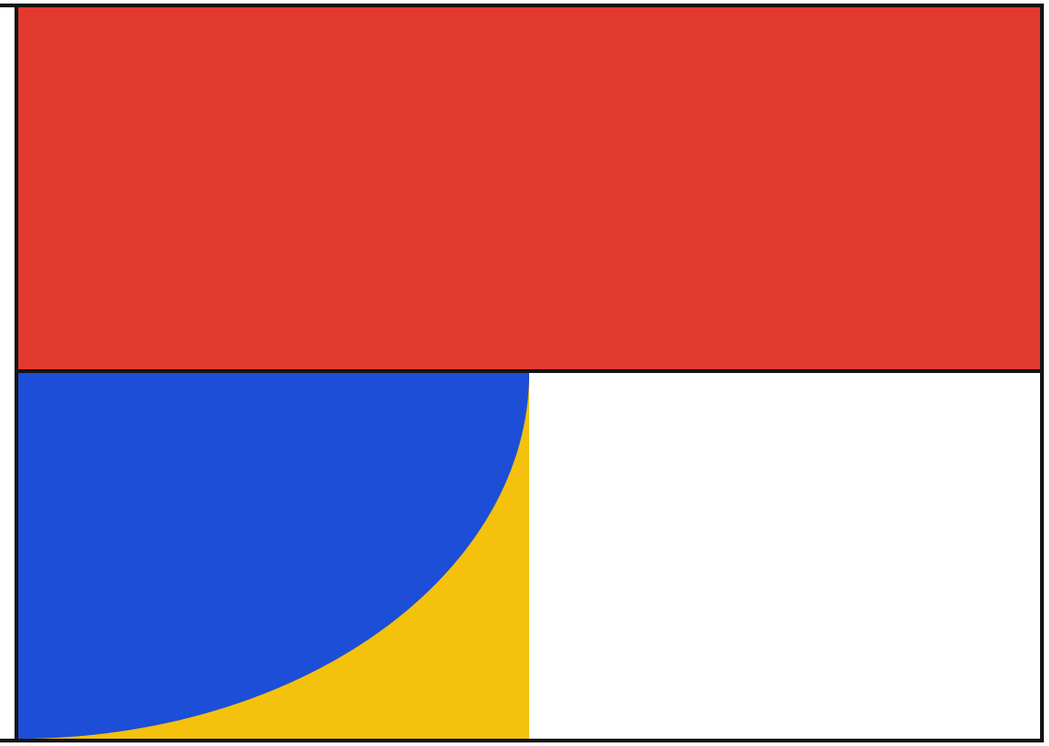
EIKE STÜRMER / AGENT-WORKFLOW

Vom Coding zum PR. Mit Agents.

Ein Setup, das mitwächst: vom ersten Tag bis zu Zero-Human-Review.

Diese Folien beschreiben einen Workflow, den ihr bei euch im Team genauso aufbauen könnt, unabhängig vom Stack. Drei Stufen, vier Bausteine, ein Lern-Loop. Mehr ist es nicht.



Warum es sich lohnt

Agents als Werkzeug helfen im Tagesgeschäft. Die drei Probleme, die Engineering-Workflows wirklich ausbremsen, lösen sie so nicht.

01

Review-Bottleneck

Pull requests warten Tage auf einen Menschen, der jede Zeile prüft. In einer guten Woche sind das vier, in einer schlechten zwei.

02

Inkonsistente Quality Gates

Was im einen PR auffällt, wird im nächsten übersehen. Niemand hat einen vollständigen Check-Katalog im Kopf.

03

Kein Lern-Loop

Was ein Agent gestern falsch gemacht hat, macht er morgen wieder. Ohne Gedächtnis bleibt jede Session ein neuer Versuch.

Die Frage, die das ganze Setup trägt: was ändert sich, wenn ein Agent den Workflow wirklich besitzt, statt nur zuzuarbeiten?

Der Ablauf in fünf Phasen

Jede größere Aufgabe läuft durch dieselben fünf Phasen. Jede hat einen klaren Owner und produziert etwas, auf das die nächste aufbaut.

PHASE	OWNER	OUTPUT
01 Discover (verstehen)	lokaler executor	Kontext, ähnliche Stellen, Constraints
02 Design (Entwurf)	`systems-architect`	Designvorschlag, der die Hierarchieebenen kennt
03 Implement (umsetzen)	executor / orchestrator	Code plus Tests, eigene Pre-Checks gefahren
04 Review (prüfen)	Bereich-Reviewer	Verdict im Standard-Vokabular
05 Verify (bestätigen)	`verifier`	PASS / FAIL mit Quality Report

Seitenphase Validate: läuft nur, wenn Daten-Korrektheit Teil der Acceptance Criteria ist. Dann sitzt sie zwischen Implement und Review als unabhängiger Daten-Check.

Vier Bausteine reichen

Egal wie komplex der Stack ist, das Setup zerfällt in genau vier Artefakt-Typen: Verhalten, Workflow, Fakten, Schnittstelle. Jede zusätzliche Achse macht das System schwerer zu warten.

Subagent prompts

Verhalten: wie ein Agent denkt, wann er stoppt, was er ausgibt. Eine Datei pro Rolle.

Skills

Workflows: Schritt-für-Schritt-Playbooks für wiederkehrende Aufgaben, einmal beschrieben.

AGENTS.md

Fakten: Konventionen pro Bereich oder Hierarchieebene im Repo. Closest-wins: die nächstliegende Datei hat Vorrang.

Handoff contract

Schnittstelle: standardisiertes Input-/Output-Format zwischen Phasen und Rollen. Reviewer werden austauschbar.

Phase 1: Mensch reviewt, Agent unterstützt

Hier starten die meisten Teams. Der Agent bereitet vor und schreibt den Code, der Mensch liest und merget. Das Setup dafür steht an einem Tag.

Was läuft beim Agent?

Discover: Kontext sammeln, relevante Dateien lesen, ähnliche Stellen suchen. **Implement:** Code schreiben, Tests dranhängen, eigene Checks laufen lassen. **First-pass review:** sich selbst nochmal lesen, bevor er übergibt.

Was bleibt beim Menschen?

Final review: jede Zeile, in Ruhe. **Merge approval:** der letzte Klick. Phase 1 ist kein Provisorium. Viele Teams bleiben hier dauerhaft, weil ihre Risiko-Toleranz nicht mehr hergibt. Das ist völlig in Ordnung.

Das braucht ihr an Tag 1: Reviewer-Rolle pro Bereich, unabhängiger Closing-Check (`verifier`), Workflow-Skill, `AGENTS.md` pro Hierarchieebene, Handoff-Contract als Stub, Journal-Hook. Der Hook muss ab Tag 1 laufen, sonst fehlt später der Lern-Loop.

Phase 2: Agents reviewen, Menschen approven

Sobald der Lern-Loop ein paar Wochen läuft, kommt der Schritt mit dem größten Geschwindigkeits-Sprung: ihr lest die Verdicts, nicht mehr den Code.

Was sich ändert

Bereich-Reviewer und `verifier` sind jetzt verpflichtende Closing Gates. Ohne deren Verdict geht der PR nicht durch. Ein `planner-reviewer` kommt vor Implement dazu, damit ihr früh merkt, wenn der Plan schon vorher krumm war.

Das größte Risiko

Agents übersehen Dinge, die ein Mensch beim Lesen sehen würde. Genau hier zahlt sich das Journal aus: jede übersehene Sache landet in einem Eintrag, und ihr seht die Muster über Wochen.

Zusätzlich nötig: handoff contract als Pflicht (Input-Packet plus Output-Vertrag), `planner-reviewer` aktiv, Pre-Handoff-Quality-Skill (CI grün, Diff im Budget). **Verdict-Vokabular:** `pass`, `pass-with-notes`, `fail`, `block`. Vier Werte, die jede Reviewer-Rolle nutzt. Damit ist parsebar, ob ein PR weiter darf.

Phase 3: Ihr klickt nur noch Merge

Der Zielzustand. Ihr seid Operator des Systems, das die PRs durchwinkt. Ein grüner Stand reicht, den PR muss niemand mehr öffnen.

Was sich ändert

Ihr klickt Merge, ohne den PR geöffnet zu haben. Der `verifier` ist fail-closed: ohne Quality Report kein Pass. CI status checks erzwingen alle Gates auf Plattform-Ebene. Auch wenn eine Reviewer-Rolle ausfällt, geht ohne den Status nichts durch.

Was das voraussetzt

Branch-Protection mit required status checks. Diff-Budget enforcement (z.B. maximal 400 geänderte Zeilen), damit große PRs gar nicht erst ankommen. Periodische Journal-Auswertung als Routine. Und ein Killswitch: bei Verdacht stoppt der `verifier` den Merge.

Phase 3 heißt: ihr vertraut dem System aus Reviewer, Verifier, CI-Gates und Journal. Der Mensch klickt Merge, weil das System grün gemeldet hat. Nicht, weil der Code gut aussieht.

Was jede Stufe freischaltet

Reihenfolge ist nicht optional. Was in einer Stufe steht, muss stabil laufen, bevor die nächste startet.

0

Setup

Die sieben Artefakte liegen im Repo. Journal-Hook läuft ab Tag 1.

1

Mensch reviewt

Agent macht Discover, Implement, First-pass review. Freigeschaltet wird Phase 2 durch: handoff contract als Gate, planner-reviewer aktiv, ~50 saubere Journal-Einträge.

2

Agent reviewt

Mensch approved, liest kein Line-Review mehr. Freigeschaltet wird Phase 3 durch: Branch-Protection, Diff-Budget, Journal-Loop produktiv und über Wochen stabil.

3

Zero-Human-Review

Alle Phasen beim Agent, fail-closed Gates. Danach: teamweites second brain, Auto-Tuning der Prompts aus den Befunden.

Das Agent-Journal: ein second brain

Ein append-only Journal, in dem jedes wichtige Task-Outcome als kurzer Eintrag landet. Ein Index über die Arbeit, kein volles Transcript.

Was reingehört

Welche subagents genutzt wurden, wie oft und wofür. Welche skills geladen wurden. Was der Task war, was der Outcome. Was funktioniert hat, was nicht. Welche Fehler der Agent gemacht hat, explizit benannt statt beschwichtigt.

Was nicht reingehört

Das volle Transcript (das liegt separat als Rohdaten-Schicht). Secrets oder Credentials. Erfundene Zahlen: `unknown` ist immer besser als geraten.

Der Agent schreibt den Eintrag selbst, am Ende jeder Session. Strukturierte Felder plus kurze Freitext-Notizen. Niemand korrigiert alte Einträge. Falsche Daten zu sehen ist ein Feature, kein Bug.

Der Lern-Loop läuft lokal, ohne extra System

Vier Komponenten, alle lokal, alle ohne Vendor-Backend. Die Einträge füllen sich von selbst, weil die IDE bei jedem Tool-Call einen Hook feuert.

01

IDE Hook Layer

Bei jedem Tool-Call, Subagent-Start und Shell-Run feuert die Agent-Umgebung ein Event. Stdin ist JSON, ein Tap-Script reicht für alle Events.

02

Append-only Log

Eine JSONL-Datei, eine Zeile pro Event. Schemafrei, Git-diffbar, unempfindlich gegen Schema-Drift.

03

Lokale SQL-Engine

Eine embedded Analytics-Engine liest die JSONL direkt. Reports und Co-Occurrence-Zählungen sind je eine SQL-Query.

04

Trace Viewer

Pro Session ein Flame-Graph: subagents als Threads, Tool-Calls als Spans. Die Konvertierung ins Trace-Format ist ein kleines Skript.

Das Log-Schema folgt der offenen GenAI-Telemetrie-Konvention. So bleibt der Sprung zu einem gehosteten Observability-Backend später eine Konfig-Frage, kein Rewrite. Erst lokal nutzen, dann entscheiden.

Wie das Journal Verbesserung treibt

Stellt euch 500 journalierte Sessions vor. Sobald ihr auszählt, tauchen zwei Klassen von Befunden auf. Beide sind direkt handlungsleitend.

A

Befund A: Workflow-Form

Beispiel: das Paar context-explorer und `verifier` taucht 80 Mal als Co-Occurrence auf. Das ist eure kanonische Workflow-Form: Kontext sammeln, bauen, bestätigen. **Aktion:** fehlt diese Form bei einer Aufgabe, lohnt der Blick in den Prompt. Warum überspringt der Agent eine Phase, die sonst die Norm ist?

B

Befund B: Anti-Pattern

Beispiel: `systems-architect` hat 0 mentions in 500 Sessions. Die Design-Phase ist unsichtbar. **Aktion:** entweder findet kein Design statt, dann fixt den Trigger. Oder es findet ohne Spur beim executor statt, dann fixt die Routing-Regel.

Das Journal ist der Feedback-Loop, der Phase 2 zu Phase 3 sicher macht. Ohne Journal ist Zero-Human-Review eine Hoffnung, kein System.

Sieben Artefakte. Mehr nicht.

Die Checkliste für Tag 1. Wie ihr die Hierarchieebenen schneidet (Domain, Layer, Service), entscheidet ihr selbst. Wichtig ist nur: die Dateien liegen da, wo der Code liegt.

ARTEFAKT	WO ES LEBT	WARUM ES EXISTIERT
`AGENTS.md` pro Hierarchieebene	verstreut im Repo, je Bereich	Closest-wins. Der Agent liest die nächstliegende Datei zuerst.
Bereich-Reviewer-Prompt	eine Datei pro Reviewer-Rolle	Bereichsspezifische Bug-Detection. Ein generischer Code-Reviewer findet höchstens die Hälfte.
`verifier`-Prompt	eine Datei, unabhängige Rolle	Fail-closed Closing-Check. Kein Pass ohne Quality Report.
Workflow-Skill	ein Playbook pro Workflow	Macht den Workflow wiederholbar. Auch für den Agent von nächster Woche.
Shared handoff contract	ein zentrales Dokument	Macht Reviewer austauschbar. Output ist parsebar, nicht nur lesbar.
Journal-Capture-Hook	Sessionende-Skript schreibt den Eintrag	Der Lern-Loop ab Tag 1. Rückwirkend lässt sich das Journal nicht füllen.
Hook-Tap plus lokales Logging	IDE-Hook, JSONL, SQL-Engine	Macht aus dem Journal eine abfragbare Datenbasis. Lokal-first, offene Telemetrie-Konvention.

Phase 3 ist die Vorbedingung, nicht das Ende.

Sobald Zero-Human-Review stabil läuft, trägt das Repo weitere agentic Workflows: Klarsprache-Queries für Stakeholder, Auto-Doku fürs Onboarding, Verfügbarkeits-Checks für Operations, Reports auf Knopfdruck. Reihenfolge ist nicht optional: erst die Leiter, dann die Plattform.

EIKE STÜRMER