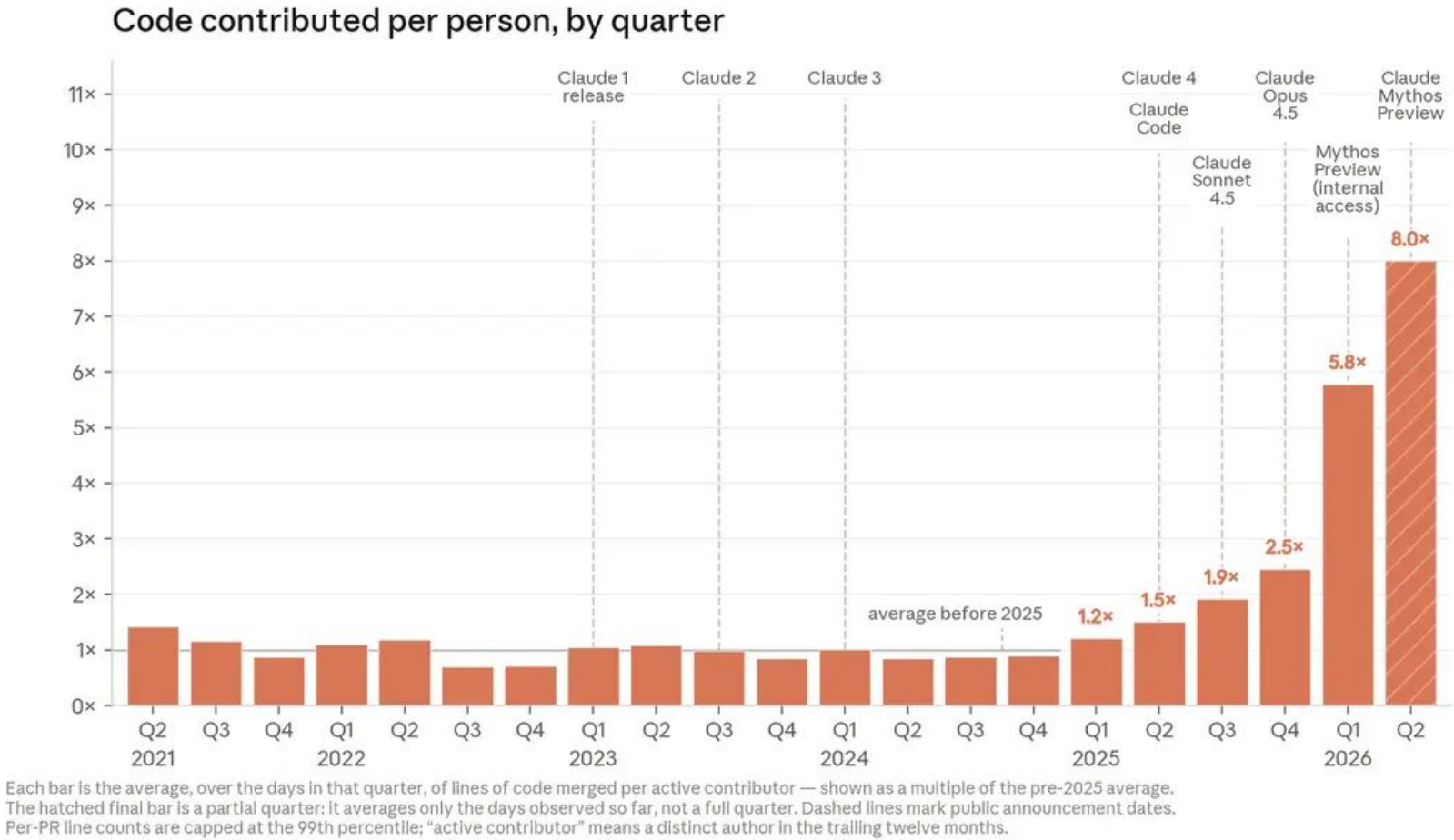
Copiloten für Agenten

Qualitätssicherung in der Ära der Coding Agents

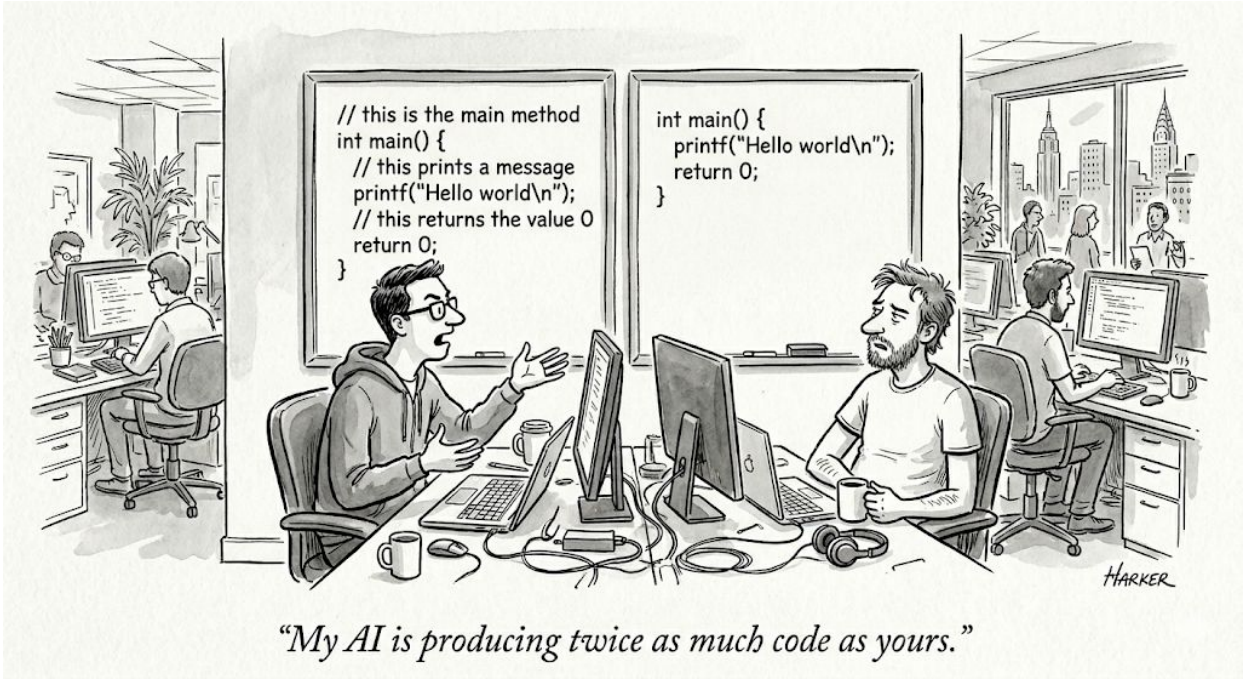
CQSE | Dr. Benjamin Hummel



Each bar is the average, over the days in that quarter, of lines of code merged per active contributor — shown as a multiple of the pre-2025 average. The hatched final bar is a partial quarter: it averages only the days observed so far, not a full quarter. Dashed lines mark public announcement dates. Per-PR line counts are capped at the 99th percentile; "active contributor" means a distinct author in the trailing twelve months.

Source: https://www.anthropic.com/institute/recursive-self-improvement

Lines of Code war eigentlich noch nie ein guter Indikator für Produktivität oder gar Qualität



BUSINESS INSIDER

Subscribe | Newsletters

AI

Jensen Huang says he would be 'deeply alarmed' if his $500,000 engineer did not consume at least $250,000 of tokens

By Lee Chong Ming + Follow

```
while true
do
  claude -p "Task $RANDOM: " \
    "Write a 1000 word essay on AI."
done
```

FARDS

AI ENGINEERING REPORT | 2026

THE ACCELERATION WHIPLASH

Engineering throughput is up. However, bugs, incidents, and rework are rising faster.

Analysis of two years of data from 22,000 developers on the real-world impact of AI adoption.

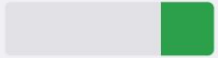
OUTPUT IS UP. NOT ALL OF IT STICKS.

Impacts of high AI adoption on engineering throughput

Epic
A sizable feature or capability, hundreds of tasks

+66.2%
Epics completed per developer

Task
Individual work item, may span multiple PRs



+33.7%
Task throughput per developer

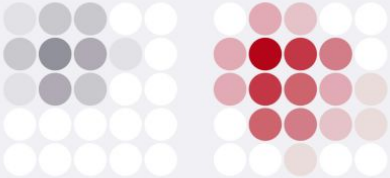
Within this, tasks with an associated PR grew +210% per team. Code-specific work accelerated far faster than overall task volume.

CODE IS GETTING LARGER AND MORE COMPLEX

Impacts of high AI adoption on code complexity

AI ADOPTION

LOW HIGH



+59.7%
files edited per PR

AI ADOPTION

LOW HIGH

+51.3%
average PR size

DEVELOPER FOOTPRINT/MONTH

+149.9%
files touched per developer

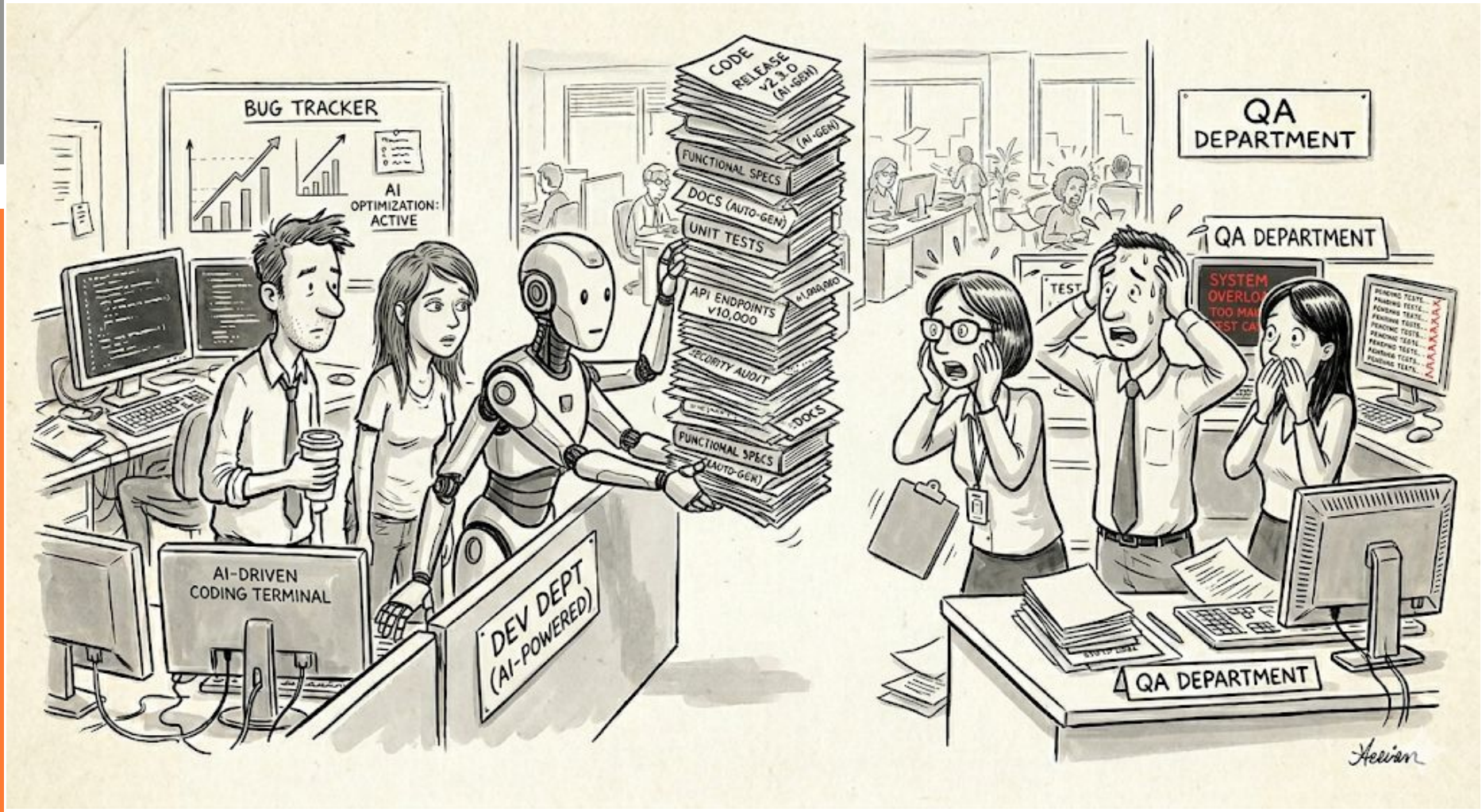
+11.7%
repositiories touched per developer

Telemetry dataset size: 22,000 developers across 4,000 teams

AI Engineering Report 2026 | The Acceleration Whiplash

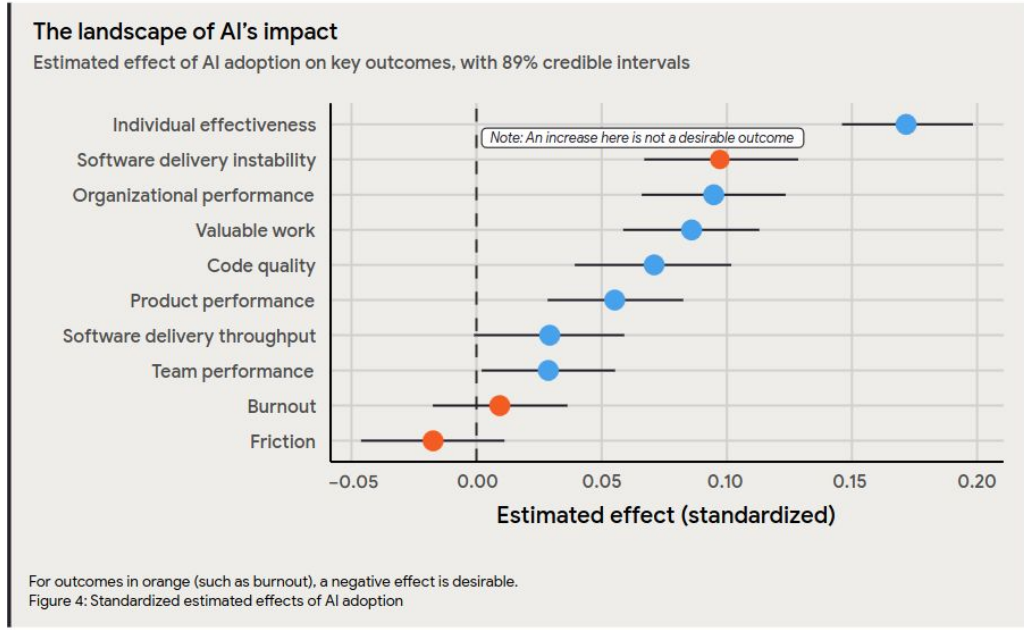
FARDS


BUG TRACKER
QA DEPARTMENT
QA DEPARTMENT
TEST
TEST CASES
FAILED
CODE RELEASE V2.3
DOCS
FUNCTIONAL SPECS
= DOCS
QA DEPARTMENT

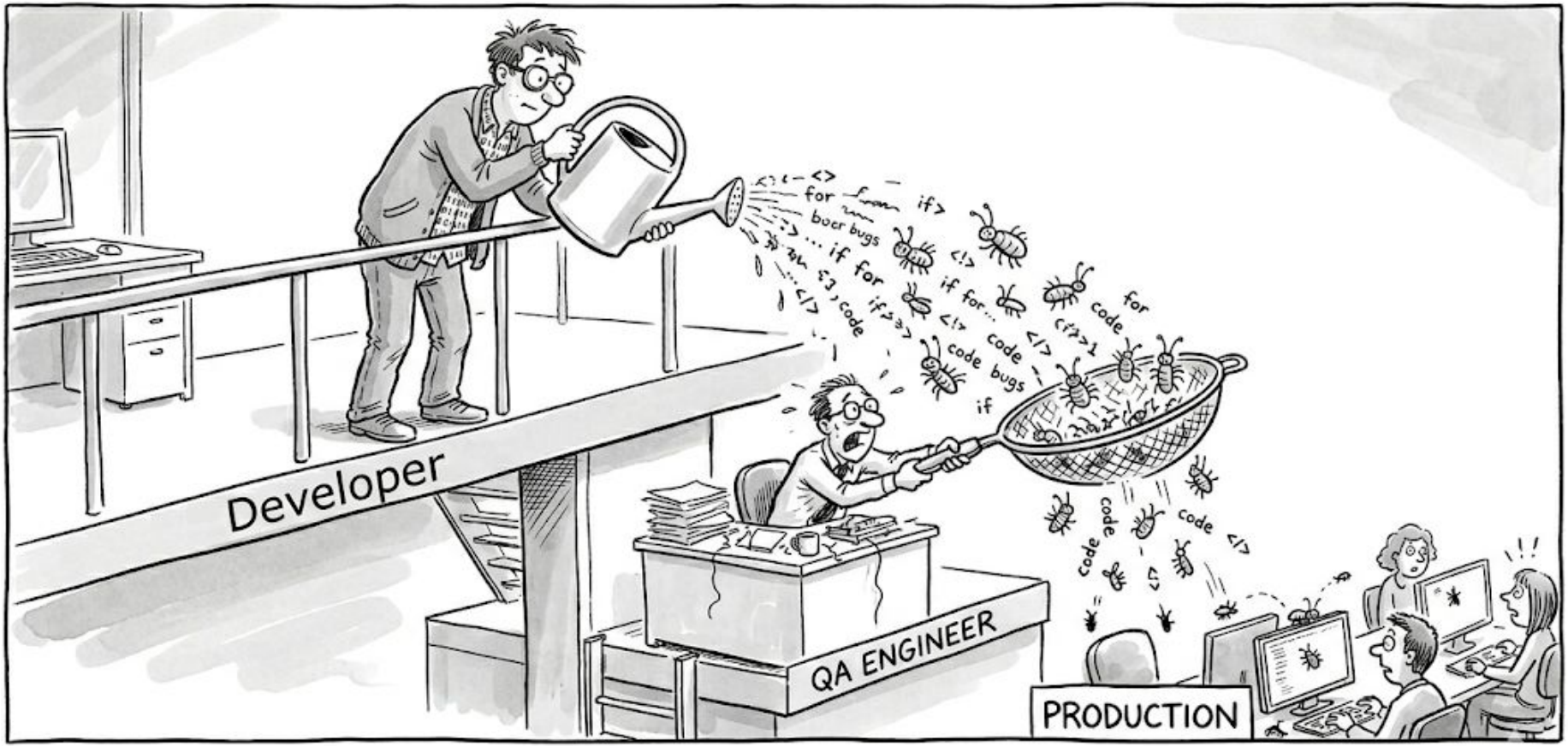

BUG TRACKER
AI OPTIMIZATION: ACTIVE
AI-DRIVEN CODING TERMINAL
DEV DEPT (AI-POWERED)
CODE RELEASE V2.3.0 (AI-GEN)
(AI-GEN)
FUNCTIONAL SPECS
DOCS (AUTO-GEN)
UNIT TESTS
API ENDPOINTS ∨10,000
SECURITY AUDIT
DOCS
FUNCTIONAL SPECS (AUTO-GEN)
QA DEPARTMENT
QA DEPARTMENT
TEST
QA DEPARTMENT

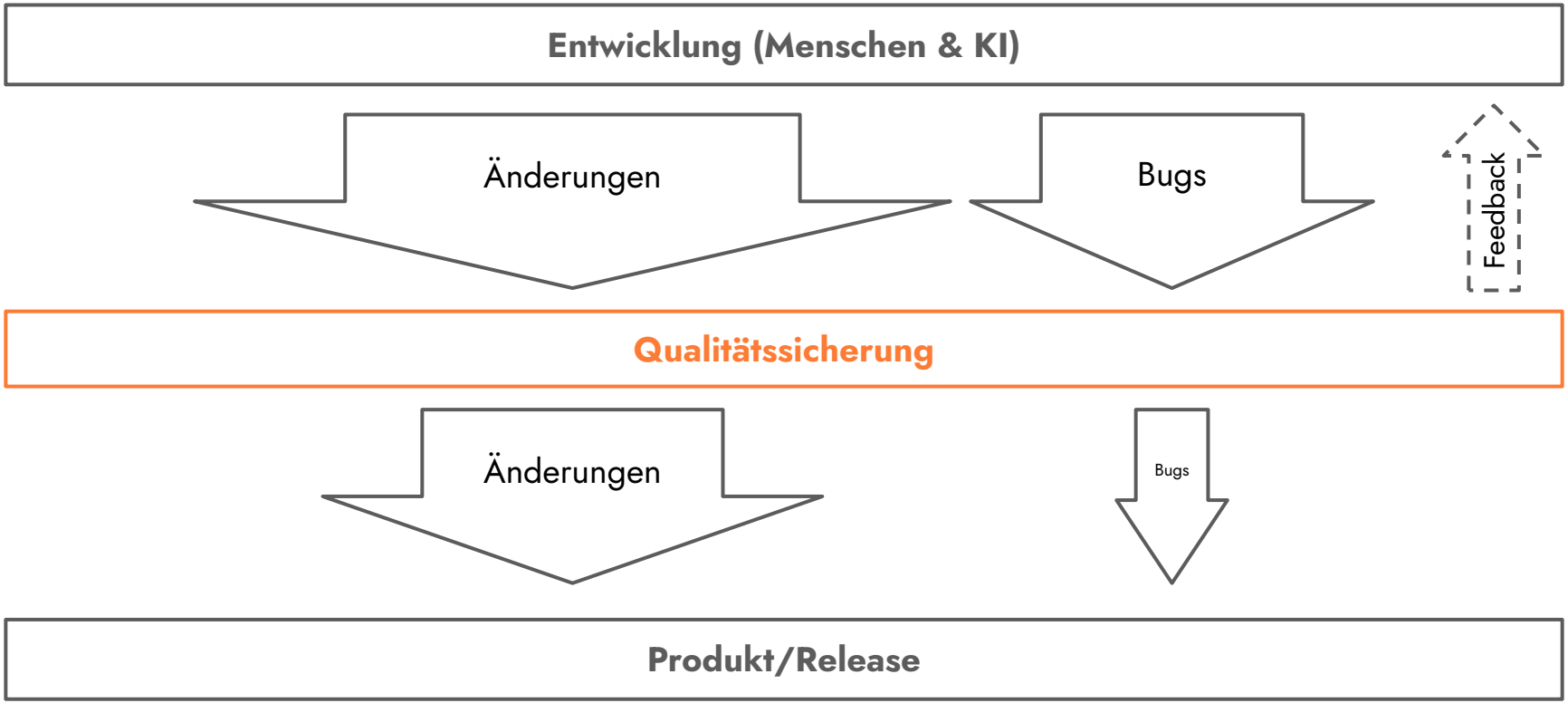
The ROI of AI-assisted Software Development

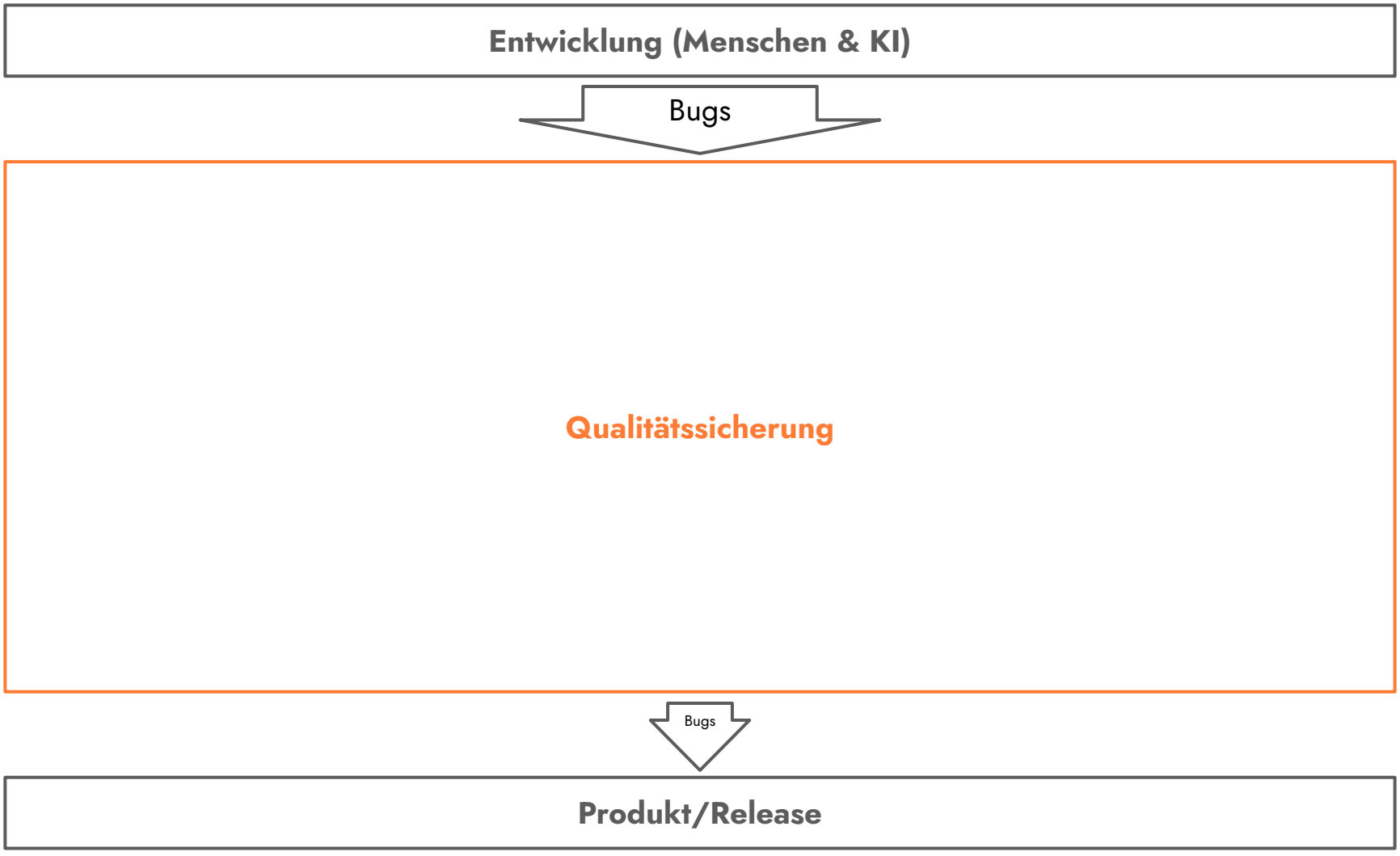
Google Cloud

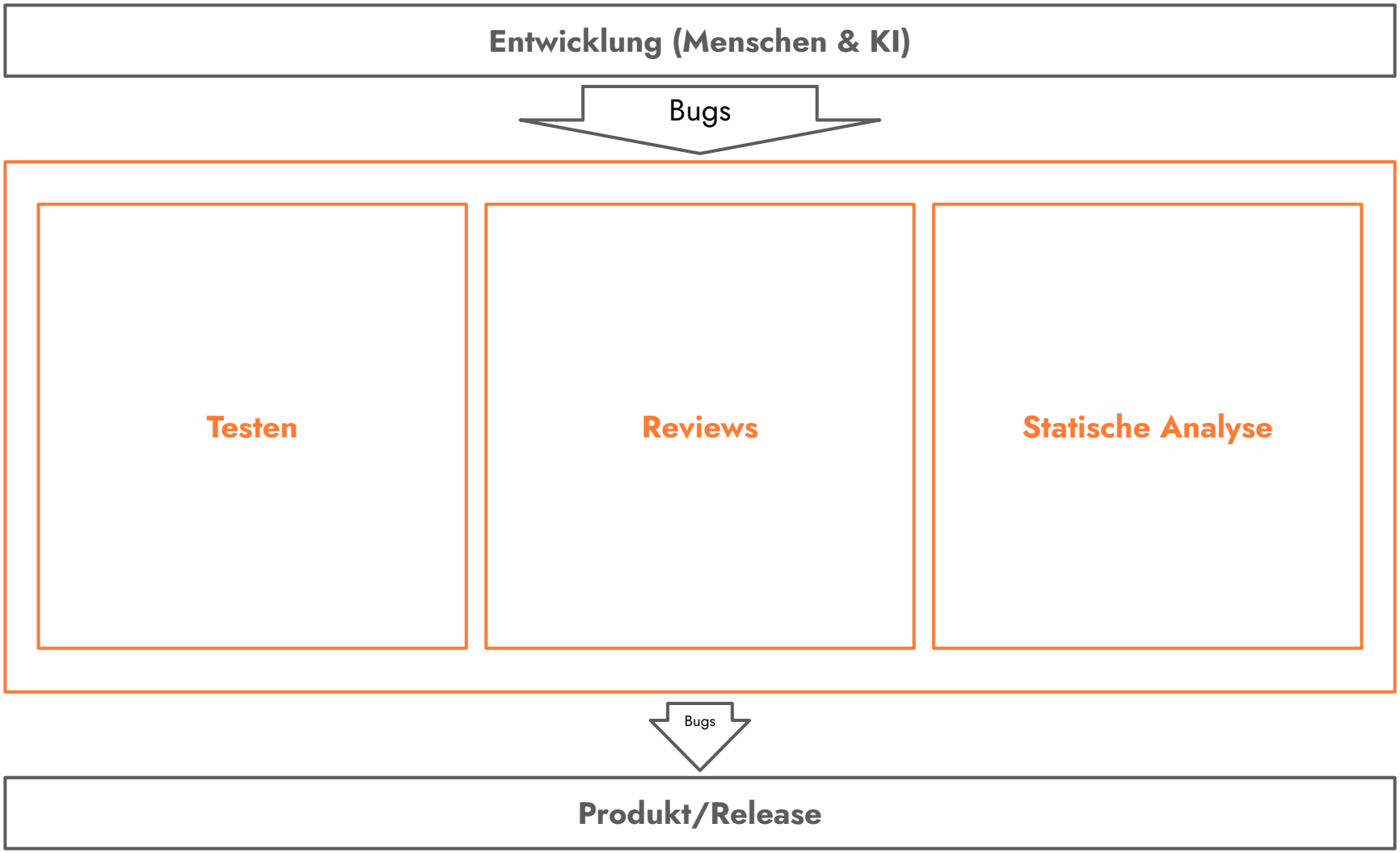


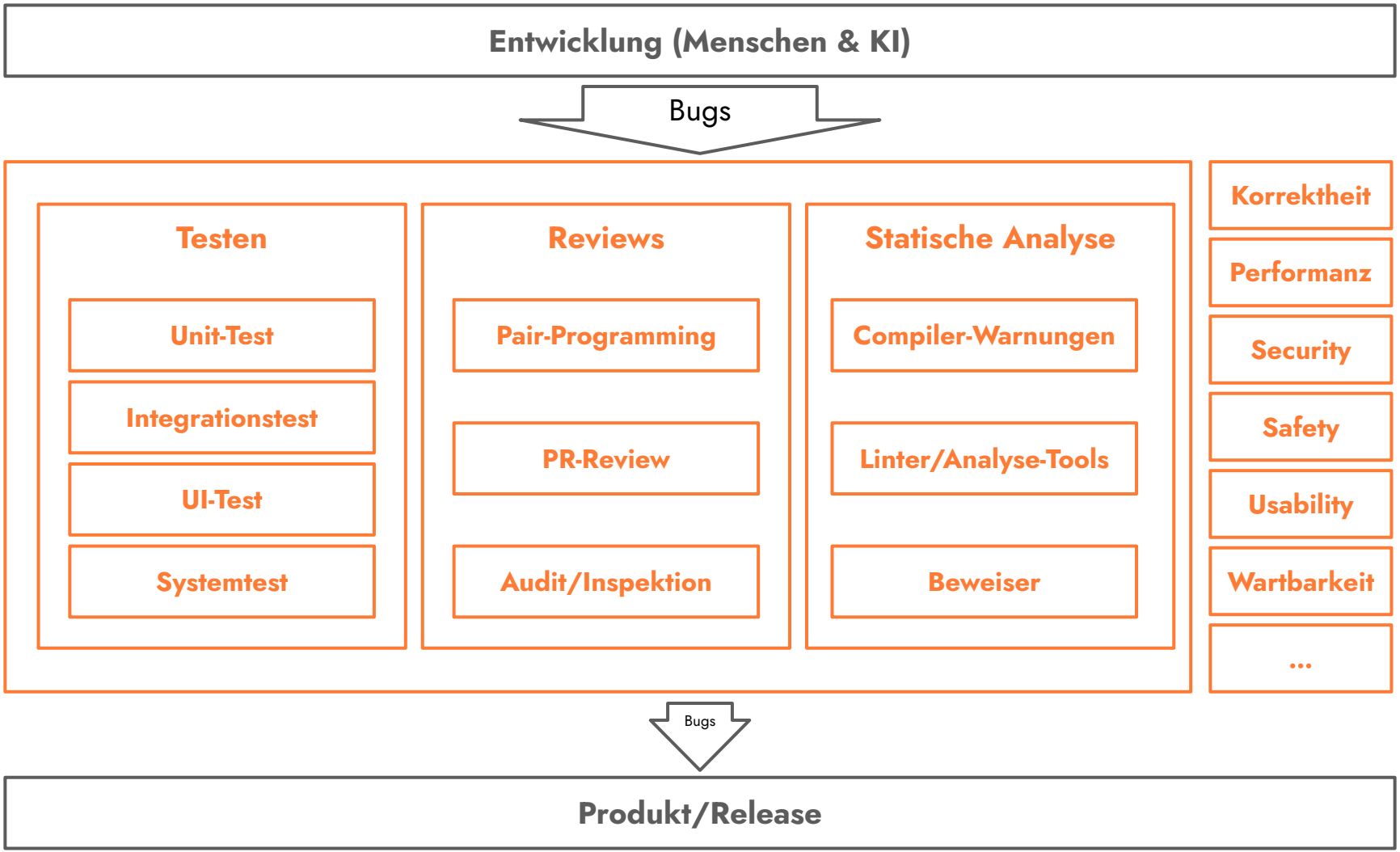
For outcomes in orange (such as burnout), a negative effect is desirable.
Figure 4: Standardized estimated effects of AI adoption


Developer
QA ENGINEER
PRODUCTION
for
if
bugs
code
bugs

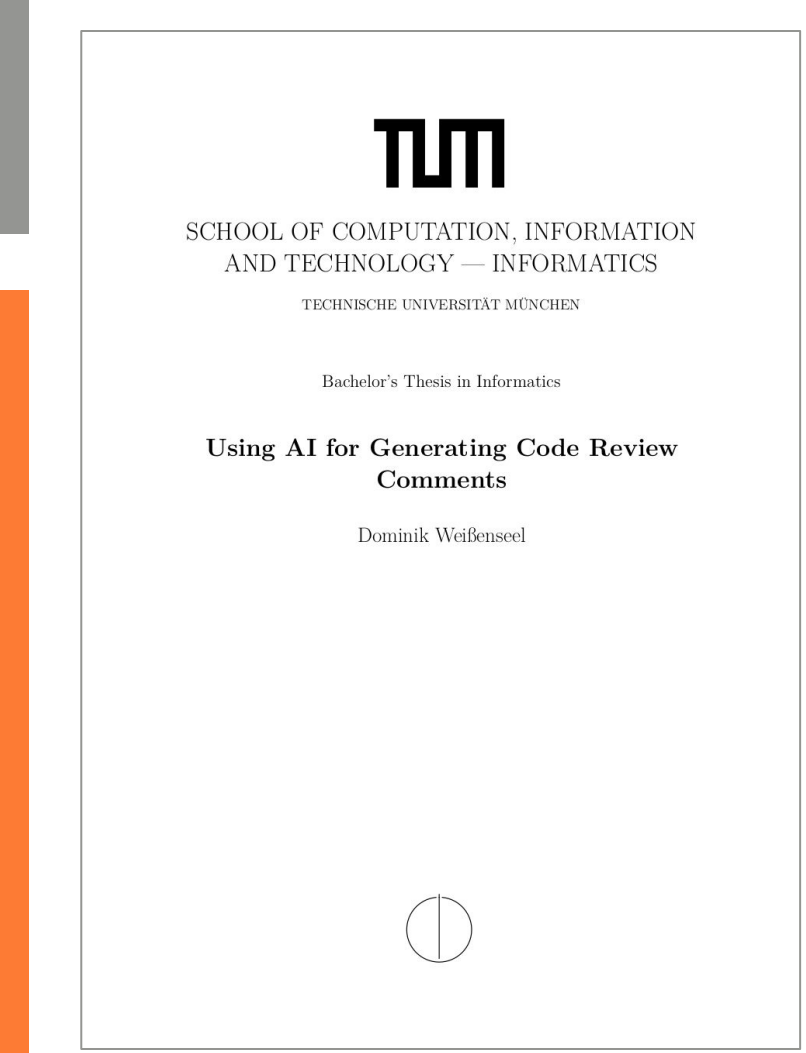

Entwicklung (Menschen & KI)
Änderungen
Bugs
Feedback
Qualitätssicherung
Änderungen
Bugs
Produkt/Release


Entwicklung (Menschen & KI)
Bugs
Qualitätssicherung
Bugs
Produkt/Release


Entwicklung (Menschen & KI)
Bugs
Testen
Reviews
Statische Analyse
Bugs
Produkt/Release


Entwicklung (Menschen & KI)
Bugs
Testen
Unit-Test
Integrationstest
UI-Test
Systemtest
Reviews
Pair-Programming
PR-Review
Audit/Inspektion
Statische Analyse
Compiler-Warnungen
Linter/Analyse-Tools
Beweiser
Korrektheit
Performanz
Security
Safety
Usability
Wartbarkeit
...
Bugs
Produkt/Release

Wie gut funktionieren AI-Reviews?



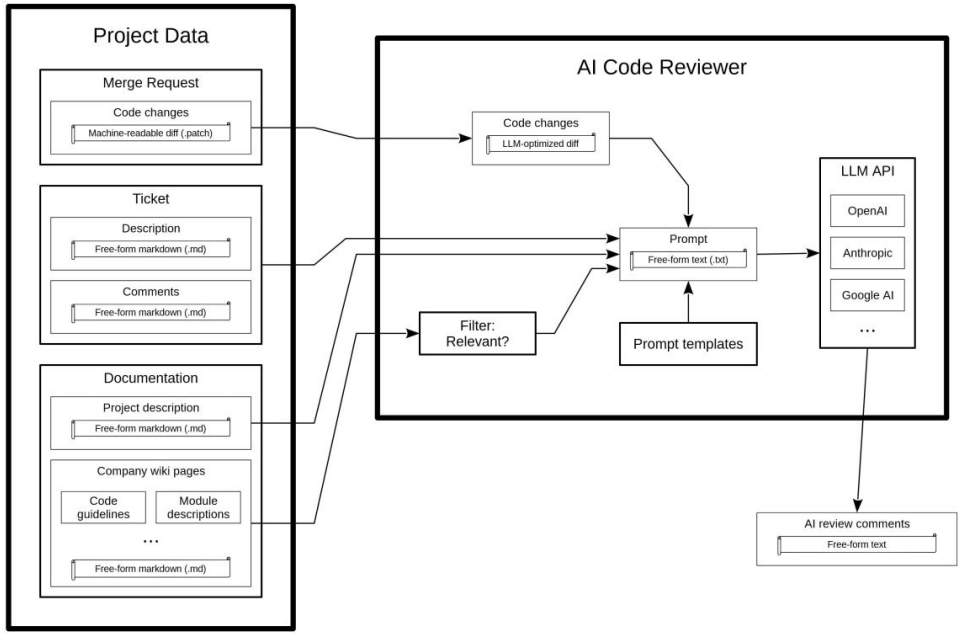
SCHOOL OF COMPUTATION, INFORMATION AND TECHNOLOGY — INFORMATICS

TECHNISCHE UNIVERSITÄT MÜNCHEN

Bachelor's Thesis in Informatics

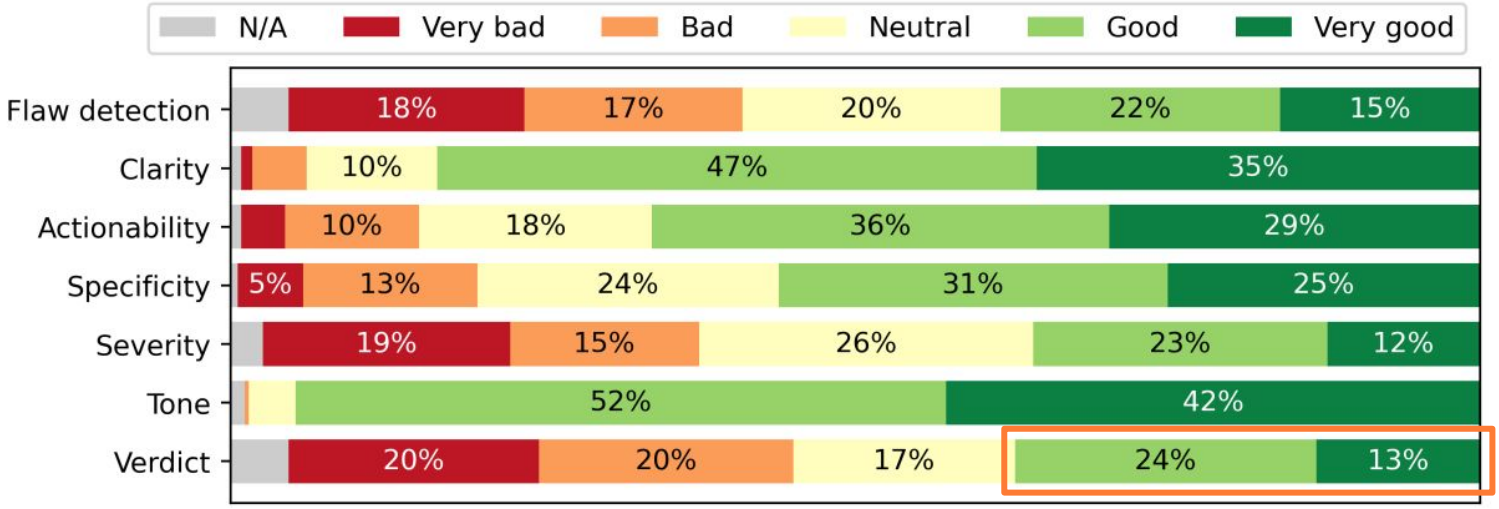
Using AI for Generating Code Review Comments

Dominik Weißenseel



	N/A	Very bad	Bad	Neutral	Good	Very good
Flaw detection	○	○	○	○	○	○
Clarity	○	○	○	○	○	○
Actionability	○	○	○	○	○	○
Specificity	○	○	○	○	○	○
Severity	○	○	○	○	○	○
Tone	○	○	○	○	○	○
Verdict	○	○	○	○	○	○

"Ein Fazit ist, dass das LLM gut darin ist, jemand mit oberflächlichem Wissen (= Junior-Entwickler?!) vom Vorhandensein eines Qualitätsproblems zu überzeugen"



Nur ca. ein Drittel der Kommentare wurde als "sinnvoll/sollte umgesetzt" eingeordnet

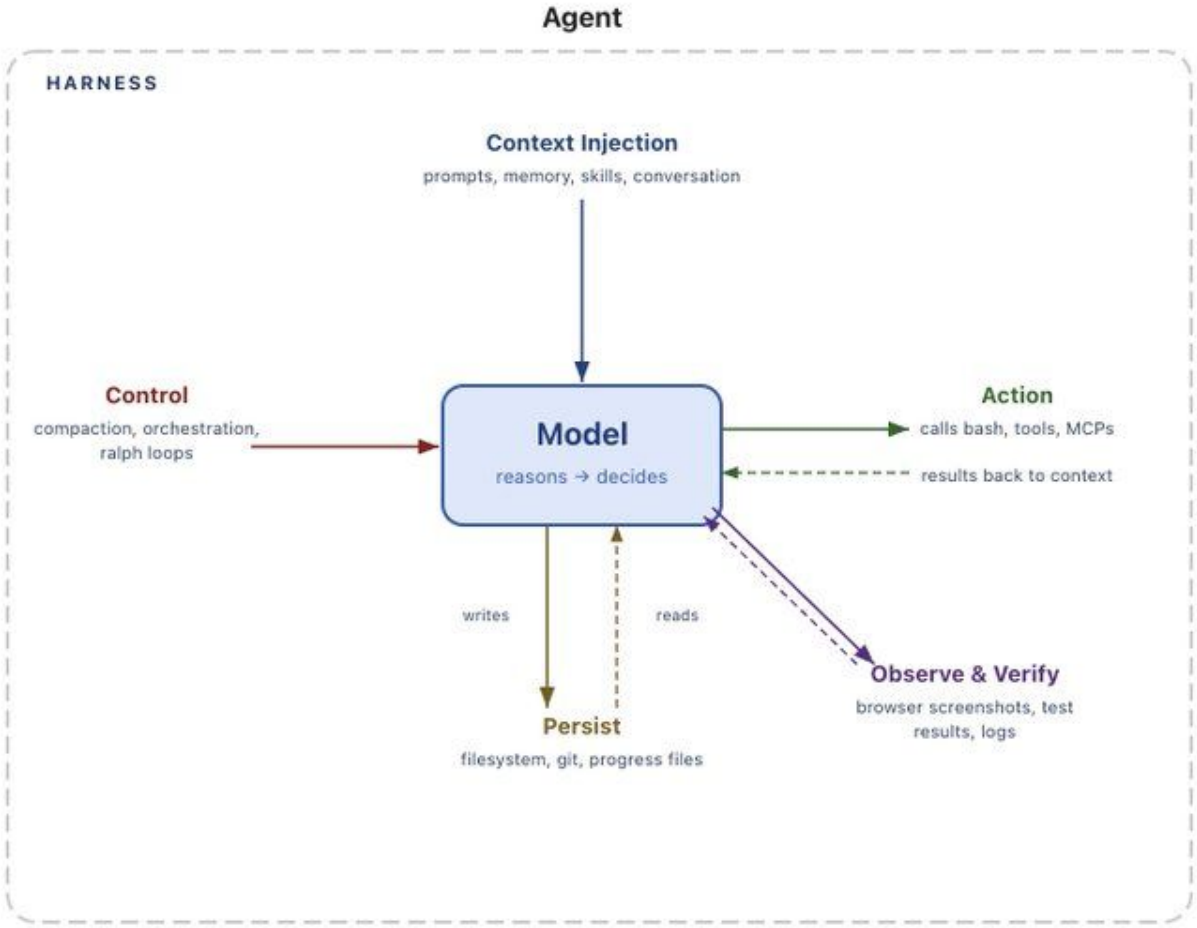
Ergebnis nach der Einführung eines führenden “AI Review Tools” bei einem großen Konzern (mehrere 1000 Entwickler:innen) begleitet durch den Tool-Hersteller über einen Zeitraum von über einem Jahr:

Akzeptanzquote der einzelnen Review-Kommentare liegt bei ca. 30%

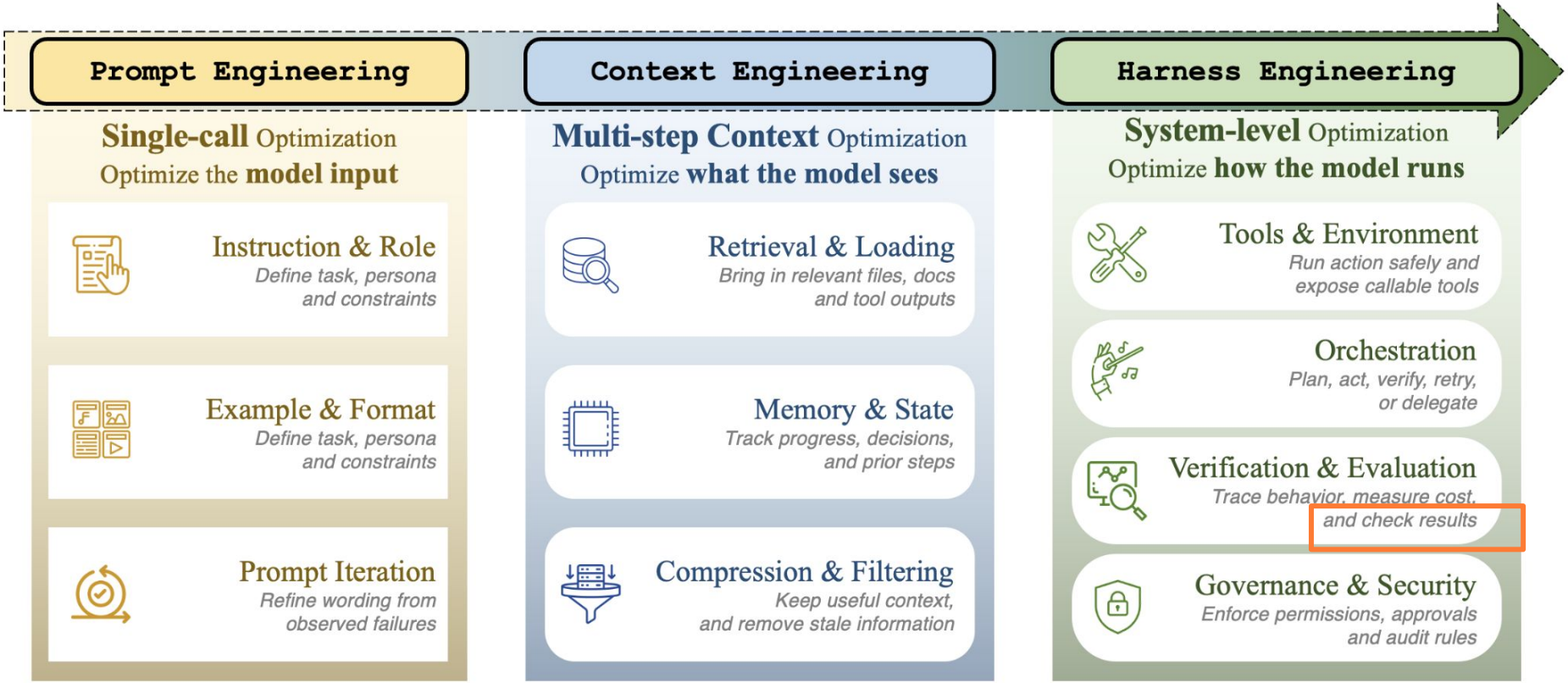
AI-Reviews: Eher **Ergänzung** als Ersatz

- AI-Reviews finden auch relevante Punkte, die der Mensch leicht übersieht
- Ohne "Human in the Loop" lassen sich richtige und falsche Ergebnisse nicht trennen
- "Menschliche" PR-Reviews finden noch viele zusätzliche Probleme, auch wenn vorher ein AI-Review gelaufen ist

Kann man Qualitätsprobleme nicht gleich im Agenten beheben?



Quelle: https://www.langchain.com/blog/the-anatomy-of-an-agent-harness


Prompt Engineering
Context Engineering
Harness Engineering
Single-call Optimization
Optimize the model input
Instruction & Role
Define task, persona and constraints
Example & Format
Define task, persona and constraints
Prompt Iteration
Refine wording from observed failures
Multi-step Context Optimization
Optimize what the model sees
Retrieval & Loading
Bring in relevant files, docs and tool outputs
Memory & State
Track progress, decisions, and prior steps
Compression & Filtering
Keep useful context, and remove stale information
System-level Optimization
Optimize how the model runs
Tools & Environment
Run action safely and expose callable tools
Orchestration
Plan, act, verify, retry, or delegate
Verification & Evaluation
Trace behavior, measure cost, and check results
Governance & Security
Enforce permissions, approvals and audit rules

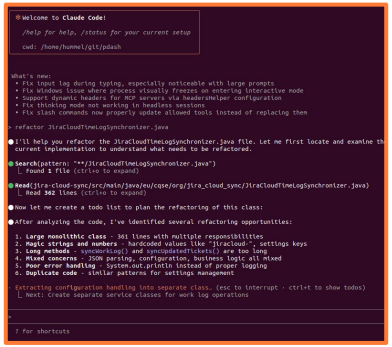
Qualitätssicherung

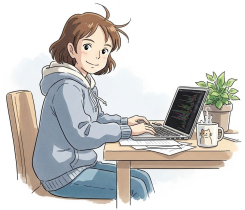
LLMs

Claude

OpenAI

Gemini





Tools

Read/Write
Shell
MCP
etc.

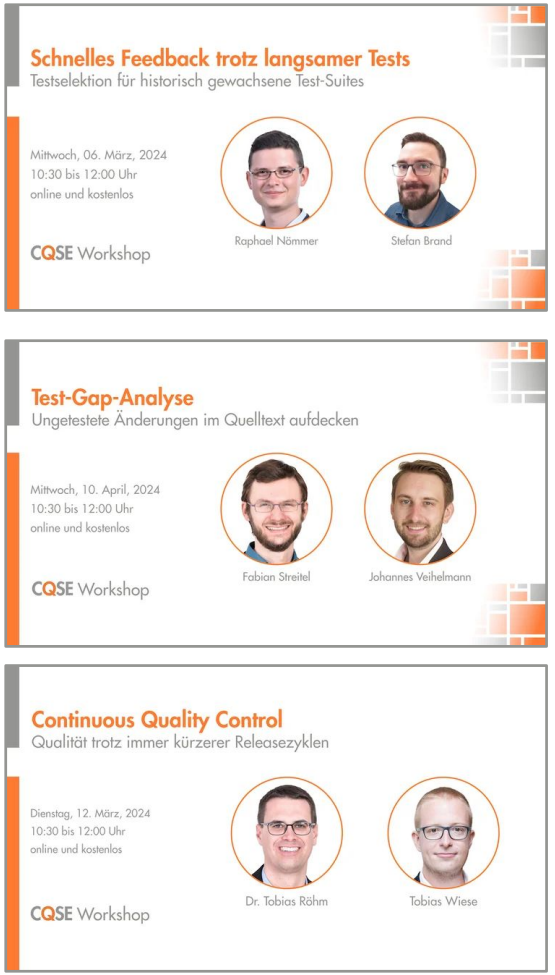
Sehr schnelles Feedback

- Laufen die Tests noch? ⇒ Test-Selektion
  - Auswahl einer kleinen Menge von Tests auf Basis der Änderungen
- Brauche ich neue Tests? ⇒ Test-Gap-Analyse
  - Geänderter/neuer Code ohne Tests finden
- “Clean” Code & Fehlermuster ⇒ Statische Analyse
  - Pre-Commit, also kein CI/CD
  - Inkrementelle Analyse (statt komplettem System)

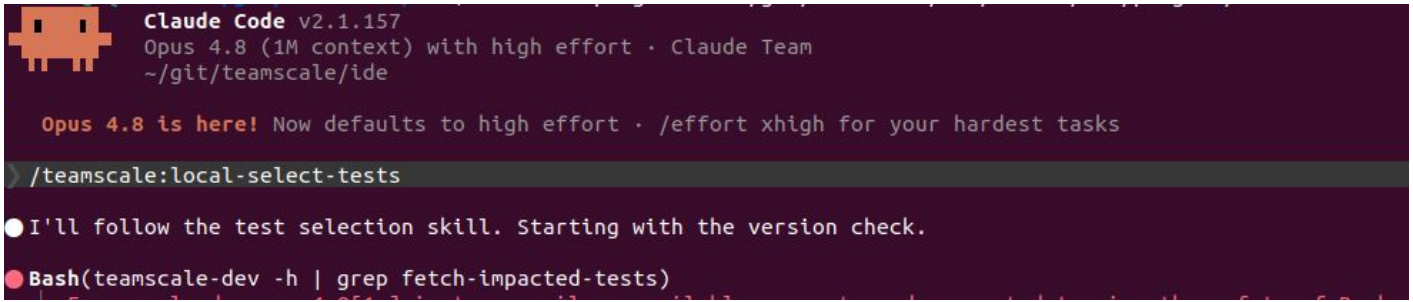
Good News

- Das ist alles schon recht gut erforscht/verstanden
- Es gibt eine Vielzahl an Tools in diesem Umfeld
- Diese Tools und Ansätze sind auch nützlich, wenn man mal ohne KI arbeitet

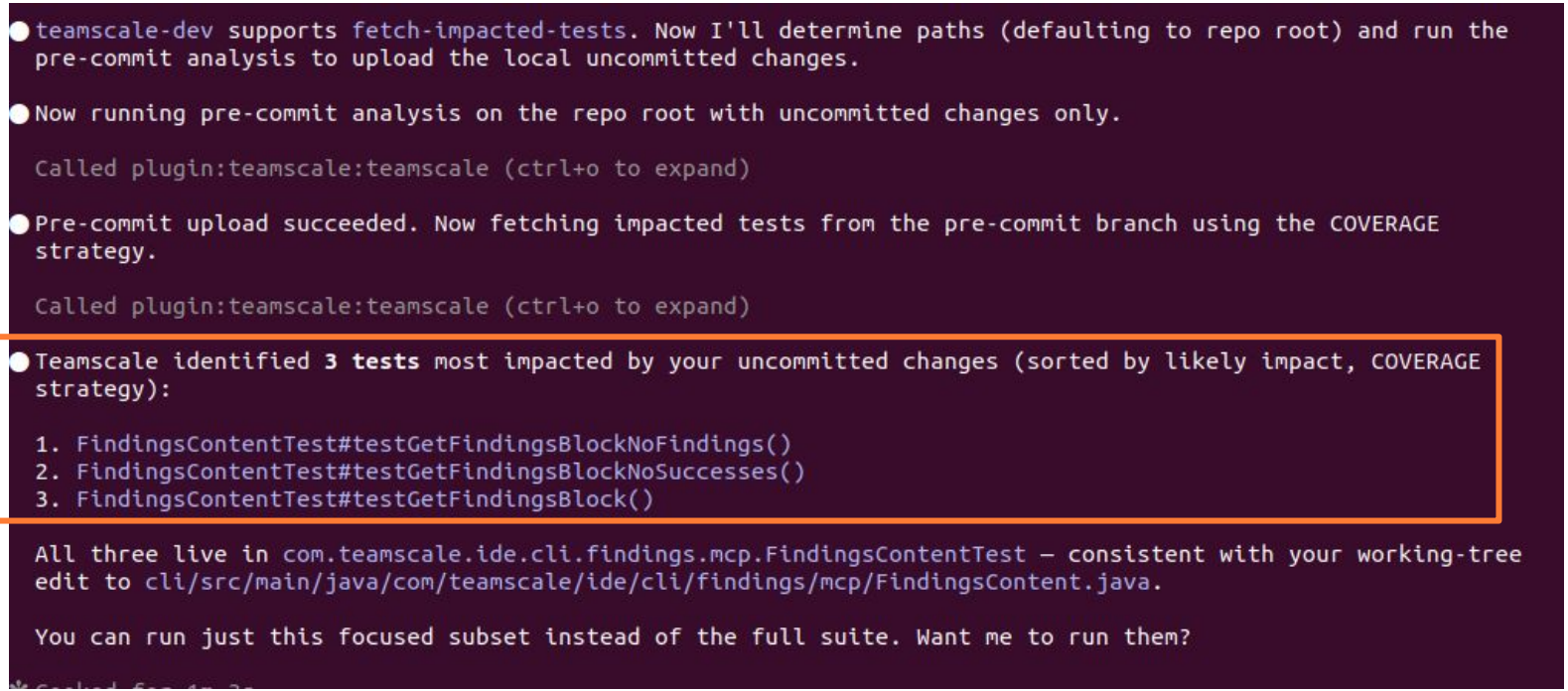
Das muss “nur noch” in den Harness integriert werden



Beispiel: Test-Selektion in Claude Code



Beispiel: Test-Selektion in Claude Code



Beispiel: Test-Selektion in Claude Code

```
● Bash(./gradlew :cli:test --tests "com.teamscale.ide.cli.findings.mcp.FindingsContentTest" 2>&1 | tail -30)
  └ com.teamscale.ide.cli.findings.mcp.FindingsContentTest

      Test testGetFindingsBlock() FAILED
    … +30 lines (ctrl+o to expand)
  └ (timeout 10m)
  └ Allowed by auto mode classifier

● Teamscale's test selection was spot-on — all 3 impacted tests ran, and one failed, exposing a real problem in
  your uncommitted edit:
```

Test	Result
testGetFindingsBlockNoFindings()	✅ PASSED
testGetFindingsBlockNoSuccesses()	✅ PASSED
testGetFindingsBlock()	❌ FAILED

```
  The failure:
  java.util.MissingFormatArgumentException: Format specifier '%s'
      at FindingsContentTest.java:44
```

Wie wird meine QA denn effektiver?

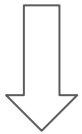
“Closing the Loop”

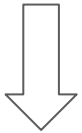
Beispiel

Review:
REST Endpoints müssen mit einer “Visibility-Annotation” versehen sein.

Beispiel

Review:
REST Endpoints müssen mit einer "Visibility-Annotation" versehen sein.





AGENTS.md/CLAUDE.md

(Wenn noch nicht "zu voll")

AI Review Instructions

Beispiel

"Teuer"

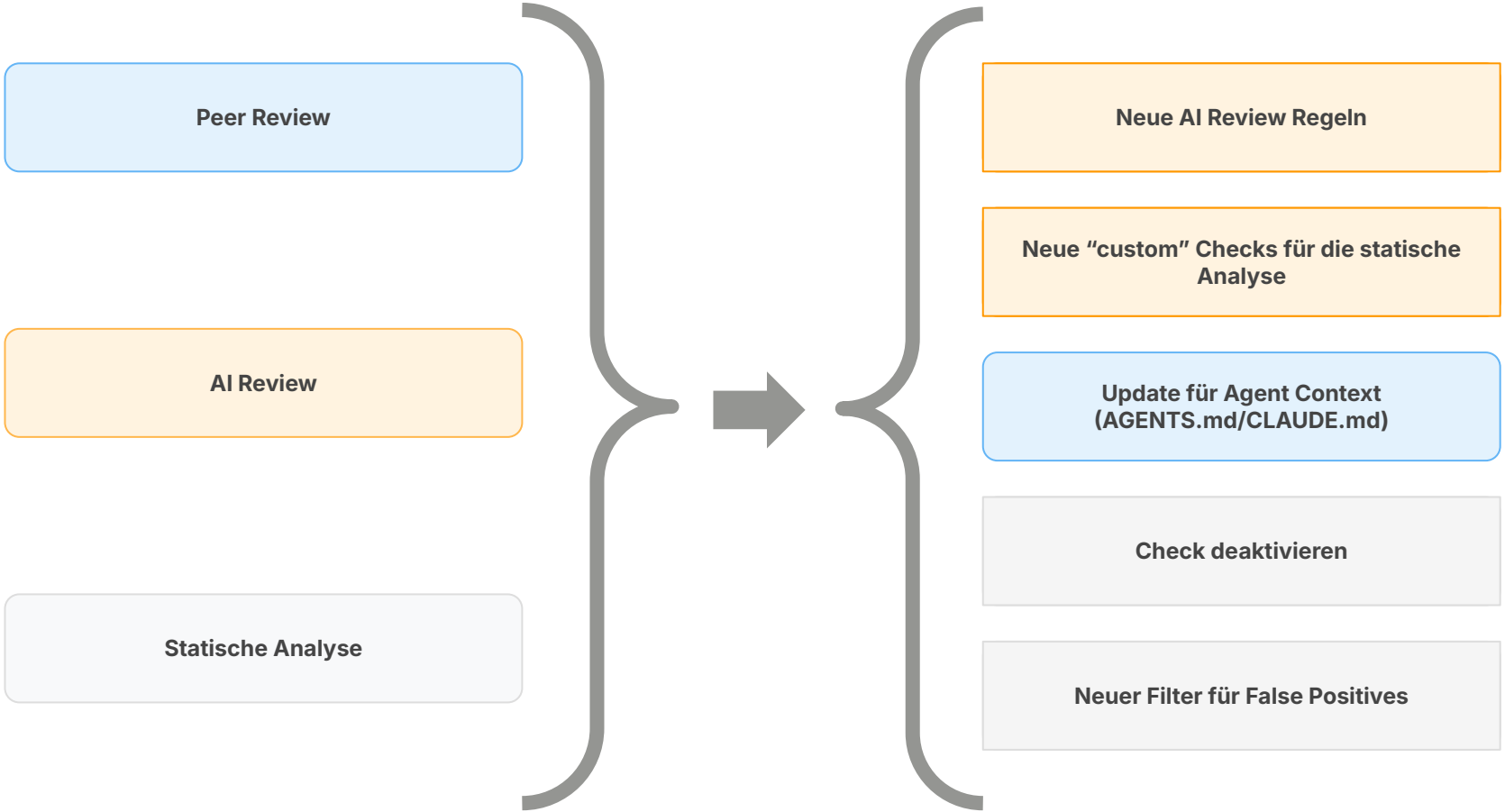
Nicht deterministisch

Deterministischer Check

```
public void execute() throws CheckException {
    List<ShallowEntity> ast = context.getAbstractSyntaxTree(getCodeViewOption());
    ShallowEntityTraversalUtils.listEntitiesOfType(ast, EShallowEntityType.METHOD)
            .forEach(this::checkAttributesOfMethod);
}

private void checkAttributesOfMethod(ShallowEntity method) {  1 usage   Yannick Kraml *
    if (!LanguageFeatureParser.JAVA.hasAnnotation(method,  ...annotations: "GET", "POST", "PI
        return;
    }
```

AI Review Instructions


Peer Review
AI Review
Statische Analyse
Neue AI Review Regeln
Neue “custom” Checks für die statische Analyse
Update für Agent Context (AGENTS.md/CLAUDE.md)
Check deaktivieren
Neuer Filter für False Positives

Warum wird das erst jetzt relevant?

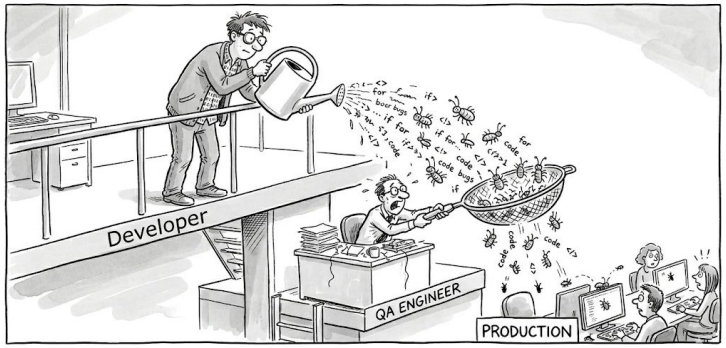
- Es gab schon immer die Möglichkeit, eigene Analyse-Regeln zu schreiben
  - Bestehende Regeln oft nicht anpassbar genug
  - Schreiben eigener Regeln oft zu aufwändig/zu teuer
  - Oft benötigtes Wissen dafür im Team nicht vorhanden

- LLMs/Agenten senken diese Schwelle
  - Checks bestehen oft nur aus wenigen Zeilen Code
  - Fehlendes Wissen durch Skills ergänzbar
  - LLMs können wiederkehrende Muster erkennen
  - Erstellungskosten im Cent-Bereich

Fazit

Meine Empfehlungen

- LLMs/Agenten als Ergänzung für die QA einsetzen
  - Qualität der Ergebnisse beobachten
  - Kosten berücksichtigen
  - "Human in the Loop"
- Agenten brauchen klare Führung
  - Projekt-Kontext (AGENTS.md/CLAUDE.md) wird nicht immer befolgt
  - Möglichst viel Qualitätssicherung in den "Harness" packen
  - Geschwindigkeit ist wichtig
- Closing the Loop
  - Ableiten von neuen Regeln/Prüfungen aus Feedback während der QA
  - Verschieben von "weichen" KI-Regeln in deterministische Checks wo möglich


Developer
QA ENGINEER
PRODUCTION


You can trust these; they filter out all unwanted elements!