

INNOQ TECHNOLOGY DAY 2026

Wasted Opportunities

Why agentic teams should slow down

INNOQ

DANIEL WESTHEIDE
SENIOR CONSULTANT

Last Year: First Agile, Then Agentic

- Agentic development needs the right foundations
- Speeding up implementation alone just creates a new bottleneck
- This year: what does the data say, and what should we do with the freed capacity?

Coding Got Faster

- Controlled experiments: tasks completed significantly faster with AI assistance
- Agentic coding: delegating whole tasks, not completing lines
- Example from my project: $75 in tokens for what I'd have estimated as 5–6 days of work

e.g. Peng et al. (2023): 55.8% faster on a controlled task with GitHub Copilot

Delivery Didn't

AI is an amplifier: it strengthens what's already there, good or bad (DORA 2025, 2026)

Without the right foundations:

Instability and verification tax

More AI adoption: higher throughput, but also higher instability. More code means more to review, maintain and operate (DORA)

Review becomes the bottleneck

Review time +441% · PRs merged without review +31% (Faros AI 2026) · AI-generated PRs wait 4.6× longer for first review (LinearB 2026)

Sources: DORA State of AI-assisted Software Development 2025; DORA ROI of AI-assisted Software Development 2026; Faros AI Engineering Report 2026; LinearB 2026

Where the Time Goes

Review

More code meets review capacity that hasn't grown

Acceptance

Implemented faster than the business can sign off

Decisions

Building takes hours, waiting for a decision takes days

The bottleneck didn't disappear. It moved.

"The bottleneck moves from building to deciding." · Melissa Perri on LinkedIn, 2026

And the Bill Comes Later

- Every line of code creates maintenance effort for years
- Double the output requires halving maintenance cost per unit

MAINTENANCE COST PER UNIT, WITH DOUBLED OUTPUT	GAINS GONE AFTER
unchanged	19 months
doubled	5 months
halved	-

Source: James Shore, *You Need AI That Reduces Your Maintenance Costs*

Queueing Theory in Practice

A Simple Model

How Queues Behave

- Every phase has a capacity, and work arrives at a certain rate
- Utilization = demand ÷ capacity
- Waiting time does not grow linearly with utilization

UTILIZATION	WAITING TIME
50%	≈ 1× processing time
75%	≈ 3× processing time
90%	≈ 9× processing time
95%	≈ 19× processing time
over 100%	queue grows without bound

Simplified (M/M/1): $\text{waiting time} \approx \frac{\text{utilization}}{1 - \text{utilization}} \times \text{processing time}$

Step 1: Before Agents

Discovery

Demand 10 · capacity 15 items/week



67%

Waiting ≈ 2.0× processing time

Build

Demand 10 · capacity 12 items/week



83%

Waiting ≈ 5.0× processing time

Review

Demand 10 · capacity 14 items/week



71%

Waiting ≈ 2.5× processing time

Busy, but flowing. Build is the most utilized phase.

Step 2: Agentic Coding

Discovery
CHANGED

Demand 20 · capacity 15 items/week

133%

Queue grows without bound

Build
CHANGED

Demand 20 · capacity 40 items/week

50%

Waiting ≈ 1.0× processing time

Review

Demand 20 · capacity 14 items/week

143%

Queue grows without bound

Build is no longer the bottleneck. Discovery and review are overloaded.

Step 3: Approve-Only Review

Discovery

Demand 20 · capacity 15 items/week



133%

Queue grows without bound

Build

Demand 20 · capacity 40 items/week



50%

Waiting ≈ 1.0× processing time

Review

CHANGED

Demand 20 · capacity 40 items/week



50%

Waiting ≈ 1.0× processing time

Defects slip through

The review queue disappears, but only because review no longer checks anything.

The Reflex: An Output Treadmill

- Build capacity is cheap now, so keep it busy
- Some "AI-native" pipelines: a feature request in Slack becomes a PR, gets reviewed and deployed, all by agents
- Requirements too slow? Accelerate them as well: "hours instead of weeks"

80% of features in the average software product are rarely or never used

Source: Pendo, 2019 · see also: *Using the Space, Not Optimizing the Treadmill*

A Critical Look at AI Discovery Tools

Agentic Planning and Spec-Driven Development

BMAD Method

Analyst, PM and architect agents: brainstorming, market research → product brief → PRD → architecture → stories

GitHub Spec Kit, Kiro & co.

specify → plan → tasks → implement, with specs as versioned artifacts

- The promise: replace "iterative discovery", from "2–3 weeks to 6 hours"
- Assumes someone at the laptop already knows the requirements
- Good at: giving agents consistent context, exposing vague answers
- But: the spec is only as good as the domain knowledge in the room

The Extraction Paradigm

The hidden assumption: domain knowledge sits in experts' heads, waiting to be retrieved by the right questions

Tacit

"We can know more than we can tell"

Polanyi, 1966

Social

Emerges through shared practice, context and trust

Nonaka, 1991 (SECI)

Takes time

Insight needs incubation, and sleep

Wallas, 1926; Cai et al., 2009

Three weeks into six hours compresses the time, and with it the result. Lossy compression.

See: *Hail Mary: Why Domain Knowledge Cannot Be Extracted from Experts*

Waterfall with a Fresh Coat of Paint

- Plan first, then implement: discovery as a phase, not a continuous activity
- The agent interviews whoever has access to the tool, often a proxy product owner
- DDD hits the same wall for the same reason: no genuine access to domain experts
- Where it works: as a listener when developers and domain experts talk to each other, not as a replacement for that conversation

Fix the organisation, not the toolchain.

See: *Spec-Driven Development is Domain-Driven Design's Impatient Cousin*

Pick a Number Between 1 and 10

Humans

28%

say 7. Because it feels the **least** obvious

Kubovy & Psotka, 1976

Language models

~80%

say 7. Because it is the **most** likely continuation

Coronado-Blázquez, 2025

Same answer, opposite reason. The model has the behaviour, not the reason.

Synthetic Users: Right Average, Wrong Distribution

- The promise: interview AI personas instead of customers. No recruiting, results in minutes
- NN/g, same open question: 604 humans spread over 50+ labels, 120 synthetic users over about a dozen
- Survey benchmarks: no model beats simple statistical baselines, segment differences inflated 2–4×

Research is how you get surprised. Synthetic users can't surprise you.

Sources: Nielsen Norman Group, *Synthetic Users vs. Human Respondents* (2026); Chen, Zhu & Zheng (2026); via Tara Maria Bosenick, *Seven. Always seven.*

AI in Discovery

AI helps with **generating and processing**, not with **contact with reality**.

Interviews	**Ideation**	**Assumptions**	**Testing**
not replaceable	widen the solution space	find blind spots	cheaper prototypes

The Thesis

Invest the freed-up capacity not in more output, but in **deeper, more rigorous discovery**.

DORA AI Capabilities Model: a user-centric focus is one of seven capabilities that amplify AI's positive impact (DORA 2025)

What Rigorous Discovery Looks Like

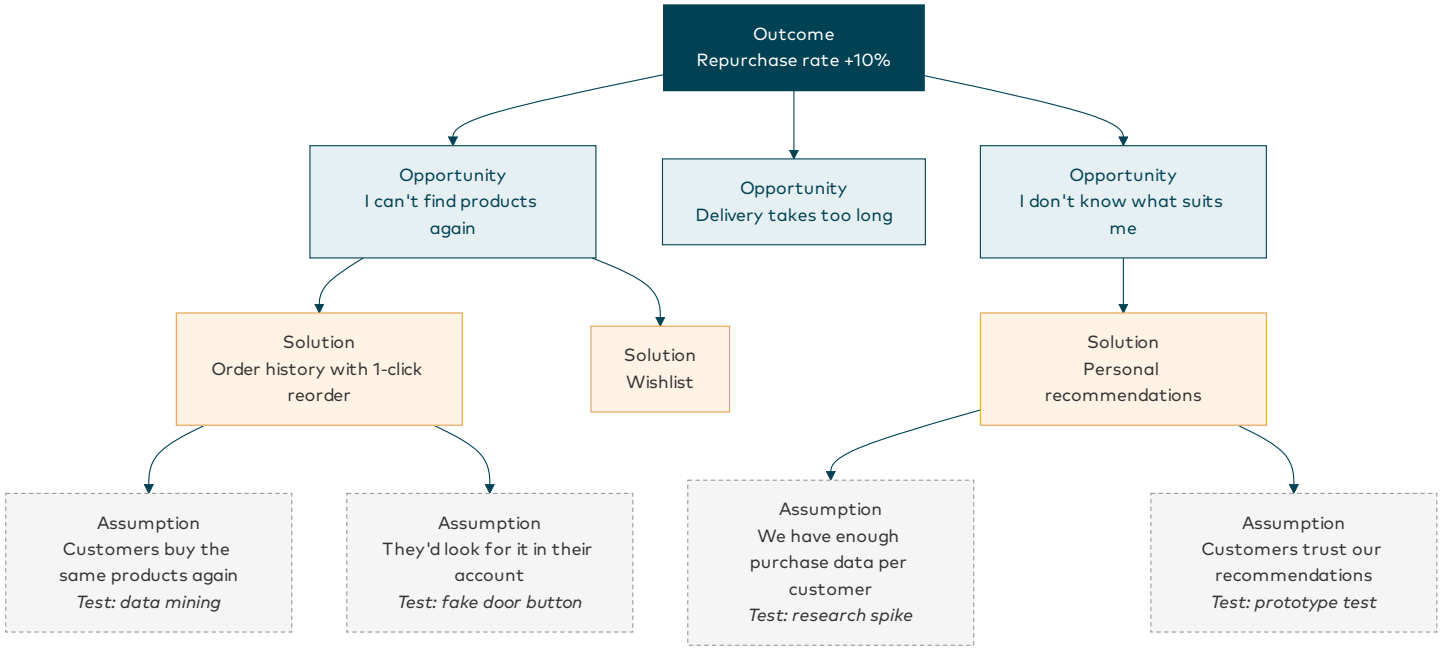
Continuous Discovery

At a minimum **weekly** touchpoints with customers, by **the team building the product**, with **small** research activities, in pursuit of a **desired outcome**.

Not a phase before delivery, but a loop with it.

After Teresa Torres, *Continuous Discovery Habits*

The Opportunity Solution Tree



After Teresa Torres, *Continuous Discovery Habits*

Stories, Not Opinions

Opinions

"How do you usually ...?"
"Would you use ...?"
"What would you like?"

Idealisation, speculation, feature requests

Stories

"Tell me about the **last time** you ..."
"What happened next?"
"How did you go about it?"

Actual behaviour, real context

After Teresa Torres, story-based interviewing

Being There Beats Measuring

Doing the work yourself · you're an actor, not an observer

Being there and observing · contextual inquiry

Regular presence · exposure hours

Story-based interviews

Structured interviews, A/B tests

Exposure hours: Jared M. Spool · see: *Faster Is Not Better*

Test Assumptions, Not Whole Ideas

Desirability	Usability	Feasibility	Viability	Ethics
Do they want it?	Can they use it?	Can we build it?	Should we?	Is it right?

- Prototype tests, one-question surveys, data mining, research spikes
- Define success criteria up front, compare solutions against each other
- Many tests fail, and that's the point

After Teresa Torres, *Assumption Testing*

Discovery as a Filter



- What's discarded in discovery needs no review, no acceptance, no maintenance
- Halving maintenance cost also works by building less
- The cheaper building gets, the greater the temptation to skip this filter

Step 4: More Capacity on the Product Side

Discovery
CHANGED

Demand 20 · capacity 28 items/week



71%

Waiting ≈ 2.5× processing time

Build
CHANGED

Demand 20 · capacity 30 items/week



67%

Waiting ≈ 2.0× processing time

Review
CHANGED

Demand 20 · capacity 21 items/week



95%

Waiting ≈ 20× processing time

Discovery recovers. Review stays dangerously close to its limit.

Step 5: Discovery Filters

Discovery

Demand 20 · capacity 28 items/week

71%

Waiting ≈ 2.5× processing time

Build

CHANGED

Demand 10 · capacity 30 items/week

33%

Waiting ≈ 0.5× processing time

Review

CHANGED

Demand 10 · capacity 21 items/week

48%

Waiting ≈ 0.9× processing time

Half of the ideas fail assumption tests. What isn't built doesn't need review.

What This Means for Teams

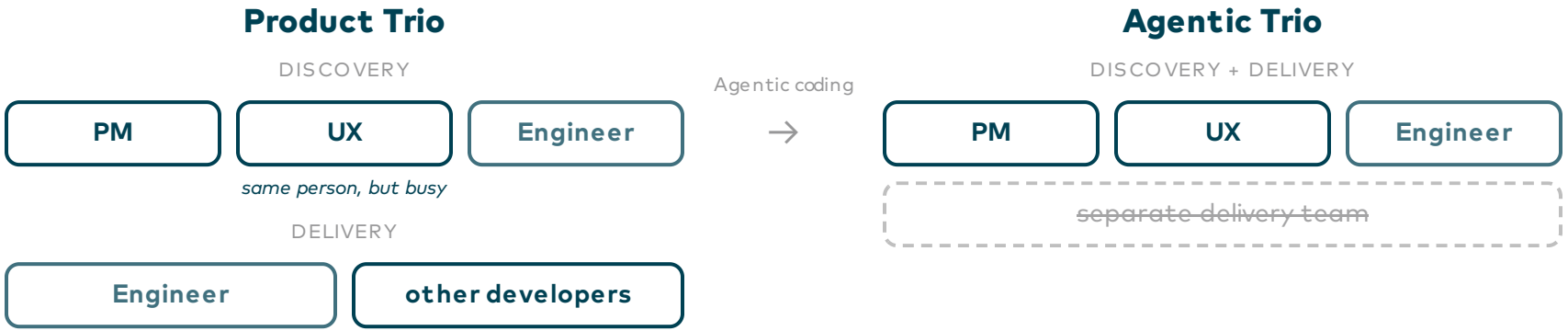
The Product Trio

- Discovery: product manager, UX and developer, together and continuously
- Delivery: the developer together with other developers, a larger team
- A small group can decide quickly and locally
- Three perspectives enrich which assumptions get questioned and which solutions get found

After Teresa Torres, *Continuous Discovery Habits*

The Agentic Trio

Why developers finally have time for discovery



- All three use coding agents to build solutions themselves
- No more developers who only build: discovery becomes part of every developer's role

Developers in the Interviews

Reading the ticket

Knows *what* to build

Understanding arrives second-hand

Hearing the story

Knows *why*

The whole trio shares the intent

Not a handover of understanding, but a shared one.

After Teresa Torres, *Continuous Discovery Habits*

What Developers Bring to Discovery

Feasibility risks

Assumptions nobody else sees

A wider solution space

Knowing what's technically possible

System constraints, early

Architecture, dependencies on other teams

Prototypes in hours

Built with agents, for an assumption test, then thrown away

Research spikes

Test feasibility before anyone commits

Data mining

Answer assumptions with data you already have

Not more code. Better decisions about which code to write.

Key Messages

1. AI made coding faster, but not delivery. The bottleneck moved to review, acceptance and decisions
2. Speeding up the output treadmill with faster specs and more features makes the queue longer
3. AI discovery tools assume knowledge can be extracted, but it is tacit, social and takes time
4. Invest the freed-up capacity in deeper, more rigorous discovery: it filters what gets built
5. Agentic development finally frees developers to contribute where their perspective matters most



Slow down where it matters:
in deciding what to build.

Everything after that gets faster.

Read More



12 theses on digital product development

agentictrio.org

On the INNOQ blog:

- Spec-Driven Development is Domain-Driven Design's Impatient Cousin
- Hail Mary: Why Domain Knowledge Cannot Be Extracted from Experts
- Using the Space, Not Optimizing the Treadmill
- Faster Is Not Better

Shift Right: How business gaps move from modeling to review, acceptance, and production
INNOQ Technology Day Meetup · Thu, 12 Nov 2026, doors open 18:00 · INNOQ Köln, Spichernstraße 44

Thank you! Questions?

INNOQ
www.innoq.com



Daniel Westheide
https://danielwestheide.com
@kaffeecoder

innoQ Deutschland GmbH

Krischerstr. 100
40789 Monheim am Rhein
(+49) 2173 3366 0

Ohlauer Str. 43
10999 Berlin

Ludwigstr. 180 E
63067 Offenbach

Kreuzstr. 16
80331 München

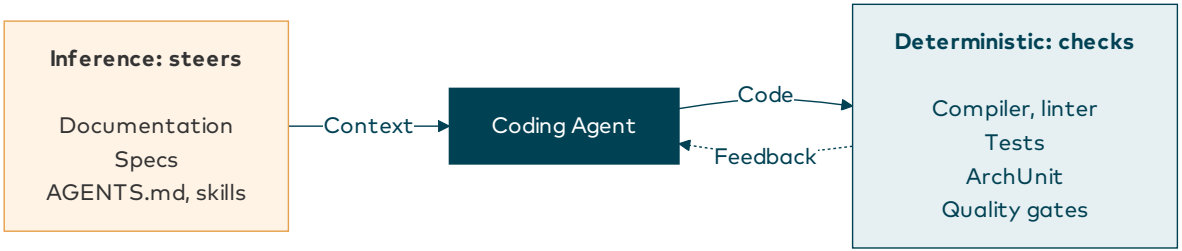
Wendenstraße 130
20537 Hamburg

Spichernstraße 44
50672 Köln

Backup

Harness Engineering

Harness: everything around the coding agent that steers and checks it, so that unreliable output reliably turns into production-ready code



Translate as much as possible from left to right: documentation becomes rules. The more is secured on the right, the more safely PM and UX can build.

Who Builds What

The developer in the trio

The system-specific harness: tests, fitness functions, architecture rules for *this* system

Choosing and adapting agents and skills for this product

Platform and enabling teams

The shared foundation: CI/CD building blocks, sandboxes, observability, curated skill libraries

Enabling teams help trios get started

Team types after Matthew Skelton & Manuel Pais, *Team Topologies*

From Producer to Steward

- The developer holds the mental model of the system
- Nobody else in the trio can take that on
- The core of software development: why the system is built the way it is, which decision justifies which structure
- Every line nobody understands anymore is maintenance burden that can't be paid down

Peter Naur, *Programming as Theory Building* (1985)

How Do PM and UX Learn to Build?

Harness first

Guardrails before anyone builds, so little can break

Ensemble work

The agent types, the humans navigate. Learning happens while working, review happens live

Graduated autonomy

Prototypes in a sandbox → small changes within guardrails → features behind flags

It's not about learning to program. It's about learning to work with agents.

AI-Moderated Interviews: A Different Story

- Real people, AI as the interviewer: hundreds of interviews in parallel
- Useful for breadth: many voices, fast synthesis
- But if nobody on the team hears the stories, the team doesn't learn
- Continuous discovery is done *by the team that builds the product*

e.g. Listen Labs, Outset, Maze