top bar is a test-harness overlay; account identity masked at top right. jQATester judged this clause met.

1 The test harness step line

Harness overlay · Not a product feature

The harness line says what is being tested and what to expect: nothing is estimated automatically after the build. It ties the screenshot to the clause. It is not part of JarviSIM.

2 AI TOKEN ESTIMATE · Not assessed

Then · Honest not-yet-calculated state

The card says Not assessed and tells the user to press Calculate metrics. The clause forbids a stale value, an invented number or an endless Assessing state. Judged met.

3 HUMAN EFFORT ESTIMATE · Not estimated

Then · Honest not-yet-calculated state

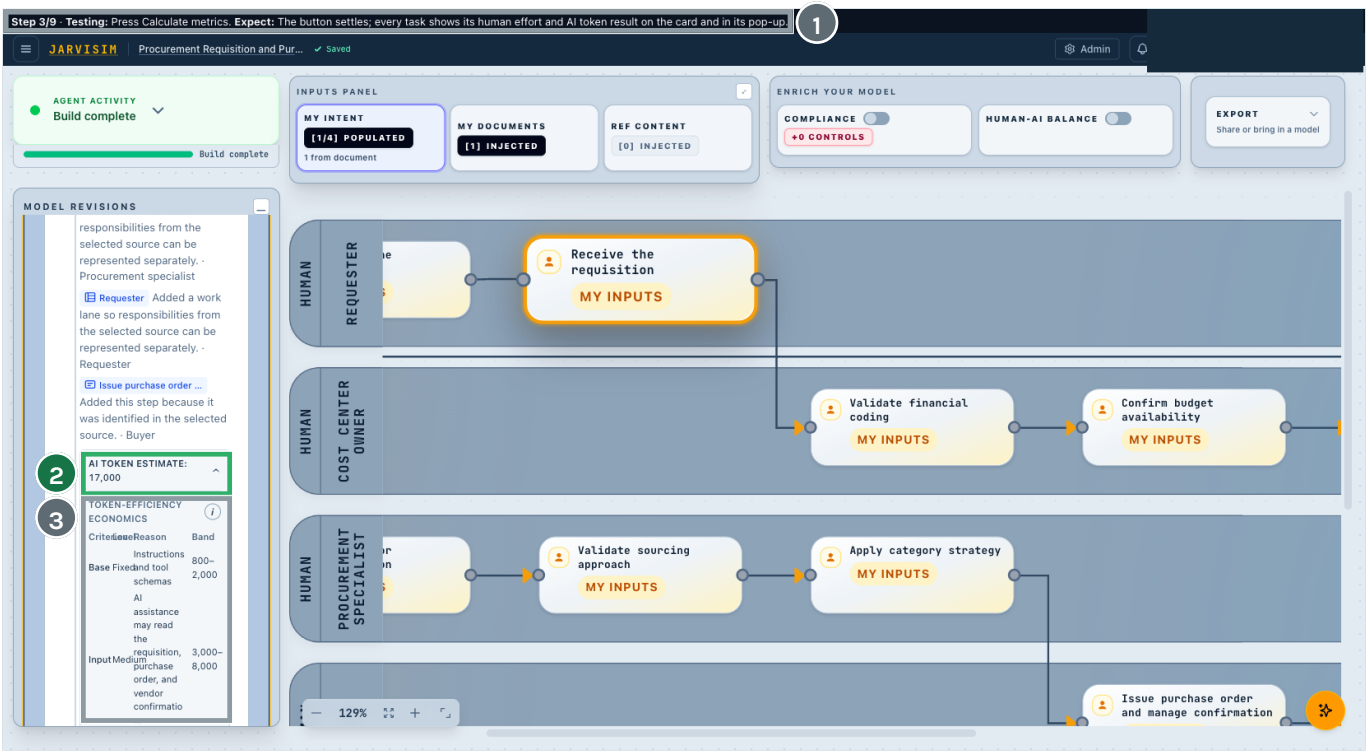
The same honest state for human effort. jQATester also read the stored values: zero for every task. Judged met.

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

PART III

After calculate: the card, the pop-up and the store agree.

The next step presses calculate. The clause says the card, the task pop-up and the stored values must agree and survive a reload. jQATester read all three and judged the clause met for all 10 tasks. The card now shows **AI TOKEN ESTIMATE: 17,000**, a displayed product estimate.



Then, after calculate. An earlier build; the top bar is a test-harness overlay; account identity masked at top right. jQATester judged this clause met.

1 The test harness step line

Harness overlay · Not a product feature

Testing: press Calculate metrics. Expect: every task shows its human effort and AI token result on the card and in its pop-up. The harness tells you which clause this step proves.

2 AI TOKEN ESTIMATE · 17,000

Then · Calculated value on the card

After the press, the card shows 17,000 for this task. It is a displayed product estimate. jQATester found the card, the pop-up and the stored values agreed for all 10 tasks.

3 Token-efficiency reasoning

Semantic · A different question

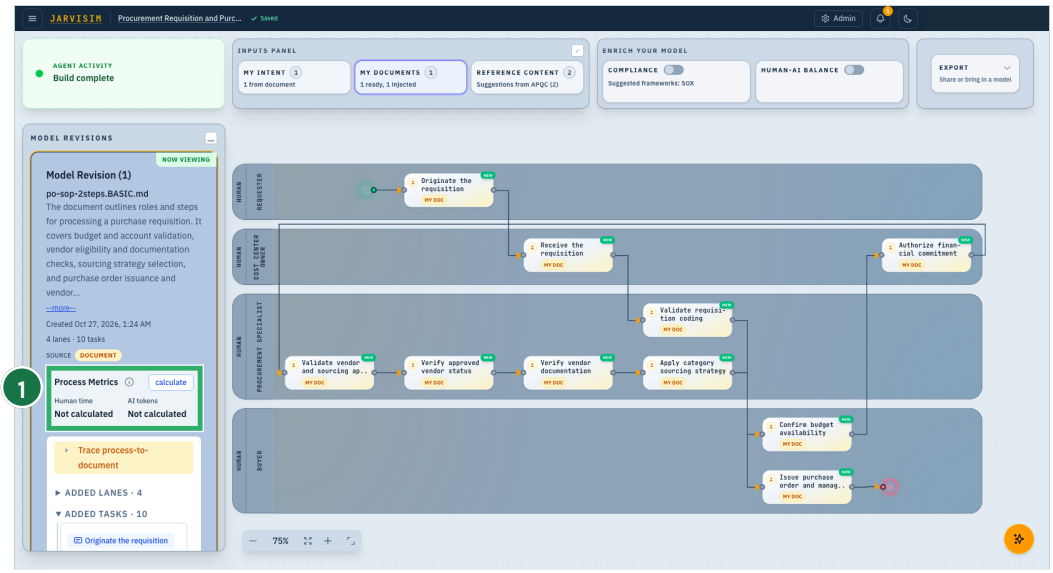
The product shows the criteria behind the estimate. Whether that reasoning is good is a semantic question with its own rubric, not a clause of this scenario.

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

PART IV

The same clause on the current build.

Intent outlives the build it was written for. The current build shows the same honest state before calculate — **Process Metrics · Not calculated** — and, after one press, **Human time 134 min · AI tokens 53.4k** for the same ten-task purchasing process. Both are displayed product estimates.

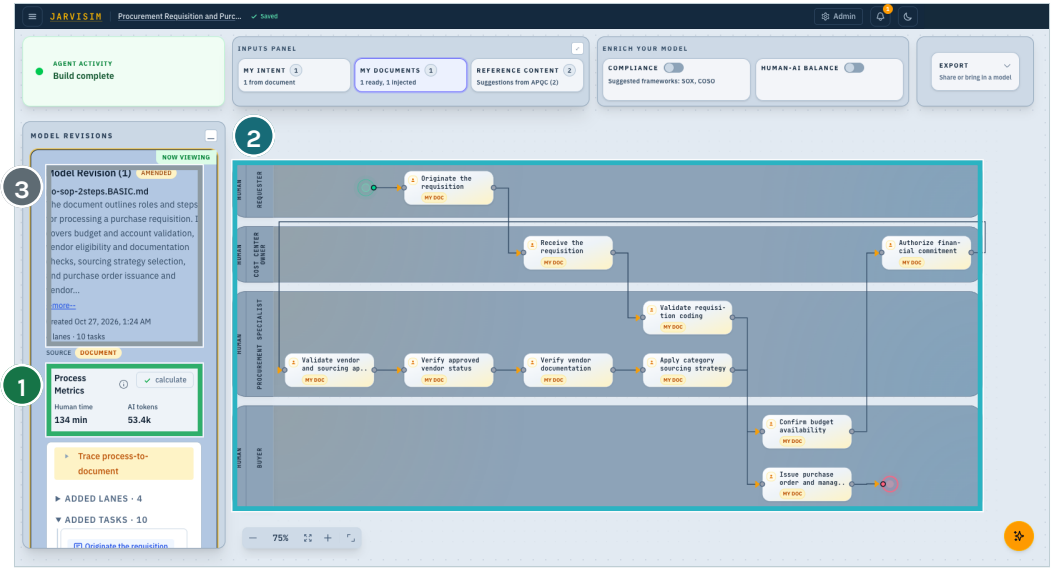


Current build, before calculate. Account identity masked at top right. The same clause on a newer screen.

1 Process Metrics · Not calculated

Then · Same clause, current build

The card reads Not calculated for human time and AI tokens, beside the calculate control. The same honest state, now on the current build.



Current build, after one calculate. Account identity masked at top right. The displayed values are product estimates.

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

PART IV · CONTINUED

1

Process Metrics · 134 min · 53.4k

Then · After one calculate press

After one press the card shows Human time 134 min and AI tokens 53.4k, with a check mark beside calculate. Displayed product estimates.

2

The ten-task purchasing process

Journey · The model the clause applies to

Four lanes and ten tasks built from the purchasing document. The Then clause applies to every task, not only to the card.

3

Model Revision (1) · 4 lanes · 10 tasks

Journey · Visible counts

The revision card names the source document and counts four lanes and ten tasks. The AMENDED tag is a product label for a revision that changed after the calculation.

The design lesson

> ## If you cannot write down what right looks like, no test can tell you when you have it.
>
> A green test is a claim about a clause. If there is no clause, the test is a claim about nothing. Writing the clause is the expensive, human part — and it is the part that makes everything the AI does afterwards cheap to trust.

Screenshot-to-story region map

Part	Numbered regions on the screenshots	Intent connection
I	Document repository; the document and its Use control; Ready to build	The Given clause: a model built from a real document
II	Test-harness step line; AI TOKEN ESTIMATE · Not assessed; HUMAN EFFORT ESTIMATE · Not estimated	First Then clause; jQATester judged it met
III	Test-harness step line; AI TOKEN ESTIMATE · 17,000; token-efficiency reasoning	Second Then clause; jQATester judged it met
IV	Process Metrics · Not calculated; Process Metrics · 134 min / 53.4k; the ten-task process; Model Revision (1) counts	The same clauses on the current build

For Claude Certified Architect–Professional readers

Anthropic’s guidance on defining success criteria and building evaluations asks for criteria that are specific and measurable before you build the evaluation. We apply this on a real product. Our Given/When/Then scenarios and non-functional requirements are those criteria, written by people and approved by the owner. The test steps and the AAT walk are the evaluation, linked back to each clause with a per-clause verdict.

Source: https://platform.claude.com/docs/en/test-and-evaluate/develop-tests

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