INNOQ TECHNOLOGY DAY 2026

No Constrain - No Gain

INNOQ



JOHANNES RAVE
SENIOR CONSULTANT · INNOQ

👋

Hi



Johannes Rave
Señor Consultant

It's *difficult to deny* that LLMs in agentic loops *produce working code*

- They write code **faster** than I do
- The code is often **quite alright** at the local level
- They **know a whole bunch** of frameworks, APIs, SO answers by heart
- They can **research and explain** concepts, technology, domain knowledge
- They **write tests, docs, and boilerplate** without complaining
- They are **great at translating** between formats, protocols, languages
- They **don't get tired, distracted, or bored** at 4pm
- They turn **a sketch into a working prototype** in an afternoon
- *Bonus:* Claude thinks **my opinions are great**
- **...**

It's *tempting* to let them *decide*, too

- They **suggest features** nobody asked for, and often good ones
- They **fill the gaps** in a story with plausible decisions
- They **name the concepts** before I've thought about them
- They **pick the defaults**: data model, error messages, what counts as done
- They **write the acceptance criteria** for their own work
- They **rarely stop to ask** when a story leaves something open
- *Bonus:* "Want me to also add …?" **Usually, yes.**
- …

SO JUST *RALPH LOOP* IT ALL

However

Risks with agent-driven projects

- **Coding agents produce more code than reviewers can read**
  In one company, reviews per reviewer doubled, and commented reviews fell from 39 % to 21 % of PRs. He et al., Jul 2026
- **Passing tests let many wrong agent patches through**
  19.8 % of previously passing SWE-bench patches fail once the tests are strengthened. SWE-ABS, Feb 2026
- **Where a story leaves a gap, the agent decides without asking**
  When a constraint is missing, agents ask in 4.9 % of runs, and none of the runs succeed. AM-Bench, Oct 2026
- **People working with an agent tend to follow the agent's choices**
  Analysts followed a preference the agent got by coin flip in 86 % of open decisions. Anthropic, Oct 2026
- **Delegating the work to an agent leaves people with less understanding of the code**
  Students with ChatGPT recalled 41 % (53 % without) and claimed 45 % of their code (81 %). Bergh et al., Sep 2026
- **Repositories with checked-in agent instructions show half the growth in complexity**
  Cognitive complexity rose 27 % with agent configuration and 53 % without, in an observational study. RAMP, Aug 2026

Who made these decisions?

From the decisions in Story 0007 of the office booking app, shortened:

- Desk positions are stored on a grid and set via seed data. There is no editor yet.
- A desk booked for any part of the day counts as occupied for the whole day.
- Zones are decoration, desks may sit inside them.

- Agents make *explicit* proposals for "perceived" holes in requirements
- Accepting a proposal is easier than deciding
- Agents make *implicit* decisions without surfacing anything

It is neither clear that no "holes" remain in our concepts , nor that all requirements have been followed, and we might be less engaged with the result.

It is *neither clear* that no "holes" remain in our concepts ,
***nor* that all requirements have been followed,**
and we are *less engaged* with the result.

It is easy to lose

interest

understanding

control

Staying *in control*

Invest time into concepts, models and decisions

(or re-establish them in brownfield projects)

People bring

empathy, *taste* and *judgement*

to the project

Designers and developers together

Designers are good at

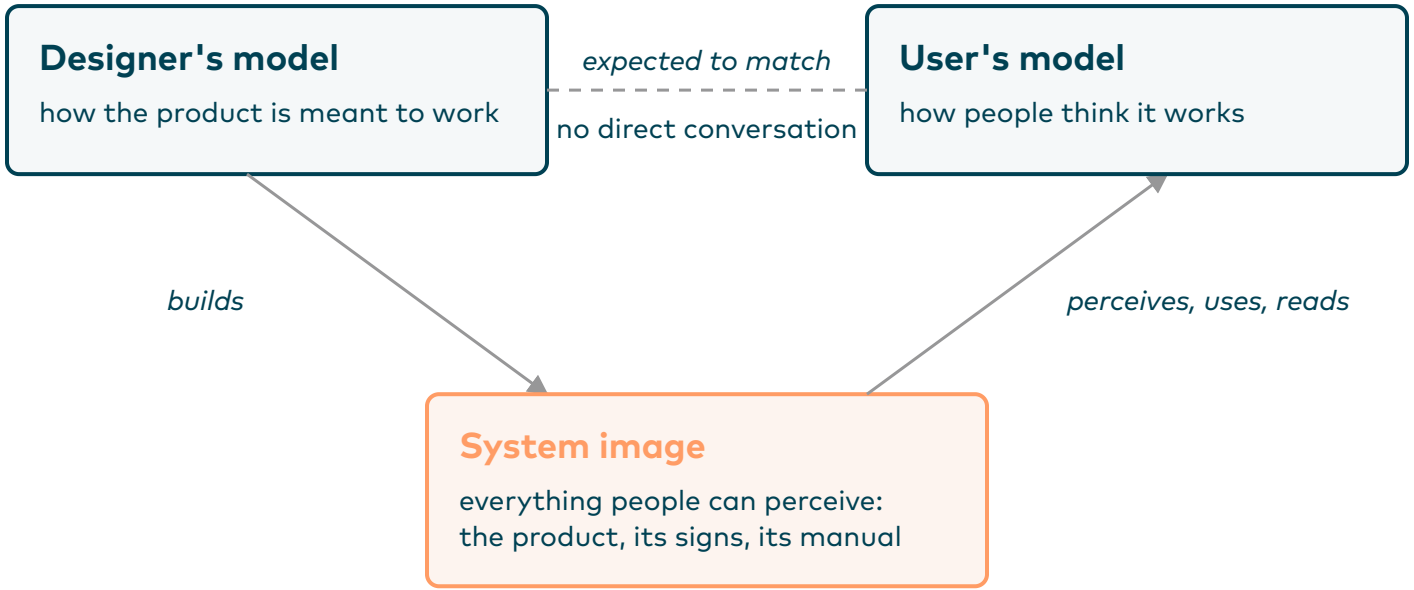
- finding out what users need and how they think about their work,
- turning that into flows, screens and words people understand,
- noticing when the product suggests something it doesn't do.

Developers are good at

- thinking in systems, states and edge cases,
- keeping rules consistent across the whole application,
- knowing which decisions will be expensive to change later.

- A team already has people who know both sides. When they move closer, they can work out the concepts together, and each builds them into their own part, the interface or the code.

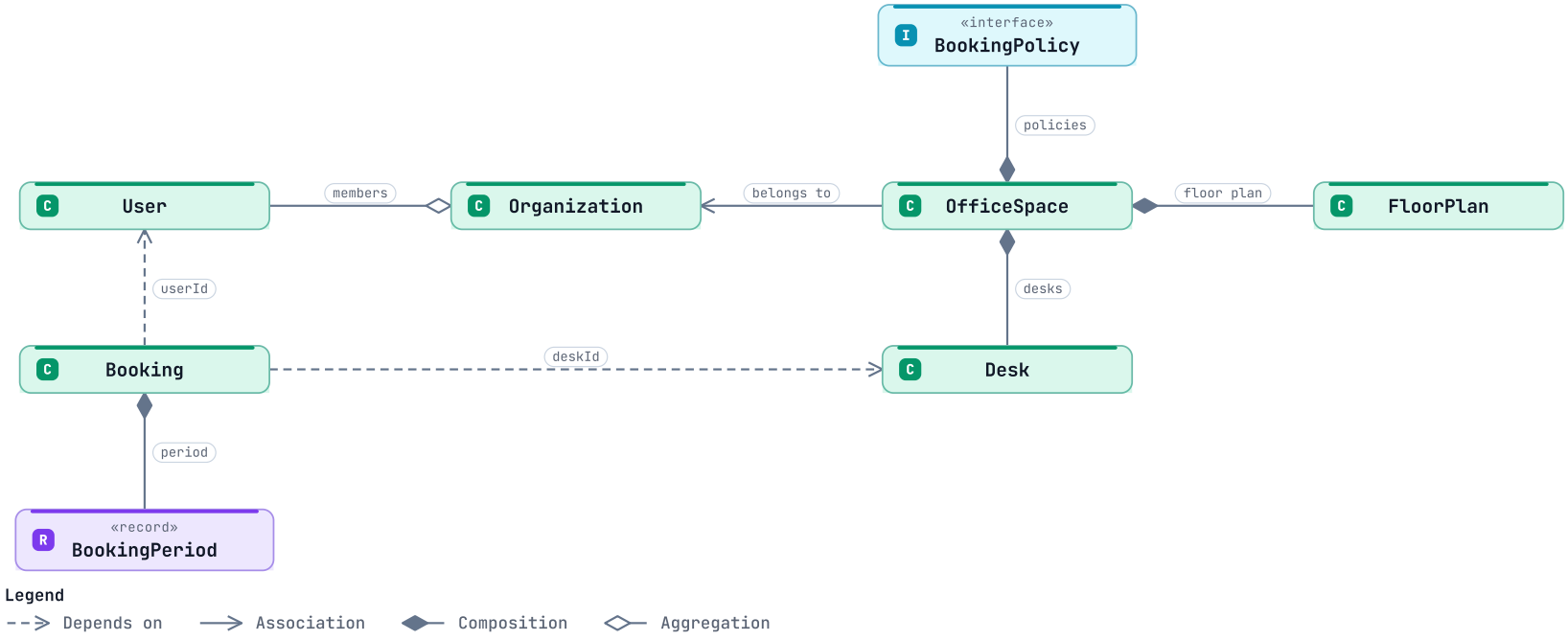
The system image



"The entire burden of communication is on the system image."

Don Norman, The Design of Everyday Things, 2013

The domain model of the office booking app



Types hold the domain model

```
data class BookingPeriod private constructor(
  val start: LocalDateTime,
  val endExclusive: LocalDateTime,
) {
  companion object {
    fun of(start: LocalDateTime, end: LocalDateTime): BookingPeriod {
      require(end > start)
      require(start.minute % 15 == 0)
      return BookingPeriod(start, end)
    }
  }
}
```

- A type gives a concept of the domain a name, and its constructor states the rules.
- The compiler makes sure every `BookingPeriod` went through these checks, for people and agents alike.
- In an experiment, an agent fixed 41 to 63 % of the errors with detailed type messages, and 25 to 41 % with only failing tests. When the types checked, 98 % of the fixes also passed all tests.

Krishnamurthi and Flatt, Type-Error Ablation and AI Coding Agents, Jun 2026

Re-enforce constraints during product development

What do we mean by *harness*?



Armour



Leather harness



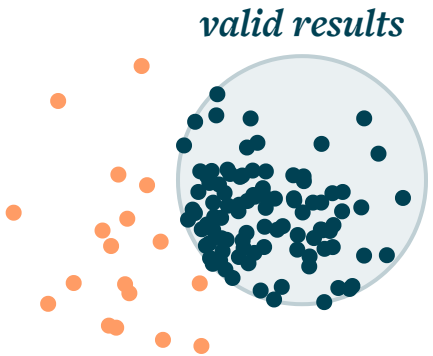
Climbing harness



Exoskeleton

Photos: Walters Art Museum (PD, Wikimedia Commons), KinkyRubberDoc, rare-gallery.com, Ekso Bionics

Making good results *likely*



AGENTS.md
moves all the results
a bit closer to valid

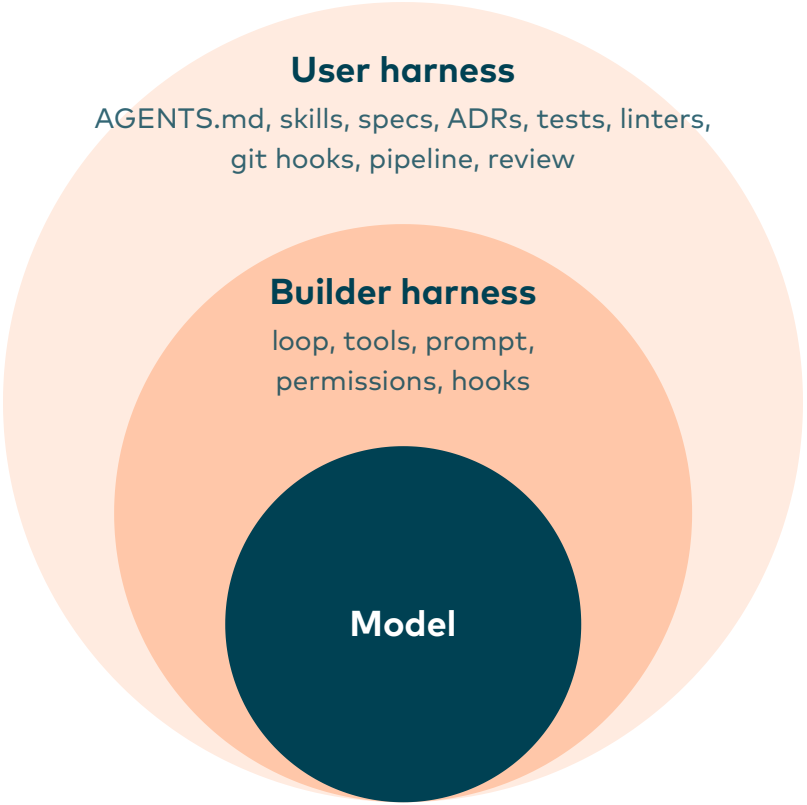
tests and types
catch the invalid results they actually check,
as long as somebody runs them

linting
makes the style more uniform,
but it barely touches behaviour

hooks
make sure the checks run every time

valid: 84 %

- valid result
- invalid result
- caught by a test, if the tests run
- all valid results



What Is the *Harness*?

- **Narrow:** the software the agent runs in. The vendor builds it.
- **Wide:** everything that keeps the agent on track. You build most of it.
- Birgitta Böckeler calls the two rings *builder harness* and *user harness* (2026)
- In this talk, *harness* means the wide sense

Birgitta Böckeler, Harness Engineering, 2026

Remain interested in your implementations

Simple and modular still pays off

- People and agents find their way.
- Changes stay local.
- We know where to review carefully.

"Once we have stable abstractions,

generating code on top of them is something we can think about in more mechanical lines."

- Martin Fowler

Domain logic

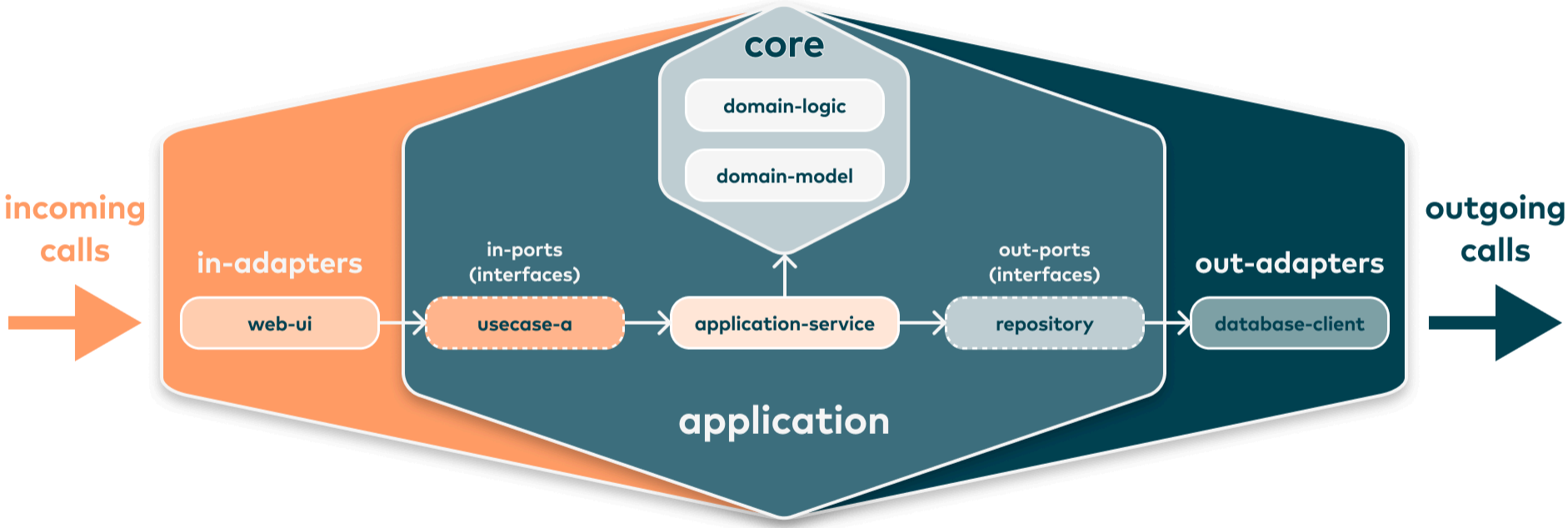
- *Critical* to get right
- Needs careful review
- Expresses what the business *does*
- … engaging to think about (I think)

Plumbing

- Tedious to set up
- Agents are great at it
- Correctness is mechanical
- Controllers, mappings, config

Splitting these well helps with reasoning, *simpler* implementation and knowning where to review rigorously!

ports and adapters



- dependencies only point inwards
- adapters and application are decoupled via interfaces
- core depends on NOTHING - it does its own thing!

Investing into re-aligning concepts and implementation

"As an evolving program is continually changed, its complexity, reflecting deteriorating structure, increases unless work is done to maintain or reduce it."

- **Meir M. Lehman**, *Laws of Program Evolution, 1980*

Meir M. Lehman, Programs, Life Cycles, and Laws of Software Evolution, Proceedings of the IEEE, 1980

Realignment and refactoring

- The goal is to find accidental complexity and remove as much of it as we can.
- We hold the code against the decisions we wrote down: stories, domain model, glossary.
- An agent can list where they no longer match. A person decides which side changes.
- It can happen while we build the next feature, as long as we plan time for it and look.

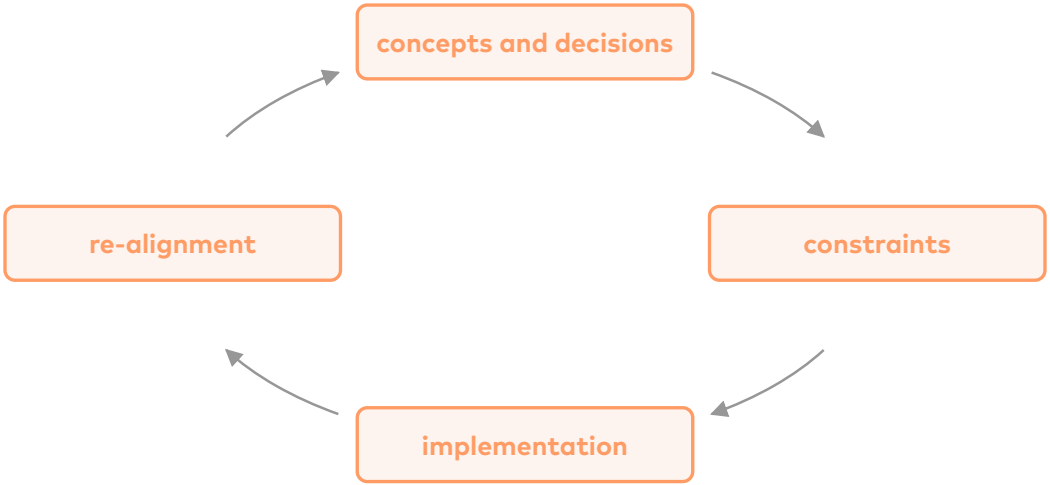
Staying *en current*

Our work in progress

The roles of developers and others working on digital products are changing. That new work however needs what people in product development already do:

- finding out what users need and how they see the product,
- naming the concepts and keeping them consistent,
- thinking in systems, states and edge cases,
- deciding what a requirement leaves open,
- and judging whether the result is right.

Supporting people who build complex products



- Agents can support people across the whole product development lifecycle. People stay in charge.

Agents *amplify* whatever you bring.

Chaos in -> more chaos. Structure in -> more structure.

"Design depends largely on constraints. The ability of the designer to recognize as many of the constraints as possible - their willingness and enthusiasm for working within these constraints."

- **Charles Eames**, *Design Q&A, 1972*



Thank you! Questions?



Johannes Rave Senior Consultant - INNOQ

johannes.rave@innoq.com