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

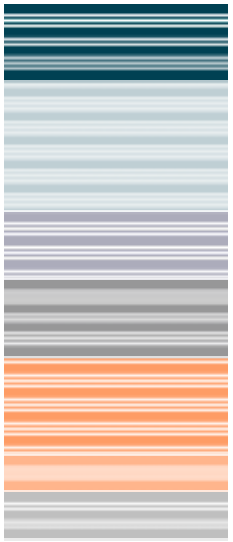
- **Coding agents produce more code than reviewers can read**
  In one company, reviews per reviewer doubled, and commented reviews fell from 39 % to 21 % of PRs. He et al., Jul 2026
- **Passing tests let many wrong agent patches through**
  19.8 % of previously passing SWE-bench patches fail once the 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

More code than we read



goal, checks, tasks, decisions 30

Story 0007 30 lines



domain 78

domain tests 130

use case and its tests 67

persistence 78

web: controller, template, CSS 98

web tests 36

seed data 49

Code and tests 536 lines

- Decision are taken quickly while the produced artifacts expand exponentially.

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*

Staying in control

In the current maelstrom, assumption get overtaken often.

This is my current thinking on *what helps* when developing products:

- Invest time into foundational decisions and concepts
  (or re-establish them in brownfield projects)
- Re-enforce these decisions during development
- Remain interested in your implementations
- Investing into re-aligning concepts and implementation

Invest time into concepts, models and decisions

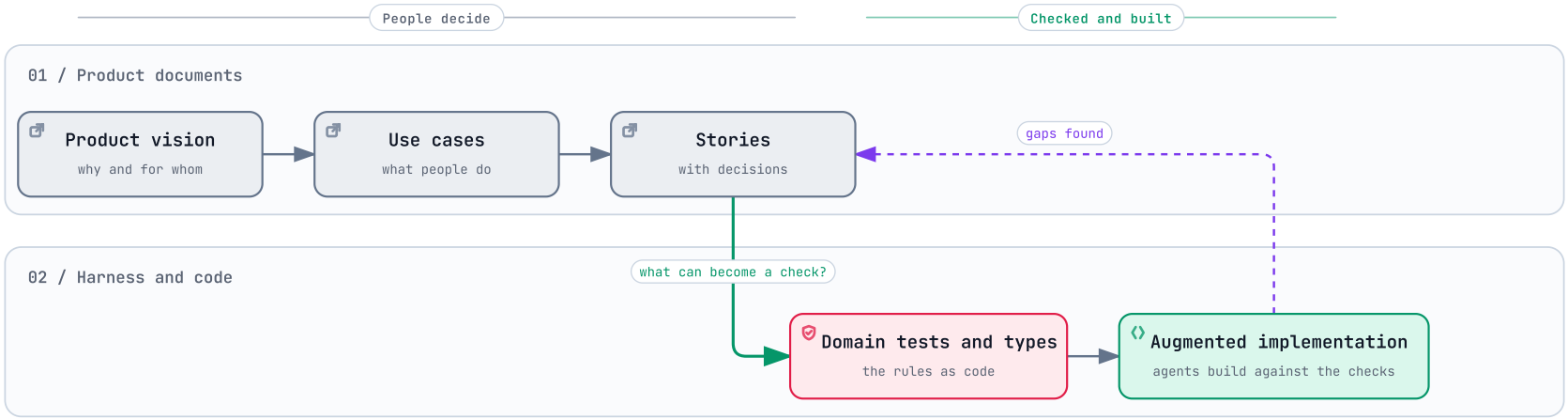
(or re-establish them in brownfield projects)

People bring

empathy, *taste* and *judgement*

to the project

From product documents to checks



- A lot of the work happens in the product documents - these form the foundation for many subsequent decisions.
- Over time, more of what the documents say is checked by the harness, and the harness gets harder.

Developers move toward product

- Developers spend less of their time writing code and more on judging and deciding. Chen and Stratton, Nov 2026

What lands on a developer's desk

- deciding what a requirement leaves open,
- naming the concepts of the product and keeping the names consistent,
- choosing among the defaults an agent proposes: data model, error messages, empty states,
- writing acceptance criteria before the code exists,
- checking whether the result does what users expect.

- They aren't trained for that, but they are good at systems thinking and at keeping things consistent. They may profit from some support.

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.

Programming as *theory building*

In a complex domain, code *isn't just* mechanical instructions for a machine:

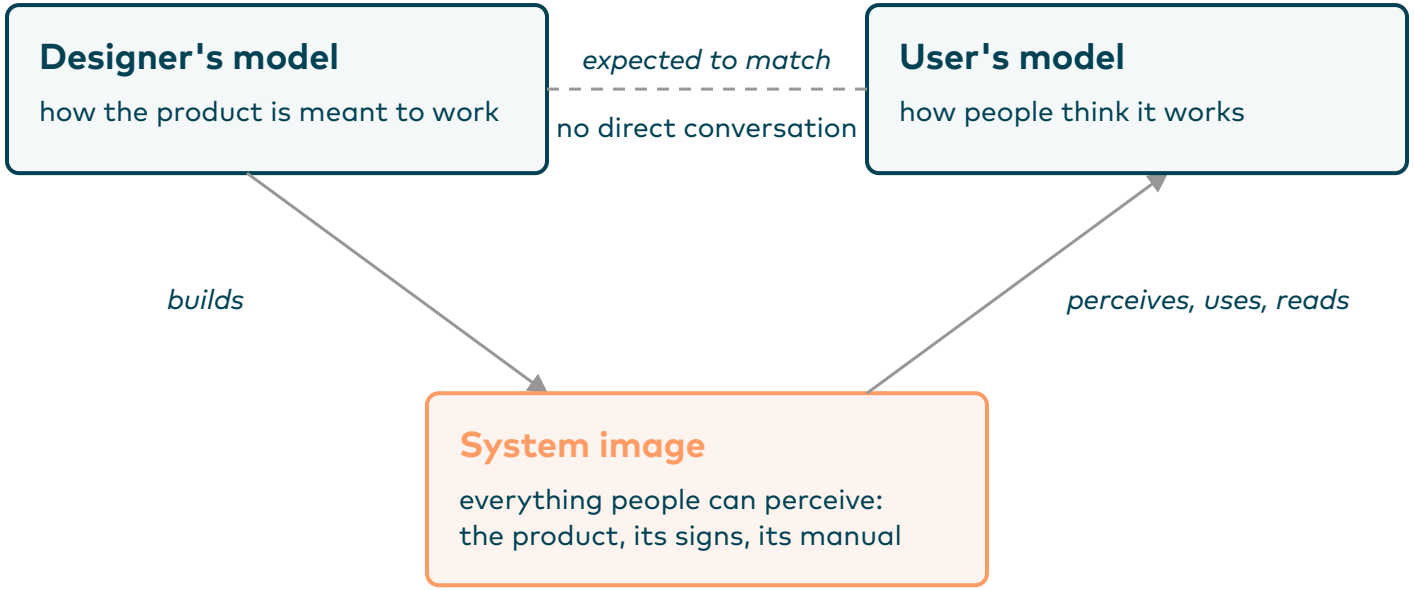
It's a **description of the behavior of the business**.

By programming, you GROW your understanding, adjust your mental model and both *prescribe* and *describe* behaviour, assumptions and expectations!

Shaping code is a *creative activity*.

Agents can greatly **enhance** your process - or *replace it*.

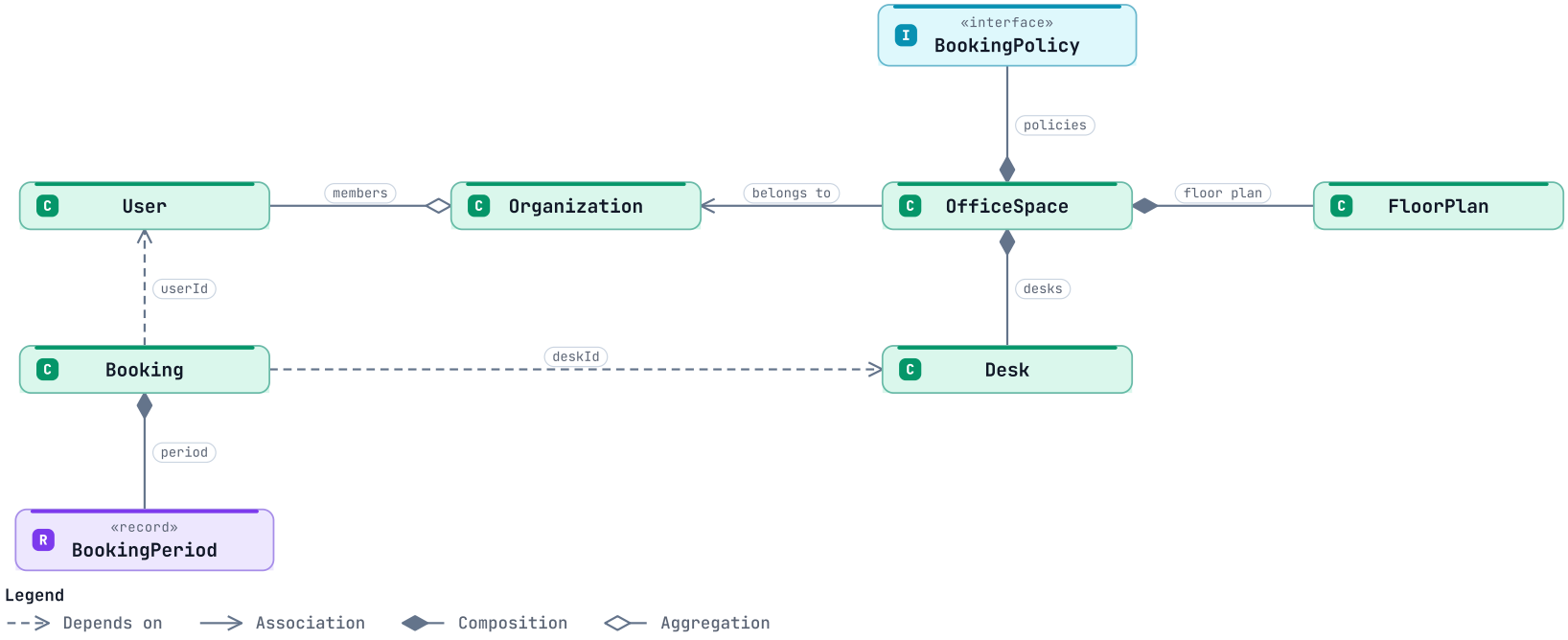
The system image



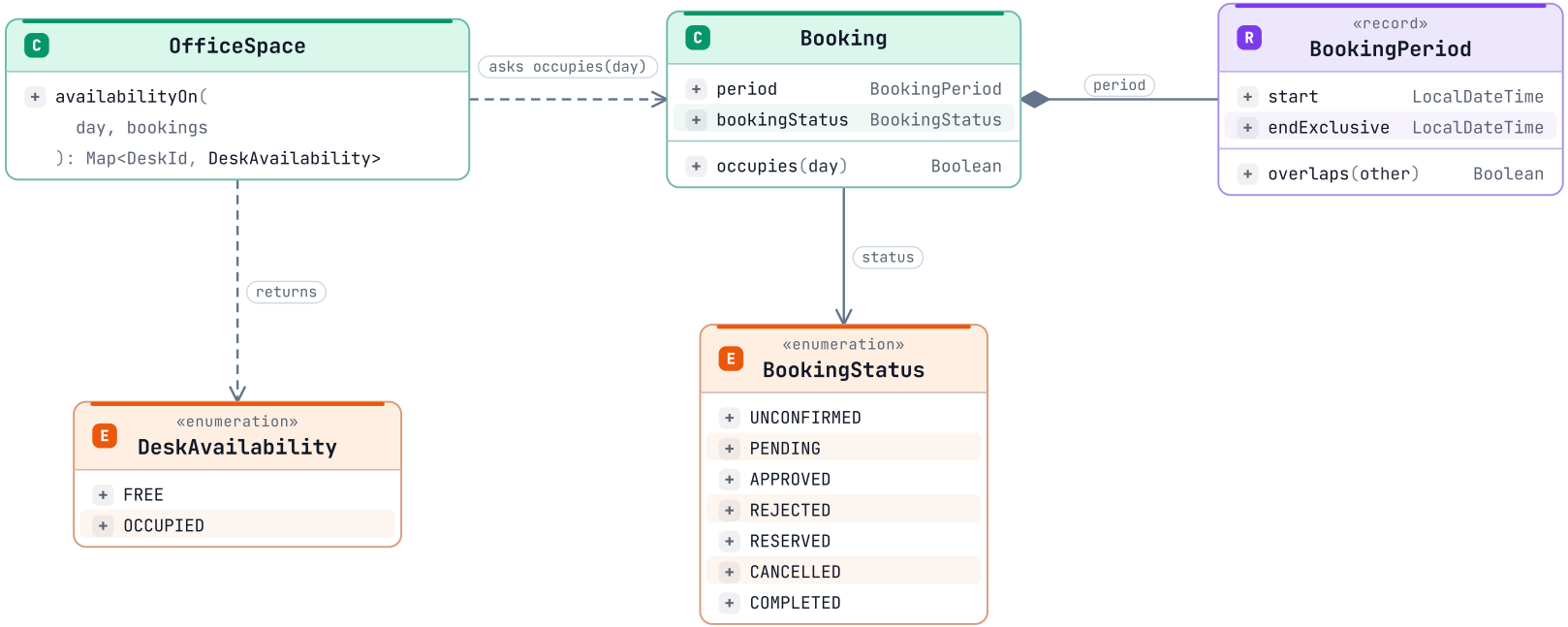
"The entire burden of communication is on the system image."

Don Norman, The Design of Everyday Things, 2013

The domain model of the office booking app



The booking rule



Legend

--> Depends on —> Association ◆— Composition

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

Great for exploration!

Not a replacement for humans.

Agents as allies in domain exploration

Agents can help:

- **Challenge** your mental models
- **Translate** between English and code, in both directions
- **Make explicit** what was implicit
- **Clarify** edge cases you hadn't considered

But they are **not a replacement for talking to domain experts**.

Everything we build has users

- APIs, MCP servers, CLIs and libraries are used by other developers and by agents.
- Their system image is the names, the error messages, the documentation and the behaviour at the edges.
- The domain model shows through there too, and it makes the use easy or hard.

Re-enforce constraints during product development

"System design is contextual design - it is inherently about boundaries."

- Ruth Malan

What do we mean by *harness*?



Armour



Leather harness

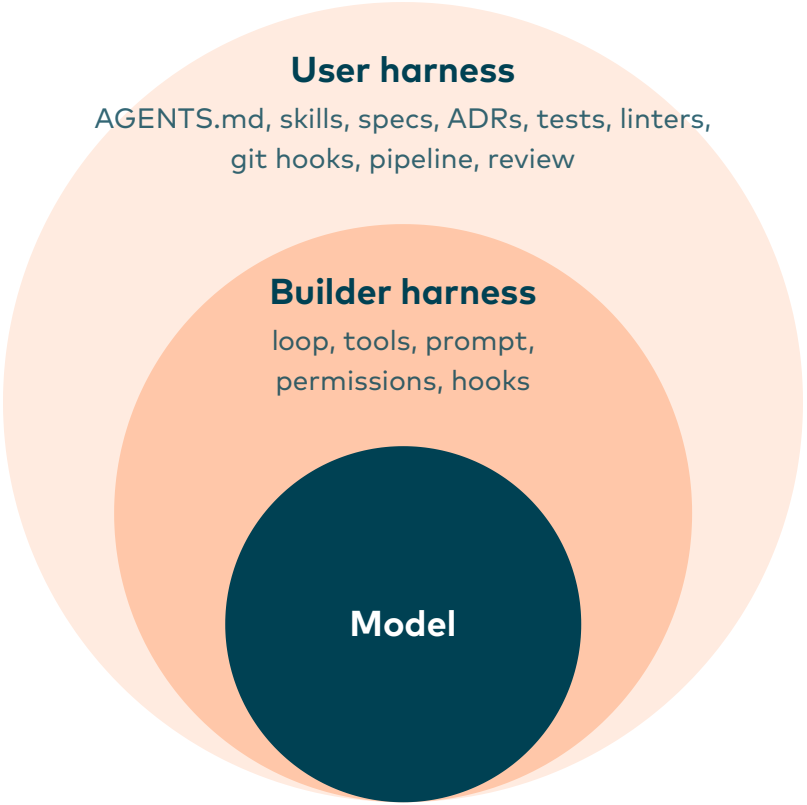


Climbing harness



Exoskeleton

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

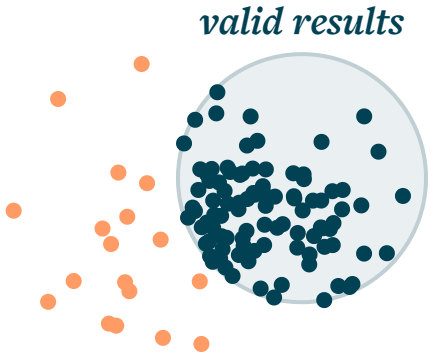


What Is the *Harness*?

- **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

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

Harness dimensions

- **Who decides:** the model (guideline) or a mechanism (guardrail)
- **When:** block the agent's action or correct after the fact
- **Determinism:** code decides or a model judges
- **Scope:** the whole repository, or one feature
- **Trigger:** every turn, every edit, every commit, every pull request, a schedule or external event
- **On a finding:** block, fix, or report
- **Loading:** always in the context, or on demand

Prefer *determinism*

- **Computational checks** come first:
  linting, tests and types are cheap, fast and have structured, often binary results.
- **Inferential checks** can express fuzzier criteria - and have fuzzier results.

Böckeler, Harness Engineering, 2026
HumanLayer, Skill Issue, 2026
INNOQ, Vom Vibe Coder zum Agentic Engineer, 2026

The harness holds *people*, too!

Using the harness for exploration, planning and conceptualizing is a *great* help.

- The harness can surface conflicts, warn against violations and support resolution.
- Importantly, it does this for *both* bots *and* humans!

You keep **ownership**: design the harness well to support your product and the people in it!

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.

Domain logic

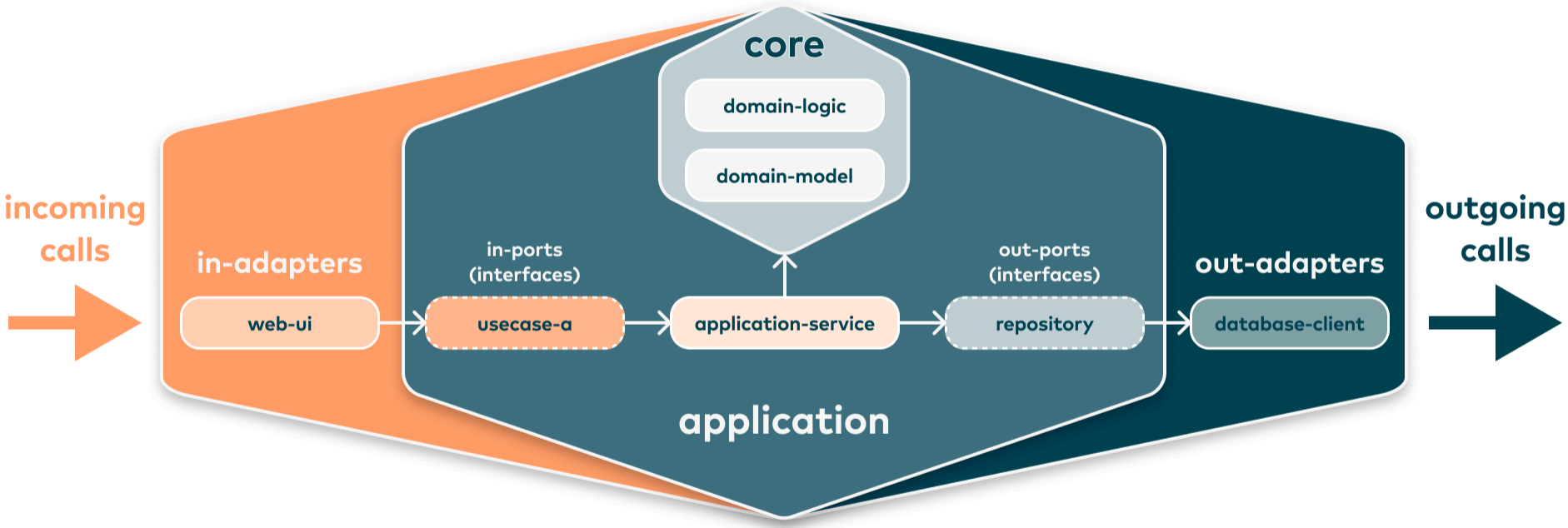
- *Critical* to get right
- Needs careful review
- Expresses what the business *does*
- … engaging to think about (I think)

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



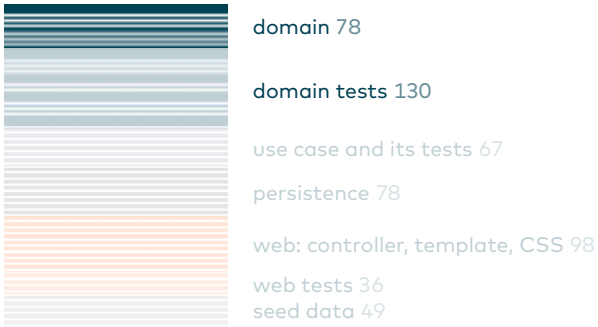
- dependencies only point inwards
- adapters and application are decoupled via interfaces
- core depends on NOTHING - it does its own thing!

"Once we have stable abstractions,

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

- Martin Fowler

Structure pays off



Story 0007, code and tests 536 lines

- The rules of the floor plan are 78 lines in one package, checked by 130 lines of tests.
- In a well-structured app, the product can follow the inner model, and both get easier to maintain.

Recipe for adding features

A solid structure enables quasi-mechanical implementation:

1. Write a **domain test** for the new behavior
2. Implement the **domain logic** to make it pass
3. Write a **use case test** – this tells you what inputs you'll need
4. Add necessary **adapter interfaces** for data fetching
5. Implement the **adapter** (database, API, whatever)
6. Write the **controller** to expose it

(... great candidate for a Claude Code *skill*, an *AGENTS.md* entry, a pre-commit *hook*...)

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

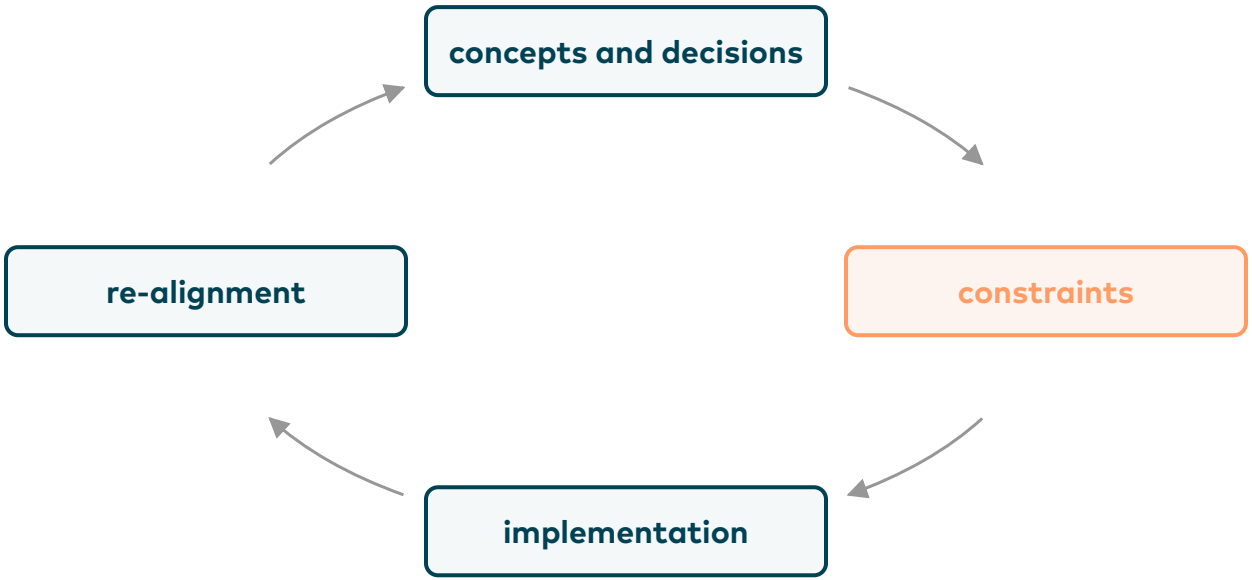
Still true with agents

- **Agent code needs more fixing later**
  After the merge, agent code needed 46 % more corrective maintenance than human code. Months with more agent code were followed by more fixes. Xia and Miller, Jul 2026
- **Complexity keeps rising without agent instructions**
  In repositories without checked-in agent configuration, cognitive complexity rose 53 % and was still rising when the observation ended. RAMP, Aug 2026
- **Agents do worse on code agents wrote**
  Building on agent-written code lowered the agents' solve rate by up to 13.1 % compared to human-written code. Patel et al., Jun 2026

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.

Re-enforce constraints during product development



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.

Our work in progress

82 %

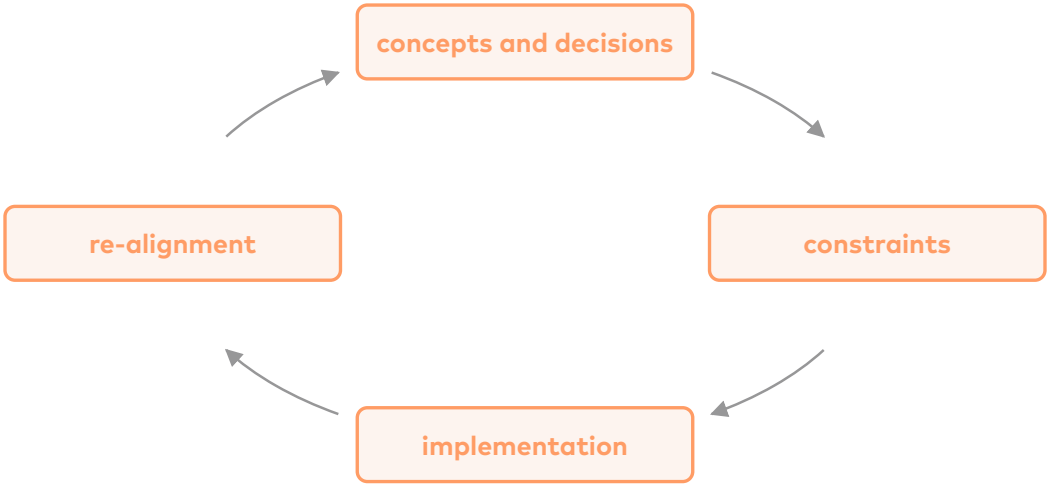
of engineers report spending less time writing code

That work 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.

Vella and Blincoe, The Impact of AI Coding Assistants on Software Engineering, May 2026

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.

No constrain, no gain

- It still takes discipline.
- Constraints give us orientation and can catch mistakes before they spread.
- Agents let us move faster. We can use that time to raise the quality along the way.

"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*



Tonight in Berlin

- Aminata Sidibe gives a short version of her talk on outcome-driven architecture, and I give one of this talk, both going a bit deeper than today.
- Doors open at 18:30, the talks start at 19:00, and afterwards there are snacks, drinks and time to talk.
- It's open to everyone, also to people who weren't at Technology Day, so feel free to pass it on.
- Please register on Meetup, so we know how many people to plan for.



Aminata Sidibe
Principal Consultant,
INNOQ

INNOQ office
Ohlauer Str. 43
10999 Berlin

Thank you! Questions?



Johannes Rave Senior Consultant - INNOQ

johannes.rave@innoq.com