

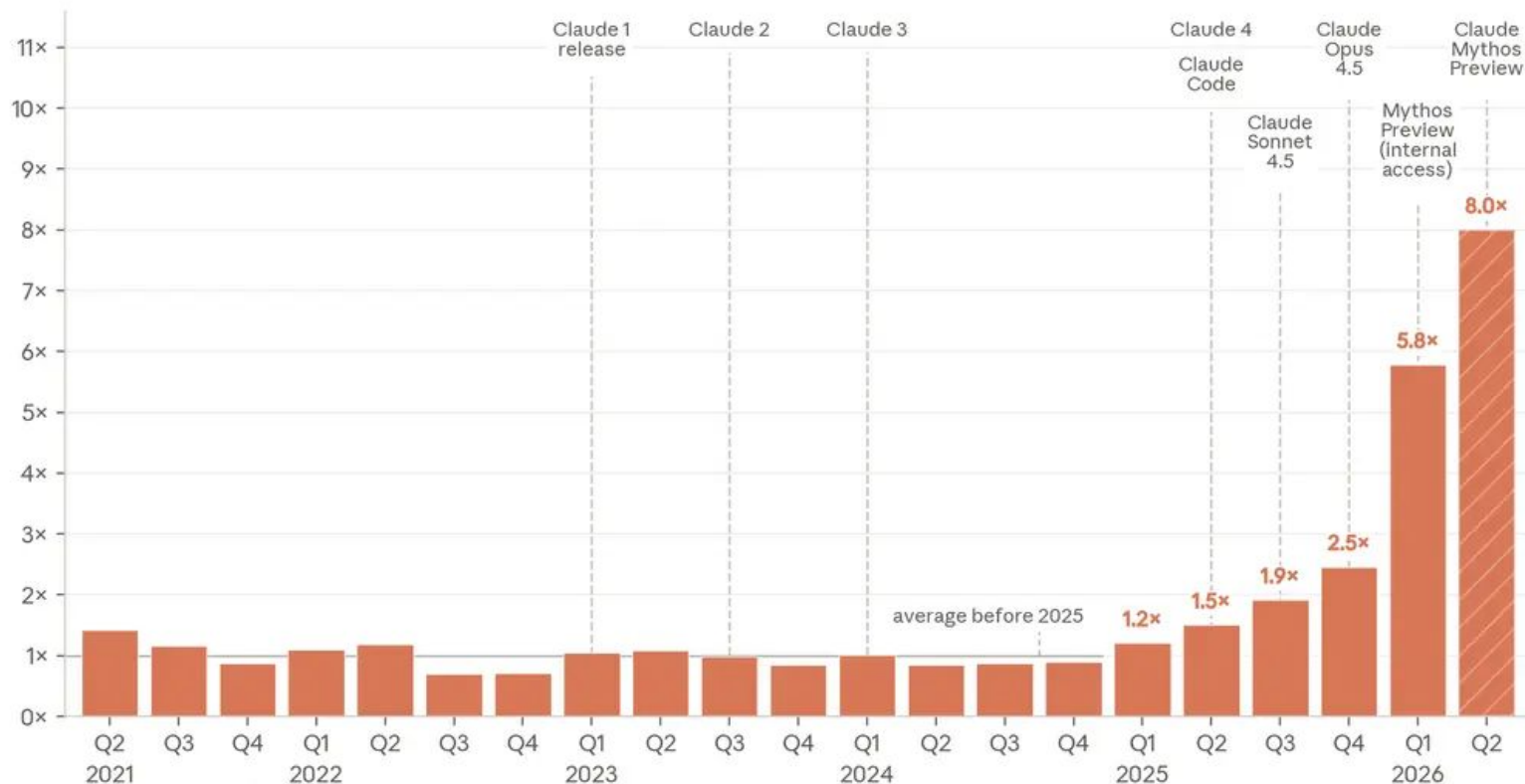
Copiloten für Agenten

Qualitätssicherung in der Ära der Coding Agents



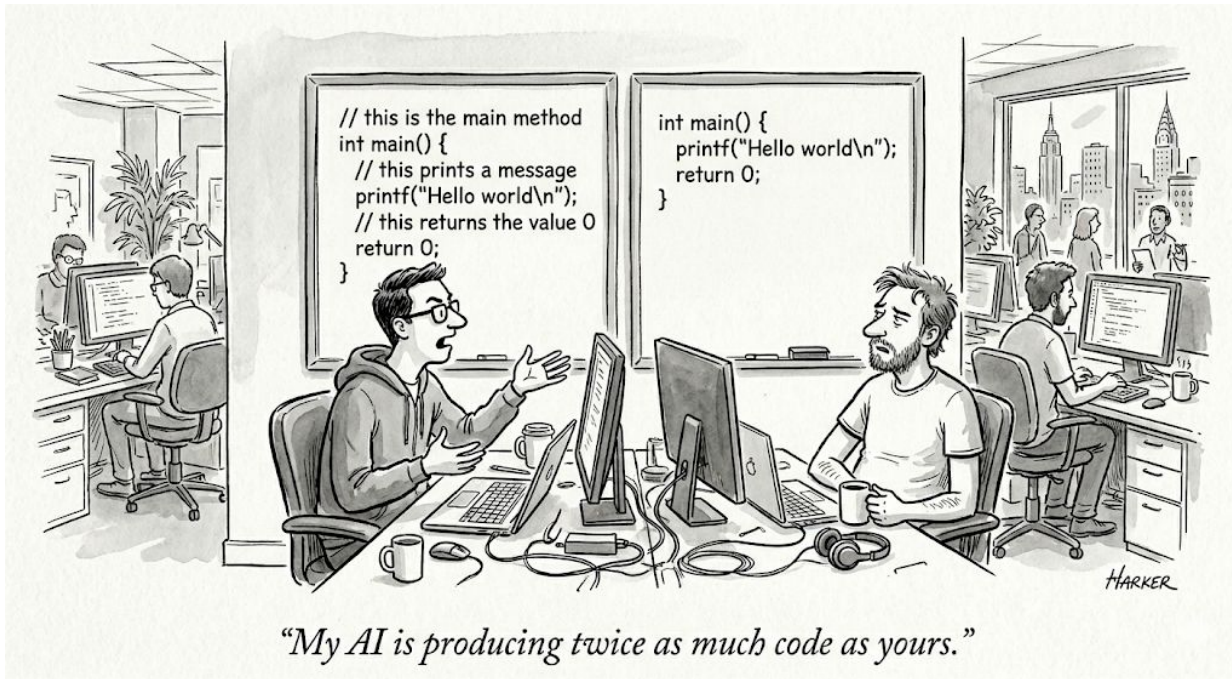
Dr. Benjamin Hummel

Code contributed per person, by quarter



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.

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





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
```



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.

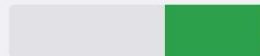
OUTPUT IS UP. NOT ALL OF IT STICKS.

Impacts of high AI adoption on engineering throughput



Epic

A sizable feature or capability, hundreds of tasks



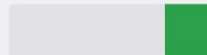
+66.2%

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

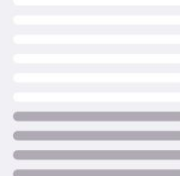


+59.7%

files edited per PR

AI ADOPTION

LOW



HIGH



+51.3%

average PR size

DEVELOPER
FOOTPRINT/MONTH



+149.9%

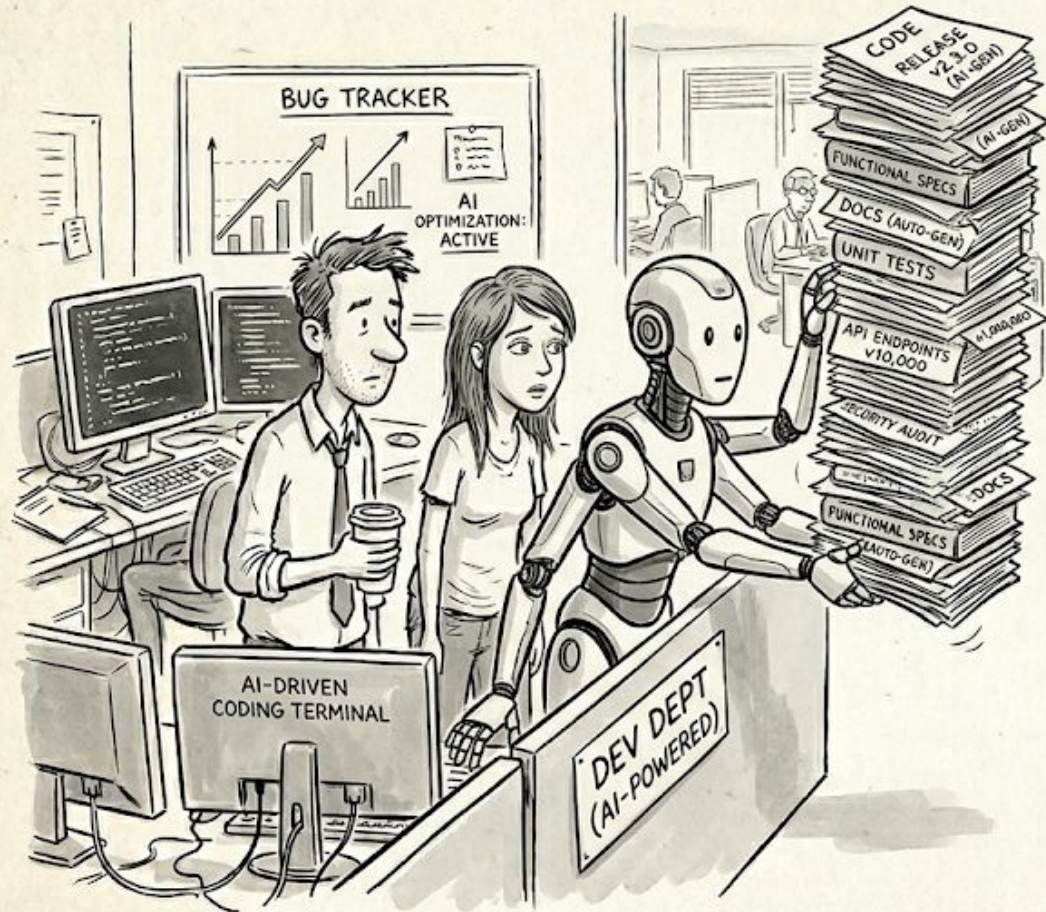
files touched per developer



+11.7%

repositories touched per developer





Heaven

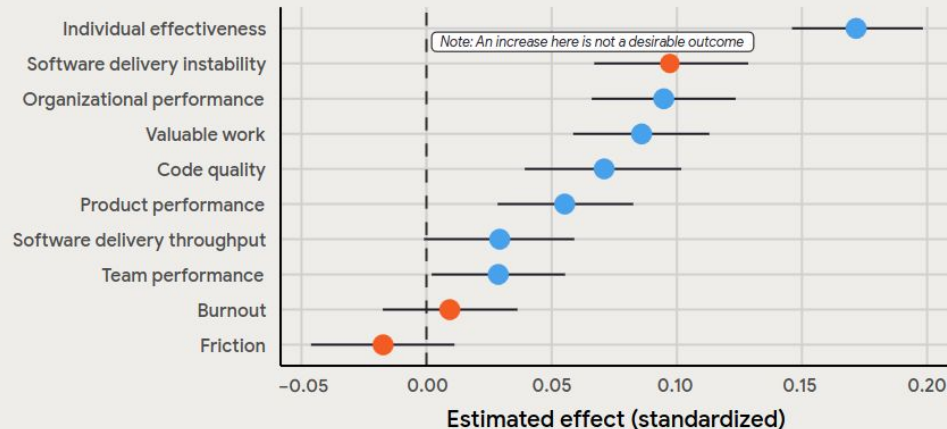
The ROI of AI-assisted Software Development

Google Cloud



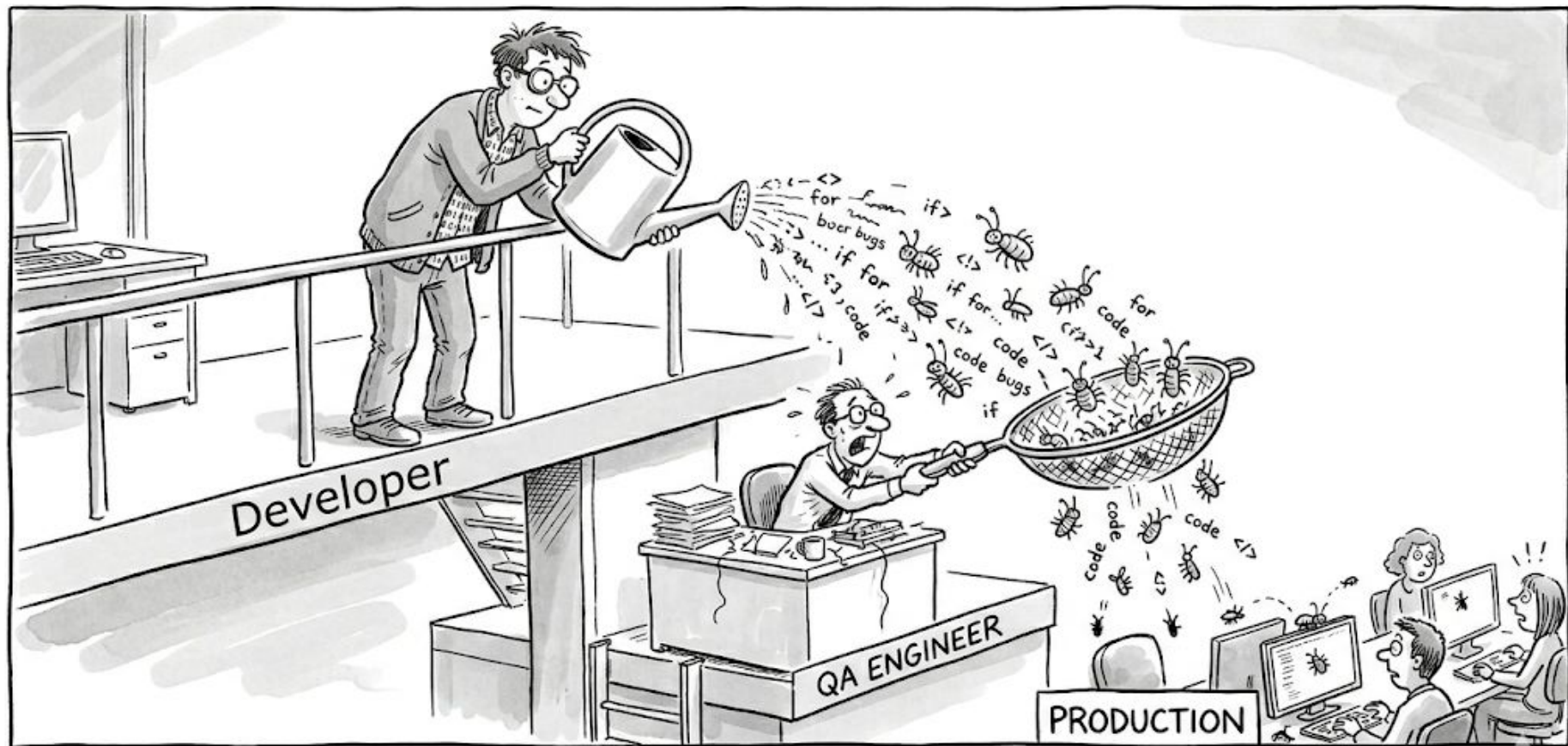
The landscape of AI's impact

Estimated effect of AI adoption on key outcomes, with 89% credible intervals



For outcomes in orange (such as burnout), a negative effect is desirable.

Figure 4: Standardized estimated effects of AI adoption



Entwicklung (Menschen & KI)

Änderungen

Bugs

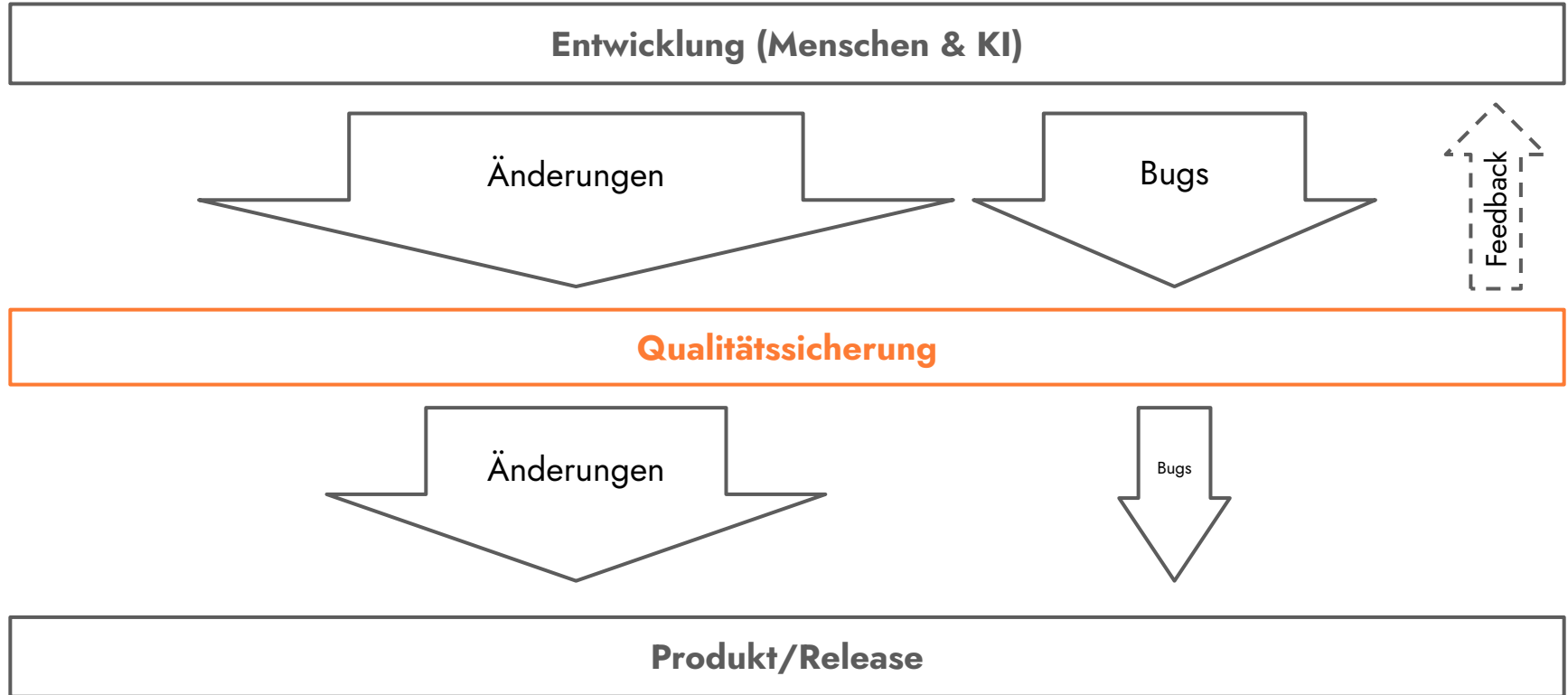
Feedback

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**?



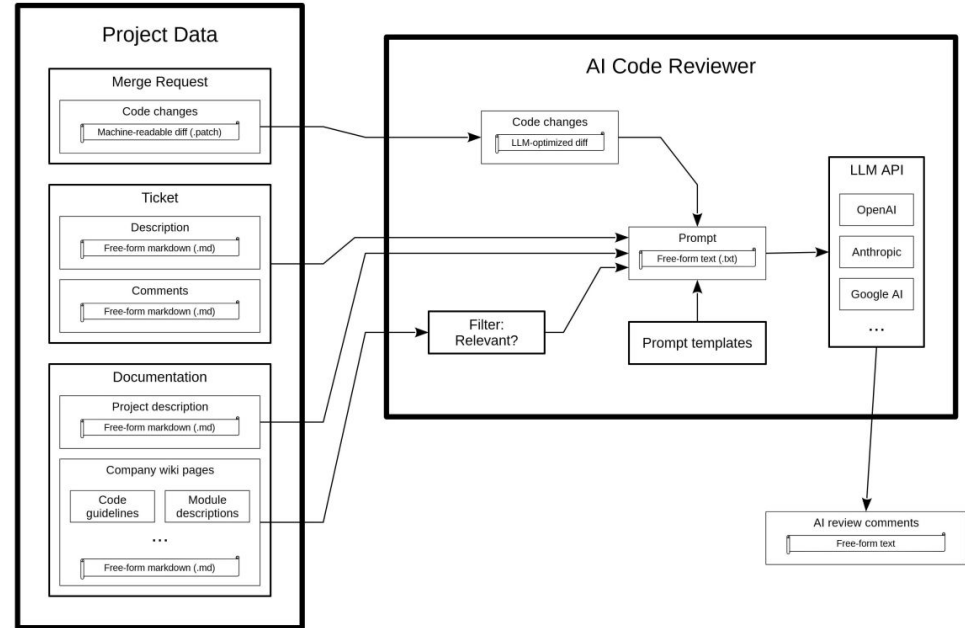
SCHOOL OF COMPUTATION, INFORMATION
AND TECHNOLOGY — INFORMATICS

TECHNISCHE UNIVERSITÄT MÜNCHEN

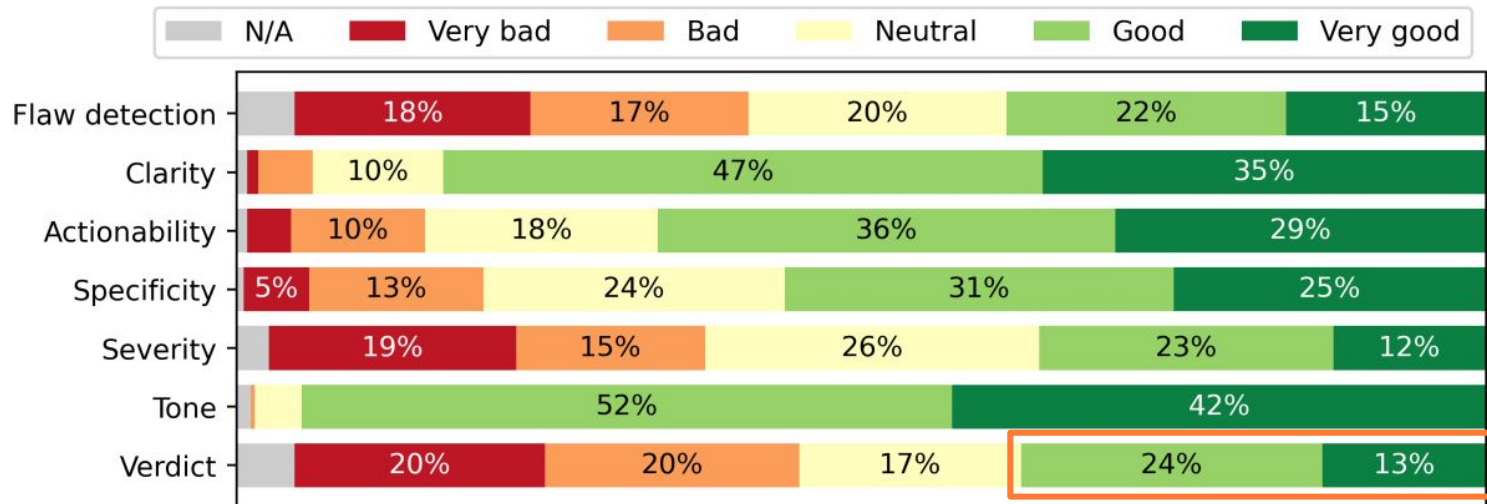
Bachelor's Thesis in Informatics

Using AI for Generating Code Review Comments

Dominik Weißenseel



“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

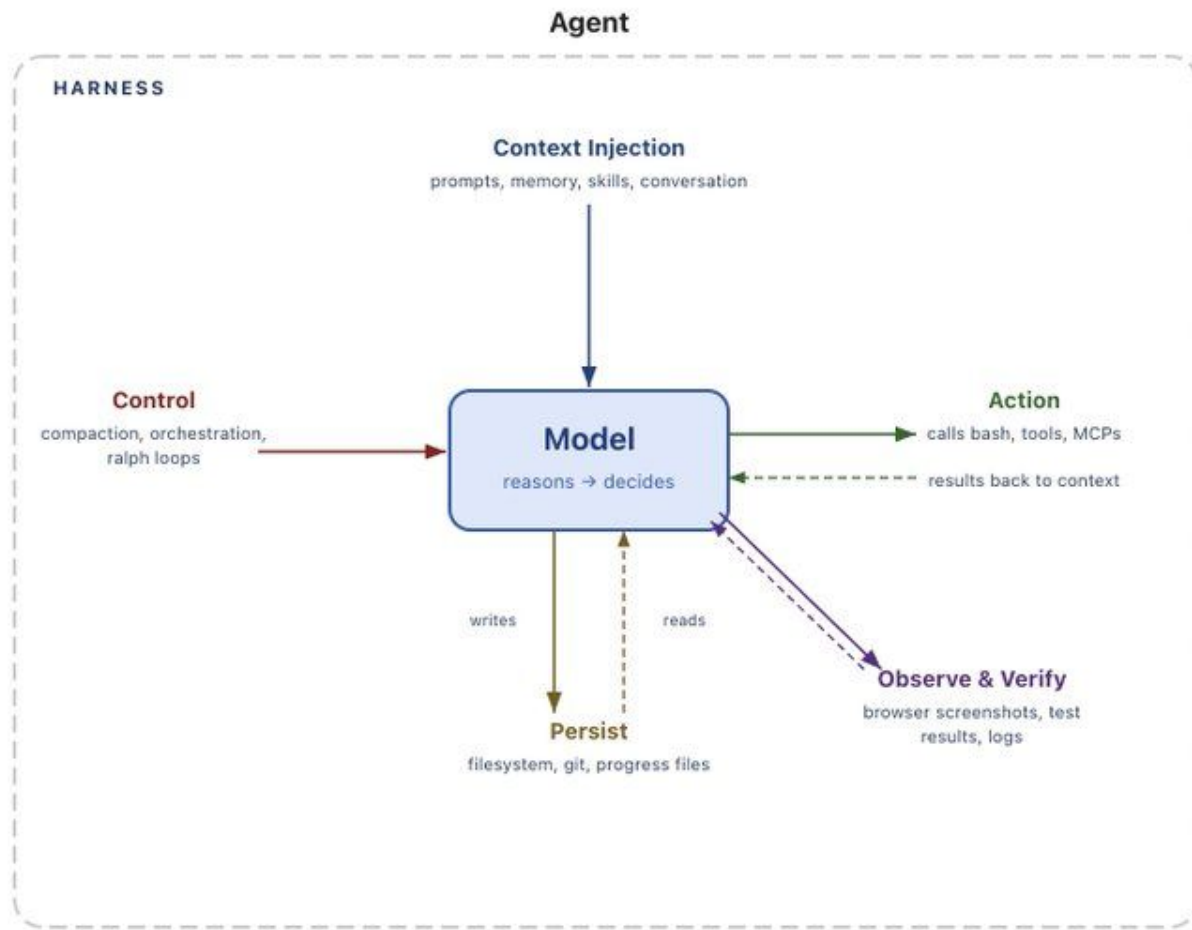
Ergebnis nach der Einführung eines führenden “AI Review Tools” bei einem großen Konzern (mehrere 1000 Entwickler:innen) begleitet durch den Tool-Hersteller über einen Zeitraum von über einem Jahr:

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**?



Prompt 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*

Context Engineering

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*

Harness Engineering

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*

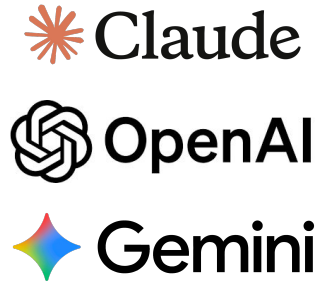


Governance & Security

*Enforce permissions, approvals
and audit rules*

Qualitätssicherung

LLMs



Tools

Read/Write
Shell
MCP
etc.

```

Welcome to Claude Code!
/help for help, /status for your current setup
cd: /home/jeff/git/pdash

What's new:
• Fix UI bug during typing, especially noticeable with large prompts
• Fix Windows issue where process visually freezes on entering interactive mode
• Support dynamic headers for MCP servers, via header/mcp_server_configuration
• Fix thinking mode not working in headless sessions
• Fix slash commands now properly update allowed tools instead of replacing them

> refactor ./src/cloud_time_log_syncronizer.java

● I'll help you refactor the ./src/cloud_time_log_syncronizer.java file. Let me first locate and examine the
current implementation to understand what needs to be refactored.

Search pattern: "**/src/cloud_time_log_syncronizer.java"
└─ Found 1 file (ctrl+q to expand)

Read: ./src/cloud_time_log_syncronizer.java
└─ Read 363 lines (ctrl+q to expand)

● Now let me create a todo list to plan the refactoring of this class:

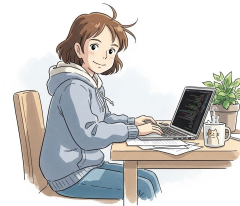
● After analyzing the code, I've identified several refactoring opportunities:

1. Large monolithic class - The class has multiple responsibilities
2. Magic strings and numbers - Hardcoded values like "cloud_time_log_syncronizer"
3. Long methods - runMonitoring() and syncDataFromGithub() are too long
4. Mixed concerns - Time parsing, configuration, business logic all mixed
5. Poor error handling - System.out.println() instead of proper logging
6. Duplicate code - Similar patterns for settings management

Extracting configuration handling into separate class. (esc to interrupt - ctrl+q to show todos)
└─ Next: Create separate service classes for work log operations

> For shortcuts

```



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

Schnelles Feedback trotz langsamer Tests

Testselektion für historisch gewachsene Test-Suites

Mittwoch, 06. März, 2024
10:30 bis 12:00 Uhr
online und kostenlos



Raphael Nämmer



Stefan Brand

CQSE Workshop

Test-Gap-Analyse

Ungetestete Änderungen im Quelltext aufdecken

Mittwoch, 10. April, 2024
10:30 bis 12:00 Uhr
online und kostenlos



Fabian Streitel



Johannes Veihelmann

CQSE Workshop

Continuous Quality Control

Qualität trotz immer kürzerer Releasezyklen

Dienstag, 12. März, 2024
10:30 bis 12:00 Uhr
online und kostenlos



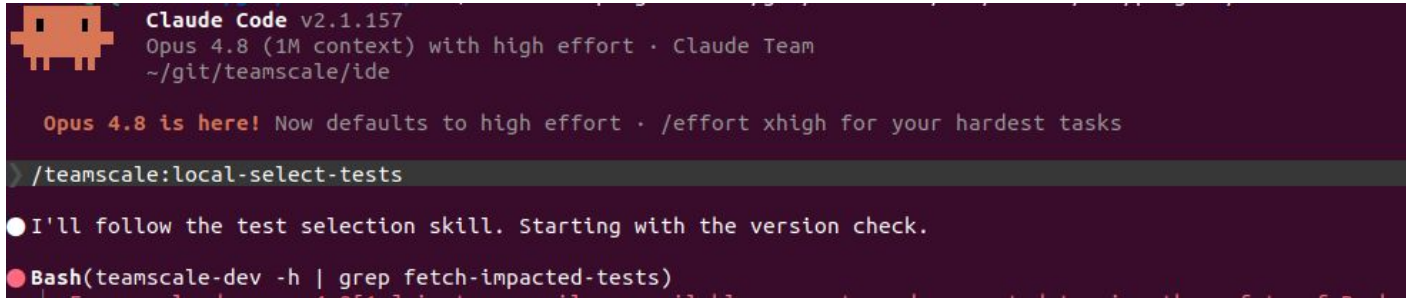
Dr. Tobias Röhm



Tobias Wiese

CQSE Workshop

Beispiel: Test-Selektion in Claude Code



The image shows a screenshot of the Claude Code interface. At the top left is the Claude Code logo, a small orange robot head. To its right, the text reads: "Claude Code v2.1.157", "Opus 4.8 (1M context) with high effort · Claude Team", and "~/git/teamscale/ide". Below this, a message states: "Opus 4.8 is here! Now defaults to high effort · /effort xhigh for your hardest tasks". A dark grey command bar contains the text "/teamscale:local-select-tests". Below the bar, a white dot indicates a response: "I'll follow the test selection skill. Starting with the version check." At the bottom, a red dot indicates a terminal command: "Bash(teamscale-dev -h | grep fetch-impacted-tests)".

```
Claude Code v2.1.157
Opus 4.8 (1M context) with high effort · Claude Team
~/git/teamscale/ide

Opus 4.8 is here! Now defaults to high effort · /effort xhigh for your hardest tasks

/teamscale:local-select-tests

● I'll follow the test selection skill. Starting with the version check.

● Bash(teamscale-dev -h | grep fetch-impacted-tests)
```

Beispiel: Test-Selektion in Claude Code

```
● teamscale-dev supports fetch-impacted-tests. Now I'll determine paths (defaulting to repo root) and run the pre-commit analysis to upload the local uncommitted changes.

● Now running pre-commit analysis on the repo root with uncommitted changes only.

  Called plugin:teamscale:teamscale (ctrl+o to expand)

● Pre-commit upload succeeded. Now fetching impacted tests from the pre-commit branch using the COVERAGE strategy.

  Called plugin:teamscale:teamscale (ctrl+o to expand)

● Teamscale identified 3 tests most impacted by your uncommitted changes (sorted by likely impact, COVERAGE strategy):

  1. FindingsContentTest#testGetFindingsBlockNoFindings()
  2. FindingsContentTest#testGetFindingsBlockNoSuccesses()
  3. FindingsContentTest#testGetFindingsBlock()

  All three live in com.teamscale.ide.cli.findings.mcp.FindingsContentTest – consistent with your working-tree edit to cli/src/main/java/com/teamscale/ide/cli/findings/mcp/FindingsContent.java.

  You can run just this focused subset instead of the full suite. Want me to run them?

% Canceled from IDE
```

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.

```
@Path("api/projects/{project}/ai/content") @ Benjamin H
public class TokenElementAiService extends ApiBase {

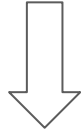
    @Inject 1 usage
    private IAIModelProvider aiModelProvi

    @GET @ Benjamin Hummel +3
    @Operation(summary = "Get a generated summary for an element", description = "Retrieves the AI summary for an element.", tags = {
        Tags.SOURCE_CODE, Tags.AI })
    @Path("summary")
    public String getAiSummary(
        @Parameter(description = TIMESTAMP_PARAMETER_DESCRIPTION) @QueryParam(TIMESTAMP_PARAMETER_NAME) UnresolvedCommitDescriptor commi
        @Parameter(description = "Uniform path to obtain the token element info for", required = true, allowEmptyValue = true) @QueryParam
        @Context HttpServletRequest servletRequest)
        throws StorageException, ServiceCallException, OidcStepUpAuthenticationRequiredException {
        TokenElementInfo element = TokenElementServiceUtils.retrieveElement(uniformPath, getProjectStorageSystem(),
            determineHistoryOption(commit));
        if (element == null) {
            throw new NotFoundException();
        }
    }
}
```

Beispiel

Review:

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



AGENTS.md/CLAUDE.md

(Wenn noch nicht "zu voll")

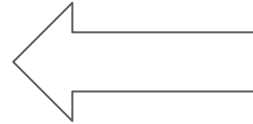


AI Review Instructions

Beispiel

Deterministischer Check

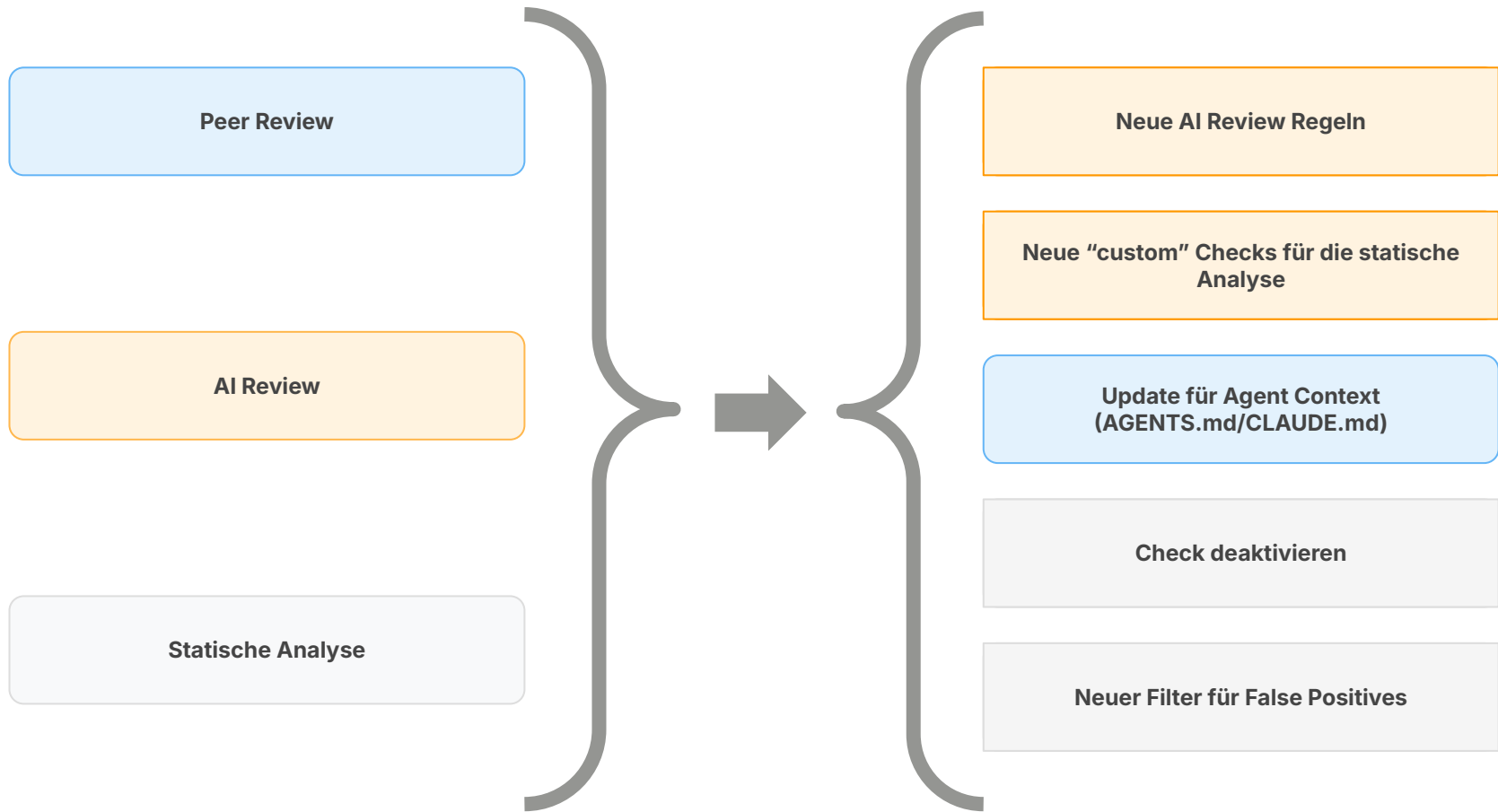
```
public void execute() throws CheckException {  
    List<ShallowEntity> ast = context.getAbstractSyntaxTree(getCodeViewOption());  
    ShallowEntityTraverserUtil.listEntitiesOfType(ast, EShallowEntityType.METHOD)  
        .forEach(this::checkAttributesOfMethod);  
}  
  
private void checkAttributesOfMethod(ShallowEntity method) {  
    // 1 usage @ Yannick Kraml  
    if (!LanguageFeatureParser.JAVA.hasAnnotation(method, "annotations: \"GET\", \"POST\", \"PI\""))  
        return;  
}
```



“Teuer”

Nicht deterministisch

AI Review Instructions



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

