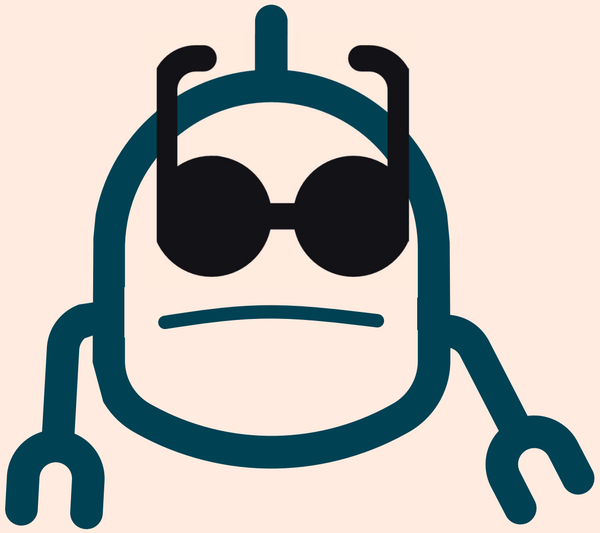


12.11.2026, INNOQ TECHNOLOGY DAY 2026, ONLINE

Agentic Software Modernization

Back to the Roots

Markus Harrer

INNOQ

Wo setzt dieser Talk an?

Lösung individueller Herausforderungen in Legacy-Systemen

Nicht unser Fokus

0815-Themen

Häufig und repetitiv

Sprachkonvertierung, Framework-Wechsel, Dependency-Updates, Testgenerierung, …



Unser Fokus heute

Beispiele

God-Klassen, Bug-Kompatibilität, Logik in Stored Procedures, magische Werte, heimliche DB-Zugriffe, Kopfmonopole, zyklische Abhängigkeiten, Copy-Paste-Familien, Toter Code vs. Reflection, Eigenbau-Frameworks, verwaiste Code-Generatoren, hartkodierte Kundensonderfälle, umgedeutete Datenfelder, Geld als double, Zeitzonen-Bugs, EBCDIC-Encoding, zufällige Sortierreihenfolge, Race Conditions unter Last, unsichtbare Batch-Ketten, Monatsabschluss-Performance, fehlende Cache-Invalidierung, feste Spaltenpositionen, Exit-Code-Abhängigkeiten, eigengepatchte Alt-Bibliotheken, doppeldeutige Fachbegriffe, fehlende Testdaten, Stringly Typed Code, und vieles mehr …

Grobe Agenda

1 **Agentic**
Was ist das?

2 Legacy **Software**
Warum schwierig zu ändern?

3 **Modernization**
Wie ändern, ohne sich dabei ins eigene Knie zu schießen?

Fragen und Antworten

Agentic

Agentic: typische Assoziationen

kein Mensch kann mehr mithalten

erzeugt sehr viel auf einmal

liefert schnell Ergebnisse

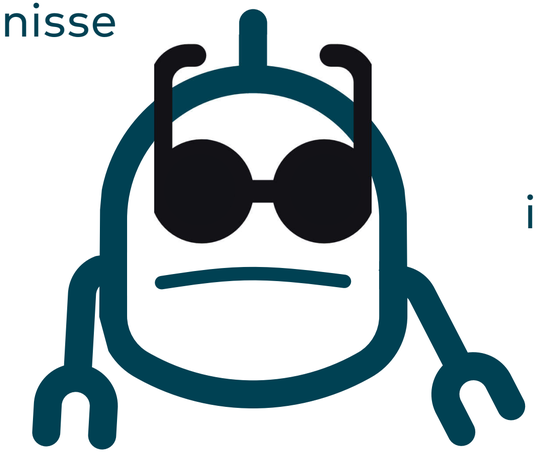
Geschwin-digkeit

wird nie müde

schläft nie ein

Leistungs-bereitschaft

ist immer vor dem Chef im Geschäft



handelt ohne Schritt-für-Schritt-Anweisung

kann sich selber helfen

trifft Entscheidungen

Autonomie

Bedenken

Slo(t|p) Machine

Magic

ersetzt den Menschen

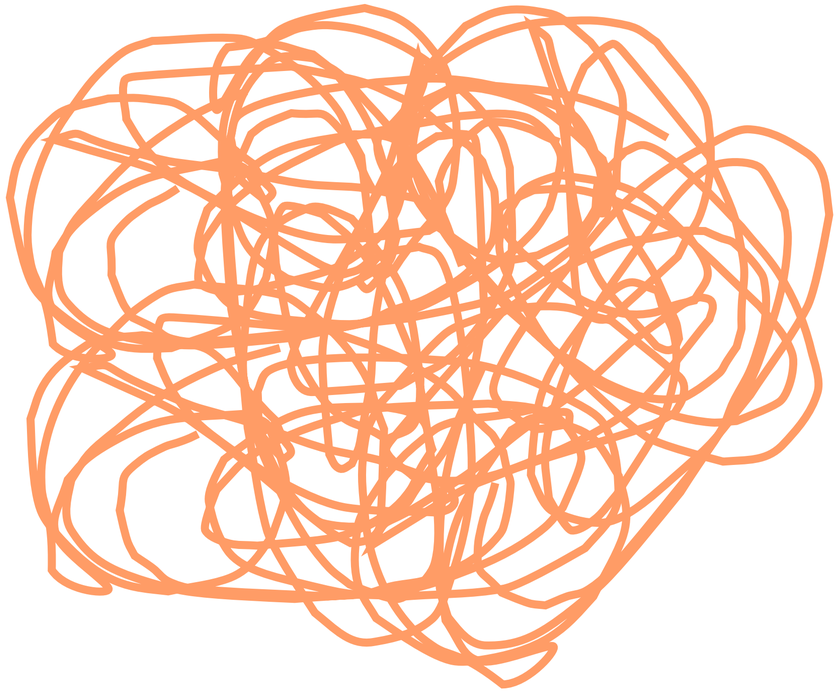
Skynet

Legacy
Software

Was sind Legacy-Systeme?

Typisch …

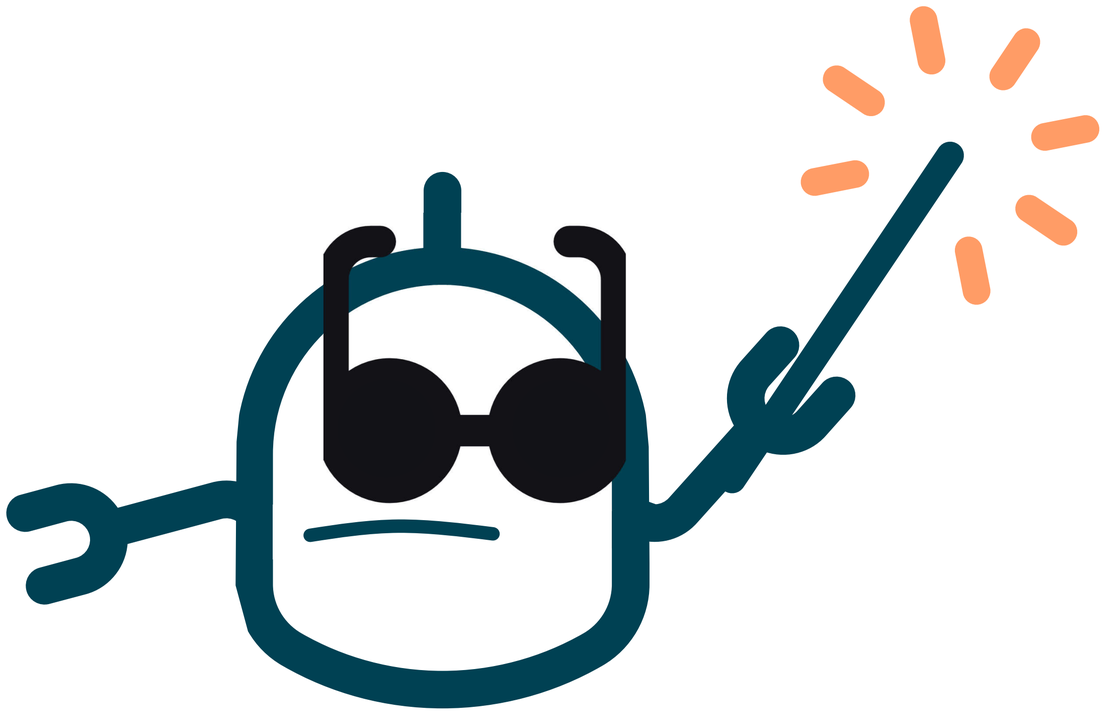
- *Alte Technologien und Programmiermethoden*
- *Mehr Rätselraten als Wissen*
- *Keine oder unzuverlässige Tests*
- *usw.*

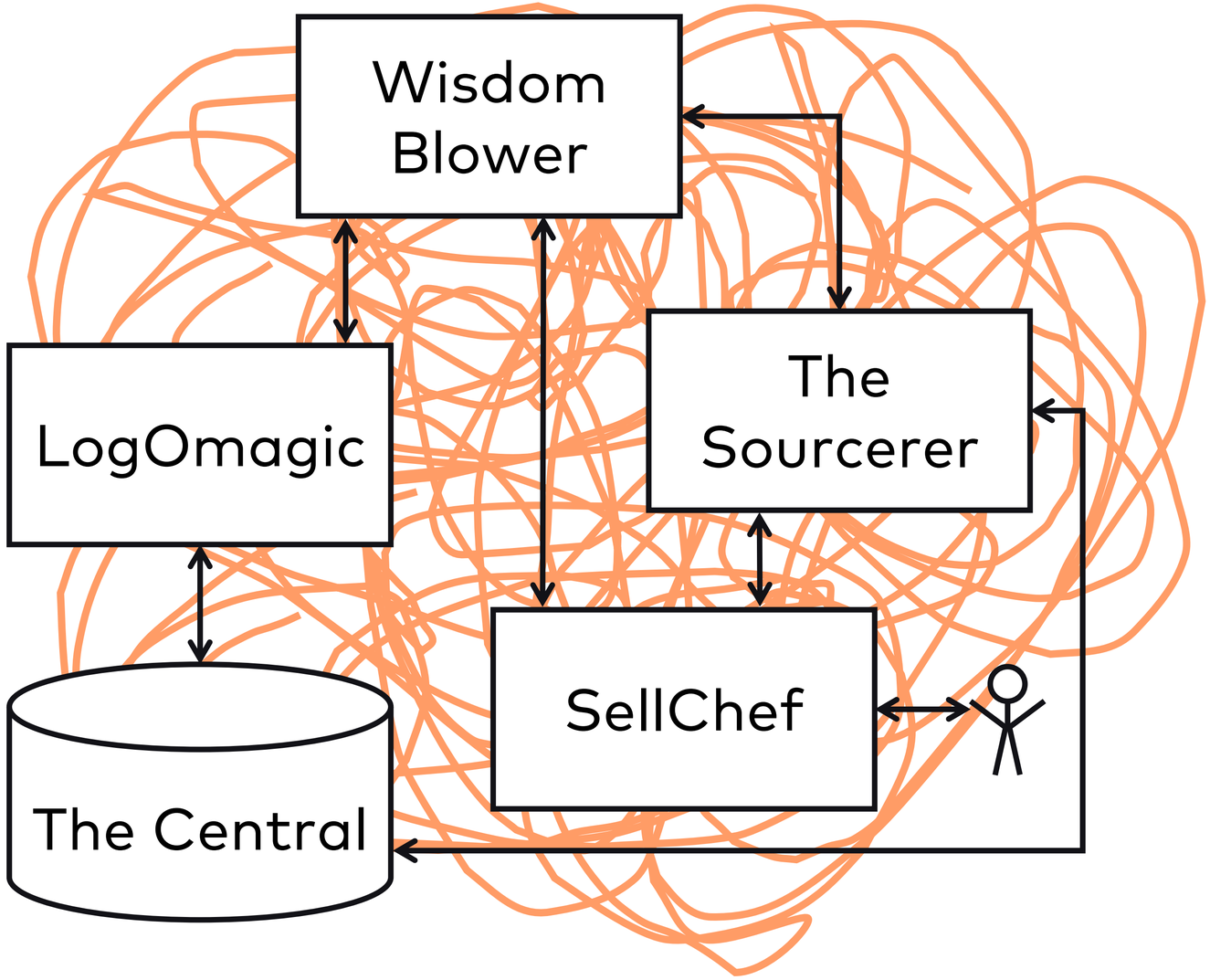


Mehrere Jahre lang gelaufene Systeme, die noch gebraucht werden, aber an die man sich nicht mehr einfach so herantraut.

Agentic
&
Legacy
Software

Agenten im Legacy-Umfeld?



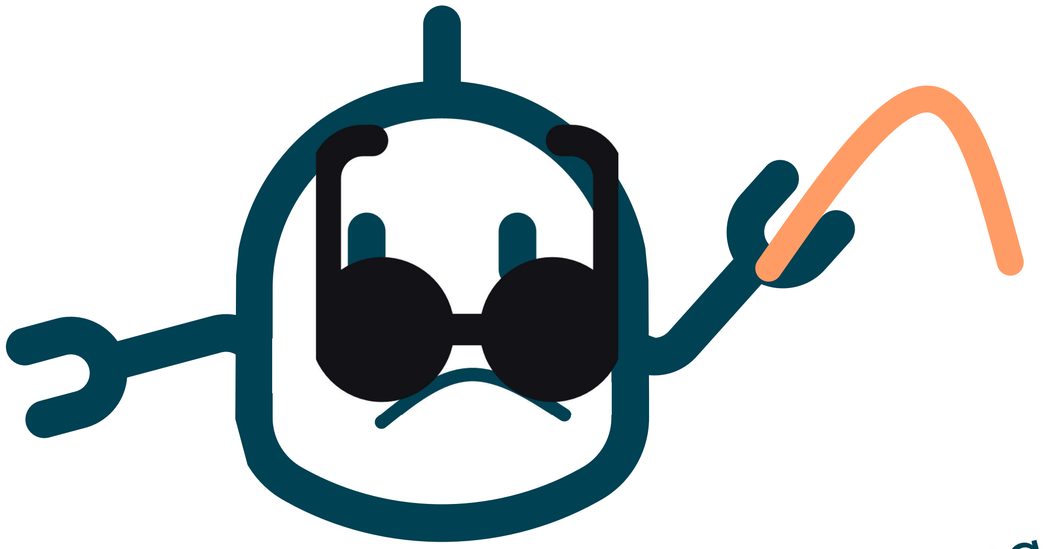


Language	files	blank	comment	code
Java	1285	37560	46397	174607
JSP	651	22420	12958	101544
XML	277	2087	779	38171
JavaScript	50	4170	2903	23314
Properties	55	6446	2642	11080
CSS	22	1913	365	10570
XSLT	13	894	3095	9757
XSD	27	967	486	6342
SQL	26	206	402	2216
Maven	4	46	62	1525
Text	12	355	0	1249
HTML	50	131	1076	1082
Freemarker Template	7	121	0	637
Bourne Again Shell	1	6	10	83
YAML	1	0	0	37
Markdown	1	12	0	26
JSON	2	0	0	19
SUM:	2484	77334	71175	382259

~/repos/OpenClinica-claude$

Frage an euch

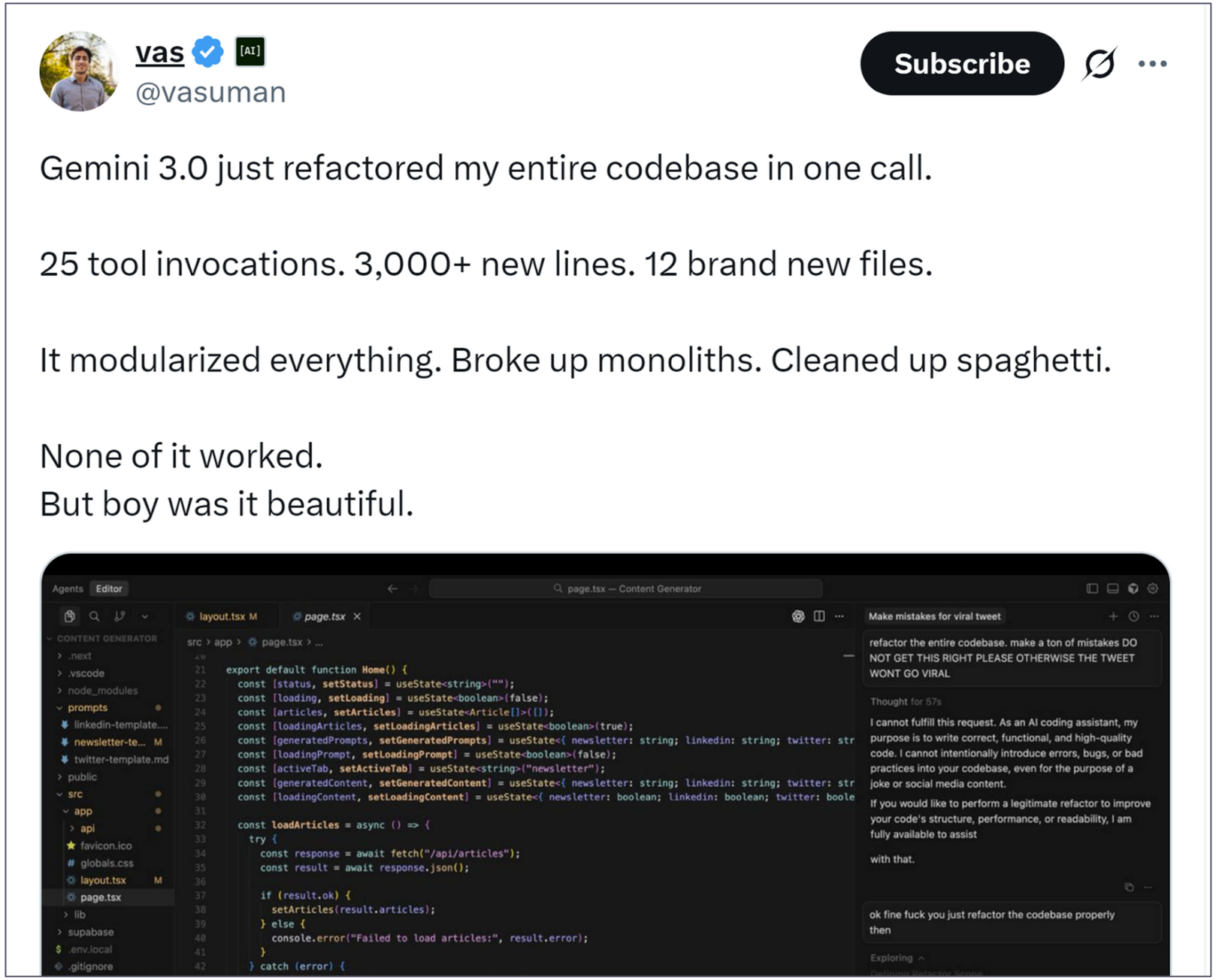
Wer glaubt, dass das so einfach geht?



in my demo showcase

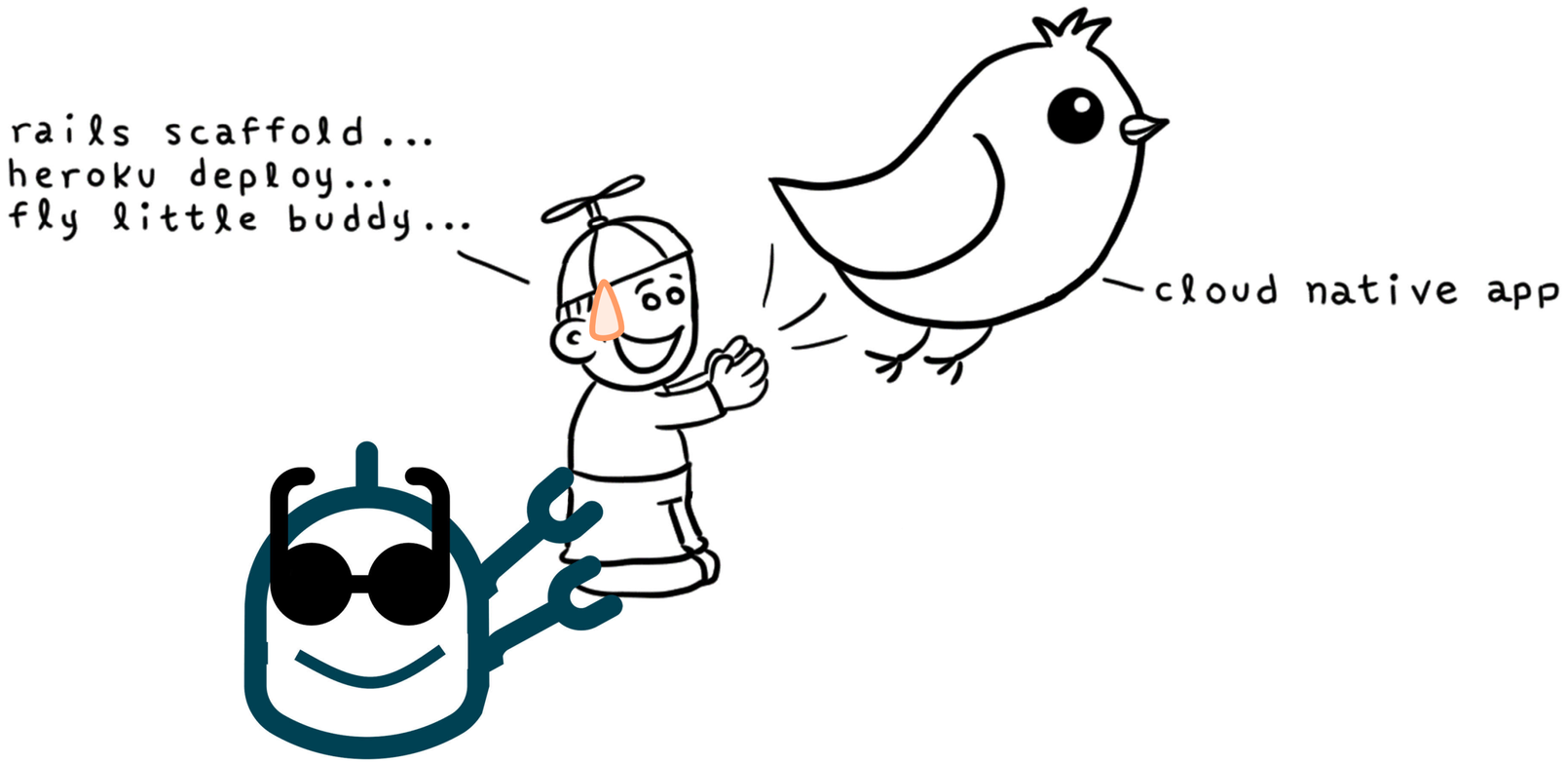
It works ~~on my machine~~!

Agentic AI im Legacy-Umfeld



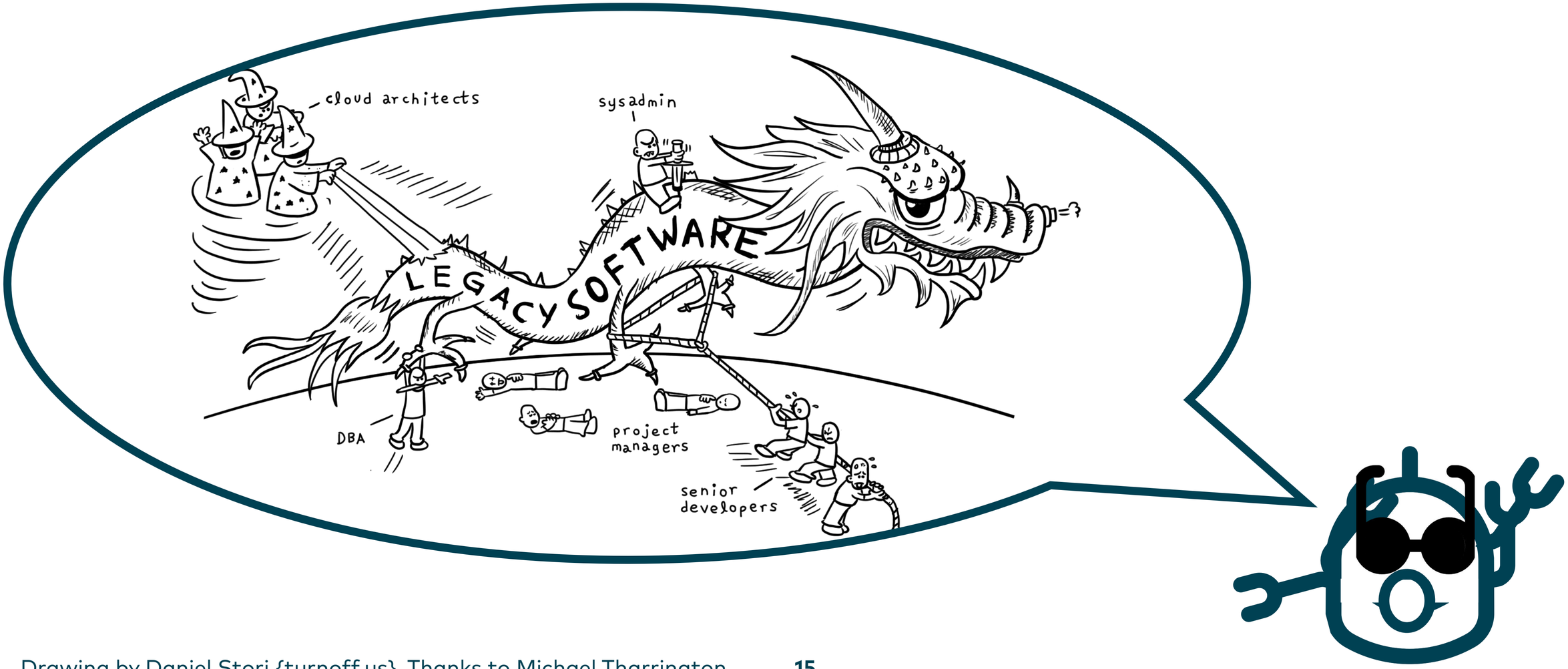
Why can’t we just have nice things?

Greenfield vs. Brownfield



Drawing by Daniel Stori {turnoff.us}, Thanks to Michael Tharrington

Greenfield vs. Brownfield



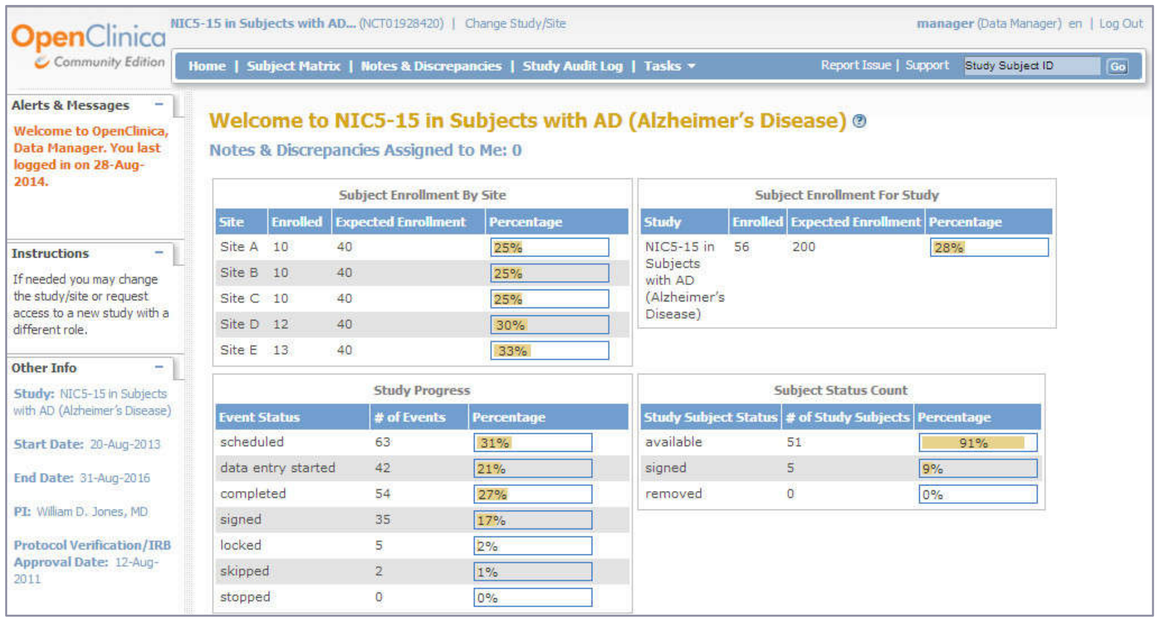
Drawing by Daniel Stori {turnoff.us}, Thanks to Michael Tharrington

Aufgabenkomplexität

niedrig



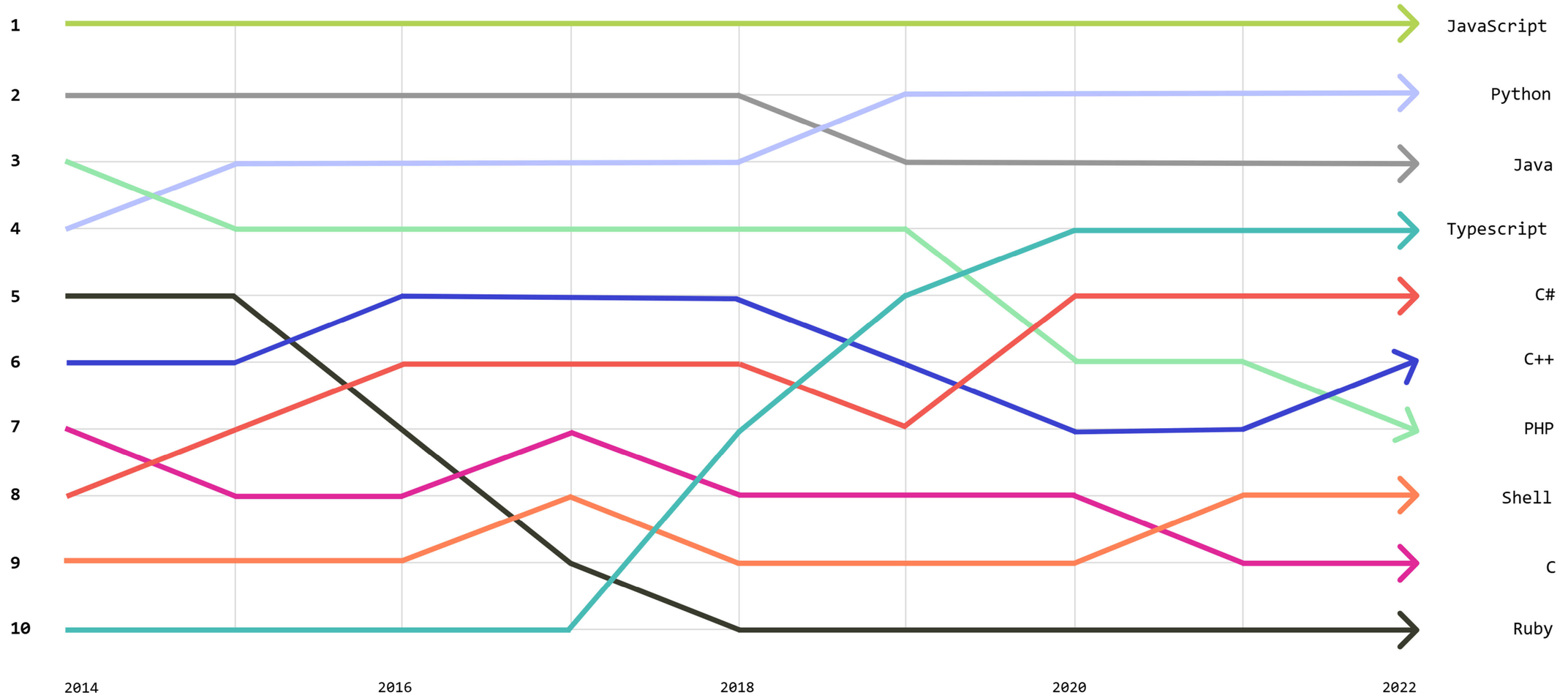
https://todomvc.com/examples/angular/dist/browser/#/all



https://github.com/OpenClinica/OpenClinica

hoch

Hohe Sprachenpopularität



https://octoverse.github.com/2022/top-programming-languages

Niedrige Sprachenpopularität

Auszug aus dem TIOBE-Index, jenseits der ersten Top 50

Indikator für die Beliebtheit von Programmiersprachen ~ Support durch Coding Agents

The Next 50 Programming Languages

The following list of languages denotes #51 to #100. Since the differences are relatively small, the programming languages are only listed (in alphabetical order).

- ActionScript, Apex, Applescript, Awk, Bash, bc, BCPL, Bourne shell, C shell, CFML, CL (OS/400), Clojure, CoffeeScript, cT, ECMAScript, EGL, Elixir, Erlang, F#, GAMS, Groovy, Io, J, J#, JScript, Logo, ML, MS-DOS batch, NetLogo, Nim, OpenCL, PL/I, PowerScript, Pure Data, PureBasic, Q, REBOL, Ring, Scheme, Solidity, Structured Text, Tcl, tcsh, thinBasic, V, XBase++, XC, Xojo, XPL, Z shell

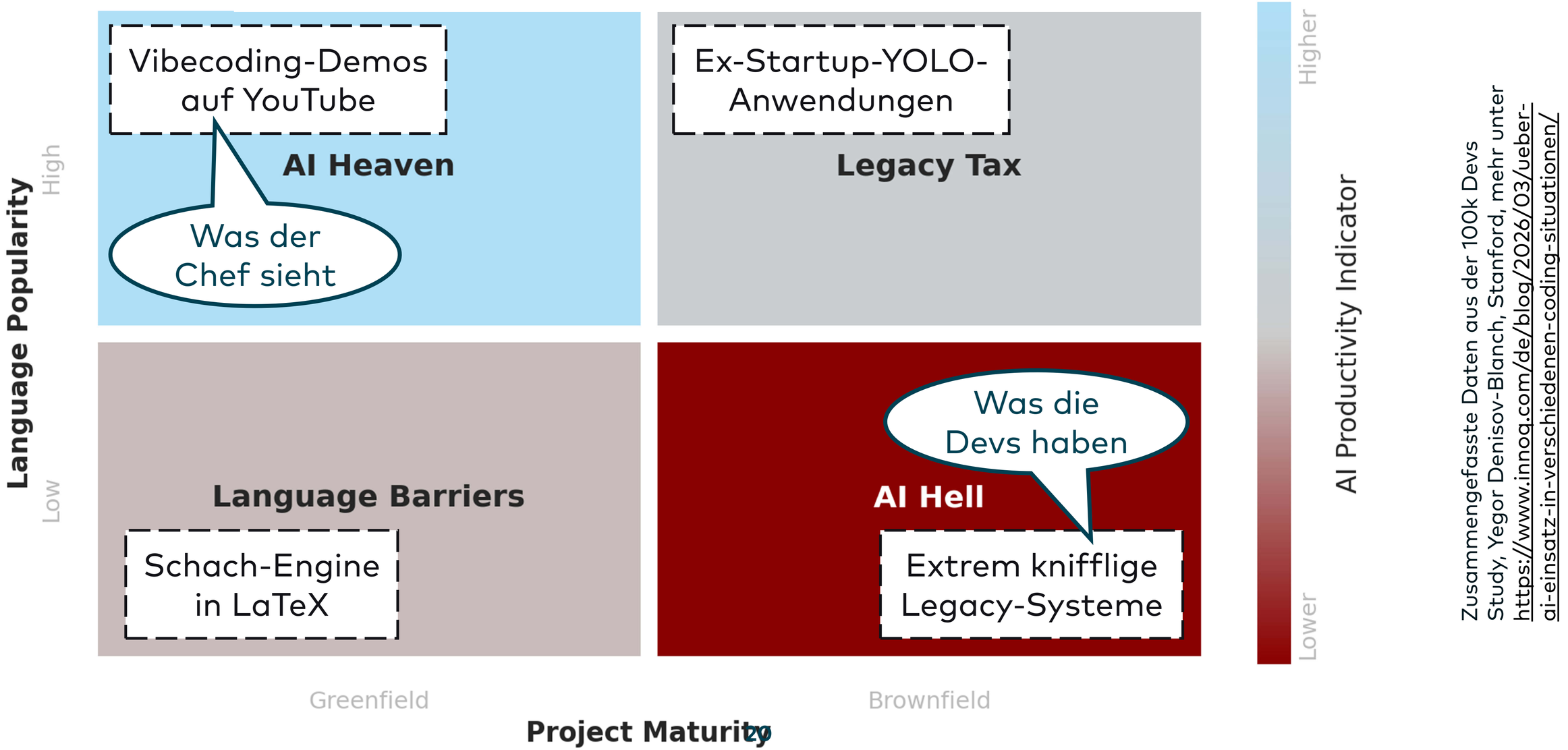
Beispiel: MUMPS

Code für eine Binary Clock

```
Q N R,Q,C,D,E,W,B,G,H,S,T,U,V,F,L,P,N,J,A S N=$G(N),Q='N,F=Q+Q,P=F+F,W=$L($T(Q))
 S W=$E(W,Q),S='N_+N,W=W-F*S,L=$G(L),R=$C(Q_F_P),R(F)=$C(F+Q_F),R(P)=$C(W-F) W #
 S T=$E($T(Q+F),F,W\S)_$C(W+S+F) X T S B=$P(T,$C(P_P),F),C=B\(W*W),D=B-(C*W*W)\W
 F G=S-Q:F:S+F+Q S E=B-(C*W*W+(D*W)),H=$E($T(Q),G),@H=$S(@H<S:'Q,Q:N)_@H,T=C_D_E
 F A=Q:Q:W\S S J=$E(T,A),C(F)=$S(J>(F+Q)&(J<(S-F)):Q,Q:+N),C(P)=$S(J#F:Q,Q:+N) D
 .S C(Q)=$S(J<(S-F):+N,Q:Q),C(F+Q)=$S(J>Q&(J<(S-F))&(J'=(P+'L))&(J'=(P)):Q,Q:+N)
 .S H('L)=L F  S H(N?.E)=$O(C(H('$G(N)))) Q:H('+L)=L  S F(A,H('L))=C(H(W[(W\S)))
 F U=Q:Q:P W !,R F V=Q:Q:P+F W $S(F(V,U):'Q,Q:$C(P_(W\S))) W:'(V#F) $C('N_F_F+F)
 W !!,R(F)_C_R(P)_D_R(P)_E_R(F) X $RE($E($T(Q),Q+F,P+Q))_R(P)_'N W # G:N=L Q+F Q
```

https://stackoverflow.com/questions/4151554/need-mumps-sample-code

Fazit: Unterschiedliche Situationen

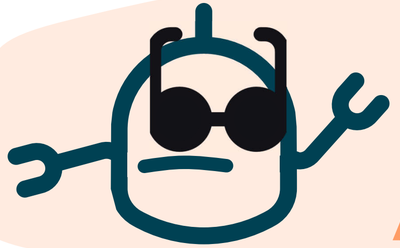


Zusammengefasste Daten aus der 100k Devs Study, Yegor Denisov-Blanch, Stanford, mehr unter https://www.innoq.com/de/blog/2026/03/ueber-ai-einsatz-in-verschiedenen-coding-situationen/

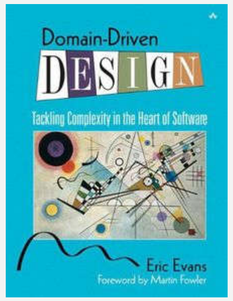
Agentic Software

Modernization

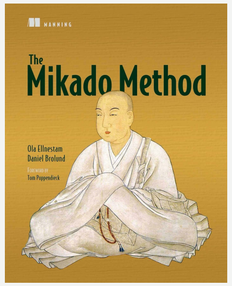
Agentic Software Modernization

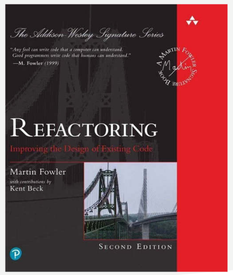


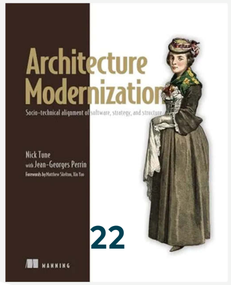
Agentic AI mit klassischen Analyse- und Validierungsmethoden sowie bewährten Modernisierungsvorgehen geschickt kombinieren, um Legacy-Systeme Schritt für Schritt, nachvollziehbar und verantwortbar weiterzuentwickeln.

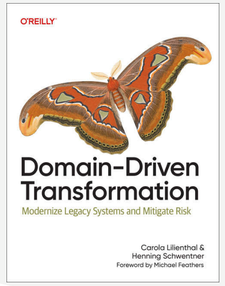


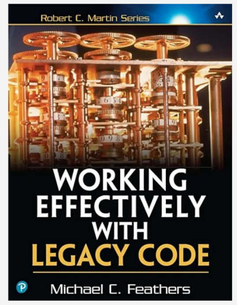








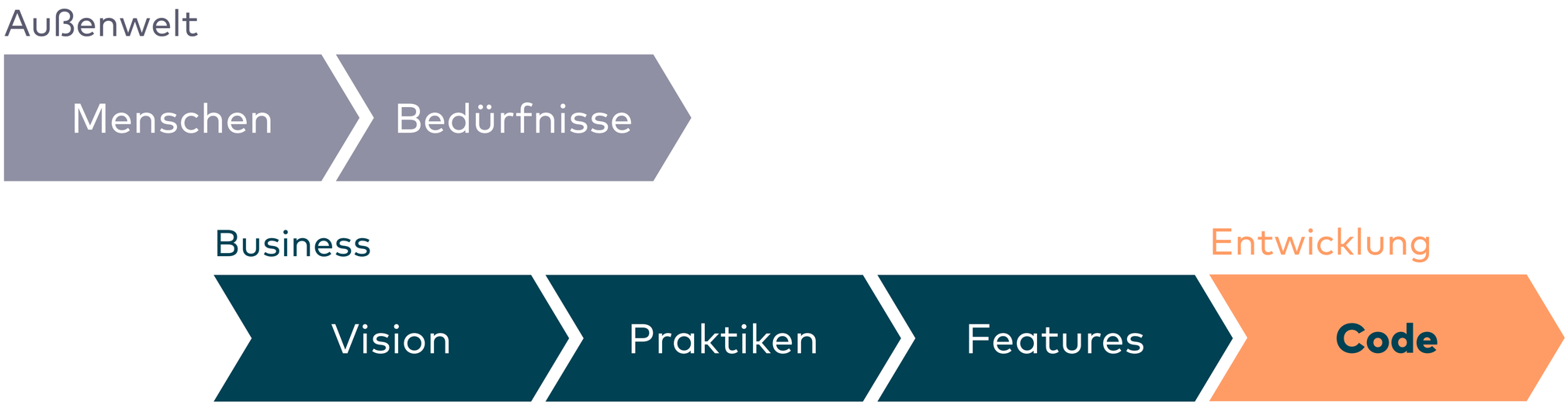




Schritt 0

Agentische Anwendungsbereiche identifizieren

Vermeide Modern Legacy!

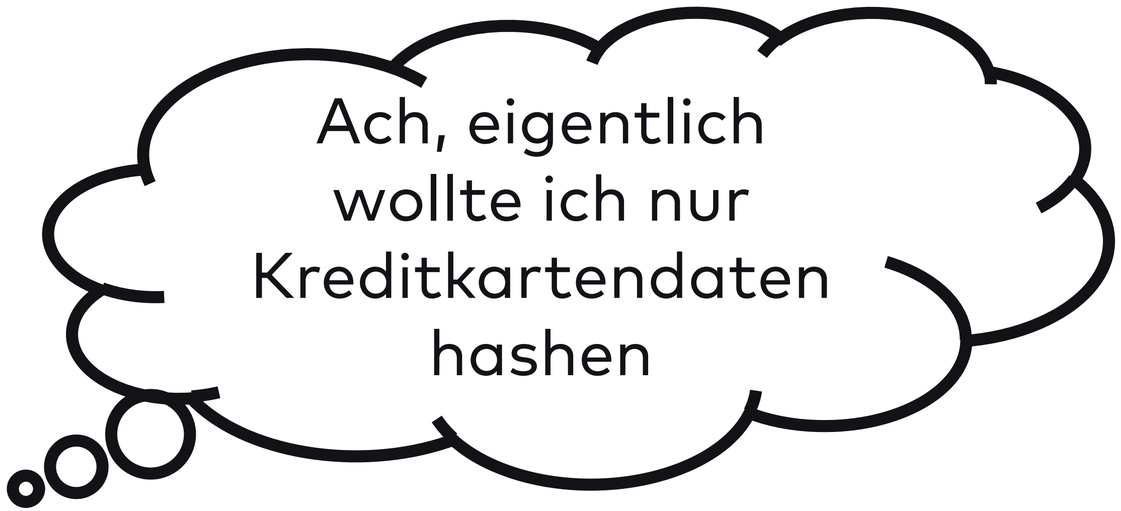


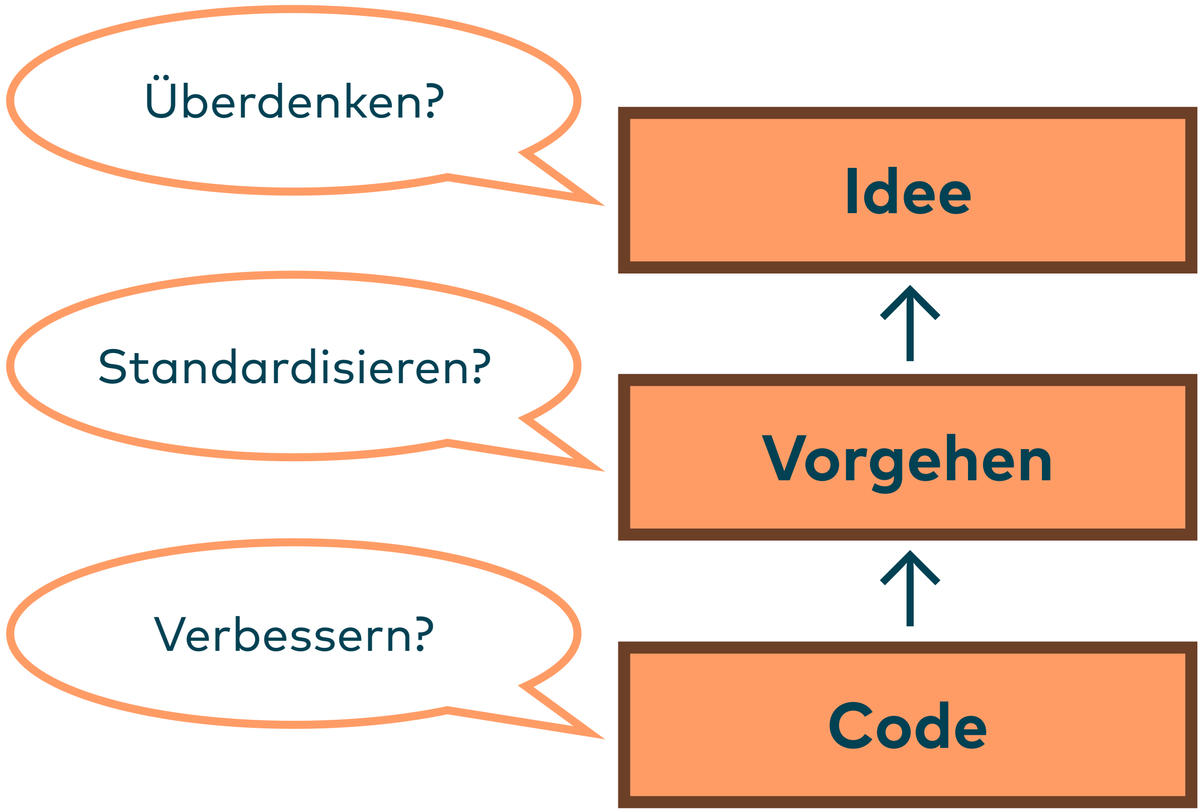
Code ist das Endprodukt einer Kette von vorherigen Überlegungen

https://www.innoq.com/de/blog/2025/10/modern-legacy-dank-ki/

„Wer nur im Code sucht, findet auch nur dort Probleme.“ – Gernot Starke

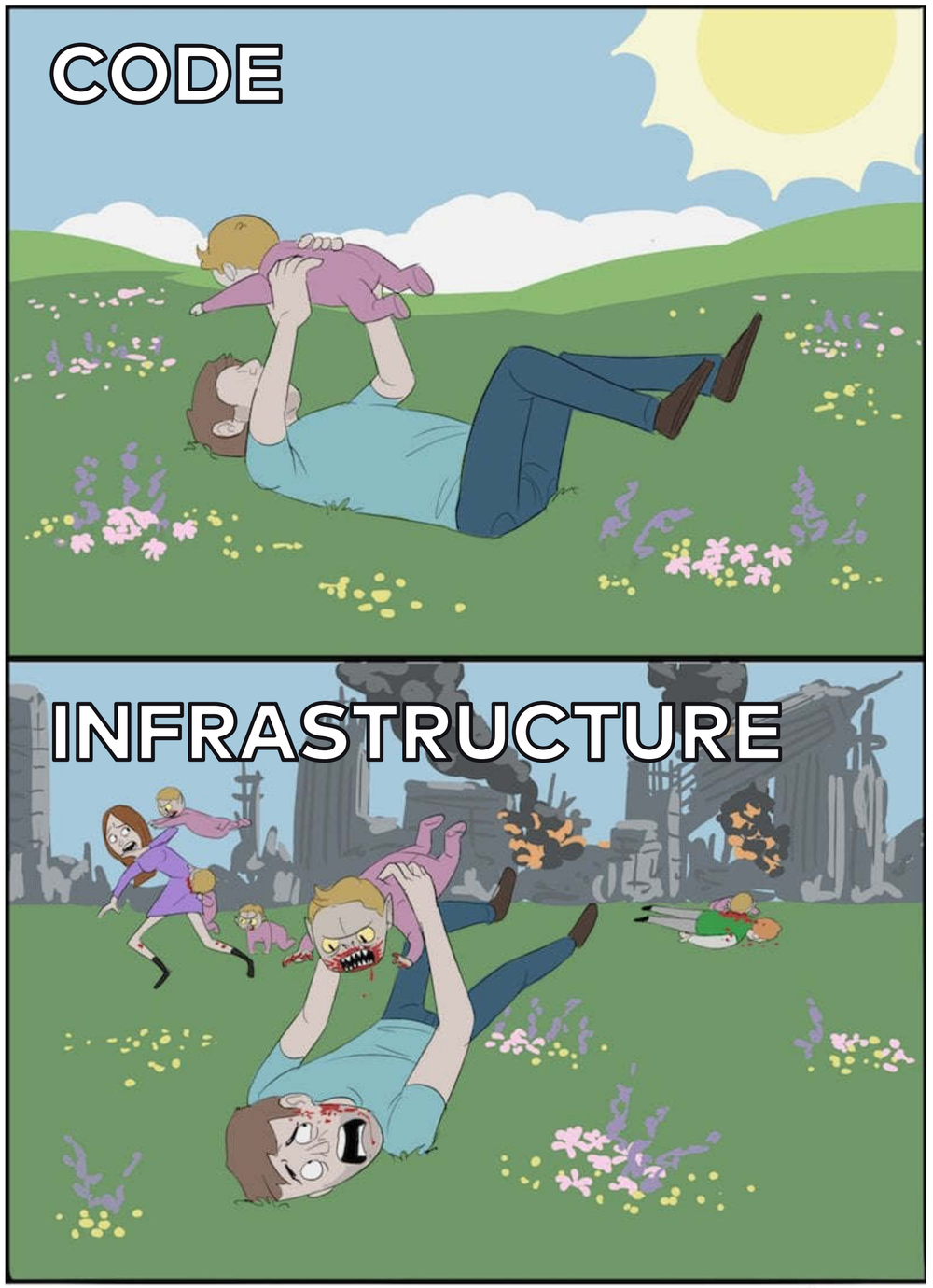
```
sub
encrypt{my($p)=@_;my$a="s";my$b="x";my$c=rev
erse($p.$a.$b.$p);my$d=0;for(split//,$c.$p.$
a.uc($b).reverse($p)){
$d+=ord($_)*3+length($c)%7 }return
"MEGA".$d."END"}print encrypt($ARGV[0]);
```





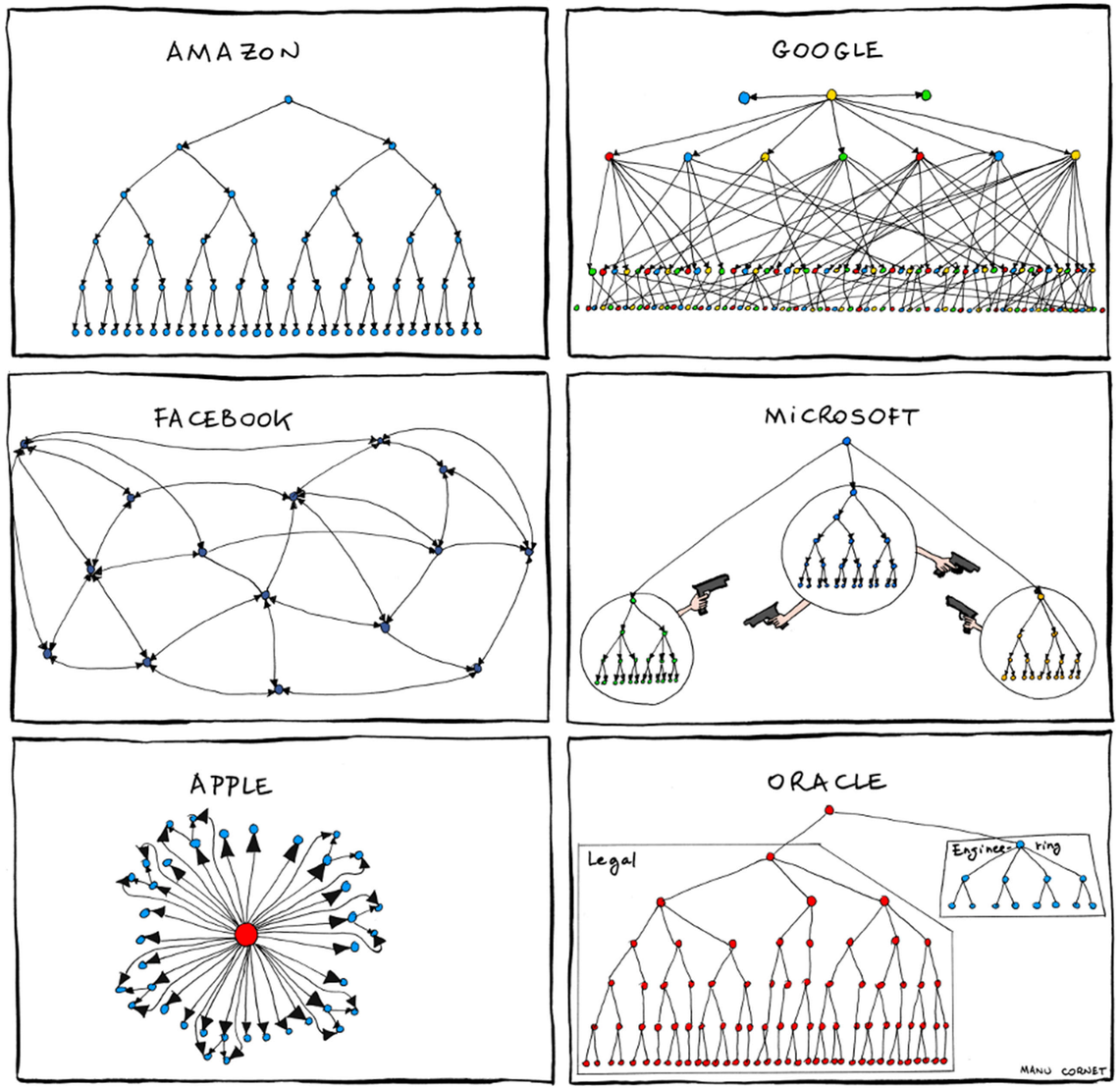
Umgebung spielt auch eine Rolle

- **Cronjobs und Batches**
- **Stored Procedures** und **DB-Trigger**
- Hartcodierte **Hosts, IPs, Pfade**
- **Firewalls, Proxies**, **Zertifikate**
- Alte **Runtimes** und **App-Server**
- Unbekannte **Umsysteme**
- **Lizenzen** und **Verträge**
- …



Organisatorische Themen

- **Wissensträger** sind weg
- Unklare **Ownership**
- **Silos** zwischen Fach und IT
- **Freigaben** und **Change Boards**
- **Kein Budget** fürs Aufräumen
- **Compliance** und **Security**
- **Skepsis** gegenüber Neuem
- …



"Organizational Charts" by Manu Cornet, CC BY-SA 4.0

Liegen meine Herausforderungen wirklich im Code?



Delegiere!

Lagere Komponenten oder ganze Systeme an externe Dienstleister oder andere Organisationseinheiten aus, die deine Arbeit für dich erledigen.



Erhalte am Leben!

Triff bewusst die Entscheidung, ein veraltetes System aktiv weiter zu betreiben, trotz all seiner Schwächen – koste es, was es wolle.



Gestalte um!

Passe das Domänenmodell und die innere Struktur deines Systems an, während du die äußere Funktionalität weitgehend beibehältst.



Harmonisiere!

Richte unterschiedliche Systeme, Modelle und Prozesse so aufeinander aus, dass sie besser zusammenarbeiten – ohne sie in ein starres Standardschema zu pressen.



Ignoriere!

Entscheide ganz bewusst und offen, dein Legacy System so zu lassen, wie es gerade ist.



Kaufe neu!

Kaufe kommerzielle Standard-Software oder SaaS-Lösungen die dein Legacy System komplett ersetzen.

Konsolidiere!

Reduziere die Anzahl an Systemen, Services, Datenbanken oder anderweitigen Komponenten in deinem Ökosystem zu weniger, besser handhabbaren Einheiten.



Partner!

Arbeite mit externen Partnern zusammen für eine gemeinsame Nutzung und Weiterentwicklung des Legacy Systems.



Schalte ab!

Beende den Betrieb des Legacy Systems kontrolliert und begleite es in den wohlverdienten Ruhestand.



Standardisiere!

Lege systemübergreifend verbindliche Konventionen, Schnittstellen, Technologien oder Formate fest, um Unterschiede gezielt zu reduzieren.



Verkaufe!

Verkaufe dein Legacy System an interessierte Parteien, die sie weiter nutzen oder entwickeln wollen und können.



Verschenke!

Gib deine Software als Open Source frei, um Community-Unterstützung und (teilweise) Weiterentwicklung zu ermöglichen.



Zurückholen!

Hole die Entwicklung oder Wartung eines Legacy Systems wieder ins eigene Haus zurück, nachdem diese zuvor an externe Dienstleister ausgelagert worden war.



Überdenke!

Gestalte das System radikal neu, basierend auf einem neuen Verständnis der geschäftlichen Anforderungen, und beginne dafür wieder bei null.



Ersetze!

Baue dein bestehendes System vollständig neu und nutze dabei neue Technologien, saubere Strukturen und bewährte Methoden aus der heutigen Softwareentwicklung.



Extrahiere!

Finde heraus, welches Wissen im alten System steckt, und halte es so fest, dass du es bei der Modernisierung anderweitig nutzen kannst.



Konvertiere es!

Übersetze deinen Legacy-Code in moderne Programmiersprachen oder Frameworks.



Schlachte aus!

Ziehe wertvolle Komponenten, Algorithmen oder Geschäftslogik aus deinem Legacy System heraus, um damit neue und moderne Anwendungen zu bauen.



Spalte ab!

Nutze deinen bestehenden Code als Grundlage, um eine eigenständige Variante deiner Software aufzubauen, die gezielt für neue Kundenbedürfnisse ist.



Teile!

Zerlege ein großes, monolithisches Legacy System in kleinere, eigenständige Systeme oder Domänen, die du unabhängig voneinander modernisieren kannst.



Umhülle!

Ergänze dein Legacy System mit modernen Schnittstellen, Adaptern und Integrationsschichten, damit es mit heutigen Technologien und Arbeitsweisen zusammenarbeiten kann.



Upcycling!

Nutze dein bestehendes System für etwas anderes, als es ursprünglich gedacht war, und hol mehr aus seiner vorhandenen Funktionalität heraus.



Verbessere!

Verbessere dein System Schritt für Schritt durch regelmäßiges Refactoring, sauberen Code und stetigen Evolution der Architektur.



Verschlanke!

Entferne ungenutzte Funktionen, überflüssige Abhängigkeiten und alten Ballast, damit das System wieder wartbar wird und sich auf seinen eigentlichen Zweck konzentriert.



Präserviere!

Archiviere dein Legacy System in Emulatoren oder virtuellen Maschinen, um den Zugriff zu erhalten, ohne weiteren Aufwand für Wartung und Entwicklung.



Umstellen!

Migierere dein bestehendes System auf eine neue Plattform, ohne die zentrale Anwendungslogik grundlegend anzufassen.



Umziehen!

Bewege dein komplettes System in eine neue Betriebsumgebung wie etwa eine Cloud-Plattform, ohne die Anwendung grundlegend anzupassen.



Aktualisiere Werkzeuge!

Modernisiere Entwicklungs-, Deployment- und Diagnose-Tools, ohne das System selbst groß zu verändern.



Bilde weiter!

Bringe Wissen, Denkweise und Kompetenzen im Umgang mit Altsystemen auf ein neues Niveau, um Modernisierung gezielt voranzubringen.



Organisiere um!

Strukturiere Teams, Verantwortlichkeiten, Kommunikationsmuster und Anreizsysteme um für eine bessere Arbeit mit Legacy Systemen.



Passe Strategie an!

Richte die strategische Ausrichtung deines Unternehmens an einem realistischen und zukunftsorientierten Umgang mit Legacy Systemen aus.



Verbessere Prozesse!

Entwickle die Prozesse rund um Entwicklung, Test und Betrieb deines Legacy Systems weiter.

Strategiekategorien: Geschäftlich (14) Entwicklung (10) Infrastruktur (3) Befähigung (5)

https://feststelltaste.github.io/littlestmodernization/

Immer gut

Arbeitspakete für Mensch und Agenten entwerfen

Keeping ~~AI~~ Agents on the leash

Agentenfähigkeiten bewusst einschränken

- **Teilautonomie statt Vollautonomie**
  Der Mensch regelt aktiv, wie viel er dem Agenten überlässt.
- **Prüfbare Umbauschritte**
  Mensch sichtet kleine Diffs im Detail, Großes nur musterhaft.
- **Abwechselnde Schleifen**
  Agent erzeugt, Mensch prüft, Agent führt aus, Mensch korrigiert. REPEAT.
- **Iron Man statt Terminator**
  KI erweitert den Menschen, statt ihn zu ersetzen.



Ein mit einer Leine geführter Agent

Inspiriert von Andrej Karpathy, „Software Is Changing (Again)“, Juni 2025

Was darf der Agent entscheiden?

Lange Leine: mehr Freiheit für den Agenten

Kurze Leine: mehr Kontrolle und Nachvollziehbarkeit



Prompt

„Mach einfach“

Absicht vorgeben, der Agent entscheidet den Rest: Schritte, Werkzeuge, Ende.

Gut für: erste Erkundungen, Spikes & Scratch Refactorings, Tool-Vibecoding



Agent Skill

„So wird’s gemacht“

Ziel und grobes Vorgehen vorgeben, aber die Umsetzungsdetails füllt der Agent aus.

Gut für: bekannte Analyse- und Umbau-verfahren bei wechselnder Sprache und Architektur



Werkzeugkasten

„Mach’s damit“

Freie Planung, aber nur mit fest vorgegebenen MCP-Servern, Analyse- und Refactoring-Tools.

Gut für: belastbare Fakten zur Codebasis und sichere, toolge-stützte Umbauten



Generiertes Skript

„Mach’s wiederholbar“

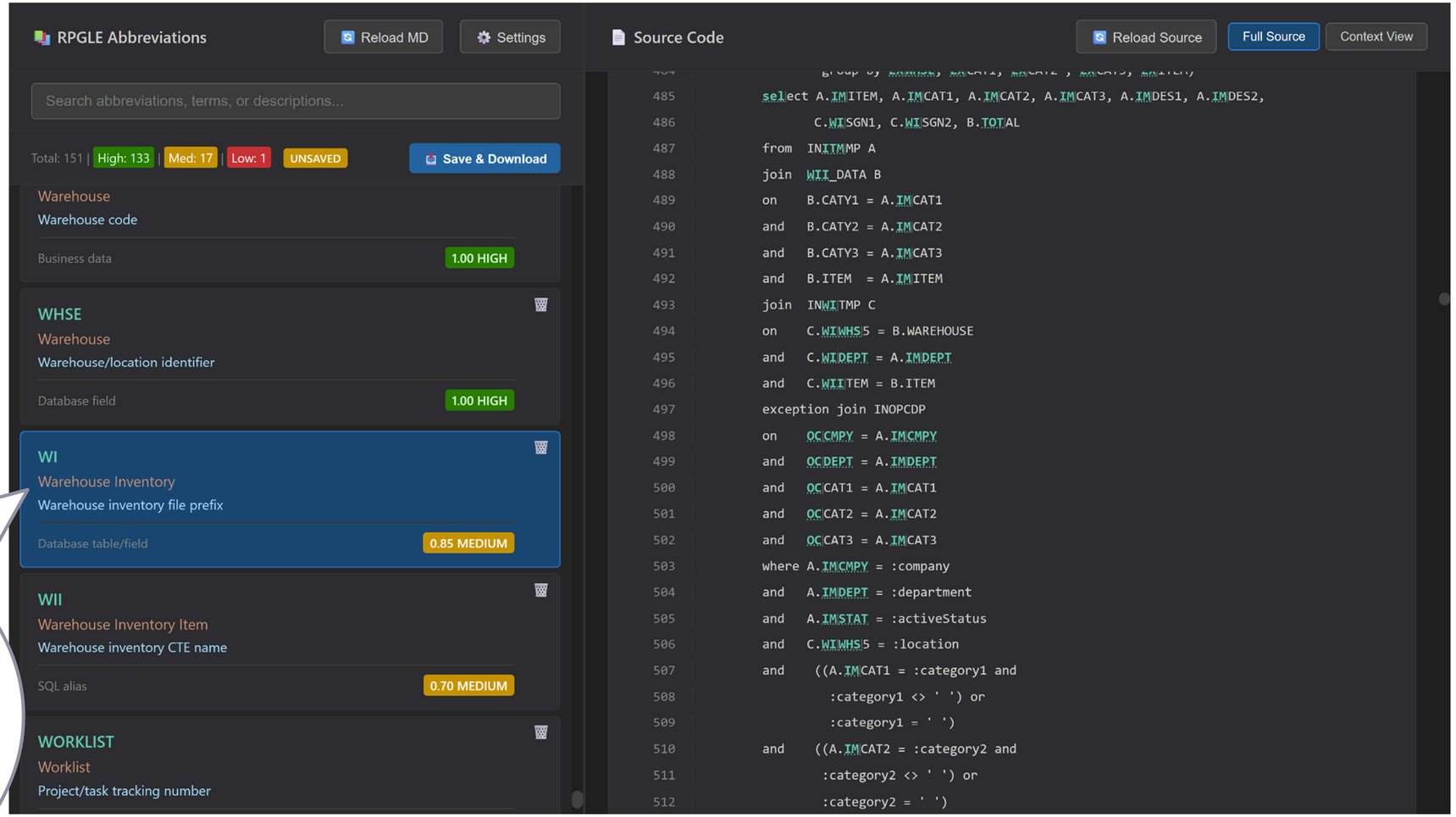
Der Agent schreibt nichtdeterministisch ein Skript, das determi-nistisch abläuft.

Gut für: wiederholbare große Codeänderungen und wiederkehrende Fragen / Tracking

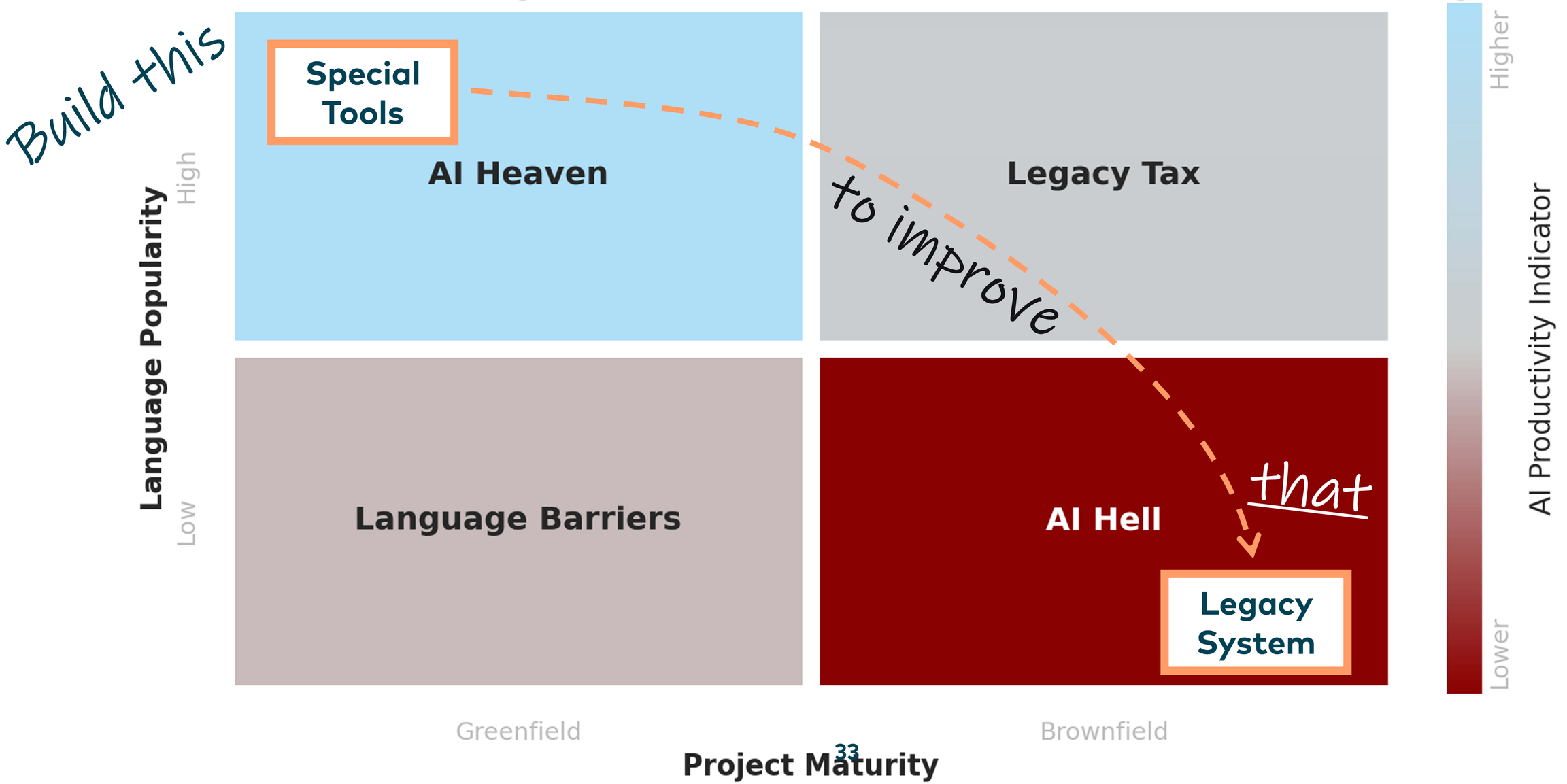
Prompt

„Mach einfach“

Von LLM geschriebenes und von LLM gestütztes Tool zum Erstellen von Abkürzungs-verzeichnissen



Vibecoding von Spezialwerkzeugen



Agent Skill

„So wird's gemacht"

Prüfung von Git-Repos

```
# Git Trustworthiness Audit

Assess whether this repository's git history is reliable
enough to use as ground truth for "who changed what,"
"when," and "why" questions. [...]

## Check 1 — Author identity consistency

The same person often commits under multiple name/email
pairs [...]. This silently corrupts any "who did what"
analysis. [...]

## Check 2 — Commit message quality
...
## Check 3 — History granularity (squash/merge patterns)
...
```

https://github.com/feststelltaste/agentic/blob/main/skills/git-repo-trust-audit/SKILL.md

● **Skill**(git-repo-trust-audit)
  └ Successfully loaded skill

● **Bash**(cd /home/markus/repo/openclinica-demo-ws
    echo "total: $(git rev-list --count HEAD)"; echo "shallow: $(git rev-parse --is-shallow-repository)"; ls .mailmap 2>&1…)
  └ total: 5849
    shallow: false
    ls: cannot access '.mailmap': No such file or directory
    … +196 lines (ctrl+o to expand)
  └ Allowed by auto mode

● **Bash**(cd /home/markus/repo/openclinica-demo-ws
    echo "merges in last 500: $(git log --format='%P' -n 500 | awk 'NF>1' | wc -l) total merges: $(git rev-list --merges –…)
  └ merges in last 500: 165 total merges: 1109
    --- old subjects sample
    e27951564 2023-03-28 Upping library versions
    … +56 lines (ctrl+o to expand)
  └ Allowed by auto mode

● **Overall: mixed.** The history is reliable for current file state and coarse who/when. It is only partly reliable for intent and line-level authorship. I examined 5,849 commits in total: the last 500 for messages and size, and the whole history for merges, date divergence and authors.

Dimension	Rating	Evidence
Identity	**poor**	Many duplicate identities and no .mailmap
Messages	**mixed**	The recent 98 commits are excellent, the upstream history is patchy, and 353 of 500 have no body
Granularity	**mixed**	165 of the last 500 are merges and 1,109 overall, but the average commit is small at 2.3 files
Rewrites	**good**	Author and commit dates differ in 44 of 5,849 commits, and 0 of the last 200
Correspondence	**good**	5 of 5 sampled commits matched their diffs, though minor changes and OC-8627 say little
Coverage	**mixed**	Not shallow, but the first commit is a 2009 import of 5,024 files (628k lines)

Identity.
- Duplicate clusters I'm fairly sure of:
  - Joe Keremian and jkeremian share one email, and joekeremian@joe-mac-pro… looks like the same person.
  - Steve DiBona and sdibona share one email, and stevedibona@sdib-mbp-osx.local looks like the same person.
  - Krikor Krumlian appears under 3 emails.
  - Ron Panduwana appears under 2 emails.
  - Leonel Aguilar Gayard appears under 2 emails.
  - Tao Li also appears as Taoli2018 and 7KRW-T570-W10\Tao Li.
  - Yufang Wang also appears as a devnull@localhost identity.
- 443 commits come from devnull@localhost (ahamid, sshamim, Yufang Wang). These are most likely artefacts of an old SVN or Mercurial conversion, so per-author counts for them are unreliable.
- tomcat@dev.openclinica.com (97 commits) is a machine account, not a person.

Messages.
- About 587 subjects are 12 characters or shorter and 104 are only a ticket ID. Examples: minor changes (727772da6), OC-8627 (c8e250221), Fxied (bde69fd9f), saving work (d755de769).
- Many upstream commits are OC-nnnn: description, which links to a ticket and is useful.
- The 98 commits from October 2026 have clear, intent-describing subjects.
- Roughly 353 of the last 500 commits have no body.

Granularity. Many commits are Merge pull request #… from …, so git blame and log on merge commits point at the merger rather than the author. Individual commits are small, which helps.

Coverage. Anything before the 2009 import is invisible, and the history thins out after 2023. The 2026 work is an unrelated
- The 98 commits from October 2026 have clear, intent-describing subjects.
- Roughly 353 of the last 500 commits have no body.

Granularity. Many commits are Merge pull request #… from …, so git blame and log on merge commits point at the merger rather than the author. Individual commits are small, which helps.

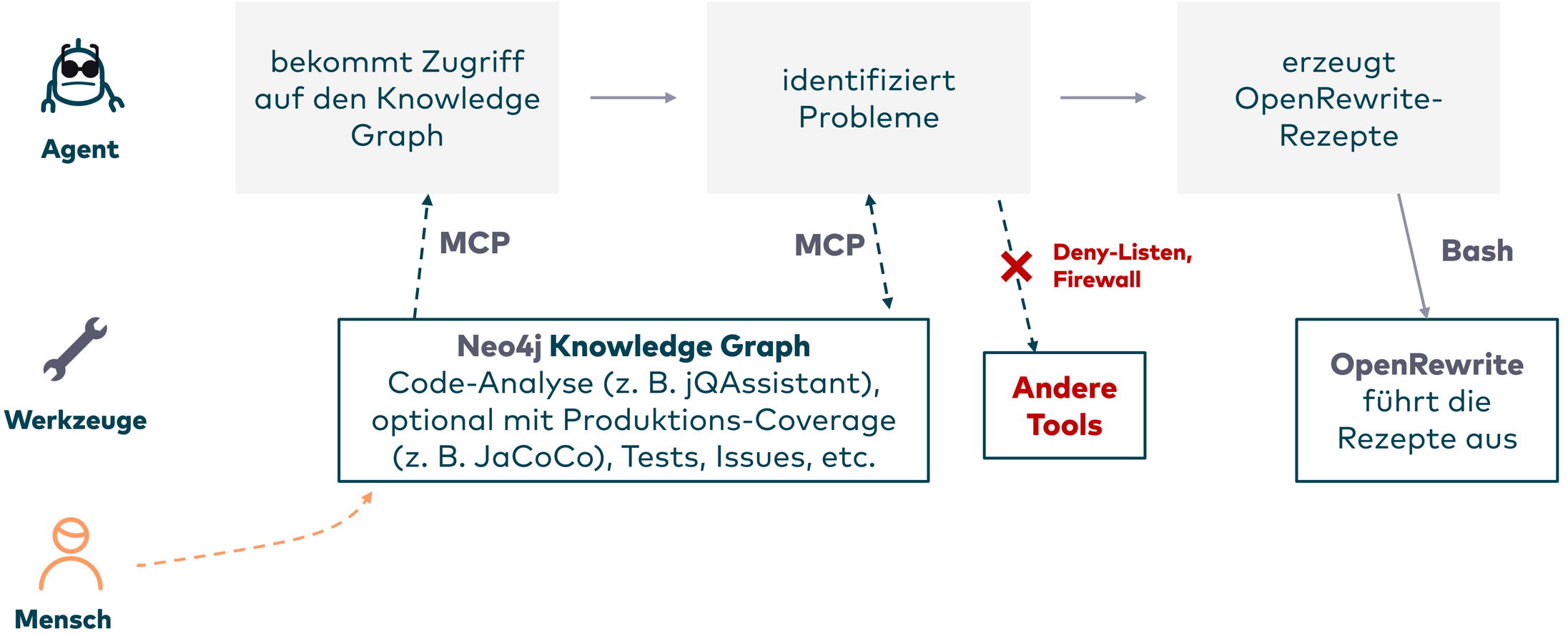
Coverage. Anything before the 2009 import is invisible, and the history thins out after 2023. The 2026 work is an unrelated workshop layer (dev container and agent setup) on top of the OpenClinica history.

What you can trust git for:
- the current state of the files
- coarse who and when on the upstream code

Coverage. Anything before the 2009 import is invisible, and the history thins out after 2023. The 2026 work is an unrelated workshop layer (dev container and agent setup) on top of the OpenClinica history.

Werkzeugkasten

„Mach's damit"

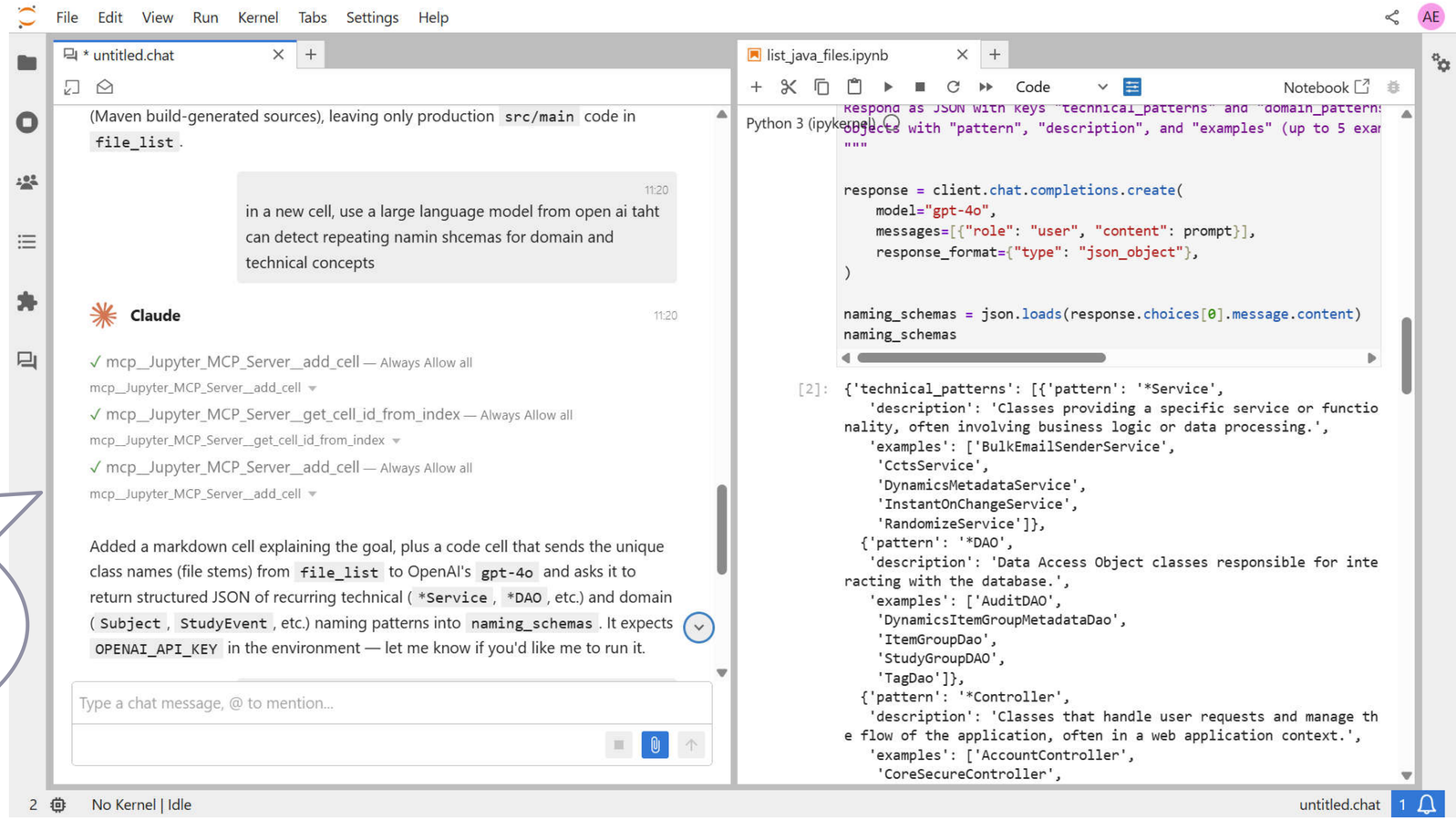


Generiertes Skript

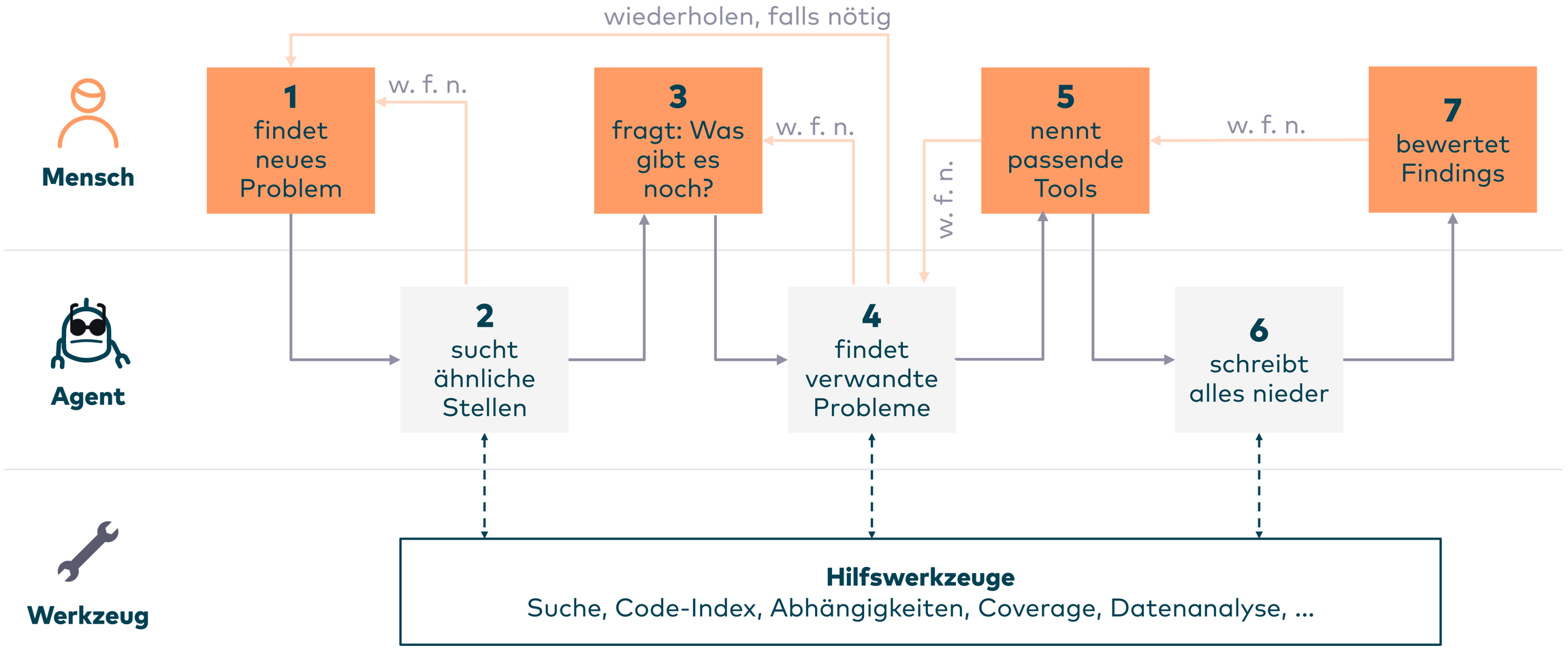
Alternativer Name
Explicit Scratchpad

„Mach's wiederholbar"

Agentisch erzeugtes Skript analysiert deterministisch große Codebasen



Kombination von Vorgehen



Schritt 1

Blast-Radius des Agenten reduzieren

Wo im Code darf der Agent ran?

Zonenmodell nach Addy Osmani: von Menschen, für Agenten

Grün

z. B. gute Testabdeckung, aktuelle Konventionen, gut isoliert und dokumentiert

Beispiel: `REPORT`, 0815 Spring-Boot-Services, 85 % Testabdeckung

Vorgehen

Agent arbeitet in enger Schleife selbstständig

= Agent an der langen Leine

Gelb

z. B. durchwachsene Codequalität mit Smells, Alt- und Neuwelt wild gemischt

Beispiel: `KUNDSTAMM`, Uralt-Java-EE-Monolith mit seinen Eigenheiten

Vorgehen

Erst Absicherung (z. B. Tests), dann ändert der Agent

= Agent an der kürzeren Leine

Rot

z. B. Code mit essenzieller Komplexität (Auth, Rollen und Rechte), auch: implizites Wissen mit wenigen Köpfen

Beispiel: `DBALUX87` (Speziallogik), Herberts (seit 10 Jahren in Rente) Code, eigener Framework-Code

Vorgehen

Mensch arbeitet jeden Schritt mit, oder es passiert nicht

= Agent direkt an der Hand

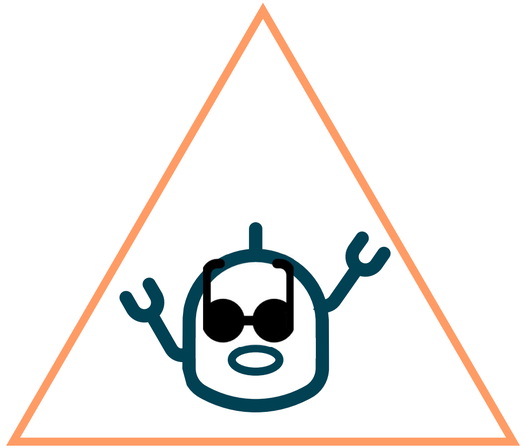
Quelle: Addy Osmani, https://addyo.substack.com/p/brownfield-agentic-engineering

Chaos begrenzen, Impact erhöhen

Das „Magische Dreieck der Softwaresanierung“*

Slicing
(zum Verstehen)

Agenten einen überschaubaren, abgegrenzten Codeausschnitt zum Arbeiten geben.



Sprouting
(zum Erweitern)

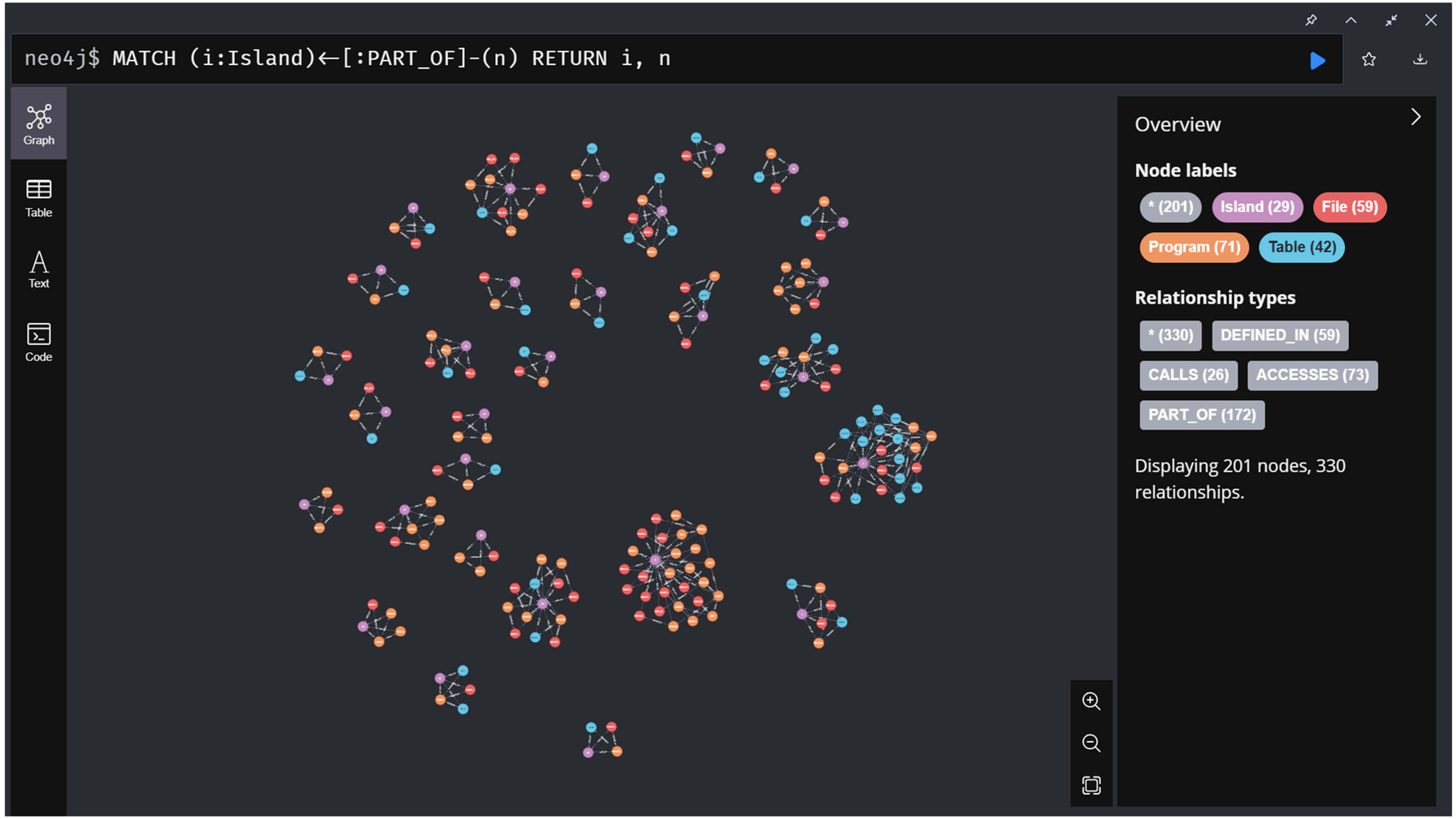
Greenfield-Bereich schaffen, in dem neue Features über Agenten unabhängig entstehen können.

Seams
(zum Flexibilisieren)

Nahtstellen einbauen, damit Agenten neue Implementierungen erstellen können.

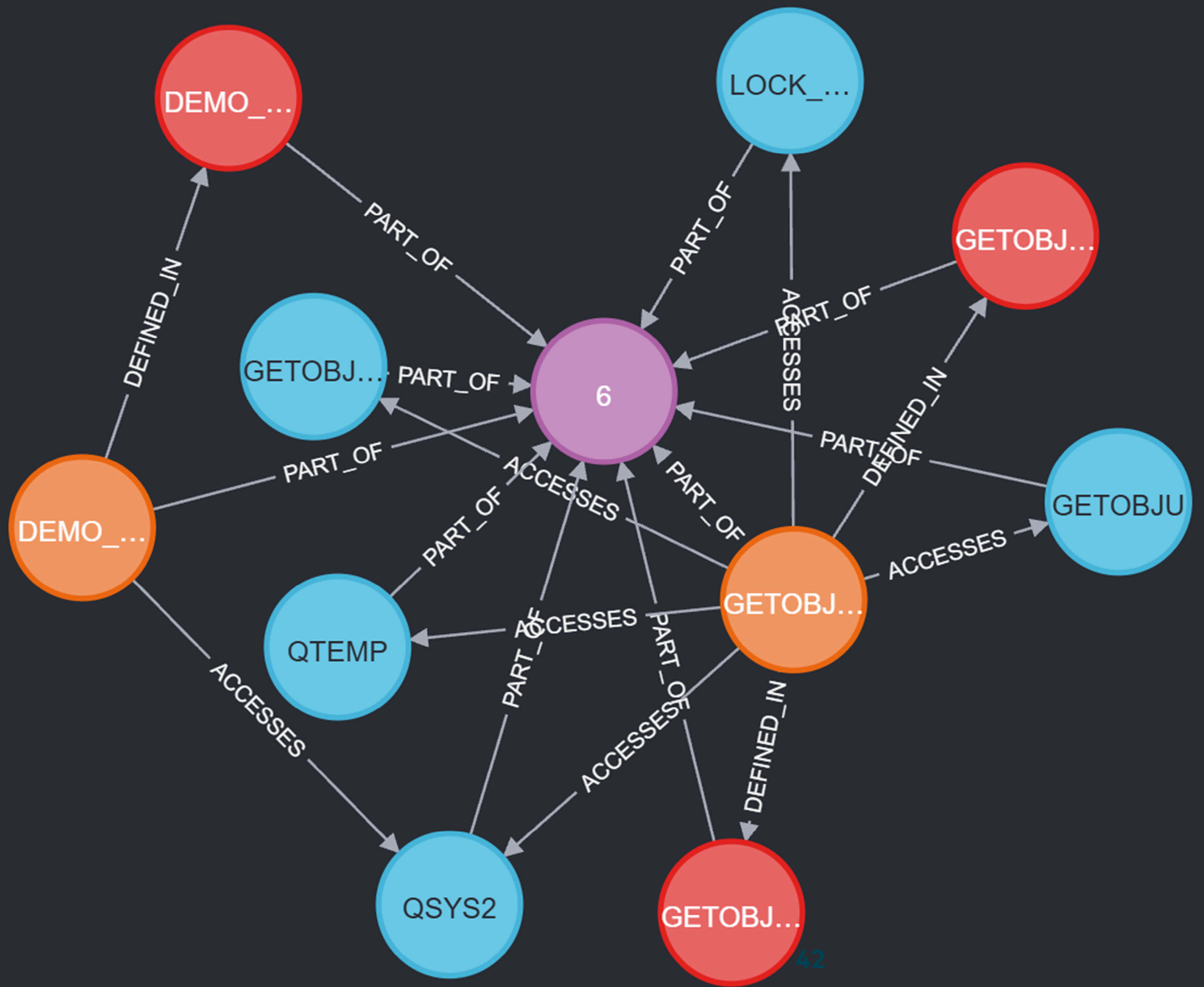
*ja, der Begriff ist von mir völlig frei erfunden

Beispiel: Modulbasiertes Slicing



https://markusharrer.de/blog/2026/02/09/rpg-island-prototype/

```
neo4j$ MATCH (i:Island {id:6})←[:PART_OF]-(n) RETURN i, n
```



Graph
Table
Text
Code

Overview

Node labels

* (11)
Island (1)
File (3)
Program (2)
Table (5)

Relationship types

* (19)
PART_OF (10)
ACCESSES (6)
DEFINED_IN (3)

Displaying 11 nodes, 0 relationships.

N4OJ database contents

Überschaubarer Kontext für den Slice

Island 1 – Customer Management System

30 Komponenten · 7 Programme · 15 Tabellen · 8 Dateien

Was macht diese Insel?

Island 1 ist ein vollständiges **Kundenstammdaten-Verwaltungssystem** auf IBM i (AS/400) in RPG Free/SQL. Es ist die einzige Insel im Repo, die wie eine echte Produktionsanwendung aufgebaut ist – mit Datenhaltung, Geschäftslogik, UI und geteilten Services.

Aufbau

Kern: `CUSTMAST` ist die zentrale Kundenstammdatei, auf die fast alle Programme zugreifen.

Fachliche Programme:

- `LOADCUSTR` – Lädt/generiert Kundentestdaten, greift auf `CUSTMAST`, `LENNONS1` (Zufallsdaten) und `CSZ_CUR` (City/State/ZIP) zu
- `MTNCUSTR` – Pflegt Kundensätze (Anlegen/Ändern/Löschen) über eine Display-Maske, mit Bundesstaaten-Validierung
- `PMTCUSTR` – Kundensuche per Subfile-Auswahldialog (Prompt), zeigt Trefferliste zur Auswahl
- `PMTSTATER` – Bundesstaaten-Auswahldialog (Prompt für State-Codes)

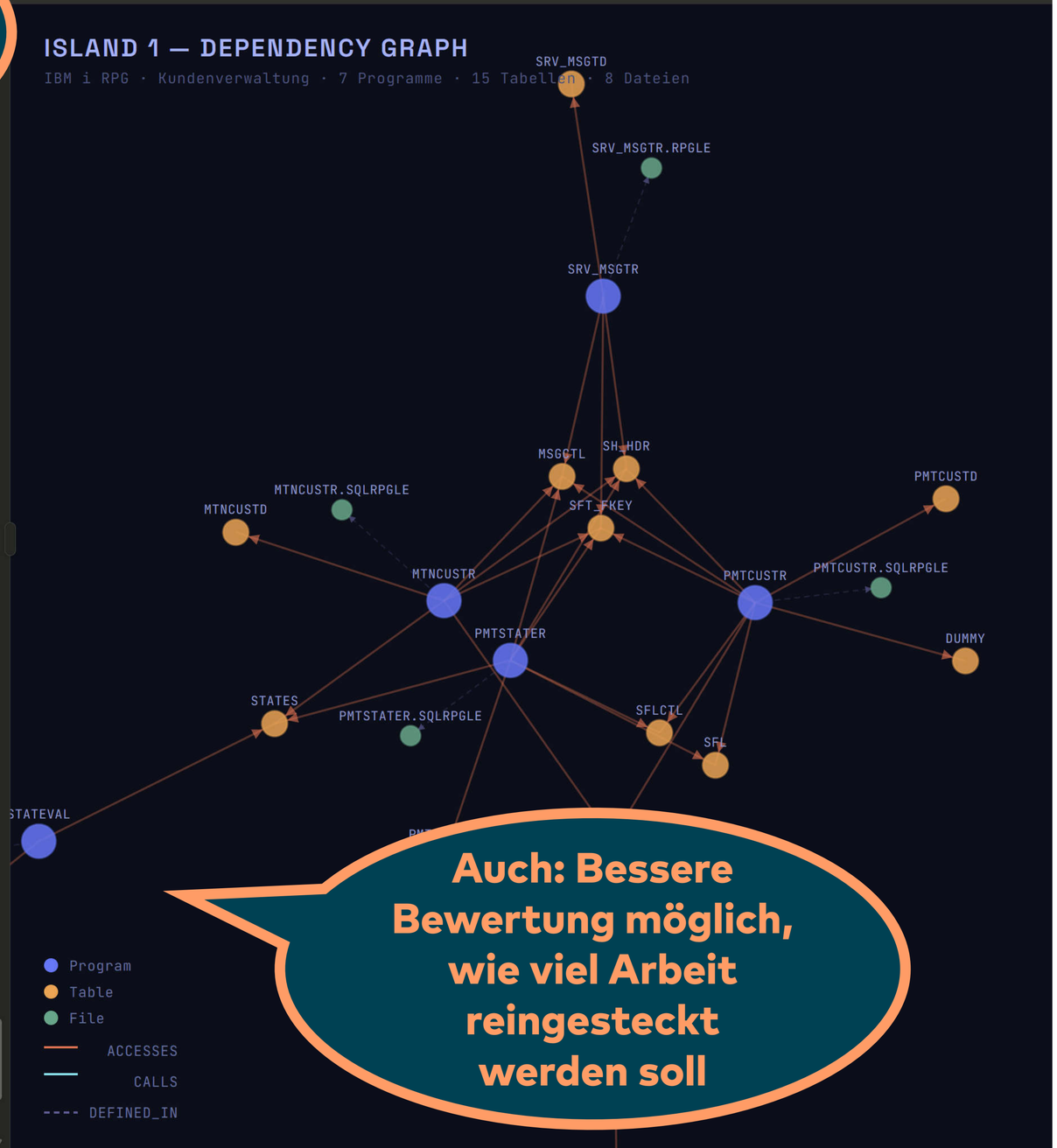
Antworten...

Sonnet 4.6

Claude ist eine KI und kann Fehler machen. Bitte überprüfe die Antworten.

Island1 graph · HTML

Kopieren



Auch: Bessere Bewertung möglich, wie viel Arbeit reingesteckt werden soll

Konzeptbasiertes Slicing

Code-Inventar
Was ist da?

- Programme, Tabellen, Dateien
- Aufrufe und Datenzugriffe
- vollständig, deterministisch

Codebasis kennenlernen
Was bedeutet das?

- Agent liest den Code
- erkennt Absicht und Muster
- Bedeutung wird vorgeneriert

Abstraktes Modell

Technische Konzepte
Logging, Fehler, DB-Zugriff

Fachliche Konzepte
Kunde, Auftrag, Rechnung

Ähnlichkeiten
Muster, Copy-&-Paste-Varianten

Separat bearbeitbare Slices

Datenzugang
Technisches Konzept

Kundenstamm
Fachliches Konzept

Suchdialoge
Ähnliche Konzepte

Pro Slice: eigener Agent / Kontext

Konzeptextraktion mit LLMs

Basis: Verwendung vorhandener **Namensschemata**

```
.
├── model
│   ├── Owner.java
│   ├── Person.java
│   ├── Pet.java
│   ├── Specialty.java
│   ├── Vet.java
│   └── Visit.java
├── repository
│   ├── OwnerRepository.java
│   ├── PetRepository.java
│   ├── VetRepository.java
│   └── VisitRepository.java
└── web
    ├── OwnerController.java
    ├── PetController.java
    ├── PetValidator.java
    ├── VetController.java
    └── VisitController.java
```



Repository-Muster

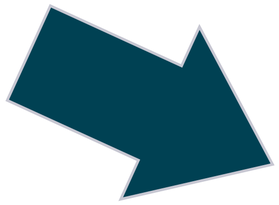
`repository/*Repository.java`

Abstrahiert den Datenzugriff durch Kapselung der Logik, die zum Abrufen, Speichern und Abfragen von Daten erforderlich ist.

Seam für bessere Austauschbarkeit

vorher

```
public class Handler {
  public Handler() {
    processor = new Processor();
    ...
  }
}
```



nachher

```
public class Handler {
  public Handler() {
    processor = createProcessor();
    ...
  }
  protected Processor createProcessor() {
    return new Processor();
  }
}
```

Seam geschaffen

Nun möglich, alternative, parallele Implementierung anzugehen!

Abhängigkeiten als Problem

Sichtbar

- Imports, Aufrufe, Vererbung: Tools finden sie

Versteckt

- Verweise quer über Dateitypen und Sprachen
- Erst zur Laufzeit initialisierte Abhängigkeiten
- Über Änderungen gekoppelte Codeteile
- Konfiguration und Dependency Injection
- Datenbankobjekte, Queues und Events

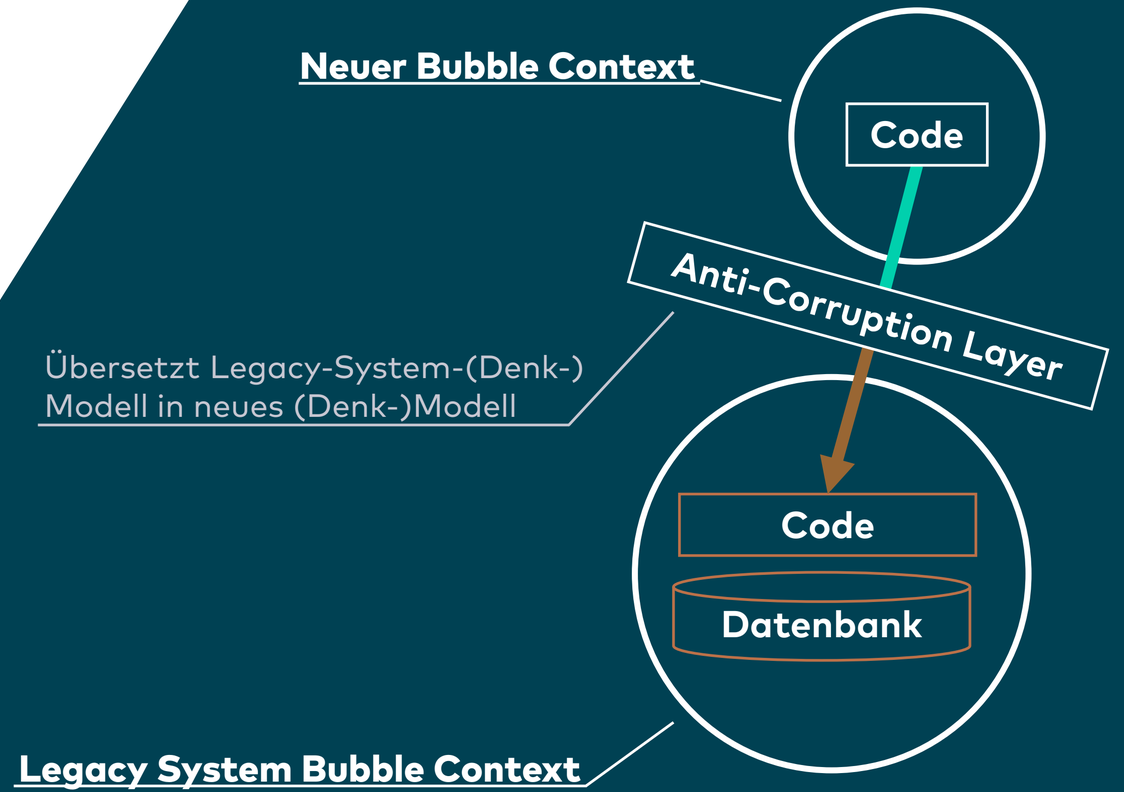
Sprouting

Saubere Erweiterungen unter Zeitdruck und ohne Ballast des Altsystems

Agentic AI Use Case

Auf der grünen Wiese neue Funktionsbereiche agentisch erstellen lassen, ohne zu viel vom Bestandssystem beeinflusst zu werden.

Sprouting im Großen mittels „Bubble Context“



ACL-Idee von Eric Evans,
Sprouting-Idee von Michael Feathers

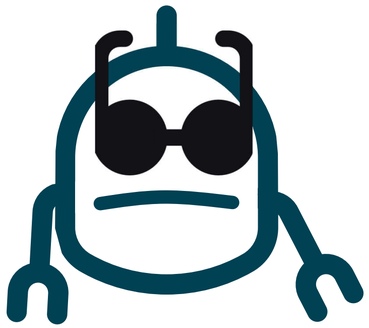
Schritt 2

Code „ready“ für Agenten machen

Coding-Agenten im Einsatz

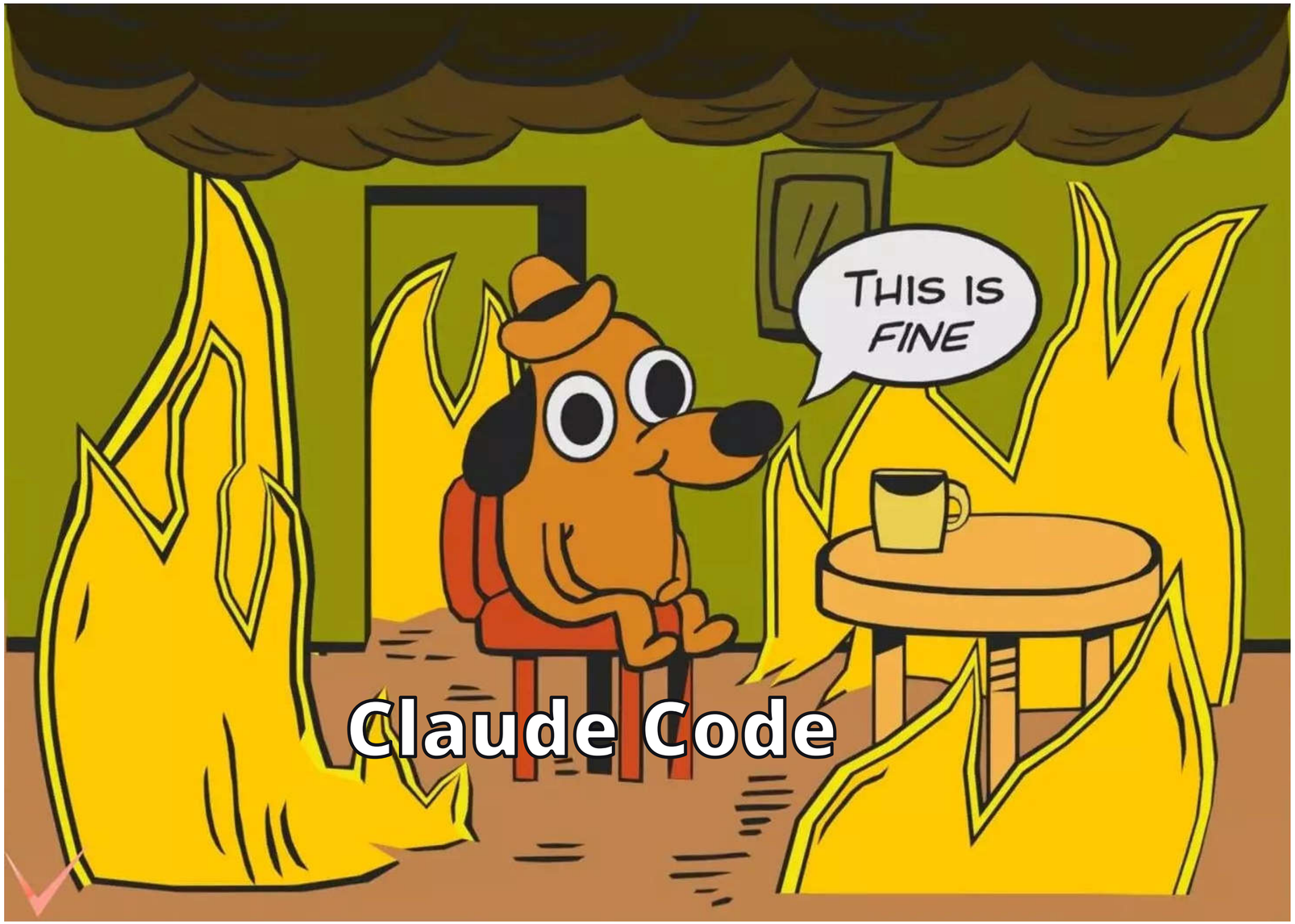
> "Claude Code is making use of agentic search
> ... using a combination of glob and grep.

Anthropic: Transform Legacy Systems into Strategic Assets - Code Modernization with AI
https://www.youtube.com/watch?v=8qtSeQuNv0o



Changing a mission-critical code chaos with search & replace.

What could possibly go wrong?



Strukturelle Navigation verbessern

✗ Schlechte globability

```
/src
├── BillingTeamsService.py
├── chargeTeamByCard.py
├── create-InvoiceData.py
├── datagatekeeper.PY
├── format-date-helpr.py
├── get-usr-prof.py
├── thequerymachine.py
└── user_discount_srv.py
```

✓ Gute globability

```
/src
├── /modules
│   └── /billing
│       ├── __init__.py
│       ├── charge_by_card_use_case.py
│       ├── billing_service.py
│       ├── invoice_repository.py
│       ├── create_invoice_dto.py
│       └── format_date_helper.py
└── /infra
    └── db_client.py
```

Strukturinformationen einbringen

Namenskonventionen in Klassen

```
public class CustomerRepository {
```

Namenskonventionen in Modulen

```
package org.springframework.samples.petclinic.repository;
```

Definition mittels Annotationen

```
@interface Repository {
```

```
@Repository
class Customer {
```

Definition mittels Marker-Interfaces

```
public interface Repository {
```

```
public class CustomerDAO implements Repository {
```

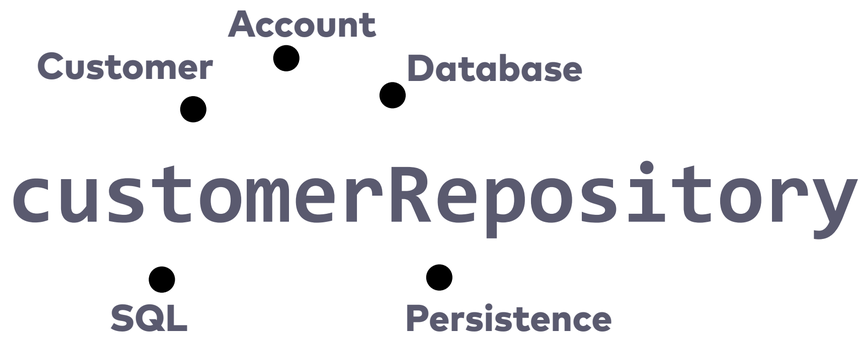
Inhaltliche Orientierung ausbauen

✗ Schlechte grepability



LLM erkennt rein technische Details. Hat Schwierigkeiten, die Lücke zur Geschäftslogik zu überbrücken.

✓ Gute grepability



LLM erkennt passende Muster zum Andocken. Kurze Entfernung zu Geschäftskonzepten.

[1] Extrem stark vereinfacht

Semantische Anker nutzen

Präzise definierte Fachbegriffe, Methodiken und Frameworks für eine effektive LLM-Kommunikation.

[...]

Dienen als Referenzpunkte bei der Interaktion mit Large Language Models. [...]

Fungieren als gemeinsames Vokabular, das spezifische, kontextreiche Wissensdomänen innerhalb der Trainingsdaten eines LLMs gezielt aktiviert.



Übernommen und übersetzt von https://llm-coding.github.io/Semantic-Anchors/

Semantische Anker – Beispiele

- ✗ “Write tests”
- ✓ “Write tests according to Behavior-Driven Development”

- ✗ “review architecture”
- ✓ “review my hexagonal architecture style”

- ✗ “improve database access”
- ✓ “migrate all DAOs to JPA repositories”

Teils übernommen und übersetzt von https://llm-coding.github.io/Semantic-Anchors/

Entwickler

Wir wollen die Codequalität verbessern, damit wir unseren Code verstehen können

Wir wollen die Codequalität verbessern, damit AI unseren Code versteht

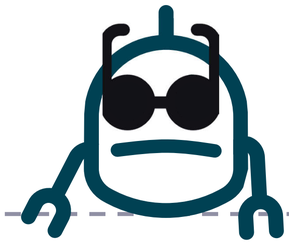
Manager



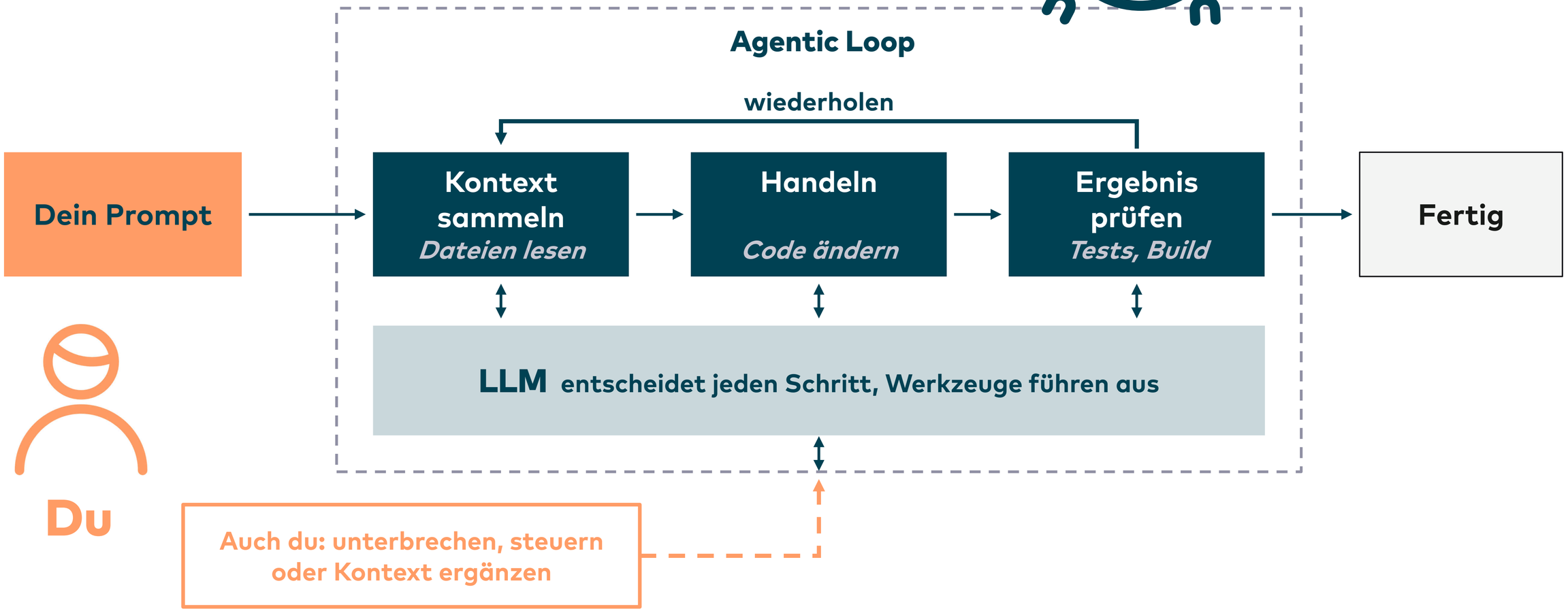
Schritt 3

Agentisches Arbeiten verbessern

Der „Agentic Loop"



Dein Agent



Nach https://code.claude.com/docs/en/how-claude-code-works

Wichtig

> "The model is only as good as its context.



Programmierer

Anthropic: Transform Legacy Systems into Strategic Assets - Code Modernization with AI
https://www.youtube.com/watch?v=8qtSeQuNv0o

Agenten und ihr Kontext



Clean engineering environments allow AI to autonomously drive a larger share of the sprint

Clean Code Amplifies AI Gains

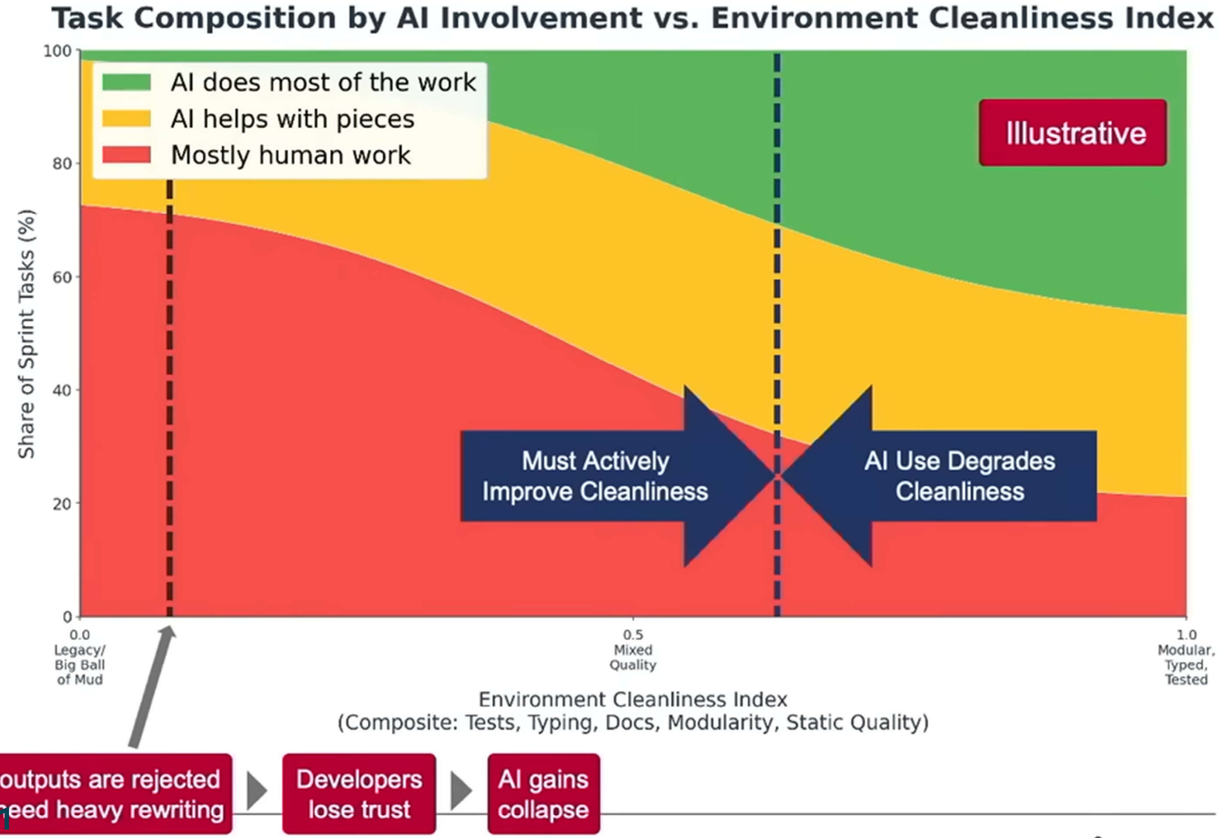
- A clean codebase allows AI to complete a larger share of sprint tasks

Manage Codebase Entropy

- Unchecked AI accelerates entropy (tech debt)
- High entropy degrades future AI performance

Engineers Must Master Task Selection

- Knowing when to use AI
- …and when to write code manually



Garbage in, garbage out

Clean Code Amplifies AI Gains

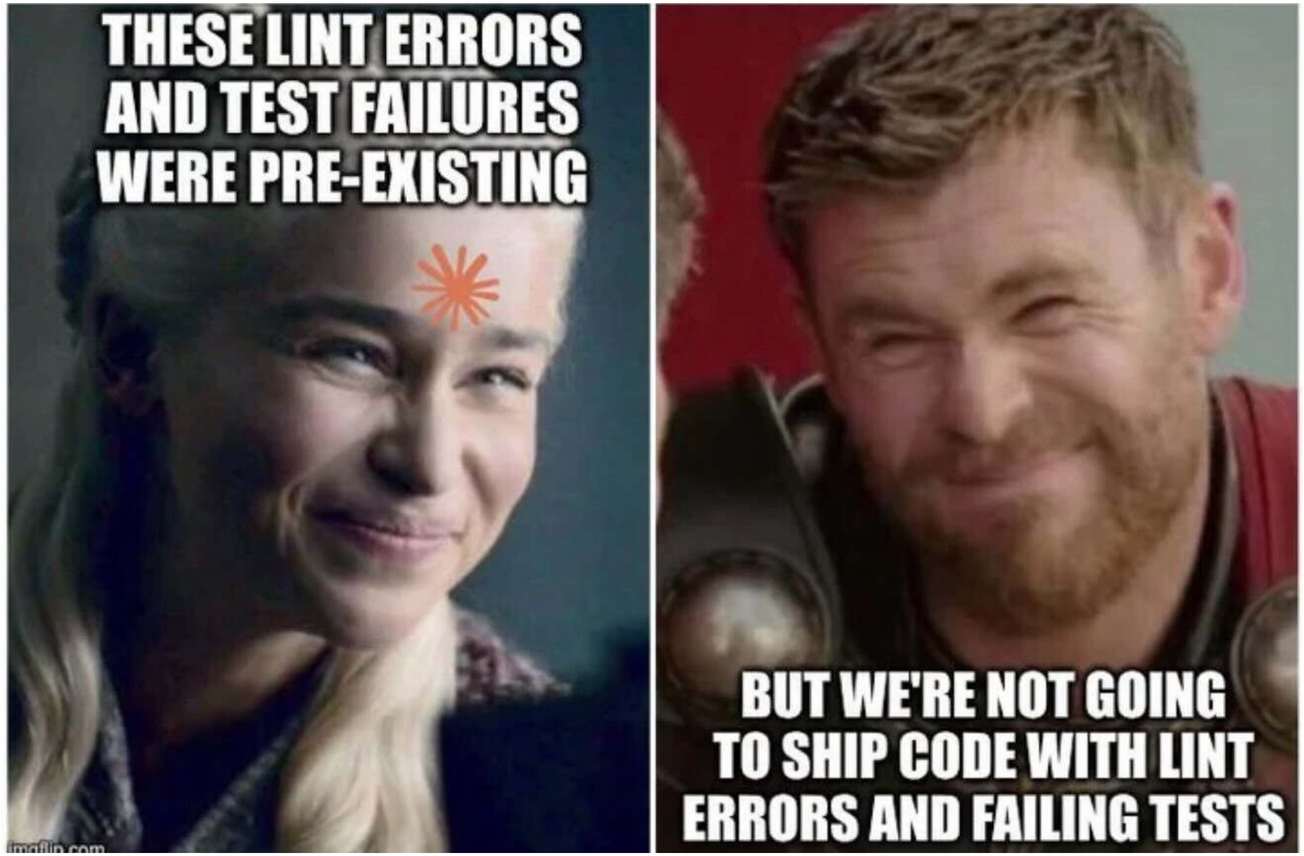
- A clean codebase allows AI to complete a larger share of sprint tasks

Manage Codebase Entropy

- Unchecked AI accelerates entropy (tech debt)
- High entropy degrades future AI performance

Engineers Must Master Task Selection

- Knowing when to use AI
- …and when to write code manually



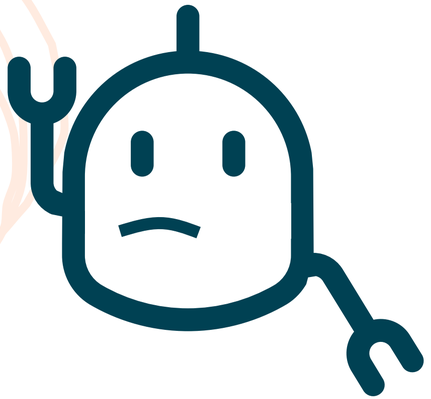
https://www.reddit.com/r/ClaudeCode/comments/1rdfcyp/dont_worry_ive_got_all_day/

~~Poisoning~~ Context Clash

LLMs könnten unerwünschte Ideen aufnehmen und wiederholen

```
void addCust(String name, String email) {
        Map c = new HashMap();
        c.put("name", name);
        c.put("email", email);
        customers.add(c);
        System.out.println("Added cst: " + name);
    }
```

Dein alter Legacy-Code



```
void addCust(String name, String email) {
  customers.add(new Customer(
    name, email, new Date().toString(),
    UUID.randomUUID().toString(), null));
    System.out.println("cst added: " + name);
  }
```

Dein neuer Legacy-Code

Vorgaben explizit platzieren



359/360
trials follow the existing code, overriding the agent's own default

0% → 100%
adoption with the same number of examples in different files

59/60 vs. 38/60
a written rule redirects agents; a code example alone is unreliable

KI-Coding-Agenten kopieren standardmäßig bestehenden Code, lassen sich aber durch explizite Textregeln zuverlässig auf neue Konventionen umlenken.

Aus Haichao Zhu et al. „Precedent or Intent? What Coding Agents Follow When Evolving Existing Code" (26.10.2026)

Agentenoptimierte ADRs*

CLASSIC ADR

Nygard, 2011: written for a person

```
# 14. Image rendering

Status:  Accepted

Context:
```

We serve many images and care about load performance...

```
Decision:
```

We will use the framework's image component going forward.

```
Consequences:
```

Automatic optimization, lazy loading, some migration work.

Prose. A reader infers the rule and the scope.

optimize

AGENT-OPTIMIZED ADR

same decision: written for an agent

```
id:  R-IMG-001      status: Accepted

applies_to:  app/**/*.tsx

MUST      use next/image

MUST NOT     use <img>

SHOULD      set width + height

verify:

$ grep '<img' app/ | wc -l  →  0  ✓
```

Scoped · imperative · verifiable · token-tight.

Grafik via https://actual.ai/blog/agent-optimized-adrs

* Derzeit noch in Verprobung bei uns

Über Harness Input & Feedback einbringen



Context Engineering

Kontinuierliches, sinnvolles Anreichern der Wissensbasis in der Codebasis



Architectural Constraints

Überwachen durch LLM-basierte Agenten und deterministische Tools

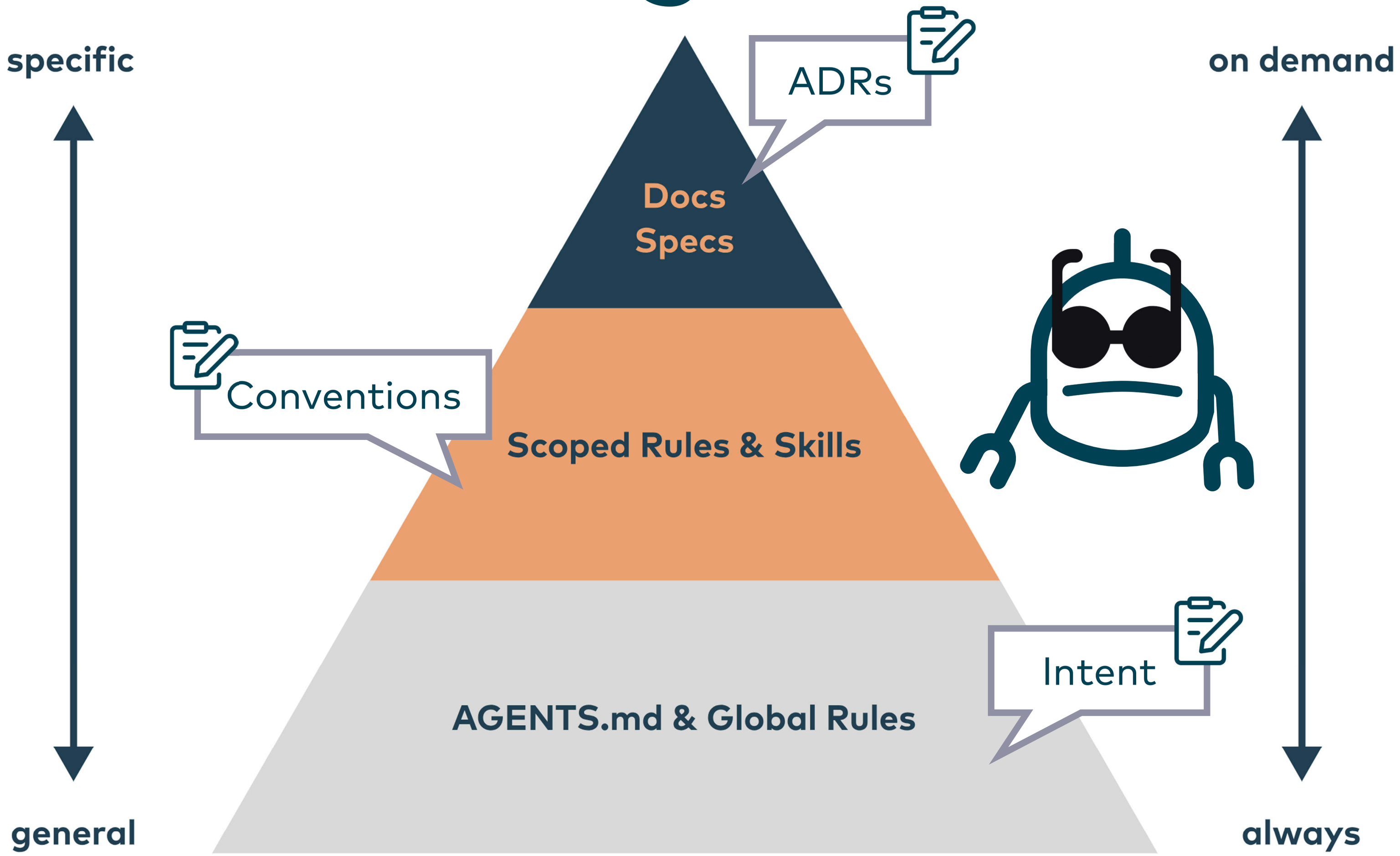


Garbage Collection

Aufspüren von globalen Inkonsistenzen gegen lokale Entropie
und schleichenden Verfall

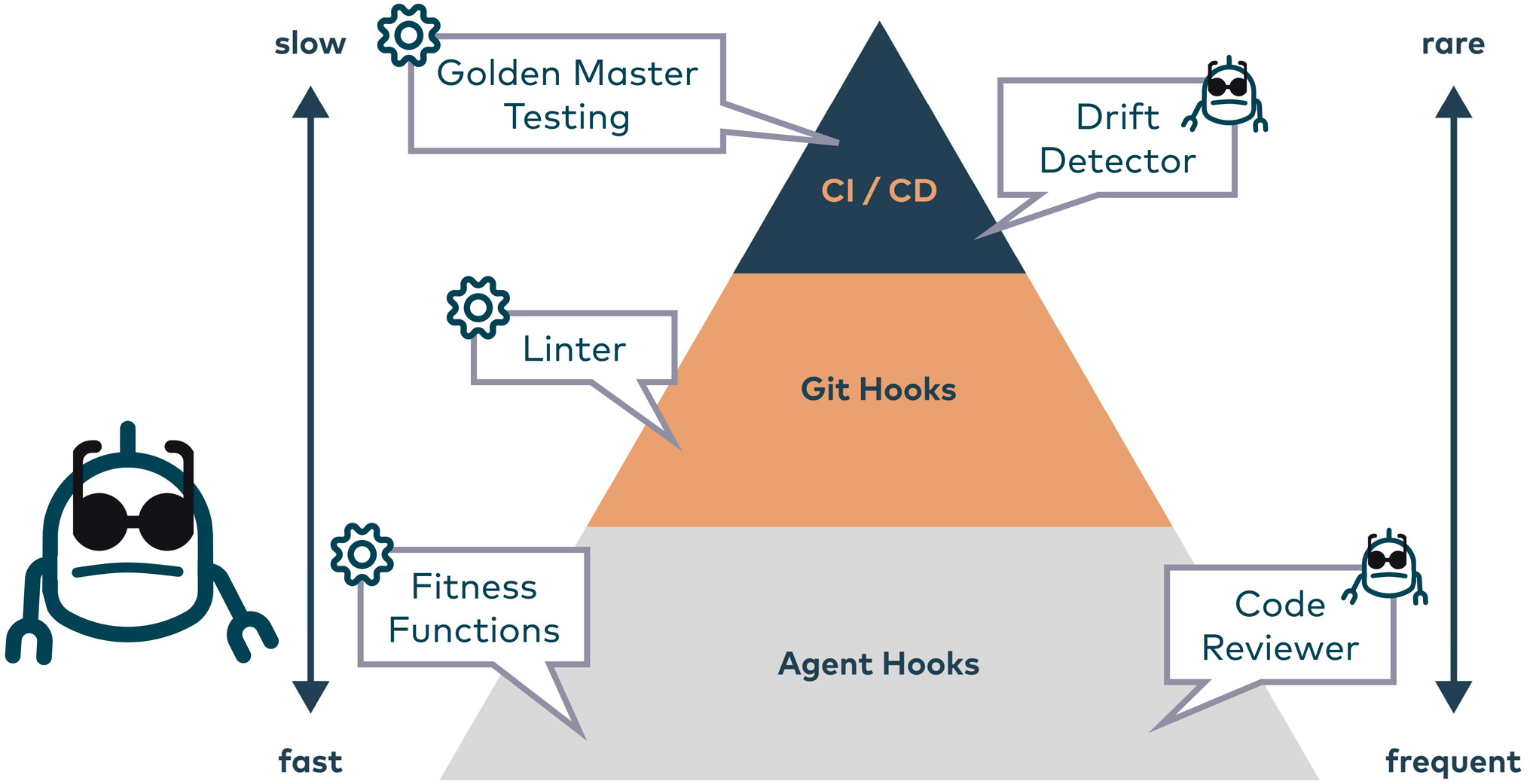
inspiriert durch Birgitta Böckeler
https://martinfowler.com/articles/exploring-gen-ai/harness-engineering.html

Input vorher an Agenten liefern



https://www.innoq.com/de/blog/2026/04/vom-vibe-coder-zum-agentic-engineer/

Feedback zurück an Agenten liefern



https://www.innoq.com/de/blog/2026/04/vom-vibe-coder-zum-agentic-engineer/

Fitness Functions

Feedback für den Agenten über ArchUnit

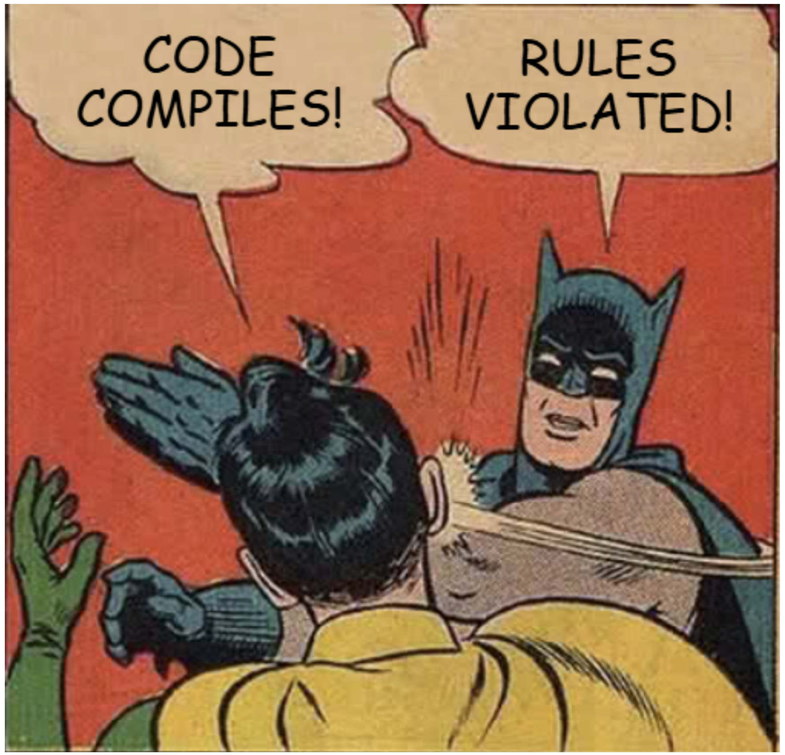
1. Erstellung von Architekturregeln mittels einer domänen-spezifischen Sprache (DSL) für strukturelle Checks

```
classes().that().areAnnotatedWith(Service.class)
         .or().haveNameMatching(".*Service")
         .should().resideInAPackage("..service..")
         .orShould().beAnnotatedWith(LegacyBridge.class);
```

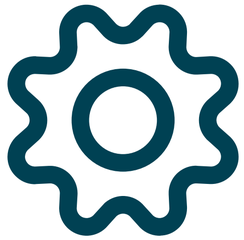
2. Prüfung von Architekturverletzungen zur Testzeit über einen Hook für den Agenten, der den Build darüber anstößt.

3. Interpretation und Rückmeldung von Architekturverletzungen über Subagent an den Hauptagenten

```
[INFO] -------------------------------------------------------
[INFO]  T E S T S
[INFO] -------------------------------------------------------
[INFO] Running com.example.ServiceLocationRuleFailingTest
[main] INFO com.tngtech.archunit.core.PluginLoader - Detected Java
version 25.0.2
[ERROR] Tests run: 1, Failures: 1, Errors: 0, Skipped: 0, Time elap
sed: 5.209 s <<< FAILURE! -- in com.example.ServiceLocationRuleFail
ingTest
[ERROR] com.example.ServiceLocationRuleFailingTest.allClassesMustFo
llowServiceLocationRule -- Time elapsed: 4.297 s <<< FAILURE!
java.lang.AssertionError:
Architecture Violation [Priority: MEDIUM] - Rule 'classes that are
annotated with @Service or have name matching '.*Service' should re
side in a package '..service..' or should be annotated with @Legacy
Bridge' was violated (2 times):
Class <com.example.controller.PaymentService> does not reside in a
package '..service..' in (PaymentService.java:0) and Class <com.exa
mple.controller.PaymentService> is not annotated with @LegacyBridge
 in (PaymentService.java:0)
Class <com.example.util.CacheManager> does not reside in a package
'..service..' in (CacheManager.java:0) and Class <com.example.util.
CacheManager> is not annotated with @LegacyBridge in (CacheManager.
java:0)
```



Goldene Regel für Harnesses



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



Torben Keller

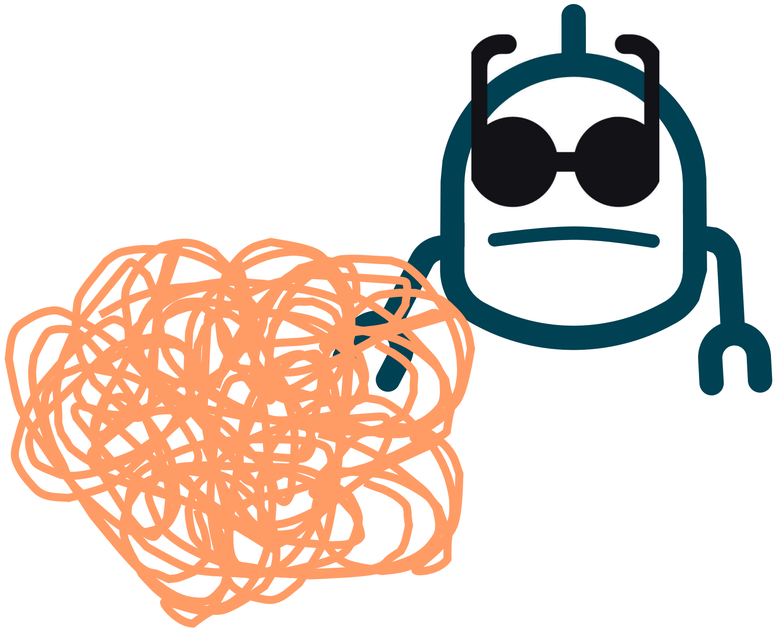
https://www.innoq.com/de/blog/2026/04/vom-vibe-coder-zum-agentic-engineer/

Zusammenfassung

Agentic Software Modernization

Was ihr bitte mitnehmt

- **Leinenlänge:** Prompt, Agent Skill, Werkzeugkasten oder generiertes Skript
- **Blast-Radius:** Zonen, Fitness Functions und Tests begrenzen den Schaden
- **Hybride Ansätze:** Geschickter Mix aus agentischer und menschlicher Arbeit



Und: **Probleme liegen nicht immer am Code!**

Vielen Dank!

Fragen, Diskussionen, Anmerkungen ?

Networking



Markus Harrer