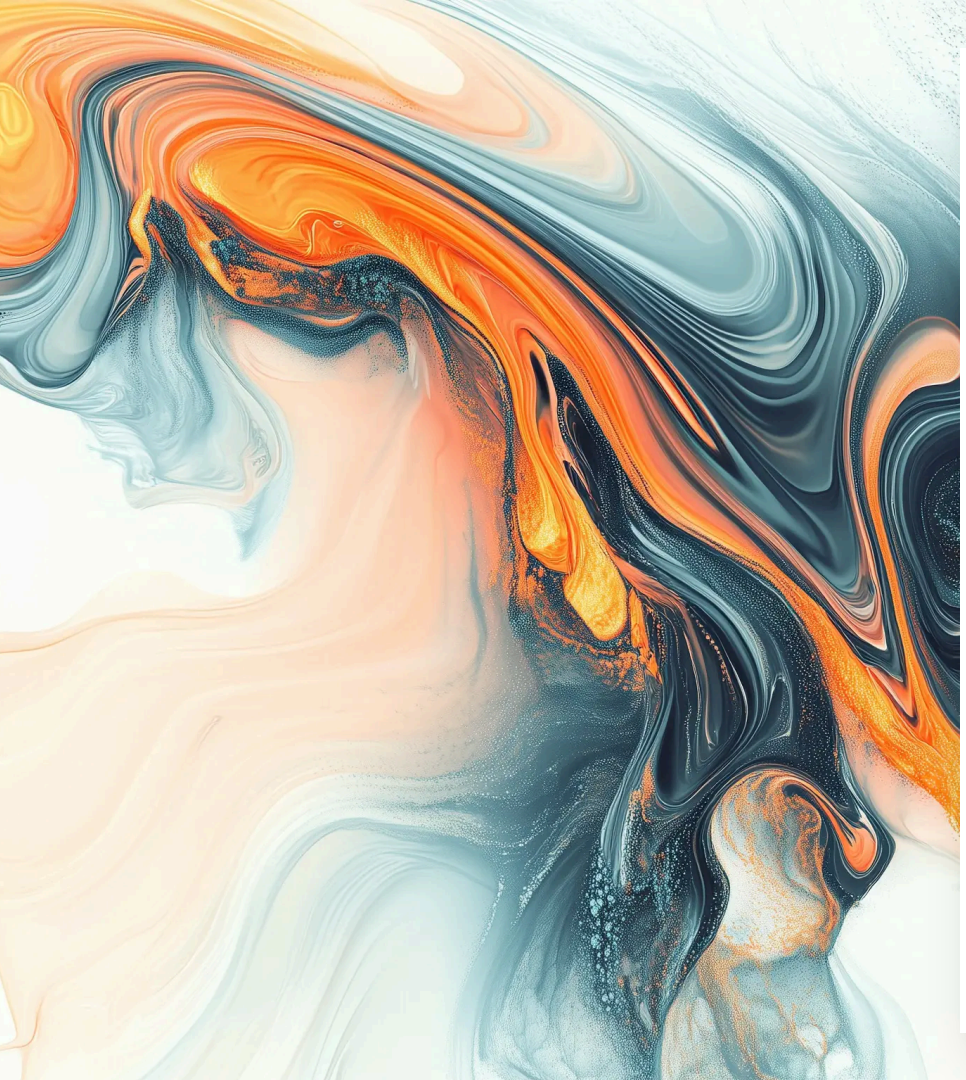


TALK

Agenten für die Business-Domain

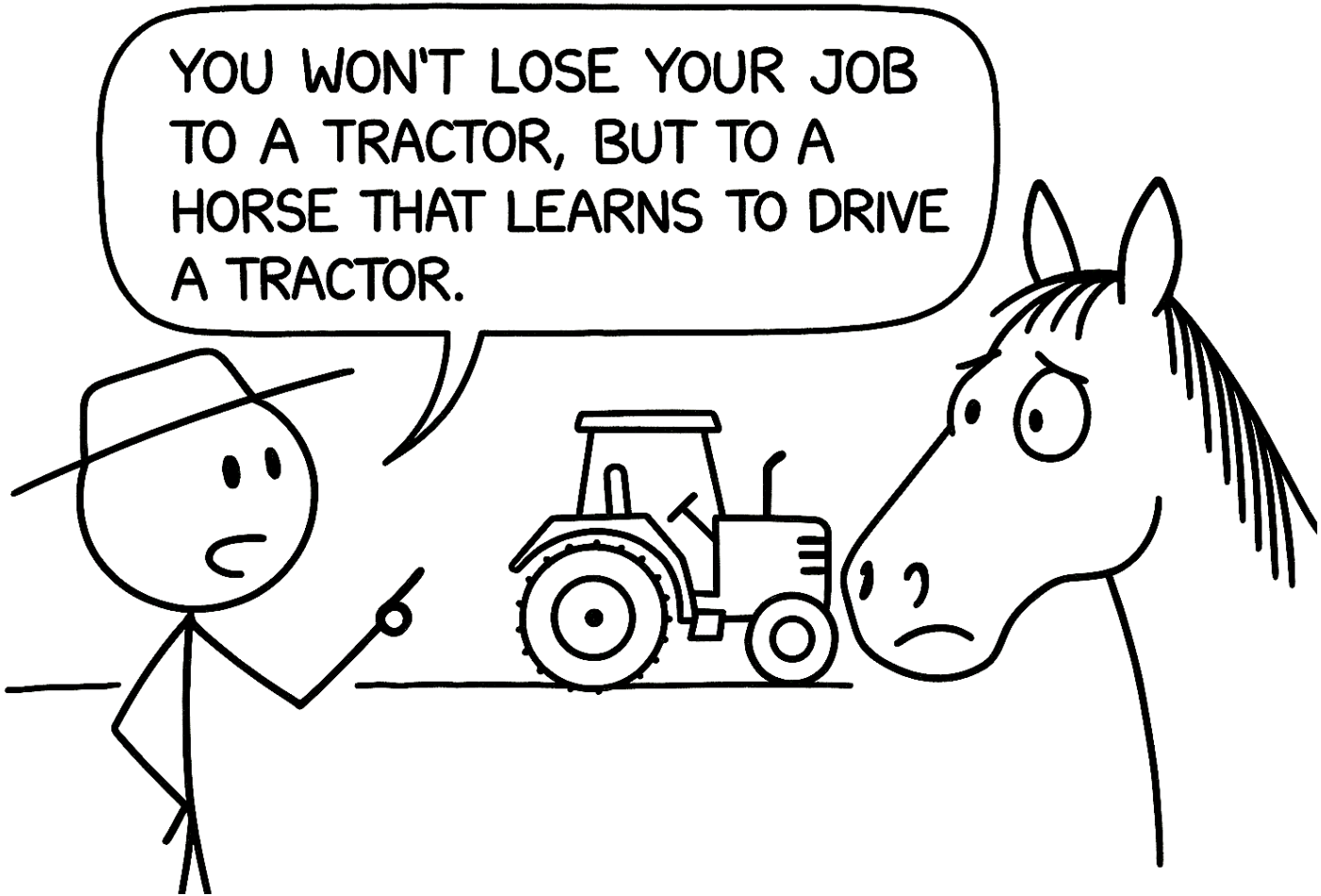
Don’t reinvent the technical wheel with every prompt

INNOQ

👋

Hi

FABIAN WALTHER
SENIOR CONSULTANT


YOU WON'T LOSE YOUR JOB TO A TRACTOR, BUT TO A HORSE THAT LEARNS TO DRIVE A TRACTOR.

Agenda

- **Context** – building the agent's long-term memory
- **Determinism** – why we need a harness
- **The Agent Harness** – the deterministic scaffold
- **Hands-on** – customizing pi: sandbox & subagents
- **Arbeitsteilung** – what really changes


Chapter 1
Building Context

Software starts long before code

- We talk to users and stakeholders
- We capture requirements
- We describe who we build for: **personas**
- We slice work into **user stories**, **journeys**, **epics**
- We make and document **decisions**

None of this is code – but all of it is *knowledge* that we use to build *models* that lead to *code*

Know who you build for

Proto Personas

What is a Proto Persona?

Origin: Jeff Gothelf and Josh Seiden (Lean UX), based on Alan Cooper's Persona concept

Purpose: Hypothetical user profiles based on assumptions and existing data. Faster than full personas, but still helpful for product decisions.

Recommended Input: User research, analytics, customer interviews, demographic data, behavioral patterns

Example |

Proto Persona

Sarah Müller - Hobby Content Creator
32 years | Munich | EUR55,000/year | Marketing Manager

Problem: Needs her own laptop for private video editing projects, current one is too weak for 4K footage and rendering

Needs: Strong performance for video editing, good display for color grading, home use only.

Buying Behavior: Researches creator hardware, reads benchmarks, compares rendering times, checks software compatibility.

Decision Factors:

1. Video editing performance (CPU/GPU, fast rendering)
2. Display quality (color accurate, large)
3. RAM + storage (16GB+, large SSD)
4. Value for money

Budget: €1.200-2.200 | **Purchase Timeline:** 2-3 weeks

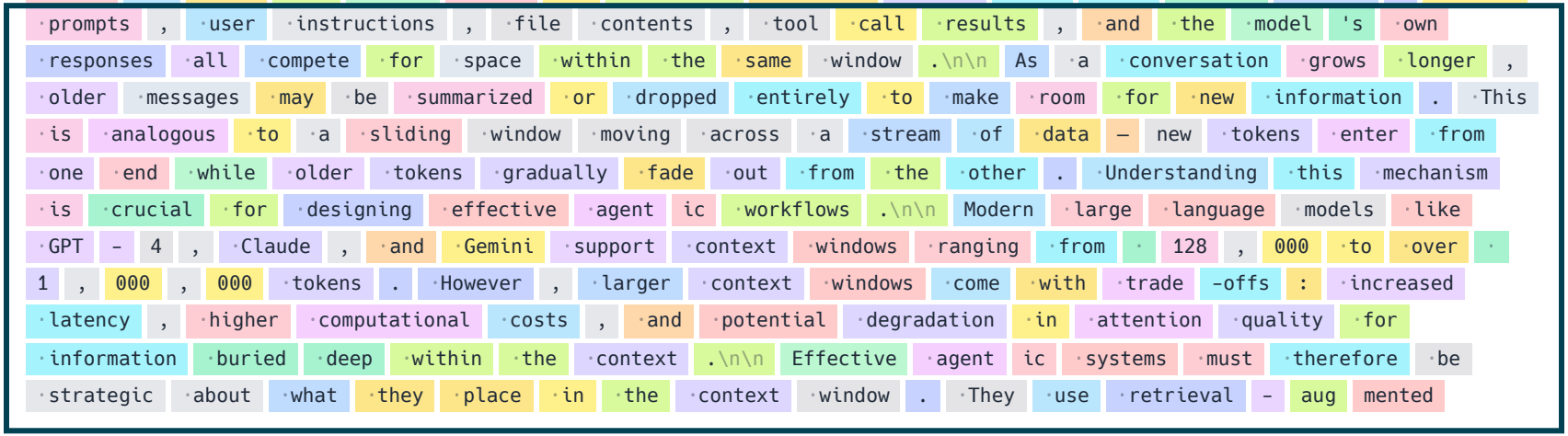
All this knowledge has to reach the model *somehow*.

Context is the only knowledge

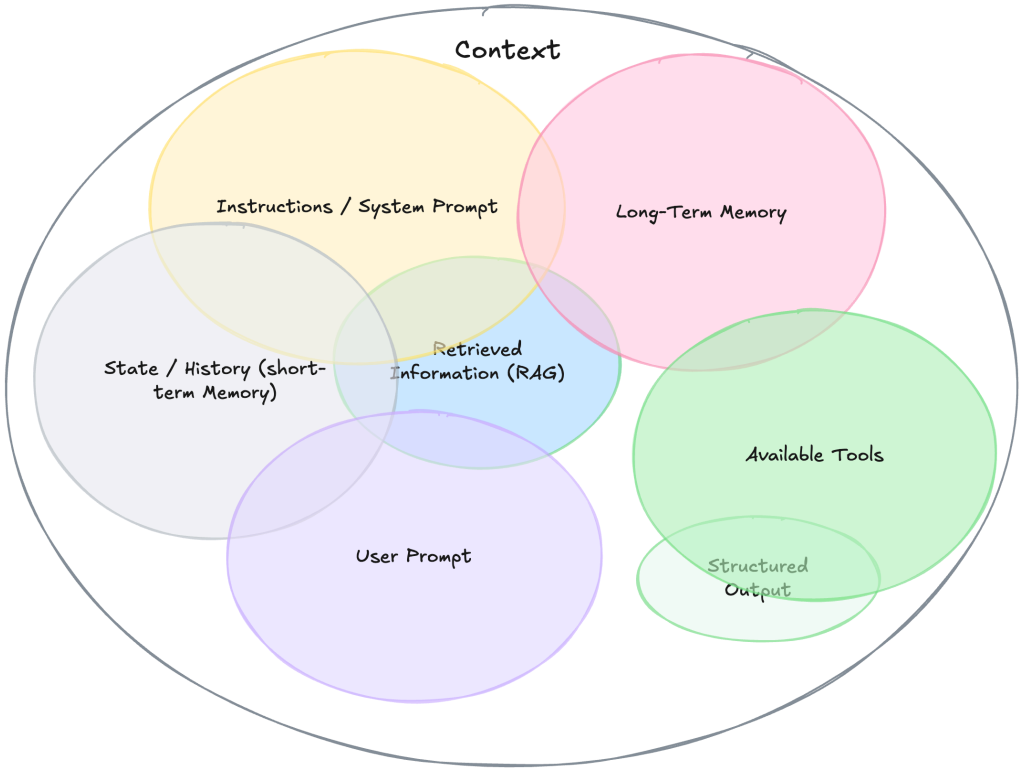
- On every new task, we start with a new context
- Everything from a previous conversation is lost
- There's no knowledge transfer between conversations
- "Learning" happens only during training, not while we work
- All information required for a task needs to be in the context

Context is king.

The Context Window



What's really *relevant*?



Short-Term Memory

vs.

Long-Term Memory

- Everything in the current session
- Temporary and limited (context window)
- Lost when the session ends
- The model generates from exactly this

- Persistent across many sessions
- Must be actively saved & retrieved
- Stored externally, e.g.
  - Markdown documents
  - Databases
  - Knowledge graphs

Long-Term Memory in practice

- `AGENTS.md` / `CLAUDE.md` – automatically attached on a new session
- **Rules** – focused memory loaded via glob patterns
- Persisted artifacts the agent can read on demand

This is where personas, requirements & decisions *live*

Personas, requirements, decisions become the agent's
long-term memory.

Chapter 2

The Determinism Problem

An LLM predicts the *next token* by probability.

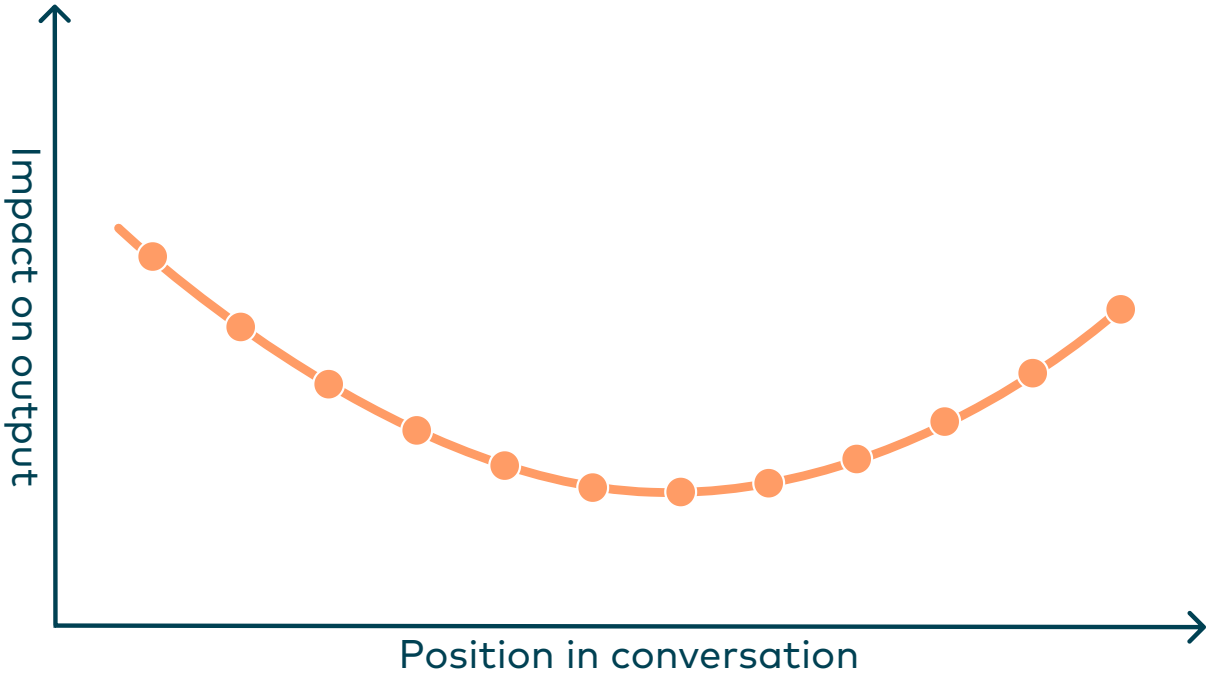
Same input can yield a *different output*.

Dealing with context is *hard*.

Context can fail in many ways

- **Overflow** – the window is exceeded, info silently drops
- **Distraction** – noise pulls the model off task
- **Poisoning** – manipulated or harmful input leaks in
- **Confusion** – vague inputs get averaged together
- **Clash** – contradictory statements, picked at random
- **Rot** – quality degrades as context grows

„Lost in the middle"



So we wrap the model in *determinism*

Non-deterministic

- The model's reasoning
- The exact wording
- Which path it takes

Deterministic (we control)

- Tools & their boundaries
- Tests & feedback loops
- Hooks, linters, type checks
- Permissions & sandboxing

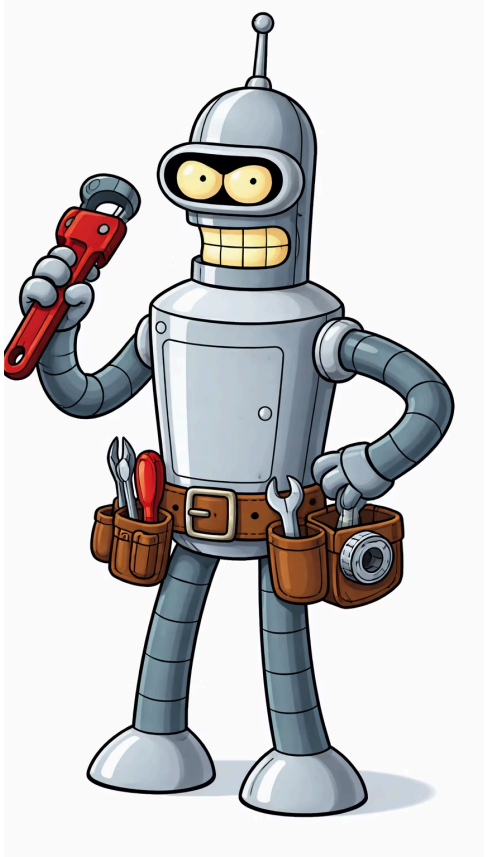
To use a non-deterministic model *efficiently*,
we need a deterministic *harness*.

Chapter 3

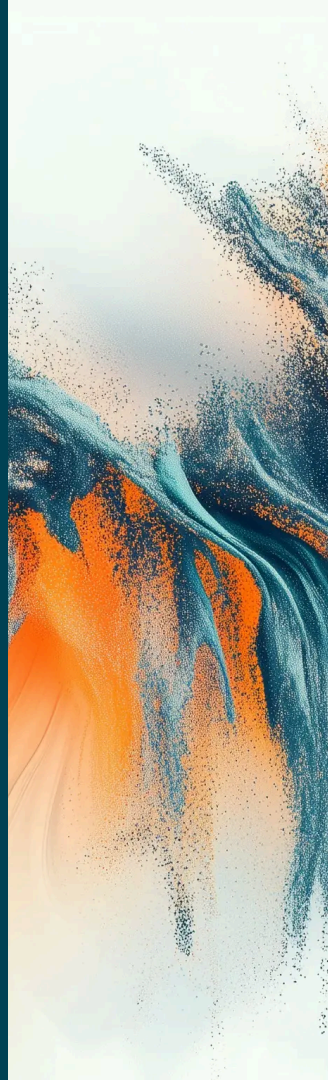
The Agent Harness

What is an agent?

- Uses an interchangeable LLM as a brain
- Has memory loss after a couple of beers
- Can change the environment
- Has access to external sources
- Has a set of tools to get the job done
- Got an attitude



What the agent *operates on*

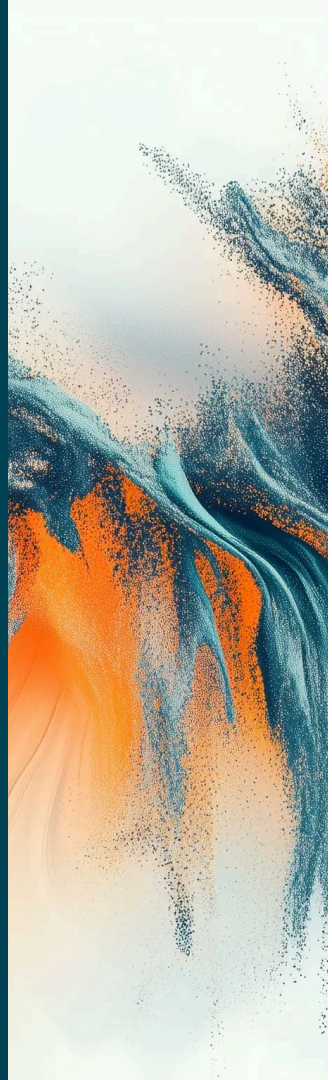


Environment

- Filesystem (code, configs, docs)
- Version Control (git)
- APIs and Services
- Databases
- Terminal / Shell
- IDE / LSP

The agent's "world"

How agents interact with the *environment*

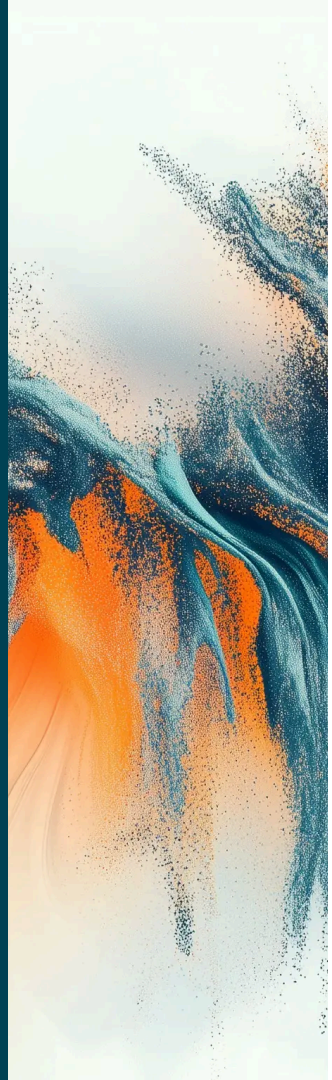


Tools

- Read (files, code, config)
- Write (create, edit, delete)
- Execute (commands, tests, builds)
- Search (grep, find, LSP)

Tools define the *capability boundary* of the agent

Closing the
loop



Observation & Feedback

What agents observe:

- Tool Results – output from actions
- Error Messages – compilation, runtime
- Test Results – pass/fail status
- Diagnostics – LSP warnings, lints
- State Changes – git diff, file changes

The Control Loop

⚡

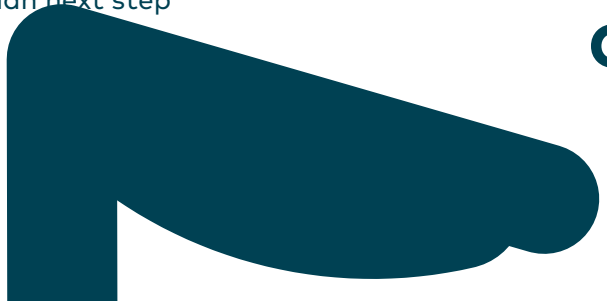
Act

Execute tool,
modify environment

🧠

Reason

Analyze situation,
plan next step



👁️

Observe

Read results,

Assistant

vs.

Agent

The harness is the *deterministic scaffold* around a non-deterministic model.

Think of agents as *interns*.
Always *verify* their work.

Chapter 4

Hands-on

pi — a minimal harness

- Open-source terminal coding agent (`pi.dev`)
- By Mario Zechner (`earendil-works/pi`)
- **Four core tools:** read, write, edit, bash
- A slim TUI around the control loop
- Provider-agnostic, bring-your-own-key

Everything else is an *extension*

- Skills, sub-agents, plan mode
- Permission gates, path protection
- Sandboxing, SSH execution
- MCP, custom editors, status bars

Composed in TypeScript, shared via *npm* / *git*

Reuse, don't *regenerate*

- Don't let the agent reinvent the technical wheel each prompt
- Plug in proven tools from the **JVM ecosystem** (build, tests, frameworks)
- The harness encodes them *once* – the agent focuses on the *business domain*
- Spare your future self from maintaining a pile of generated **SLOP**

We bend the harness to our workflow

Customize pi

But first: run it *safely*

- An agent runs `bash` – it can do real damage
- Solution: run pi inside a **Docker sandbox** (Docker SBX)
- Isolated filesystem, no access to host secrets
- Fail fast, throw the container away

Deep dive: Joy Heron on sandboxing AI agents -- INNOQ blog & Podcast #186

Our playground

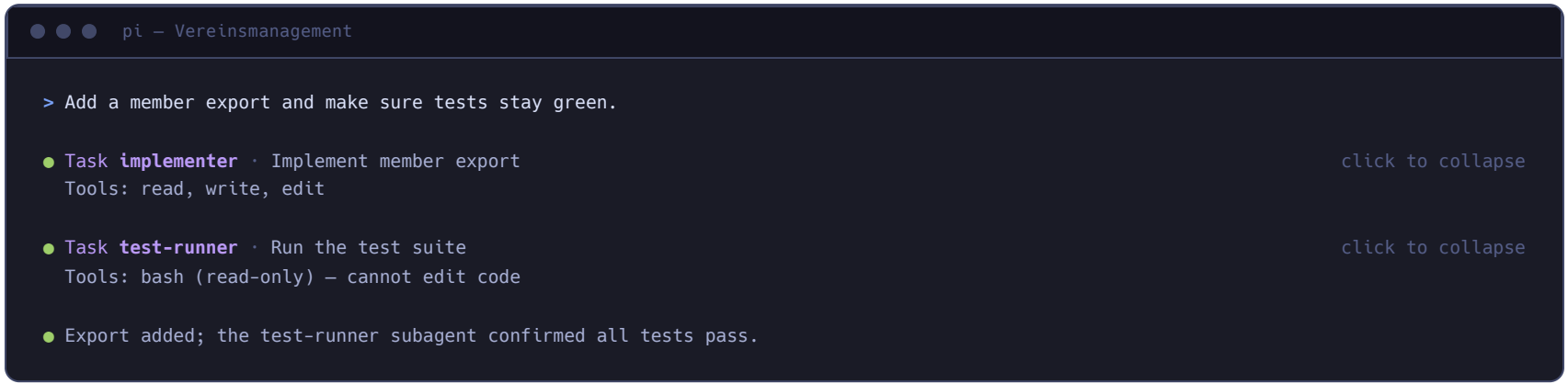
Vereinsmanagement

The example app

- A small **club management** application
- A plain **Java / JVM** project – Gradle, JUnit, Spring
- Members, contributions, events
- Small enough to grasp in minutes, real enough to matter

The agent reuses the existing JVM tooling – and works on the *domain*

Tool-restricted subagents



Demo

https://pi.dev

Live: pi + Sandbox + Subagents

Chapter 5

Arbeitsteilung

Generating code got *fast*.

Writing code was *never* the bottleneck.

What actually takes the time

- Understanding the problem
- Coordinating across people and teams
- Integrating into existing systems
- Reviewing and taking responsibility
- Keeping quality up over time

In other words: *Arbeitsteilung*

LLMs are a *neutral accelerator*.

They speed up the good *and* the bad.

So good practice matters *more*

- Clear requirements and decisions
- Tests and fast feedback loops
- Reviews and ownership
- Clean structure and documentation

Everything that was good engineering before is now *amplified*

Arbeitsteilung, part 1

Among Agents

Division of labor among agents

- Different **scopes** – narrow, well-defined tasks
- Different **tools** – restricted to what's needed
- Different **models** – cheap for routine, strong for hard work
- An orchestrator delegates and integrates

Curated context per agent beats one giant context

Arbeitsteilung, part 2

In the Team

The developer role shifts

- From writing every line …
- … to building the **harness** others work in
- Defining quality criteria and guardrails
- Enabling non-technical members to produce good code

Generation is democratized – *responsibility* is not

Someone still has to *own* what goes to production.

You push it, you own it.

A word to

Product Owners & Managers

Most time is lost *waiting*

- Don Reinertsen: value is lost in **queues**, not in the work itself
- Most time & money is spent *waiting* – for reviews, decisions, handoffs
- Speeding up coding optimizes a step that was rarely the bottleneck
- Local speed-up ≠ organizational efficiency

Fix the *flow*, not just the typing speed

No Silver Bullet

- Fred Brooks (1986): *essential* vs *accidental* complexity
- AI mostly attacks the **accidental** – boilerplate, syntax, glue
- The **essential** complexity – understanding the domain – remains
- There is no single tool that gives an order-of-magnitude gain

Agents make this distinction *clearer*, not obsolete

The Agentic Trio

- **Product Manager** – owns business outcomes & opportunities
- **UX Designer** – brings the user's perspective into discovery & delivery
- **Engineer** – guardian of sustainable delivery, builds & maintains the *harness*

A small trio now does discovery *and* delivery – after Teresa Torres' Product Trio

Chapter 5

Key Takeaways

1. Good context comes from good upstream work
2. Wrap the non-deterministic model in a deterministic harness
3. Reuse proven tools – don't regenerate SLOP
4. Code is cheap; flow & Arbeitsteilung are the real work
5. LLMs amplify your practices, good and bad
6. You push it, you own it

Vielen Dank! Fragen?!

FABIAN WALTHER
SENIOR CONSULTANT

INNOQ