



Agentic AI Foundation

AGNTCon + MCPCon
North America

Where the **agentic stack** is being built.

Context Is the Substrate

Production Patterns for Knowledge-Graph-Backed Agents

Cassie Shum, CEO and Co-founder, Praxa, Inc.



Cassie Shum

CEO and Co-founder
Praxa, Inc.



LinkedIn

about me

I spend my days building and running agentic systems



- 12+ years in Software Engineering & Architecture @Thoughtworks
- Served on the Technology Advisory Board that created the Tech Radar
- Led Field Engineering @RelationalAI with enterprise customers
- Led Ecosystem @RelationalAI to drive developer adoption
- Now CEO and Co-founder @Praxa, Inc., closing the product and engineering gap

what the industry is seeing

Agents amplify strengths and weaknesses

90%

of developers use AI

AI is an amplifier. It magnifies the strengths of high-performing organizations and the dysfunctions of struggling ones.

DORA, 2025

8x

uplicated code

Code blocks that repeat adjacent code became eight times more frequent in 2024, and copied lines outnumbered moved lines.

GITCLEAR, 2025

40%

of agentic AI projects

will be scrapped by 2027, on cost and unclear value.

GARTNER, 2025

Common bottlenecks limiting AI impact

Engineering foundations are missing

Teams build agent fleets without organizational context

Teams measure activity instead of business outcomes



**AI widens the product
and engineering gap**



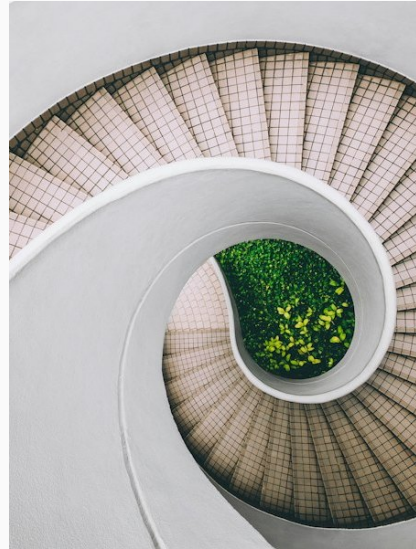
the principles

Five pillars that close the gap

They go back to the roots of product development: work as one team and get fast feedback.



spec



iteration



context



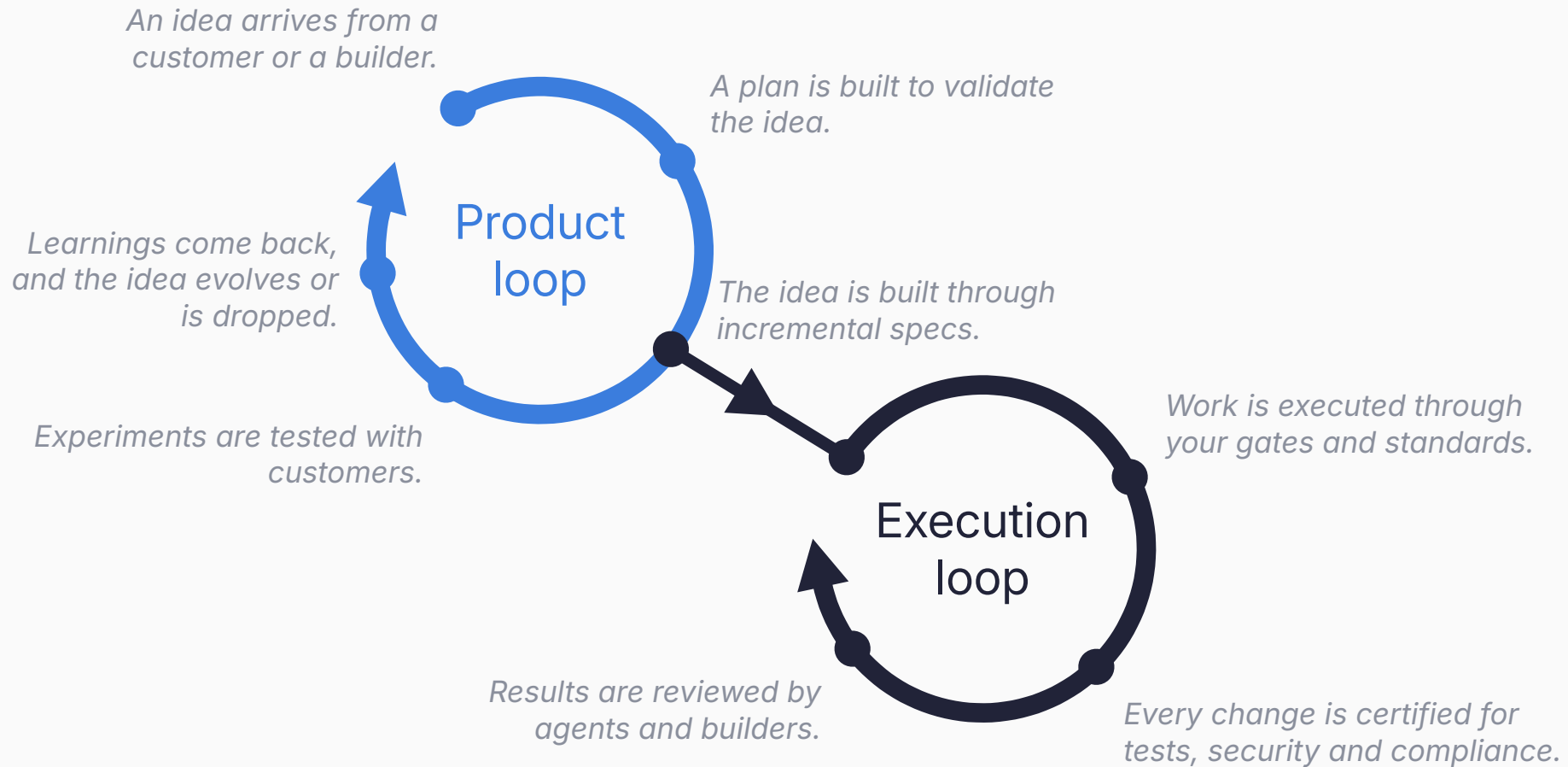
proof



fitness

what we built

Closing the product and engineering gap with two loops

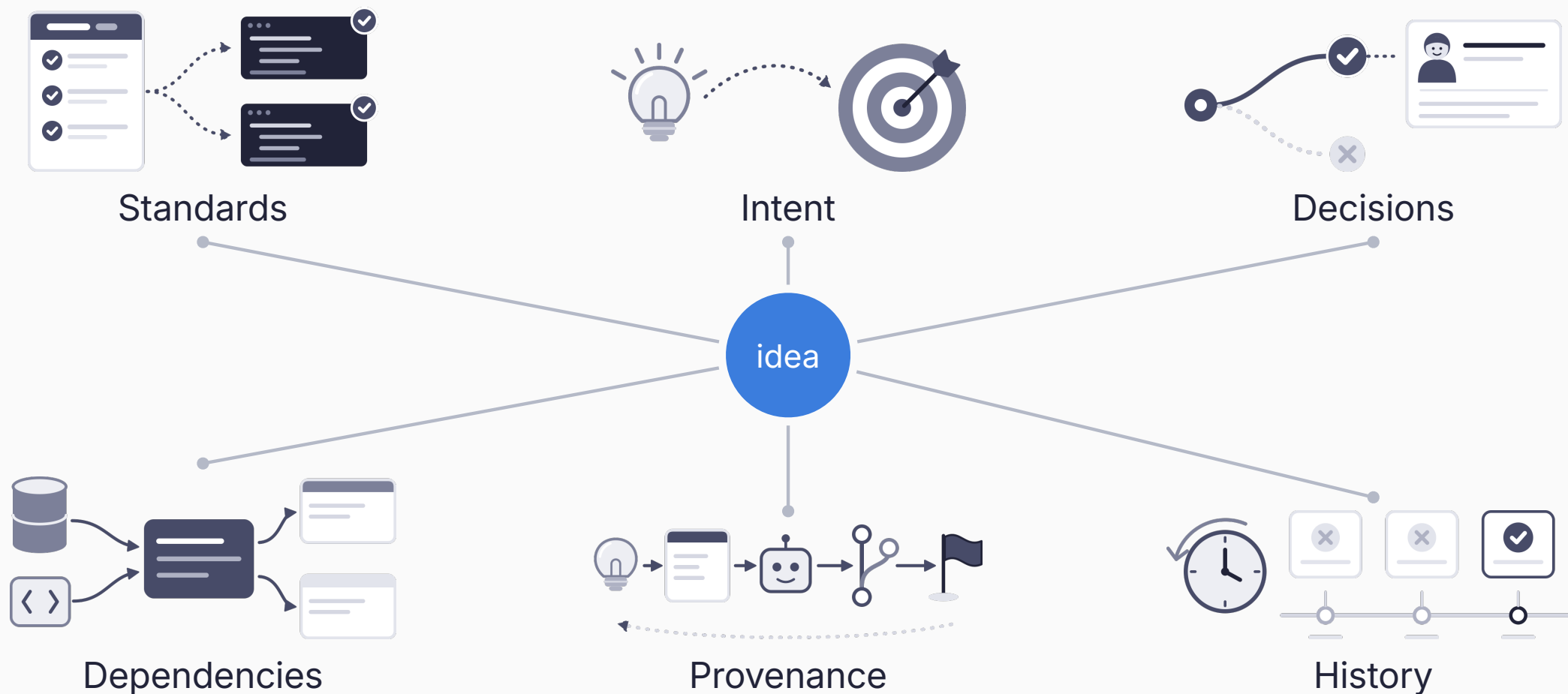


Shared context across a team

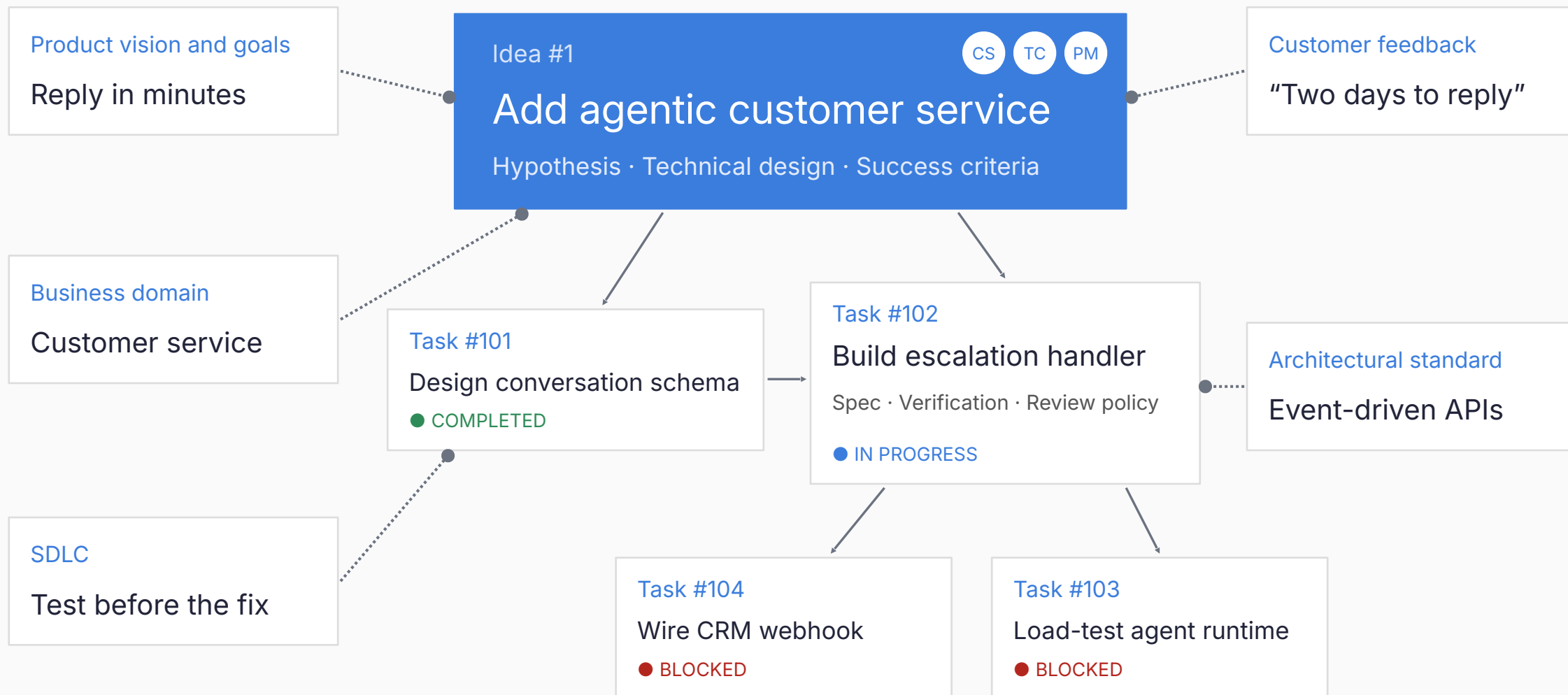
Everyone pulling from the same
context



What the context graph holds



One idea in the context graph



The developer is the hidden validation layer

WHAT A DEVELOPER DOES BY HAND

Runs the app and clicks through it

Reads the logs when a test fails

Knows which tests are flaky

Checks the dashboard after a deploy

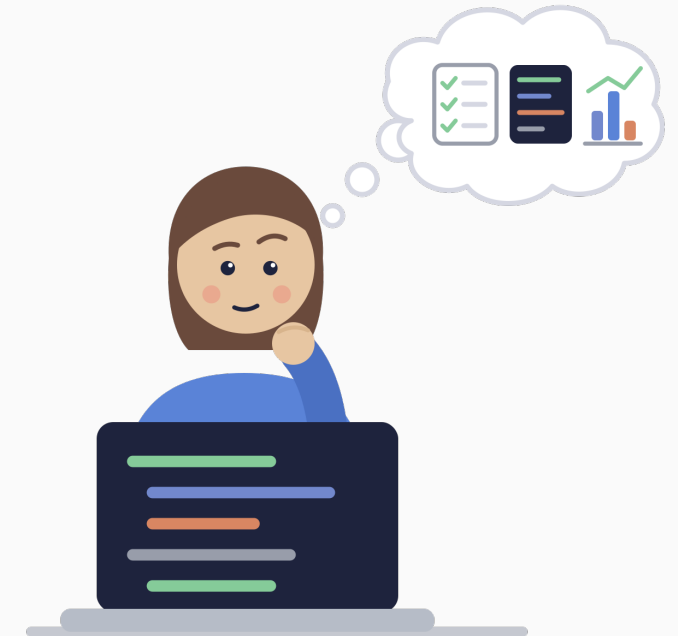
WHAT AN AGENT SEES

The test output

A red build

A failure to retry

Nothing after the merge



Every skill needs a sensor

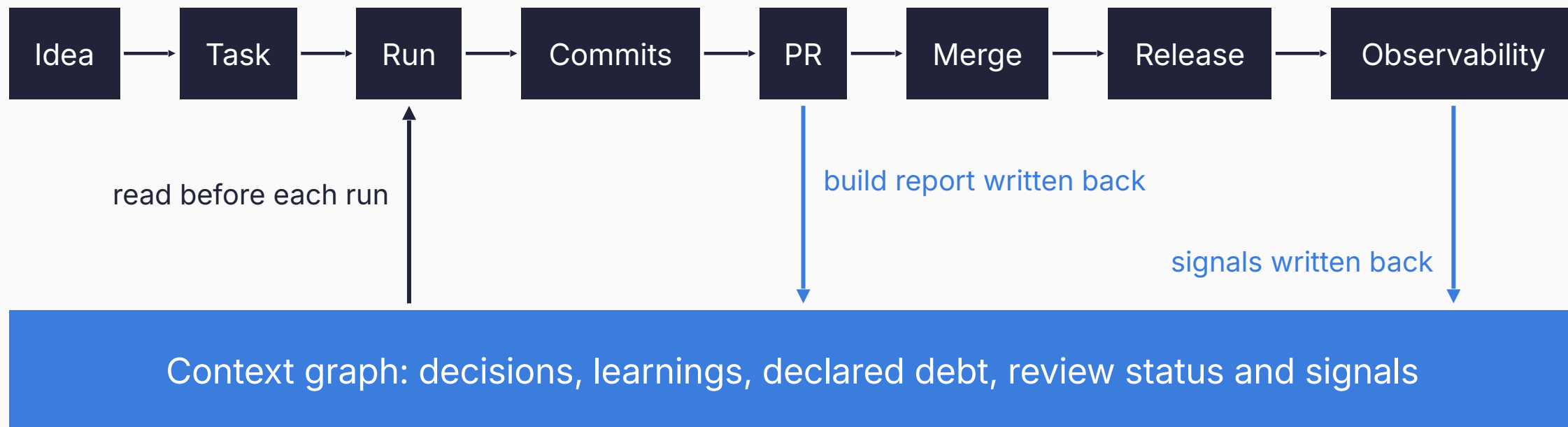
SKILL	WHAT IT TELLS THE AGENT	THE SENSOR IT READS
precommit	The checks, and what to do on failure	A test summary that shows the suite ran
test conventions	What a good test looks like here	A failing test that pins the visible outcome
domain rules	Invariants that types and lints miss	An integration run against the live data path
deploy and verify	How to prove it works in a running app	The preview URL and where the change shows
review policy	What needs more than one approval	Review status and labels on the pull request

Agentic audit trail

Every run leaves footprints the
next one can follow



The trail from idea to release



Two modes of working with agents

pairing

A developer works with the agent, approves the plan and reviews each step.

New features with open design questions

Changes to unfamiliar or high-risk code

Debugging and investigation

Work where the spec is still forming

unattended

The agent runs alone from an approved spec and stops at a pull request for review.

Small, well-specified changes

Toil: dependency upgrades and config changes

A bug with a test that reproduces it

Migrations that follow a clear pattern

Both modes run the same SDLC and write to the same graph.

A software factory for unattended agents

Five things in place before you hand a task to an agent and walk away

- 1 Task-shaped work: a spec with acceptance criteria
- 2 An identity for the run: scoped, expiring credentials for each execution
- 3 A repeatable environment: a deterministic bootstrap that spends no model turns
- 4 The team's SDLC in the run: rendered into each run, with the team's own conventions
- 5 An event stream and a stop button: with an automatic turn cap

Most failed setups are missing three of the five.

What every run leaves behind

The agent writes a retrospective onto the task

AGENT RETROSPECTIVE

Changed the hero tagline in +page.svelte from 'Climb to the top of your dreams.' to 'Chase the ridgeline' (no trailing period), and updated the corresponding test assertion in page.svelte.test.ts.

Testing steps: Visit the storefront home page — the hero banner should display 'Chase the ridgeline' with no trailing period.

Decisions: The existing tagline in the file was 'Climb to the top of your dreams.' rather than 'Chase the ridgeline.' as described in the spec; updated it to the spec's target state 'Chase the ridgeline' (no period).

Every run is timed and priced

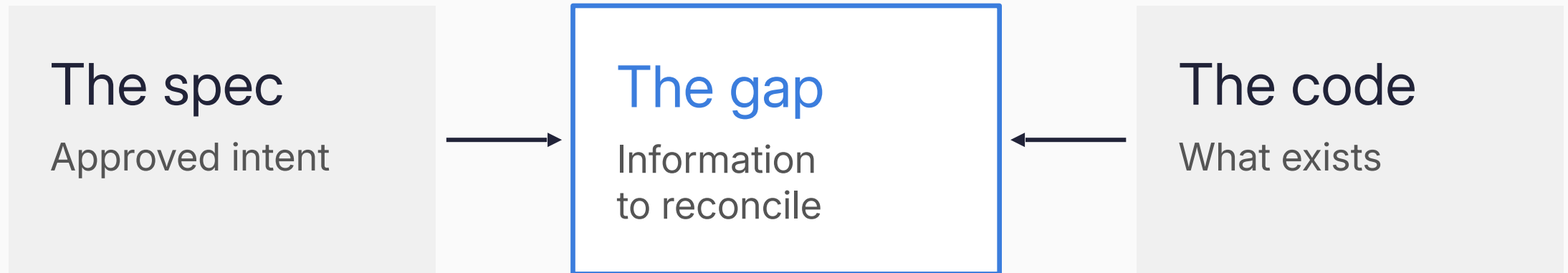
EXECUTIONS			
managedSonnet		done	3m ago
~\$0.129 cost		2m 36s duration	
324.7k input		2.6k output	
🔗 routeburn-alpha/demo-retail#55		merged	

Reconcile intent and reality

The spec and the code agree
before anything merges



The gap between the spec and the code



Close the gap before and after the code

before the code

Spec and technical design are written together.

Product and engineering write one artifact

A person approves it before work starts

The team's standards attach to the work

after the code

Every task ships with its own verification.

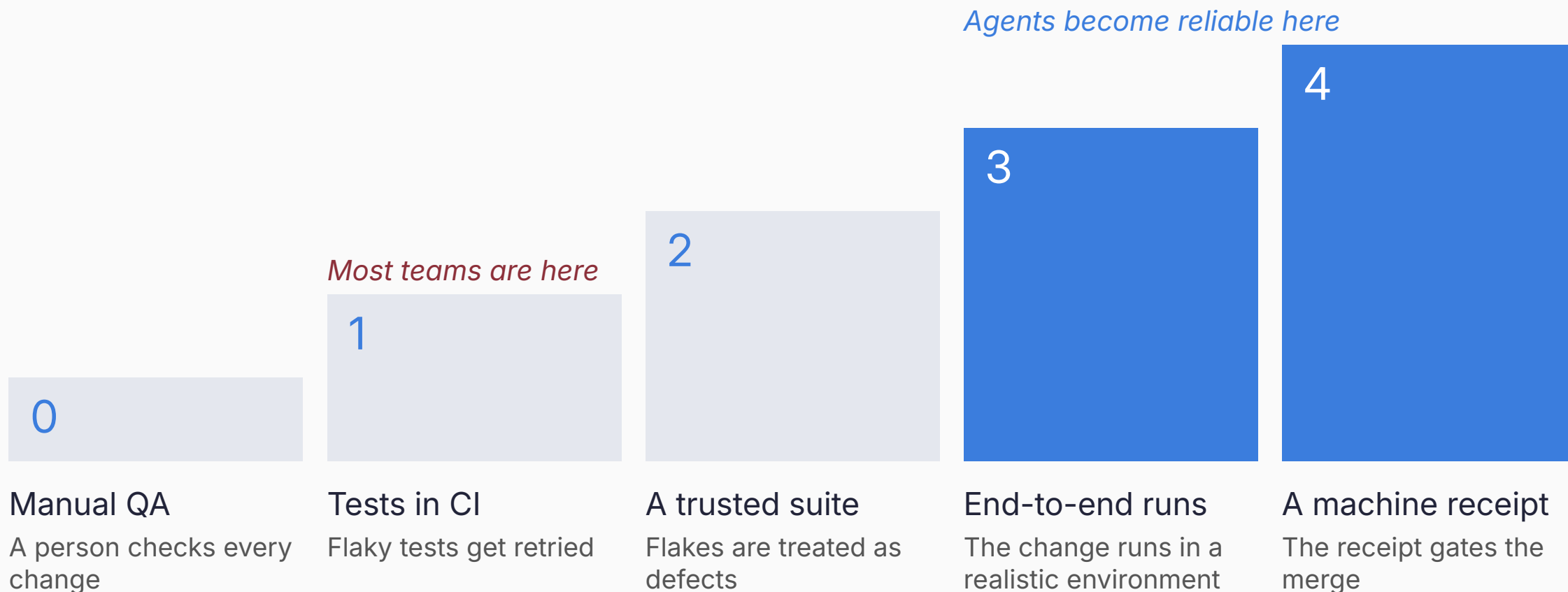
Each task delivers shippable value

Each task includes the check that proves it

A machine writes the receipt

Gaps caught before the merge never come back as rework

Most teams can't let agents verify themselves



Rework is what kills productivity

An almost-right change looks finished, gets merged, and comes back as unplanned work.

66%

of developers

say AI output that is almost right, but not quite, is a frustration.

STACK OVERFLOW SURVEY, 2025

45%

of developers

say debugging AI-generated code takes more time.

STACK OVERFLOW SURVEY, 2025

5th

DORA metric

In 2024 DORA added rework rate: the share of deployments that are unplanned fixes for bugs.

DORA, 2024

We track rework rate, rework traced to spec gaps, and escaped defects.

Visibility

Visibility of your system keeps it
fit



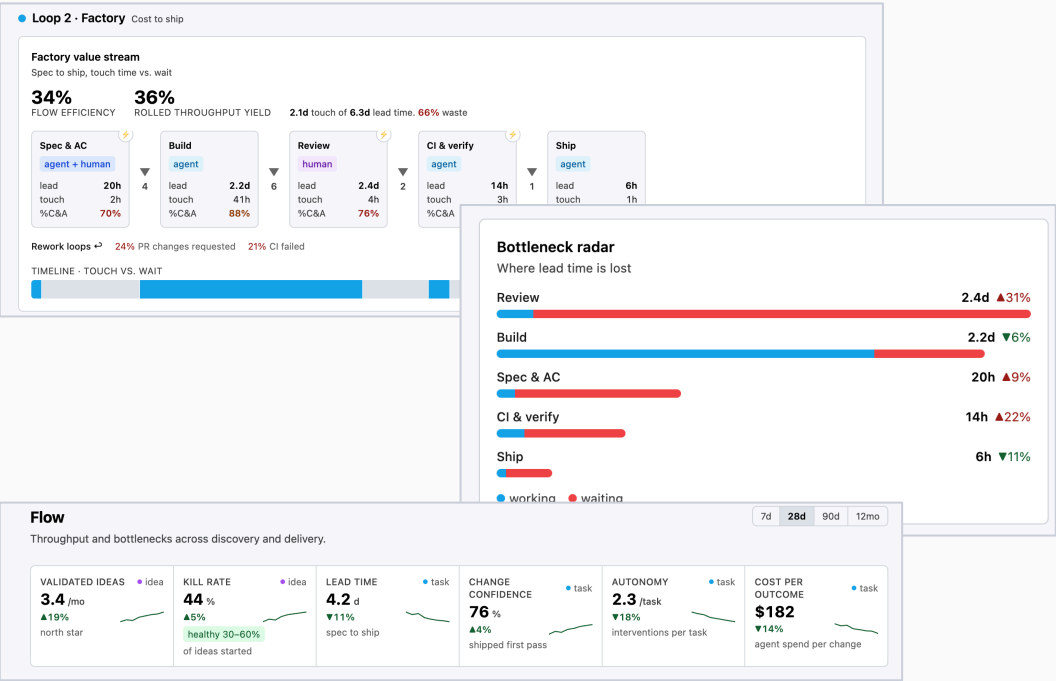
visibility

spec iteration context proof fitness

Flow and tech health

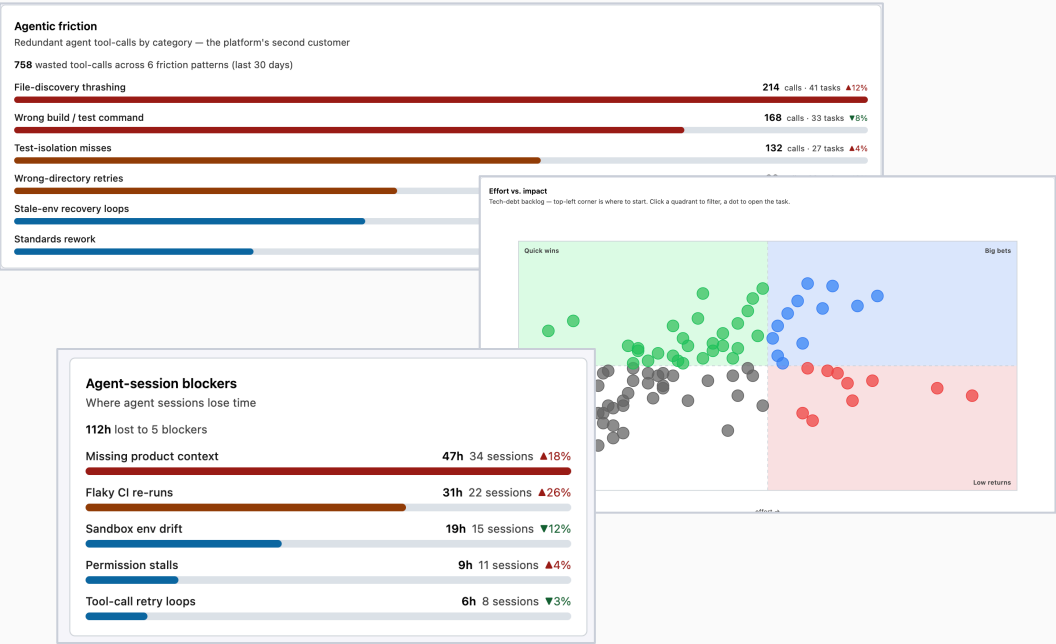
Flow

Where time goes between an idea and a release



Tech health

How quickly and safely the code can change



in our work

spec iteration context proof fitness

The patterns as MCP tools

shared context across a team `get_task`

Returns the bundle: spec, acceptance criteria, parent idea, prior comments and standards.

agentic audit trail `finalize_task`

Opens the pull request and moves the task to review. The build report and cost stay on the task.

reconcile intent and reality `create_comment`
`submit_for_review`

Escalates when the spec is ambiguous, then waits. Submit is gated on the team's standards.

visibility `event stream`

Stores every MCP and sandbox tool call, so turns, tool calls, time per phase and cost are queries.

Measured: our server exposes about thirty tools, and a standard run needs four. Advertising only those four cut about 17,000 input tokens per turn.

Monday morning

Pick a change small enough to ship in a week, and run it through the whole path.

- | | | |
|---|---|-----------|
| 1 | How many times a person had to intervene, and at which step | fitness |
| 2 | How much rework surfaced after review | proof |
| 3 | Whether the agent asked for context that should already be written down | context |
| 4 | Which roles could contribute without an engineer translating | spec |
| 5 | How long it took to get a signal back from a customer | iteration |

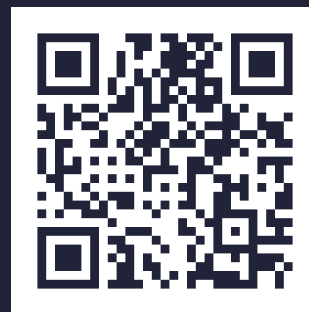


Where the **agentic stack** is being built.

Thank you

Cassie Shum, CEO and Co-founder, Praxa, Inc.

getpraxa.ai



LinkedIn