

INNOQ MEETUP COLOGNE

Shift Right

How business gaps move from modeling to review, acceptance, and production

INNOQ



DANIEL WESTHEIDE
SENIOR CONSULTANT

Show of Hands

- Requirements arrive as tickets, and the PO accepts at the end
- You still review the code yourself, not just the AI's summary
- Your merge requests have a description

Shift Left, Shift Right

Shift left

Test and find defects as early as possible in development

Shift right

Keep testing and observing in production: usage, performance, security

Shift right adds feedback on top of shift left. Both: fail small, recover quickly.

Definitions after Red Hat

With coding agents, business gaps drift right.

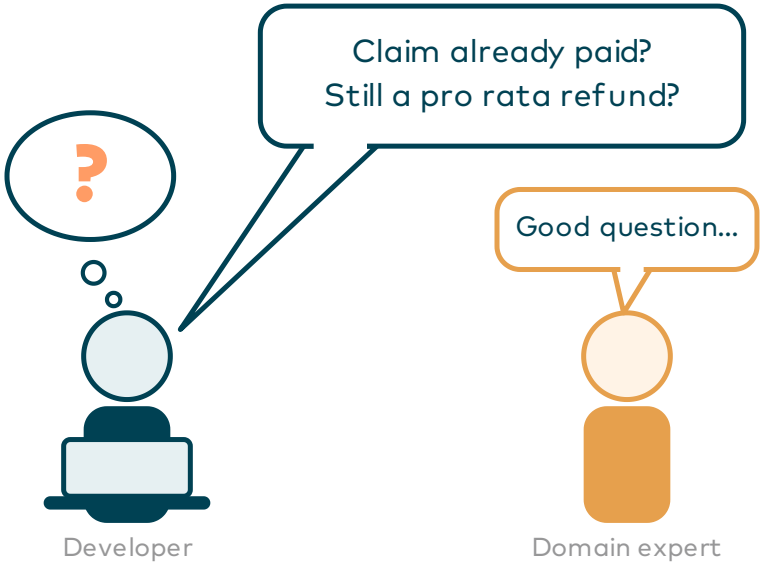
Not as an extra source of feedback, but as a substitute for the one that went missing.

A hypothesis, not a finding.

We Used to Stumble Over Gaps

```
def refund(policy: Policy, date: LocalDate): Money =
  val unused = policy.unusedShare(date)
  policy match
    case p: Active    => p.premium * unused
    case p: ClaimPaid => ???
```

Policy is a sum type: Active or ClaimPaid. Matching on it, you meet both cases, and gaps tend to surface as holes you write down yourself.



Evans (knowledge crunching) · Naur (programming as theory building) · Reeves (code as design)

The Model Bridges Silently

```
def refund(policy: Policy, date: LocalDate): Money =
    val unused = policy.unusedShare(date)
```

1. ClaimPaid is a case of Policy, but nothing looks at it
2. `policy.premium * unused`

It compiles. It reads as complete. The tests are green.

1 A decision belongs here

Does a paid claim change the refund? Matching on Policy, you would likely hit this case.

2 The model's default

Pro rata, always. Plausible, so nobody questions it.

No ???, no question. Nobody decided anything.

The Gap Moves Right

Modeling
Found through friction

Review
A convincing story

Acceptance
Green

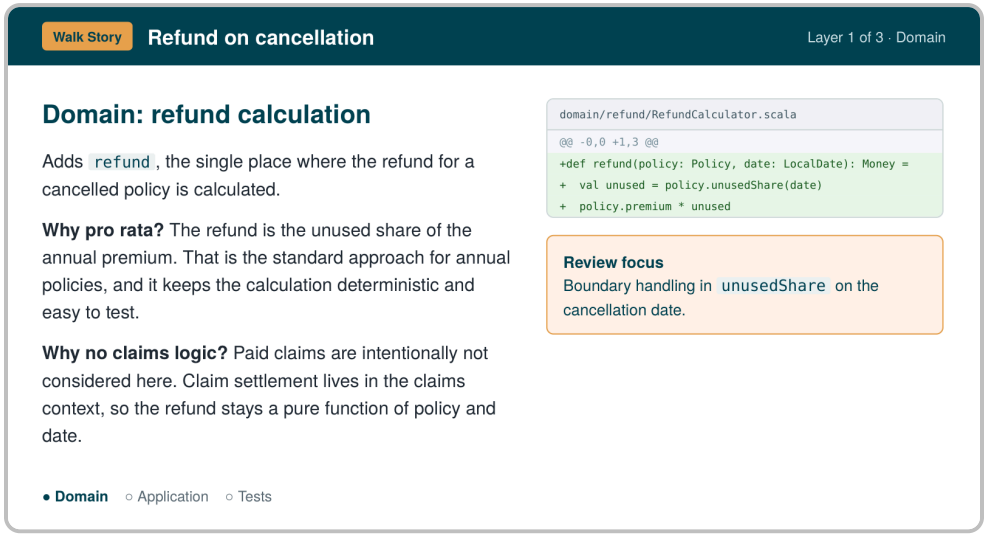
Production
An incident

The gap doesn't disappear. Only the later stages are left, and each one is a weaker detector.

Review: A Convincing Story

A generated story gives a plausible reason for each decision, including the ones nobody made.

Plausibility used to be a rough sign of understanding. Now it is cheap.



"Why no claims logic?" Nobody decided that. The model made up the reason after the fact.

A gap is, by definition, something nobody wrote an acceptance criterion for.

Spec, code and criteria come from the same source. They are consistent, but not independent.

Our refund example passes acceptance ✓

Given a policy cancelled on day 90 of 365, the unused 275/365 of the premium is refunded.

Production

- Some gaps are found there, by a customer rather than by the team
- It costs an incident instead of a conversation

In DevOps, shift right adds feedback. Here, it replaces feedback that went missing.

Consistent With the Data

+54%
bugs per developer

+243%
incidents per PR

+31%
more PRs merged without review

These numbers don't show business gaps. They're what you'd expect to see if there were many.

Sources: Faros AI Engineering Report 2026 (vendor telemetry); DORA 2025: AI adoption lifts throughput, instability still rises (survey correlation)

What Does This Mean for How We Work?

Requirements in. Acceptance out. A clean line between business and development.

The Handover Model Hid a Detector

- Developers were the accidental gap detector: whoever had to implement stumbled over unclear requirements
- The separation absorbed errors it created itself
- The agent takes over the stumbling, and bridges instead of asking

It's not news that handover doesn't work. What's new is that we can no longer get away with it.

The People With Context Belong Where Decisions Happen

- Business and developer at the same machine, working with the agent
- The agent bridges a gap, the business side sees it and says "we never do it that way"
- Discovery moves left again, through watching and objecting instead of stumbling

The price

Scarce domain expert time, at the very moment we promised speed

The caveat

Presence only helps if the agent's choices are visible to them. And experts only notice what they know consciously

Spec-Driven Grilling

BMAD and similar approaches: an agent interviews you before anything is built.

What it gets right

Gap hunting becomes the agent's job

Best with developer and domain expert grilled together

Consistent context for every later step

Open questions and assumptions are written down

Tacit Knowledge

Experts know more than they can tell.

Knowledge is not a deposit you mine with questions. It tends to emerge when people work with the thing.

Imagined Questions

The developer, by hand

Asks because she can't write the next line

The code forces the question, at a concrete case

The interviewing agent

Has to imagine its questions

Imagined questions tend to be generic

Not human against agent: an agent working from concrete cases and types can ask forced questions too.

Prose Is Not Formal

Prose

Can look complete without deciding every case

Formal structure

Gives the question a place to show up: an exhaustive match, a type with a case that has no answer yet

Likelier, never certain: it can only force what the model distinguishes.

Some questions only come up when you implement.

The interviewing agent doesn't implement. The implementing agent doesn't ask.

Ideas for the Discussion

Ask first

The agent grills before building (spec-driven, BMAD-style). Works for gaps it can imagine

Stories first

Domain Storytelling or Event Storming, with the people who know. The stories become the examples

Examples first

The trio writes examples as tests before the agent builds. The expected value forces the decision

Make assumptions visible

The agent lists what it assumed, in domain language, or interviews the author about why (diffwalk)

Each one should bring in something that doesn't come from the same source.



Where would you bring the friction back?