Agentic Coding im Team

Erfahrungen aus dem Projektalltag

iteratec

David Schowalter · Senior Lead Software Architect · iteratec GmbH · 29. April 2026

Quick Show of Hands

- **Wer nutzt KI-Tools beim Coden?**
- **Wer nutzt agentic Tools?** (Claude Code, Cursor Agent, Codex…)
- **Wer hat Team-Konventionen dafür?** (Shared Instructions, Skills, Specs…)

10 Minuten.

Dann sinkt eure **eigenständige Problemlösefähigkeit** messbar.

Liu et al. 2026 · RCT · N = 1.222

77%

scheitern an Wartungsaufgaben — ohne KI.

Sankaranarayanan 2026 · N = 78 Studierende · Cursor + Claude

+20%

schneller — *schätzen* erfahrene Devs. Gemessen: **kein Speedup**.

METR 2025/26 · RCT · 57 OSS-Devs · 800+ Tasks

> *"KI macht Software Engineering nicht einfacher. Sie macht Engineering-Disziplin wichtiger denn je."*

ThoughtWorks Radar Vol. 34 (April 2026)

Die Produktivitäts-Illusion

Der Perception Gap

-4%

gemessene Produktivität (nicht signifikant)

METR 2025/26 · RCT · 57 erfahrene OSS-Devs · 800+ Tasks

+20% schneller

Selbsteinschätzung *derselben* Devs

Subjektiv: Speedup. Objektiv: keiner messbar.

≥24 Punkte

Wahrnehmungslücke (Perception Gap)

Differenz zwischen gefühltem (+20%) und gemessenem (-4%) Effekt

Gegenbefund: Cui/Demirer 2025

- 3 Enterprise-RCTs (Microsoft, Accenture, Fortune 100)
- **+26% Tasks** — stärkste positive Studie
- Aber: Effekt fast nur bei **Juniors**
- Erfahrene Devs: **kein messbarer Effekt**

Management Science 2025 · 7 Monate Laufzeit

Bestätigt: Chen et al. 2026

- **N = 2.989** Entwickler bei BNY Mellon
- Zufriedenheit korreliert **nicht** mit Zeitersparnis
- Langzeit-Faktoren (Expertise, Ownership) werden ignoriert

ICSE-SEIP 2026 · arxiv.org/abs/2602.03593

Quellen: METR 2025/26 (metr.org, RCT, N=57) · Cui/Demirer et al. 2025 (Management Science, 3 RCTs) · Chen et al. 2026 (ICSE-SEIP, N=2.989)

Das Delivery-Paradox

Output steigt

+66%

Epics completed per Developer

+98%

PRs merged

Faros AI 2025/26 · Telemetrie · 10.000+ Devs · 1.255 Orgs

Delivery bleibt gleich

0%

Verbesserung bei Deployment Frequency, Lead Time, MTTR, Change Failure Rate

DORA 2025 · Survey · ~5.000 Befragte

> *"AI does not automatically improve delivery performance — it amplifies existing engineering conditions."*

Quellen: Faros AI 2025/26 (Telemetrie, 10.000+ Devs, 1.255 Orgs) · DORA 2025 (Survey, ~5.000 Befragte)

Comprehension Debt

-17%

weniger Code-Verständnis nach KI-gestütztem Coding

Anthropic 2026 · RCT · N = 52 Devs

77%

Failure Rate ohne KI nach KI-gestütztem Training

Sankaranarayanan 2026 · N = 78 Studierende

304k

AI-Commits mit statisch messbaren Issues

Liu et al. 2026 · 6.275 Repos · 484k Issues

> *"The risk isn't that AI writes bad code. It's that we stop understanding the code we ship."*
>
> — Addy Osmani, Google Chrome Engineering

Der Triple-Debt-Effekt

Technical Debt

Im Code. Wartungskosten steigen.

304k AI-Commits · 24% der Issues persistieren
Liu et al. 2026 · 6.275 Repos

Cognitive Debt

In den Köpfen. Verständnis erodiert.

EEG: Gehirn-Konnektivität sinkt bei LLM-Nutzern
Kosmyna et al. 2025 · N = 54

Intent Debt

Im Wissen. Das "Warum" geht verloren.

Niemand hat die Design-Entscheidung bewusst getroffen
Denkmodell: Storey 2026

Wenn individuelle KI-Nutzung nicht zuverlässig hilft – was dann?

> *Es geht nicht ums Tool. Es geht darum, wie das **Team** es einsetzt.*

Die Evolution des KI-Einsatzes

Prompt Engineering

2022–23

Bessere Fragen stellen

→

Context Engineering

2024–25

Den richtigen Kontext zur richtigen Zeit

→

Harness Engineering

2026

Systemische Rückkopplung

> *"Context Engineering ist die Kunst, den richtigen Kontext zum richtigen Zeitpunkt zu geben."*
>
> — Tobi Lütke, Shopify CEO

> *"Engineering a user harness is a specific form of context engineering."*
>
> — Martin Fowler, 2026

Säule 1: Context Engineering

Context Engineering – Warum?

"Lost in the Middle"
Relevante Infos in der Mitte großer Kontexte werden ignoriert
Liu et al. 2024 · NeurIPS

32% der Agent-Failures
entstehen durch schlechten oder fehlenden Kontext
LangChain 2025 · State of AI Agents

52.8% → 66.5% SWE-Bench
nur durch besseren Kontext und Scaffolding
Bui 2026 · arxiv 2603.05344

ThoughtWorks Radar

Vol. 34 · April 2026

ADOPT	Context Engineering
ADOPT	Curated Shared Instructions
TRIAL	Progressive Context Disclosure
TRIAL	Agent Skills (modularer Kontext)
CAUTION	Agent Instruction Bloat

CLAUDE.md in der Praxis

Context Engineering

```
# Projekt: PaymentService

## Architektur
- Hexagonal Architecture (Ports & Adapters)
- PostgreSQL via JPA, Redis für Cache
- Keine direkten DB-Zugriffe aus Services

## Code-Konventionen
- Tests zuerst (TDD) -- nie ohne Test committen
- Deutsche Kommentare, englische Bezeichner
- Max 200 Zeilen pro Klasse

## Wichtige Befehle
- `./gradlew test` -- alle Tests
- `./gradlew check` -- inkl. Linter
- `docker-compose up` -- lokale Infra

## Was du NICHT tun sollst
- Keine @Transactional auf Service-Layer
- Keine Lombok -- explizite Konstruktoren
```

Was hineingehört

✓ Architektur-Entscheidungen

✓ Team-Konventionen

✓ Wichtige Befehle

✓ Explizite Verbote

✓ Domain-Vokabular

Was NICHT hineingehört

✕ Jede Bibliotheks-Doku

✕ Generische Best Practices

✕ Dinge, die im Code stehen

CAUTION Agent Instruction Bloat

Sweet Spot: **100–150 Zeilen.** Darüber hinaus sinkt die Performance wieder.

ETH Zürich 2026 · SWE-bench · arxiv 2602.11988

Progressive Context Disclosure

Context Engineering

Problem: Context Rot
Alles laden = nichts beachten.

Lösung: Just-in-Time
Nur relevanten Kontext laden.

TRIAL ThoughtWorks Radar Vol. 34

Aufgabe → Kontext

Neues Feature
ADRs + Domain-Glossar

Bug Fix
Error-Patterns + Testkonventionen

DB-Migration
Schema-Konventionen + Migrationen

Umsetzung: Skills

```
# .claude/skills/feature.md

Lies zuerst:
@docs/architecture/ADRs/
@docs/domain-glossar.md

Dann implementiere nach
Hexagonal Architecture.
```

Dev tippt: `/feature Rabattberechnung`
→ Agent hat den richtigen Kontext.

TRIAL Agent Skills · TW Radar Vol. 34

Säule 2: Harness Engineering

Das Harness-Modell

Harness Engineering

Feedforward Guides

→ CLAUDE.md / .cursorrules
→ Aufgaben-Templates
→ Architektur-Constraints
→ Domain-Vokabular

Agent

Feedback Sensors

→ Test-Suite (Unit + Integration)
→ Static Analysis / Linter
→ Architecture Tests (ArchUnit)
→ Mutation Testing

> *"A coding agent harness wraps an AI coding agent with feedforward guides and feedback sensors."*
> — Huang et al. via Martin Fowler, 2026

"Hatten wir doch alles schon?"

Harness Engineering

Vorher: Qualitätssicherung

- **Tests** — validieren menschlichen Code
- **Linter** — Stilkonsistenz für Devs
- **CI** — fängt Fehler vor dem Merge
- **ArchUnit** — dokumentiert Regeln

Jetzt: Agent-Infrastruktur

- **Tests** — die **Augen** des Agents
- **Linter** — **Feedforward**-Signal vorab
- **CI** — **Feedback-Loop**, Agent korrigiert sich
- **ArchUnit** — einzige **Garantie** gegen Drift

52.8% → 66.5%
SWE-Bench mit Harness
Bui 2026

80% → 91%
Reflexion-Pattern
Shinn et al. 2023

> *"LLMs cannot self-correct reasoning without external feedback."*
> — Huang et al. 2023

Mutation Testing: Coverage lügt

Harness Engineering

Schritt 1

100%

KI schreibt Code + Tests.
Alle Tests grün. Coverage perfekt.

Aber: **beweisen die Tests etwas?**

Schritt 2: Mutant

`>` wird zu `≥`

Tool ändert den Code
minimal und automatisch.

Erkennt dein Test die Änderung?

Mutant überlebt

Test ist immer noch grün
→ Test ist **wertlos**

Mutant getötet

Test schlägt fehl
→ Test **beweist etwas**

TRIAL Mutation Testing als KI-Guardrail · Tools: PIT (Java), Stryker (JS/TS)

Architecture Drift

Harness Engineering

Das Problem mit KI-generiertem Code

Jeder Agent-Run ist zustandslos – der Agent "vergisst" Architektur-Entscheidungen. Ohne Guardrails entstehen Inkonsistenzen.

Deterministisch (empfohlen)

```java
// ArchUnit -- läuft in CI
@ArchTest
static final ArchRule noDirectDbAccess =
  noClasses()
    .that().resideInPackage("..service..")
    .should().accessClassesThat()
    .resideInPackage("..repository..");
```

ADOPT ArchUnit, Dependency Cruiser

LLM-basiert (experimentell)

```yaml
# arch-check.yml
prompt: |
  Analysiere diese PR-Änderungen.
  Verletzen sie unsere Hexagonal
  Architecture? Begründe deine
  Antwort mit konkreten Stellen.
model: claude-opus-4
on: pull_request
```

ASSESS LLM-basierte Arch-Reviews

Säule 3: Team-Zusammenarbeit

Der Wandel in der Teamdynamik

Team-Zusammenarbeit

2025: Individuell

- Jeder nutzt KI anders
- Keine gemeinsamen Prompts
- Kein geteiltes Institutional Knowledge
- "Geheimwaffe" statt Team-Tool

2026: Team-Infrastruktur

- Gemeinsame CLAUDE.md / .cursorrules
- Mob/Pair mit Agents
- Geteilte Skills und Prompt-Bibliotheken
- Harness in shared CI/CD

Warum Review-Schleifen nicht reichen

Team-Zusammenarbeit

Nachträgliches Review

AI-Output sieht **plausibel** aus → Reviewer nickt ab

Fehlerhafte Muster replizieren sich **unbemerkt**

Wissen über Entscheidungen bleibt **beim Einzelnen**

Gemeinsam am Output arbeiten

Fehler werden **sofort** erkannt — nicht erst im Review

Entscheidungen entstehen **gemeinsam**

Wissen verteilt sich **automatisch** im Team

Mob/Pair Programming mit Agents

Team-Zusammenarbeit

Warum

Wissen bleibt im Team
Alle verstehen, was entstanden ist und warum.

Erfahrung wird weitergegeben
Seniors zeigen live, wann man dem Agent vertraut.

Bessere Entscheidungen
Vier Augen sehen mehr — weniger Rework.

Wie

Mob: Screen Share + Rotating Driver (15-20 min)

Parallel: Einzeln in Worktrees, dann Mob-Review

Mob für Komplexes. Parallel für Routine. Je nach Risiko entscheiden.

Die drei Säulen in Aktion

Vom Prozess zur Anwendung — Drei Säulen in Aktion

Context Engineering

Ausgangslage: Excel, implizites Wissen

→ Interviews transkribieren

→ Wissen explizit machen

→ Kontext für den Agent bereitstellen

Harness Engineering

→ Walking Skeleton zuerst

→ Architektur-Entscheidungen früh

→ Guardrails setzen (ArchUnit, Tests)

→ Deterministisches System aufbauen

Team-Zusammenarbeit

→ Schnell prototypen

→ PO validiert iterativ

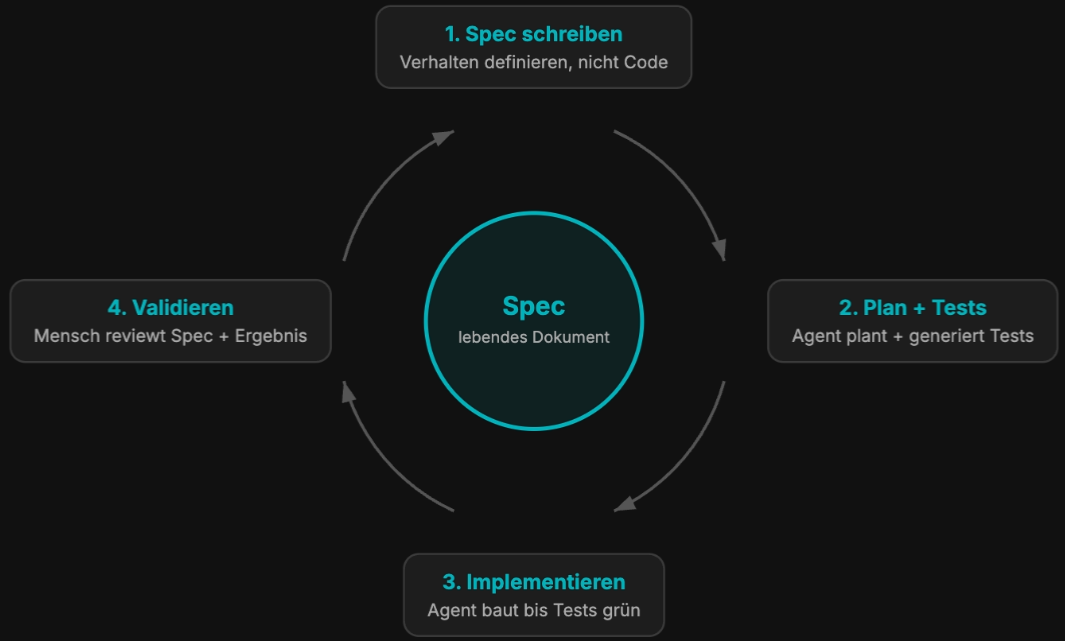
→ Mob/Pair am Agent

→ Spec als lebende Doku

> *Aber wie wird aus dem PO-Wissen eine Spec, die der Agent versteht — und die aktuell bleibt?*

Spec-Driven Development

> *"A development paradigm that uses well-crafted software requirement specifications as prompts, aided by AI coding agents, to generate executable code."*
>
> — Liu Shangqi, ThoughtWorks · December 2025



Spec-Tooling im Vergleich

Superpowers

Brainstorm → Spec → Plan → Execute

Visual Companion (Mockups im Browser)

Hard Gates + TDD integriert

Agent: Claude, Copilot, Gemini CLI

Maintenance: manuell

github.com/obra/superpowers · 171k Stars

Spec Kit

constitution → specify → plan → tasks

Drift Detection (Spec Sync)

CI Guard blockiert ohne Spec

Agent: jeder

Maintenance: Drift Detection

github.com/github/spec-kit · 91k Stars

AWS Kiro

Prompt → Requirements → Architecture

EARS-Notation (formaler)

IDE-nativ (VS Code Fork)

Agent: built-in (Claude)

Maintenance: regenerieren

kiro.dev · Preview 2026

BMAD Method

Persona-basiert (PM, Architect, Dev...)

Story → Spec → Checklist → Code

Umfangreiche Templates + Checklisten

Agent: jeder

Maintenance: manuell (schwergewichtig)

github.com/bmad-code-org/BMAD-METHOD

> *Die entscheidende Frage ist nicht Tag 1, sondern **Tag 30**: Bleibt die Spec aktuell — oder wird sie zu toter Dokumentation?*

Spec-Tooling-Empfehlungen

Superpowers

`/brainstorm` Interaktive Fragen, 2-3 Ansätze

→ **Design Spec** wird automatisch geschrieben

`/write-plan` TDD-Schritte mit Code pro Task

`/execute-plan` Subagent pro Task + Review

Hard Gate: kein Code vor Spec-Approval · Visual Companion für Mockups

```
## Domain Model
record Ablehnungsnotiz(zeitpunkt, autor, kommentar)
## Ports
ablehnen(UUID, String, String)
```

→ Sagt dem **Entwickler** was er bauen soll

Spec Kit

`/speckit-constitution` Projekt-Prinzipien

`/speckit-specify` User Stories, Given/When/Then

`/speckit-plan` Architektur + Tech-Entscheidungen

`/speckit-tasks` + `/speckit-implement`

+ `/speckit-analyze` für Drift Detection · `/speckit-checklist` für Qualität

```
### User Story 1 (P1)
Given eingereichte Reise, When Teamlead ablehnt,
Then Kommentar + Zeitstempel gespeichert
### Success Criteria
SC-001: Ablehnung in unter 30 Sekunden
```

→ Sagt dem **PO** was er bekommt

Tag 30: Spec Drift in der Praxis

Was passiert ist

PO Maria: "Wir brauchen doch 'Erneut einreichen'." — Ein Entwickler implementiert das Feature. Beide Specs sagen noch: **"out-of-scope"**.

Superpowers

```
## Abgrenzung
~~Kein "erneut einreichen"~~
```

Niemand merkt den Widerspruch.

Spec Kit + Analyze

```
DRIFT DETECTED:
Spec sagt "out-of-scope",
Code hat erneutEinreichen()
```

Drift Detection findet den Widerspruch.

> *Specs driften nicht irgendwann — sie driften **in der ersten Woche**. Ohne Tooling merkt es niemand.*

Ausblick

Die Rolle des Entwicklers

Context Engineer

Stories & Epics → **Lebende Specs** mit Agent

Wissen, welcher Kontext wann benötigt wird

Quality Guardian

Manuelle QA → **Harness, Mutation Tests, Arch Guards**

Feedback-Systeme bauen und Agent-Output validieren

Code Versteher

Blind mergen → **AI-Code lesen, hinterfragen, besitzen**

Gegen Comprehension Debt: verstehen, was man shipped

Team-Orchestrator

Async PR-Review → **Mob/Pair am Agent**

Mensch-Maschine-Zusammenarbeit koordinieren

> *"The role of the programmer is changing from writing code to defining what good code looks like -- and then verifying the machine got it right."*
>
> — Kent Beck

Key Takeaways

1. **Zusammen arbeiten, nicht nachträglich reviewen**

   Mob/Pair mit Agents statt Review-Schleifen. Wissen bleibt im Team, Erfahrung wird live weitergegeben.

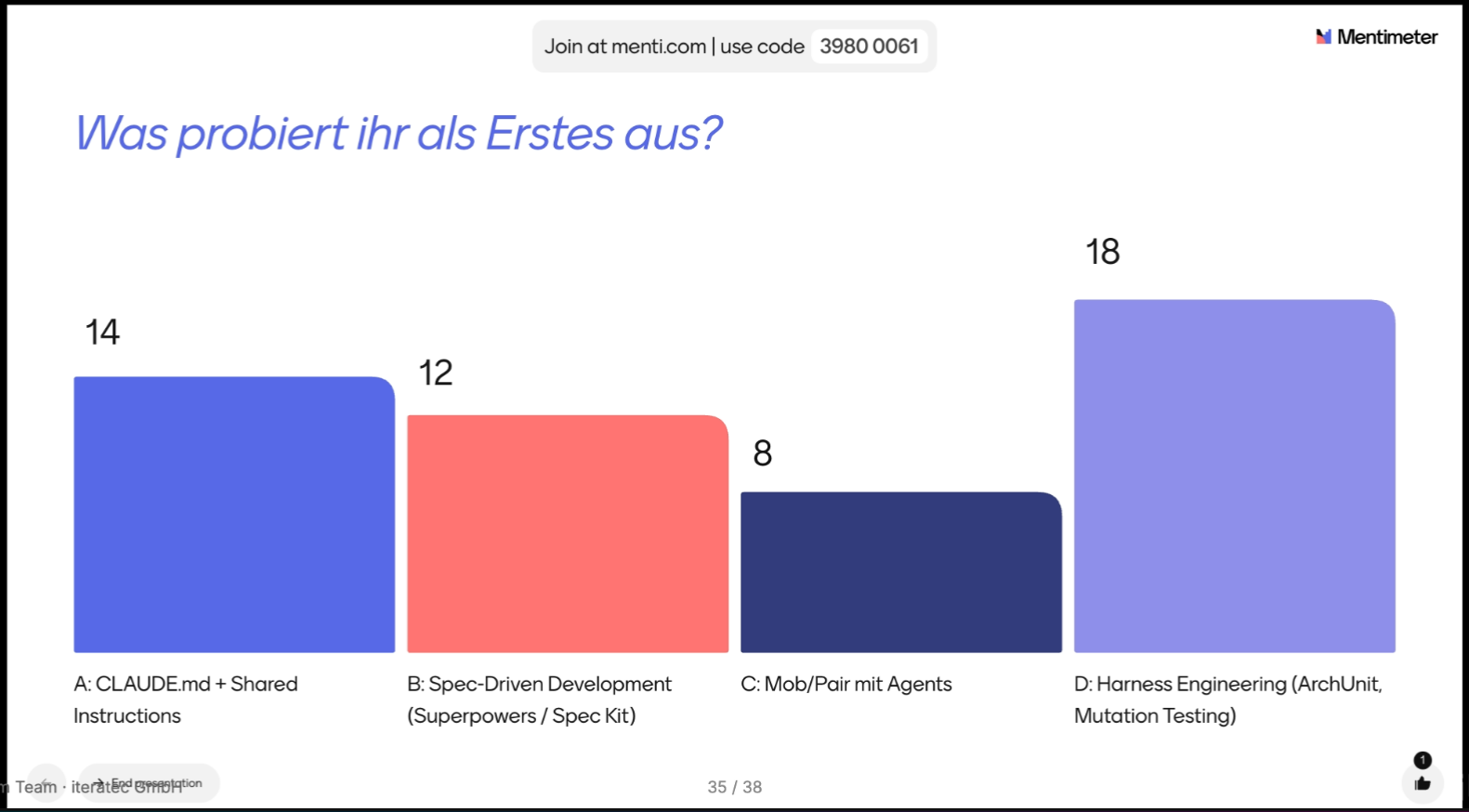
2. **Context + Harness + Spec = Infrastruktur**

   CLAUDE.md, Test-Harness und Spec-Driven Development sind keine Extras — sie sind die Basis für produktive Agent-Nutzung.

3. **Specs driften — plant dafür**

   Tag 1 ist einfach. Tag 30 entscheidet. Nutzt Tooling (Spec Kit, Superpowers) und haltet Specs lebendig.

Was probiert ihr als Erstes aus?



Quellen: Produktivität + Cognitive Debt

METR 2026 — Measuring AI Impact on Dev Productivity. N=57, 800+ Tasks. metr.org

Cui/Demirer et al. 2025 — Effects of GenAI on High-Skilled Work. Management Science. 3 RCTs.

Chen et al. 2026 — Beyond the Commit. ICSE-SEIP. N=2.989, BNY Mellon.

Faros AI 2026 — AI Productivity Paradox. Telemetrie. 10.000+ Devs. faros.ai

DORA 2025 — State of AI-Assisted Software Development. ~5.000 Befragte.

Anthropic 2026 — How AI Impacts Skill Formation. RCT, N=52. arxiv 2601.20245

Sankaranarayanan 2026 — Epistemic Debt / Fragile Experts. N=78. arxiv 2602.20206

Liu et al. 2026 — Debt Behind the AI Boom. 304k Commits, 6.275 Repos. arxiv 2603.28592

Liu et al. 2024 — Lost in the Middle. NeurIPS.

Osmani 2026 — Comprehension Debt. Google Chrome Engineering.

Storey 2026 — Triple Debt Model (Denkmodell). arxiv 2603.22106

Huang et al. 2023 — LLMs Cannot Self-Correct. arxiv

Quellen: Engineering + Tooling

ThoughtWorks Radar Vol. 34 — April 2026. thoughtworks.com/radar

Fowler 2026 — Harness Engineering. martinfowler.com/articles/harness-engineering.html

Lütke 2025 — Context Engineering. Shopify CEO.

Kent Beck 2023-25 — AI makes XP more important.

Shinn et al. 2023 — Reflexion: Self-Correction via Test Feedback. arxiv

Bui 2026 — Building Effective AI Coding Agents. arxiv 2603.05344

ETH Zürich 2026 — Evaluating AGENTS.md. arxiv 2602.11988

LangChain 2025 — State of AI Agents. 32% Context Failures.

Superpowers — github.com/obra/superpowers · 171k Stars

Spec Kit — github.com/github/spec-kit · 91k Stars

Danke!

Kontakt

David Schowalter

david.schowalter@iteratec.com

github.com/schowave

LinkedIn

iteratec