

Your AI writes code fast.
Your codebase has opinions.

ROMAN LOBUS
LOBUS

ACT I · THE CHANGE THAT SHIPPED

MONDAY · 09:12

*“Automatically refund premium shipping when the
when the delivery promise is missed.”*

— POLICY, CURRENT

The agent does its job.

STEP 01

Reads the ticket.

STEP 02

→ Reads the codebase.

STEP 03

→ Implements the happy path.
path.

✓ Diff is clean

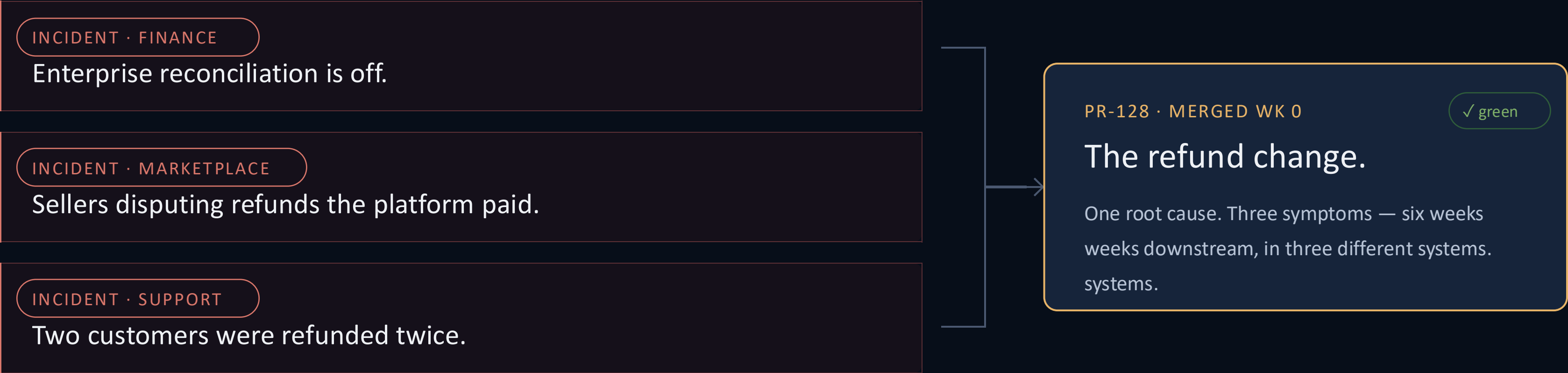
✓ Tests pass

✓ Merged Thursday

Sprint review calls it a win — by every signal anyone can see, it is one.

WEEK 6

Three tickets arrive in the same fortnight.



None of the three tickets mentions it.

One line visible. Seven behaviours below.

ON THE TICKET

“Automatically refund premium shipping when the delivery promise is missed.”

BELOW THE LINE · SEVEN INSTALLED BEHAVIOURS, MOSTLY UNWRITTEN

01 Direct retail —SR_SHIP_DELAY to original tender

02 Managed enterprise —invoice credit, not card reversal

03 Marketplace orders —excluded, seller-owned settlement

04 Split shipments —manual, asynchronous carrier evidence

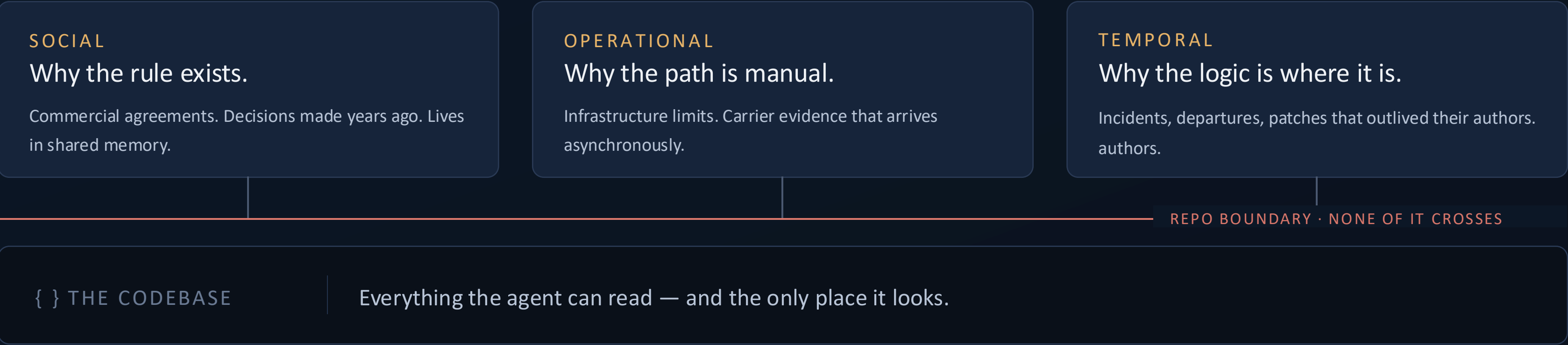
05 Duplicate prevention — blocks a second adjustment

06 Already-invoiced — tax reversal may be required

07 Promotional free shipping — no fee, no adjustment

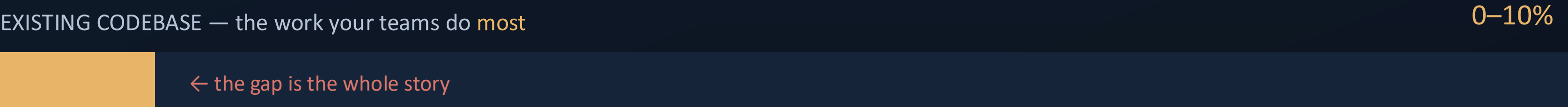
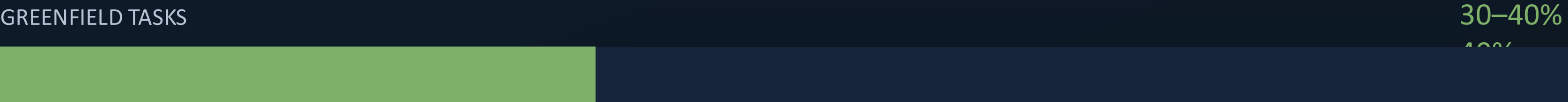
None of it was in the ticket. Most isn't written anywhere.

Why the agent couldn't see it.



The knowledge that was missing isn't in the code.

This is the normal case, not the horror story.



STANFORD · 100,000 DEVELOPERS

+98% PRs · +91% review time

No company-level gain — Faros AI, 10,000 developers

96% distrust AI code · 48% verify it

Half-trusted, under-verified — Sonar, January 2026

Same mechanism, different domain.

“Add a button to export the customer list to CSV.”



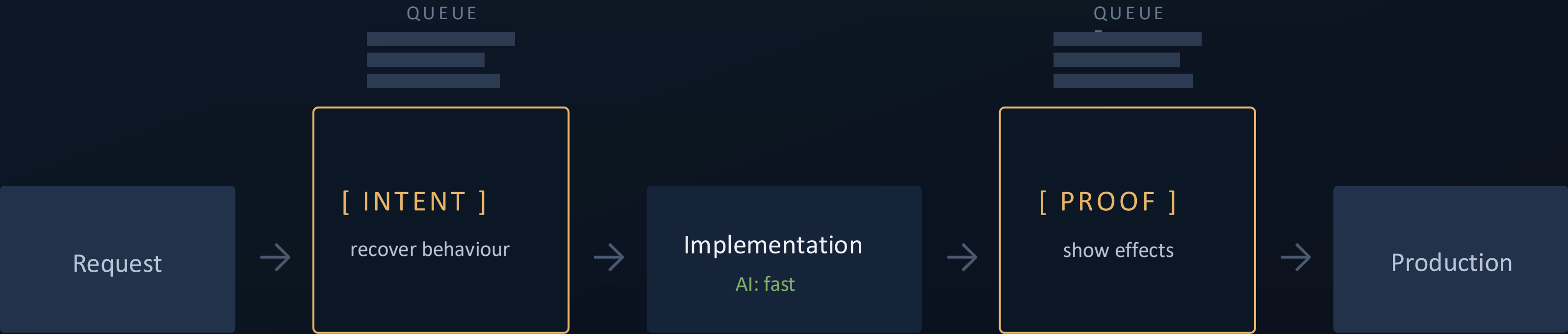
– return repo.visibleFor(tenantId);

+ return db.findAll();

The export contained every tenant’s customers, deleted records, and staff-only notes.

The agent’s own test stayed green. It tested what the agent intended, not what the system guaranteed.

The bottleneck moved.



Speed was never the bottleneck. Intent and proof always were — AI exposed it.

The model.

STAGE 01 · SPECIFY

Two specs

Actual-behaviour spec + intended delta. Kept separate.



STAGE 02 · SCOPE

Bounded delta

What changes. What stays. What is explicitly out of scope.



STAGE 03 · PROVE

Evidence-first review

Proof packet travels with the change.
Preserved behaviour first, diff second.

Every part of it is something a delivery lead can ask for, by name, before work starts.

Two specs: the structural claim.

SPEC 01 · EVIDENCE

Actual-behaviour spec

Records only verified current behaviour.

SPEC 02 · INTENT

Intended delta

Records only authorised changes.

A line cannot belong to *both*.

What you need to ask: “ *Are evidence and intent in two documents — or one?* ”

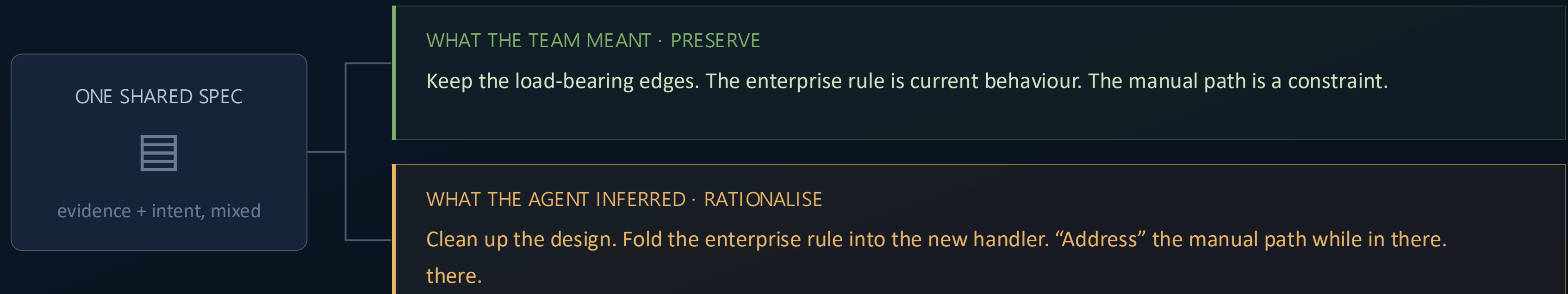
The same billing case, on two sides of the wall.

ACTUAL-BEHAVIOUR SPEC Only what we confirmed the system does today ✓ VERIFIED ● Direct retail — SR_SHIP_DELAY to original tender ● Enterprise — invoice credit, monthly reconciliation ● Marketplace — excluded, seller-owned settlement ● Split shipments — manual, async carrier evidence ● Duplicate guard — one adjustment per order	INTENDED DELTA Only what we authorised to change ◆ AUTHORISED ◆ Automate SR_SHIP_DELAY for direct retail · domestic · premium ◆ Trigger once carrier evidence confirms the missed window ◆ Amount and mechanism identical to the manual path Five lines of evidence. Three lines of intent. Nothing shared.
---	--

Every fact that ambushed the team in week six — now on the page, on the side the agent can’t redesign.

Why conflation is the root cause.

Share one document, and the agent resolves the ambiguity — toward coherence, not toward what was authorised.



What you need to ask: “ *If they’ re mixed, who resolves the ambiguity — us, or the agent?*”

Characterisation tests: the baseline you can run.

Written before implementation. Not to prove the feature — to pin the reality.



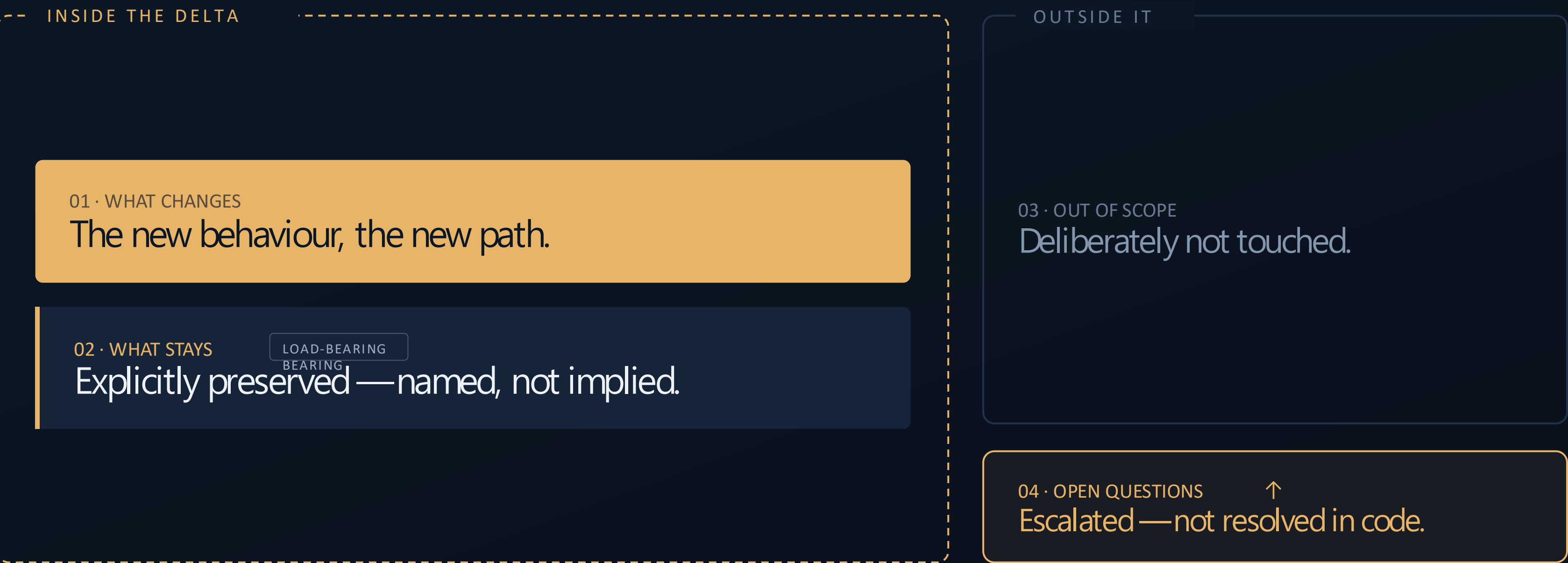
```
# Records current behaviour. Turns red if the implementation moves it.  
def test_enterprise_late_delivery_produces_invoice_credit():  
    adjustment = billing.apply_late_delivery(enterprise_order)  
    assert adjustment.code == "SR_SHIP_DELAY"  
    assert adjustment.kind == "INVOICE_CREDIT"# not a card reversal
```

Your question as a delivery lead:

“Do these exist for the behaviour we’re preserving?”

The bounded delta draws the boundary.

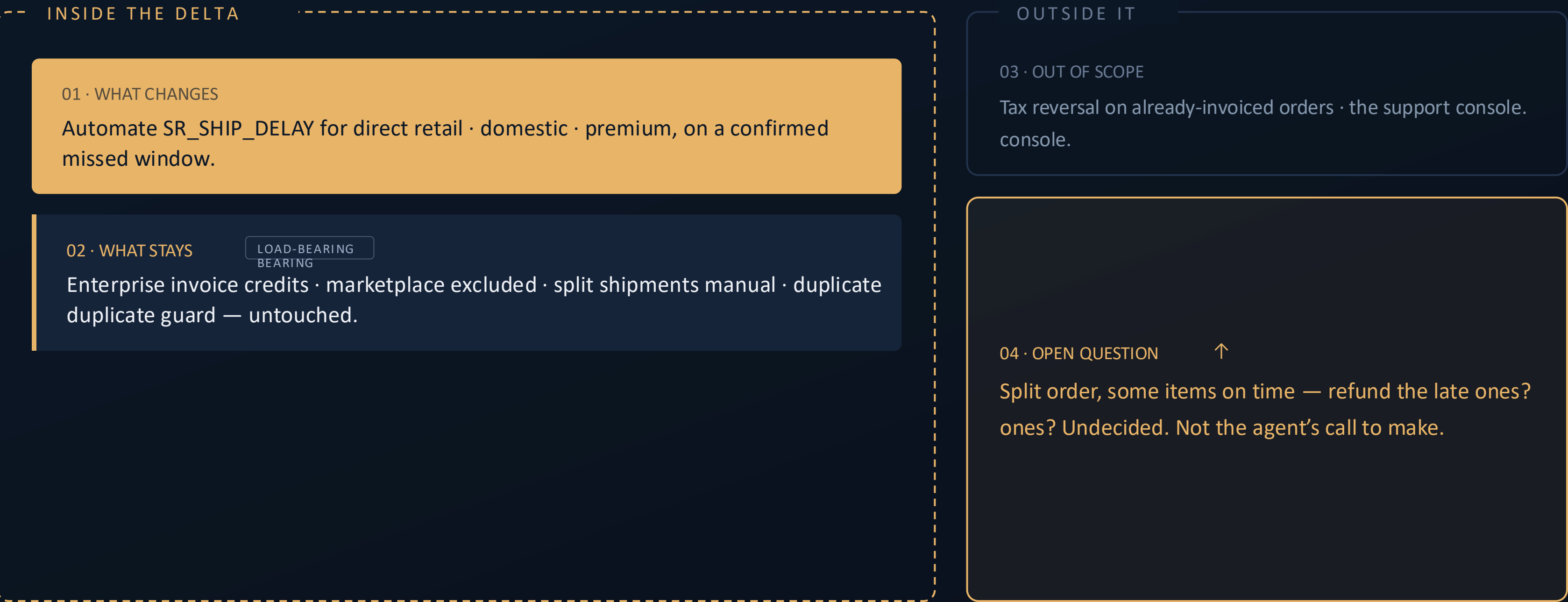
Four parts — two inside the change, two outside it.



What you need to ask: *“Can the delta name what stays — not just what changes?”*

The bounded delta, filled in.

The refund change — what each part actually said.



What you need to ask: “ *Does the delta mark the load-bearing walls before anyone knocks one through?*”

Evidence-first review: the proof packet.

PROOF PACKET		TRAVELS WITH THE CHANGE · PR-128
01	What preserved behaviour have I verified? The characterisation tests, named — and what each confirmed.	
02	What changed —and is there proof it changed correctly? The test that exercises the new path, and why it's the right acceptance condition.	
03	What downstream effects did I check? Settlement export · finance reconciliation · operations console — checked, not assumed.	
VERIFICATION BOUNDARY		
04	What am I <i>not</i> claiming to have proven? What separates a proof packet from optimism.	

What you need to ask: *“Does proof travel with the change —or do we take it’s done’ on trust?”*

The proof packet, filled in.

PROOF PACKET · PR-128		LATE-DELIVERY REFUND AUTOMATION
01	✓ Three characterisation tests green. Enterprise invoice-credit path · marketplace exclusion · duplicate guard.	
02	✓ New test exercises the authorised path. Direct-retail premium late delivery → SR_SHIP_DELAY to the original tender.	
03	✓ Downstream effects checked, not assumed. Settlement export diffed · finance reconciliation sampled · ops console unaffected.	
VERIFICATION BOUNDARY		
04	◆ What it does <i>not</i> claim. Split shipments untouched — no assertion. Tax reversal never exercised — no assertion.	

What you need to ask: *Does the packet state its own boundary—including what it didn't prove?"*

Specification compliance ≠ behavioural correctness.

A wrong red costs an investigation. A wrong green costs an incident.

SPECIFICATION COMPLIANCE

Did the output match the visible instruction set?

✓ Answerable from the diff alone.



BEHAVIOURAL CORRECTNESS

Did the system behave as intended — preserving what should stay fixed?

🔄 Requires a verified baseline — not just a passing suite.

A change can satisfy the first *while failing the second*.

Precisely the wrong thing, built with confidence.

✓ diff clean

✓ tests pass

behaviour unverified

What you need to ask: *“Did anyone verify behaviour — or did we just read the diff?”*
the diff?”

The same Monday, run properly.

MONDAY · 09:12 · SAME MEETING

“What do we actually know about the current behaviour?”

01 · RECOVER

Verified actual-behaviour
baseline

02 · AUTHORISE

Narrow intended delta

03 · PIN

Characterisation tests before
code

04 · PROVE

Proof packet travels with the
change

Six months later, the next change on this path takes *days, not weeks*.

The questions to ask before the keyboard.

Not one line of code between them — just the handful that would have stopped Monday's change.

RECOVER

What the system actually does

“What do we actually know about the about the current behaviour?”

“Do characterisation tests exist for for what we’re preserving?”

AUTHORISE

What we are allowed to change

“Evidence and intent — two documents, or one?”

“Does the delta name what stays — not — not just what changes?”

PROVE

What the change actually did

“Does proof travel with the change — or do we take ‘done’ on trust?”

“Did anyone verify behaviour — or did we just read the diff?”

You won’t write any of the code. You ask these first.

Your turn: four changes, one Monday.

Place each one. Which gets the full treatment — which should nobody touch yet?



90 seconds with a neighbour. Then hands.

Rigour is a dial, not a doctrine.

Match the rigour to the change. Maximum everywhere is bureaucracy; minimum everywhere ships regressions.

LIGHT	STANDARD	HEAVY	OFF THE DIAL Not ready
Finance rounding bug confined, reversible	Billing delta baseline recovered	Payment approval rule regulated; audit trail	Warehouse orchestration behaviour disputed — recover first

Under-specification is the dominant risk. AI turns vague work into faster mistakes.

What sets the dial.

Five variables, three questions — feeding one decision: how much rigour does this change earn?

POTENTIAL IMPACT

Risk & compliance exposure

Blast radius & reversibility

UNCERTAINTY

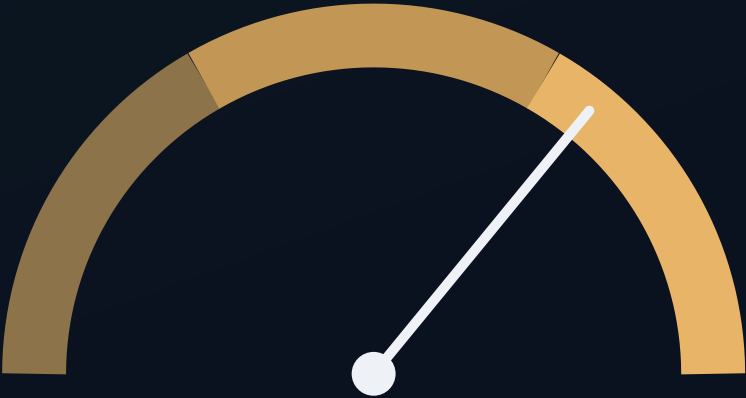
Novelty & ambiguity

Codebase condition

Familiarity

EVIDENCE

The recovered baseline



LIGHT STANDARD HEAVY
Y

The reading sets the rigour — not a blanket policy.

What you need to ask: *“How much could this hurt, how sure are we, and what have we verified?”*
we verified?”

Three levels of rigour.

Each setting on the dial has an artefact. The dial chooses; the artefact delivers.

LEVEL 1	LEVEL 2	LEVEL 3
Context brief Change boundary and ordinary review. The lightest possible record. RIGHT FOR Routine, well-tested, low-blast-radius work	Living spec Intent, constraints, preserved behaviour and open questions — maintained as the work evolves. RIGHT FOR Changes where several roles must stay aligned aligned	High-control pipeline Requirements, tasks, validation and traceability linked throughout. RIGHT FOR Regulated or high-consequence work

A change whose current behaviour is disputed fits none of these — recover the baseline first.

Who owns what.

The model assigns the decisions — it does not dissolve them.

THE QUESTION	THE OWNER
Local truth Does the actual-behaviour spec match reality?	The engineer who defined the slice
The delta What is authorised to change?	Product
The change boundary How wide is too wide?	Engineering leadership
Proceeding under uncertainty Is the residual uncertainty acceptable?	Reviewer

“The agent said it was done” answers none of these.

“Isn’t this just the process we killed?”

A fair objection — and the one this room might have already been forming. The honest answer is no, and the distinction is the whole point.

NOT THIS

A stack of approvals after the fact.

Governance theatre — process bolted on once the work is already done. That is what we killed, and we were right to.

BUT THIS

The right work easier to produce — the wrong work harder to merge.

Structure that shapes the work upstream, not paperwork that inspects it downstream.

AND IT DOESN’T LIFT WITH BETTER TOOLS

Stanford: 51 enterprise deployments: coding alone stays human—AI collaboration — the review obligation does not lift as the tooling improves.

What you need to ask: “*Are we shaping the work upstream — or inspecting it after fact?*”

The tools will change.

The underlying problem does not.

What to do Monday.

Start with *one workflow*. Not one mandate.

THE RIGHT FIRST PILOT ANSWERS YES TO ALL FOUR

- ✓ Runs in production today.
- ✓ Real downstream consequences if it breaks.
- ✓ Changes often enough that the baseline pays back.
- ✓ Not shared core infrastructure.

When it works, copy the conditions that made it honest — not the pilot.

Measure the pilot on one question: how many changes were specification-compliant but behaviourally incorrect?

Two specs. A bounded delta. Evidence-first review.



Download the slides and the supplementary guide.

Brownfield AI — coming soon.

Shipping Safe Changes to Existing Systems with Coding Agents

ROMAN LOBUS · PMI AGILE 2026

Questions?