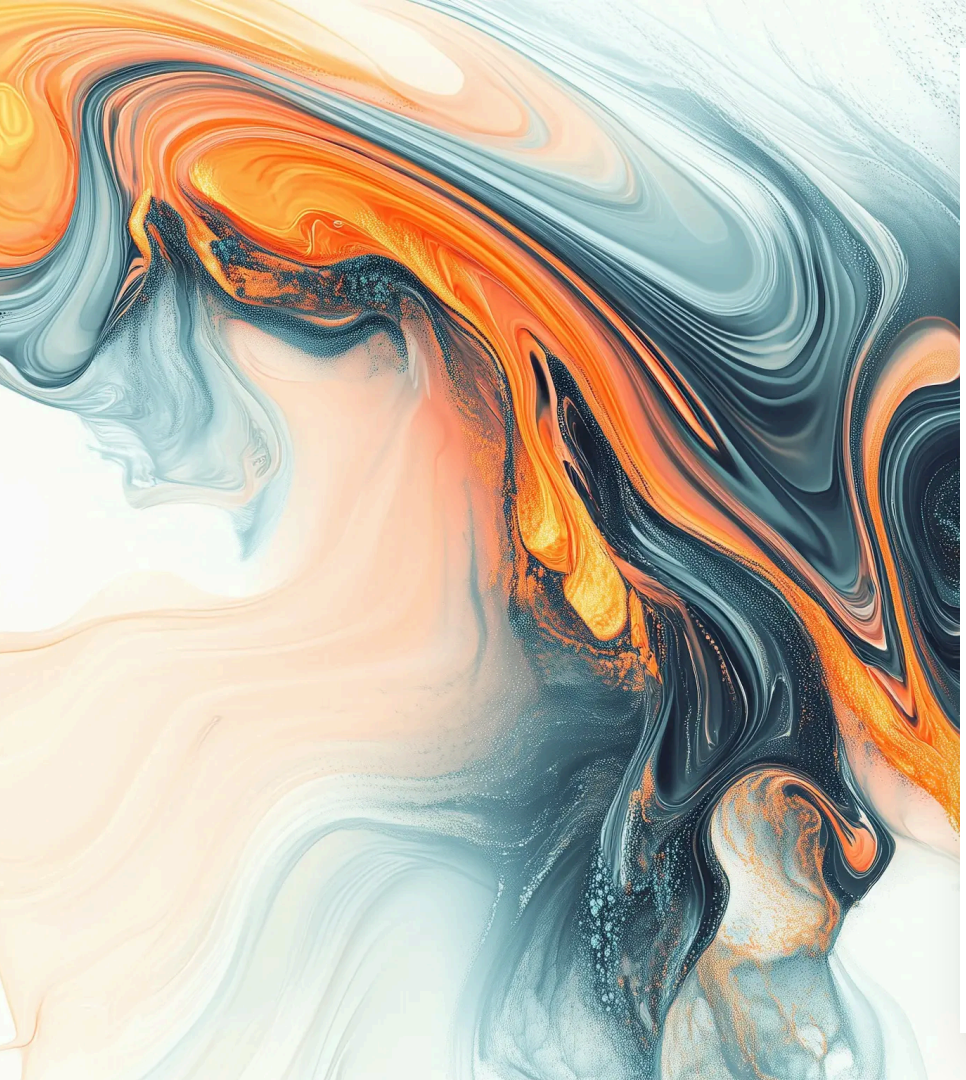


TECHNOLOGY DAY · LIGHTNING TALK · 2026

From Slop to Harness

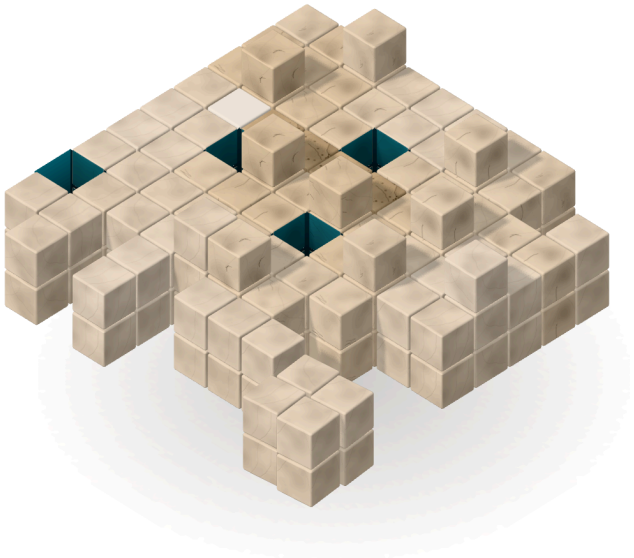
5 Steps to make a codebase agent-ready

INNOQ



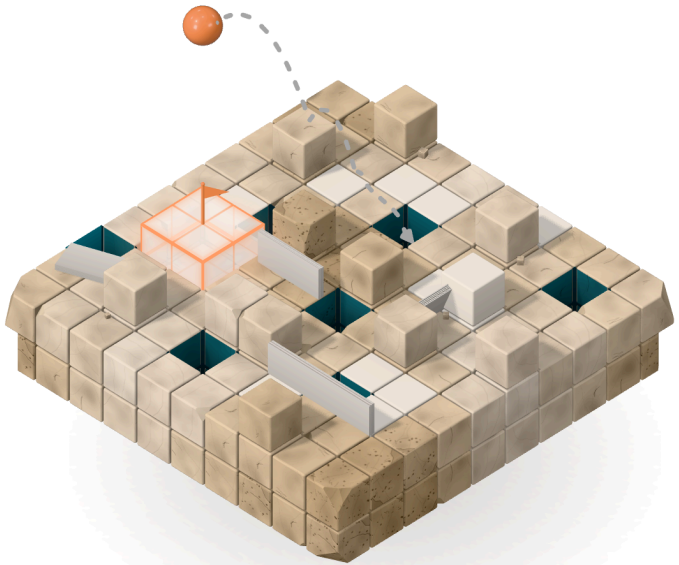
Roman Stranghöner
Senior Consultant

A codebase grows *stone by stone*



- 🪨 Bootstrap, setup, README, conventions
- 🏗️ Feature after feature
- 🕳️ No time for tests: the first pits
- 🏝️ A side project docks on, never integrated
- ⏳ The foundation ages, one stone gets replaced

Your codebase is a *tilt maze*



The board

your codebase: fresh, old and ancient stones

The grey sheets

its docs: they help a bit, but they are incomplete and scattered

A pit

a weak spot: old dependencies, missing tests, no formatter

The build site

what the spec asks for

The ball

the agent: let go, and it rolls

So you tilt. Every single move.

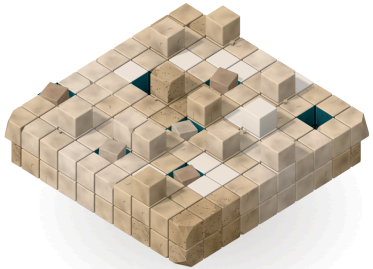
When output gets *cheap*, the *expensive* things change

Code output
gets cheaper & faster

but

- Defects increase
- Bottlenecks increase
- Maintenance increases
- Accountability increases
- Knowledge loss increases

Three flavours of *brownfield*



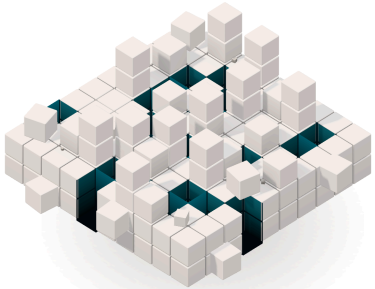
Human only

built by hand, knowledge lives in heads



Human + AI

new stones that don't fit the old grid



AI only

shiny, busy, and nothing fits together

Brownfield is *harder* for agents

PRODUCTIVITY GAIN WITH AI

	simple	complex
Greenfield	35–40 %	10–15 %
Brownfield	15–20 %	0–10 %

Stanford study, 136 teams (Y. Denisov-Blanch)

🕳️ **It falls into pits**
weak spots nobody wrote down

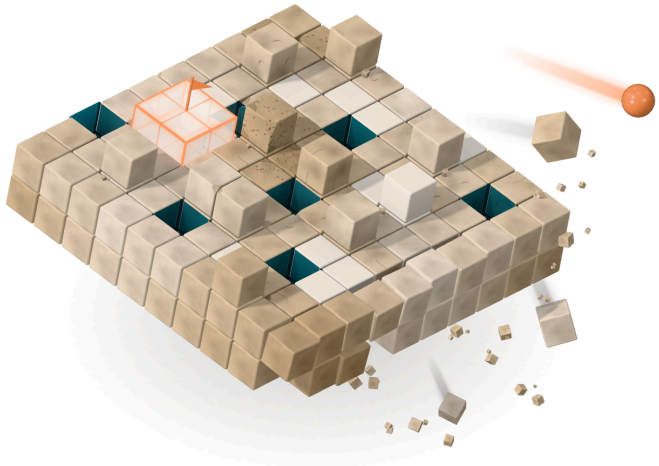
🧭 **It gets lost**
lots of context: some of it outdated, the rest in people's heads

🏗️ **It builds the wrong thing**
against the architecture, against the intent

And many of us work in some form of brownfield.

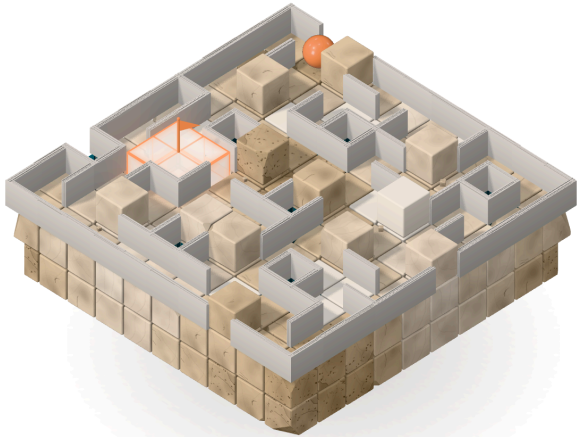
https://www.innoq.com/de/blog/2026/03/ueber-ai-einsatz-in-verschiedenen-coding-situationen/

Two ways to *lose the ball*



🎲 Shake and hope

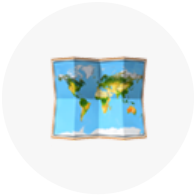
🏄 Vibe coding: no walls, no checks, sometimes lucky



📚 The over-built maze

📋 Spec inflation: paper walls first, the ball much later

Five *steps*



1

Understand

Where are the pits?



2

Condense

What does the agent really need?



3

Anchor

What can a tool check?



4

Let go

Where does it still fall?



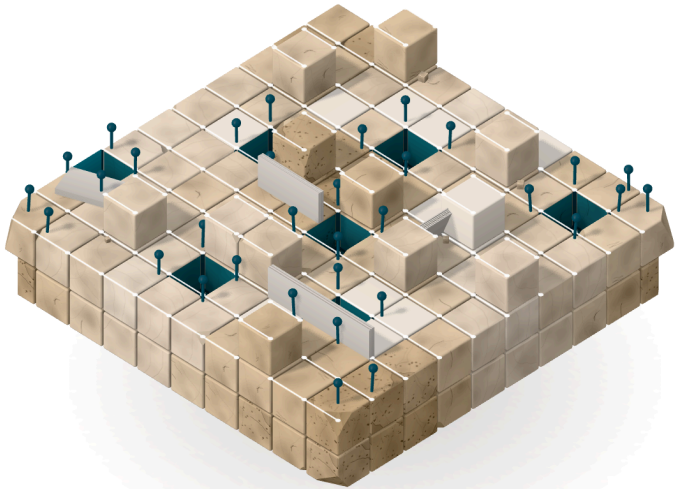
5

Learn

Which miss keeps coming back?

↻ again and again

1 · Understand: *map the maze*



Where are the pits?

Facts by tools, meaning by model.

Tools collect layers, dependencies and cycles. The model reads the result and explains what it means.

Three ways to *collect the facts*

Write a script

A few dozen lines for your questions: module graph, cycles, entities. Same answers every run.

Use what exists

Every ecosystem has tools for analysis: `dependency-cruiser`, `madge`, `jdeps`, `ArchUnit`. Sometimes it is already in the repo.

Explore in a notebook

Markus Harrer's `jupyter-notebook` skill: one step per cell, every step visible, rerun anytime.

Different tools, same rule: facts first, meaning on top.

https://github.com/feststelltaste/agentic/blob/main/skills/jupyter-notebook/SKILL.md

2 · Condense: from prose *to checks*



Would the agent build it differently without this line?

If not: delete it.

Go through specs, AGENTS.md and docs line by line. A rule a tool can check becomes a test or a lint rule. What no tool knows stays, kept short.

Every line goes *into one bucket*

Take the docs line by line. Each line lands in exactly one place:

🧪
Acceptance test
checkable behavior

📐
Lint or architecture rule
"`lib/` never imports stores"

🪄
Formatter / `lint --fix`
style that fixes itself

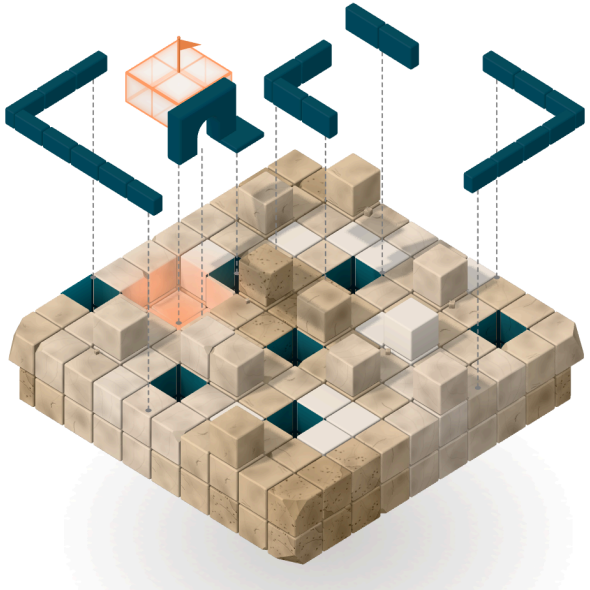
🧭
AGENTS.md, skill, ADR
context no tool knows

🗑️
Delete
already in the code

🗑️
Delete
pre-empts the implementation

a tool checks it · the agent reads it, kept short · gone

3 · Anchor: *build the harness*

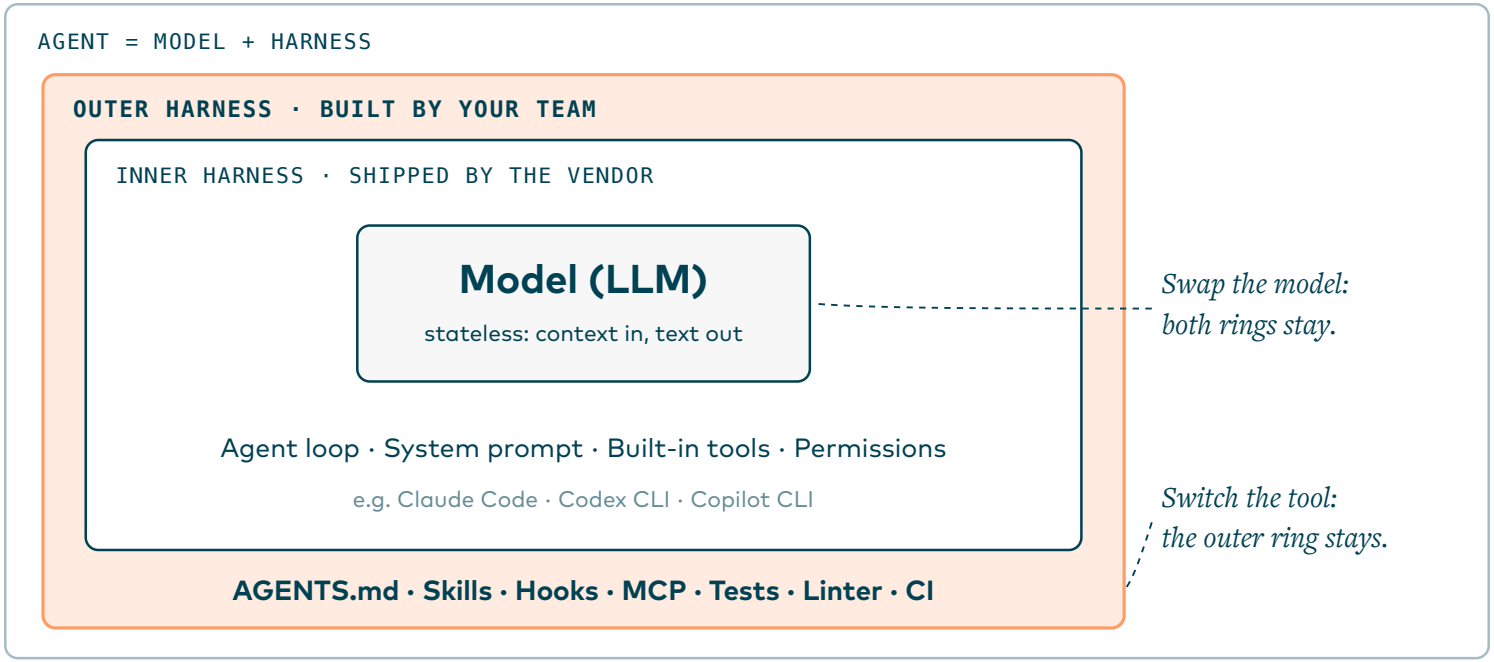


What can a tool check?

If you can tool it, tool it.
If you can't, prompt it.

Rails and walls steer before: AGENTS.md, skills. Plates and the gate check after: tests, linters, hooks, CI.

Agent = Model + *Harness*



Steer before, *check after*

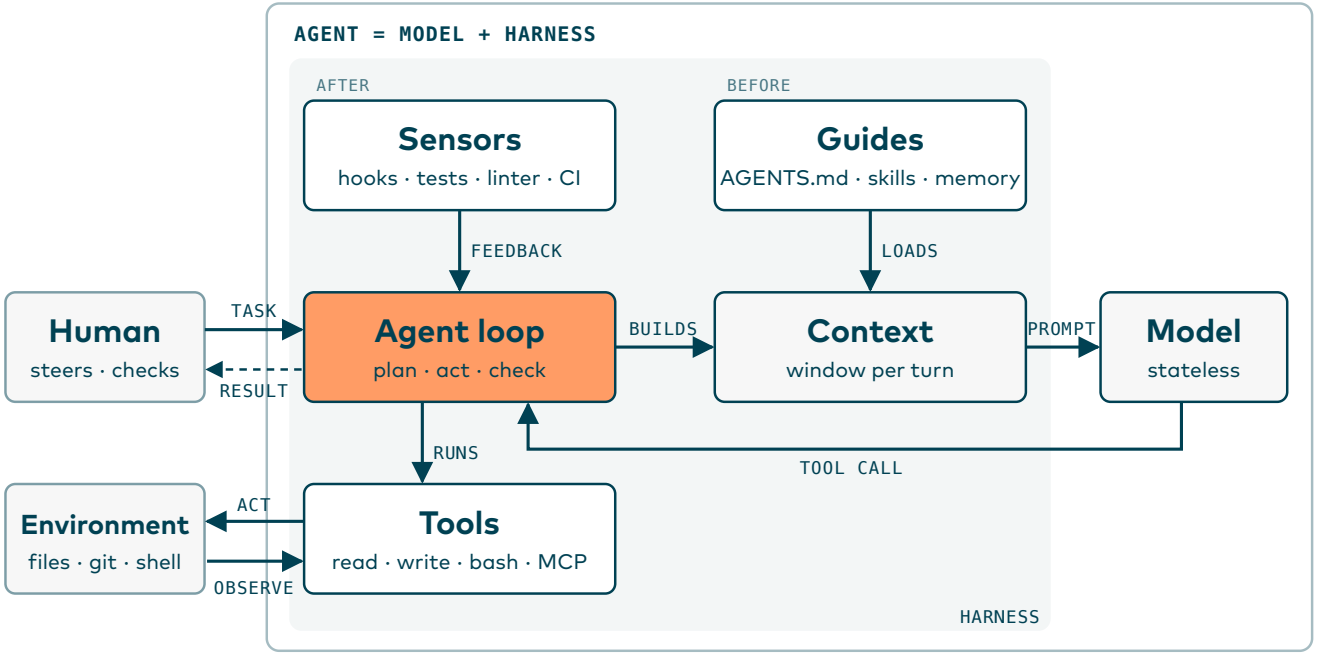
	🧭 **Guides** steer before	📡 **Sensors** check after
⚙️ **Computational**	formatter, codemods, scripts, templates	typecheck, linter, tests, architecture tests
🧠 **Inferential**	AGENTS.md, skills, architecture docs	review agent, LLM as judge

Only guides: the agent knows the rules but never learns that it broke them.

Only sensors: the agent keeps making the same mistakes.

https://martinfowler.com/articles/harness-engineering.html

The harness *at work*



99 problems, but a *rule* ain't one.

📄 AGENTS.md

```
- Timer engine is pure TypeScript:
  no React, no JSX, no hooks in lib/
```

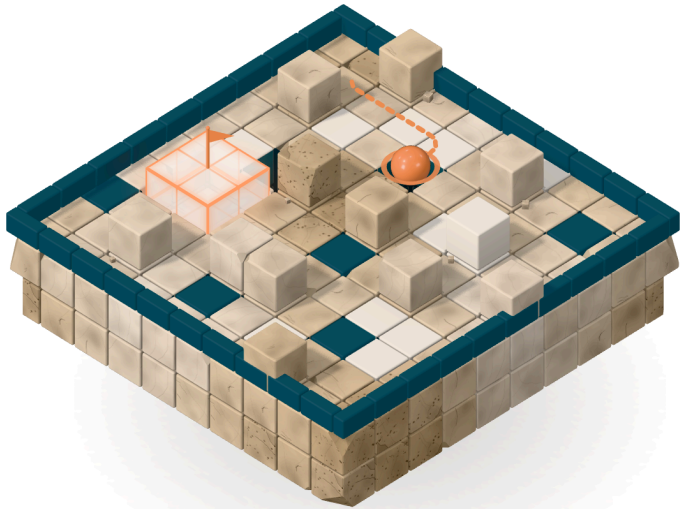
An agent may read this. Or not.

📐 ESLint

```
{
  files: ["src/lib/**"],
  rules: {
    "no-restricted-imports": ["error", {
      patterns: ["react", "@/stores/*"],
    }],
  },
}
```

If the agent breaks it, the run fails.

4 • Let go: *watch where it falls*

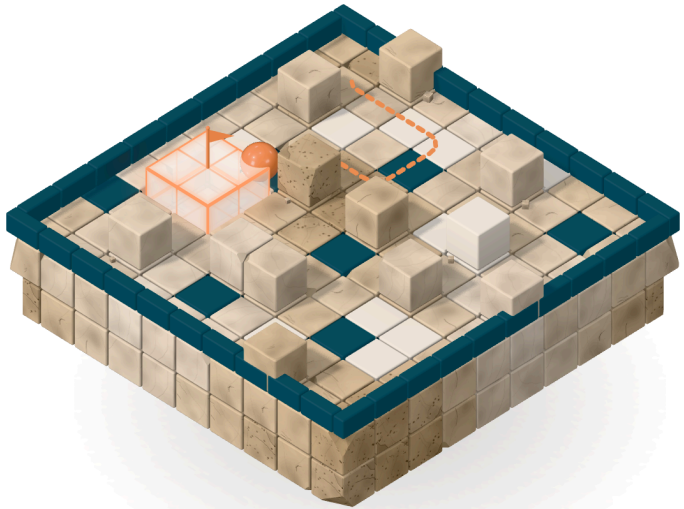


Where does it still fail?

Hands off, eyes on.

Let the agent run on its own. It will still fall somewhere: into a pit the map did not show. Mark and remember the spot.

5 • Learn: *close the loop*



Which miss keeps coming back?

Every repeated miss becomes a check.

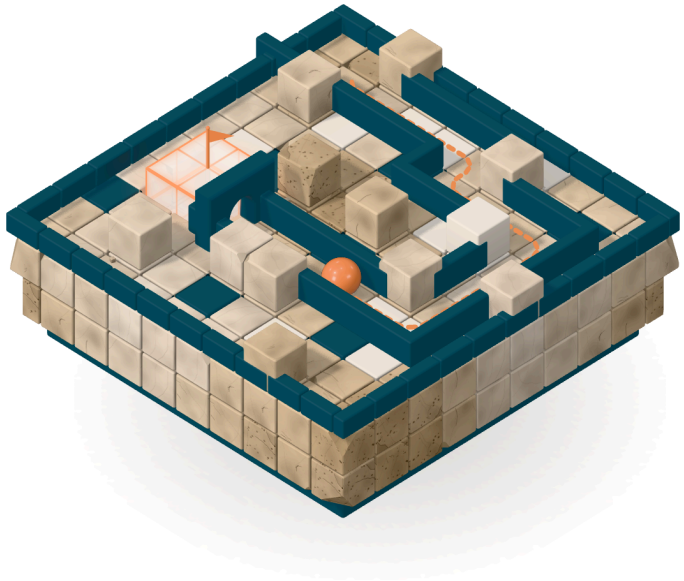
▶ Run

📍 Note where it fell

🔁 Again? Turn it into a plate, a wall, a rule in the gate

↻ **Run again**

From *in* the loop to *on* the loop



Same board, many loops later:

Edge rails
AGENTS.md, kept short

Guide walls
skills that shape the path

Patch plates
tests and lint rules over the pits

The gate
one check before it lands

The ball rolls on its own. You judge where it lands.

Less specs, more checks.

Every rule a tool can check belongs in a *gate*, and *out* of the docs.

You push it, you own it.

Deterministic where *possible*.
Probabilistic where *necessary*.
Human where it gets *risky*.

Want more? *Three days*, hands-on.

INNOQ TRAINING

Agentic Software Engineering

- 🤖 Foundations of AI agents
- 📋 Requirements and product
- 🏛️ Architecture and decisions
- ⌨️ Implementation with agents
- 🧪 Testing and quality assurance
- 🚀 CI/CD and automation
- 📈 Operations and monitoring

For developers, tech leads, architects and platform engineers. In-house for your team.

Scan for details

https://www.innoq.com/de/topics/agentic-software-engineering-training/

Thank you! *Questions?*

INNOQ
www.innoq.com



Roman Stranghöner
Senior Consultant

LinkedIn

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

c/o betahaus Hamburg
Eifflerstraße 43
22769 Hamburg

Spichernstraße 44
50672 Köln