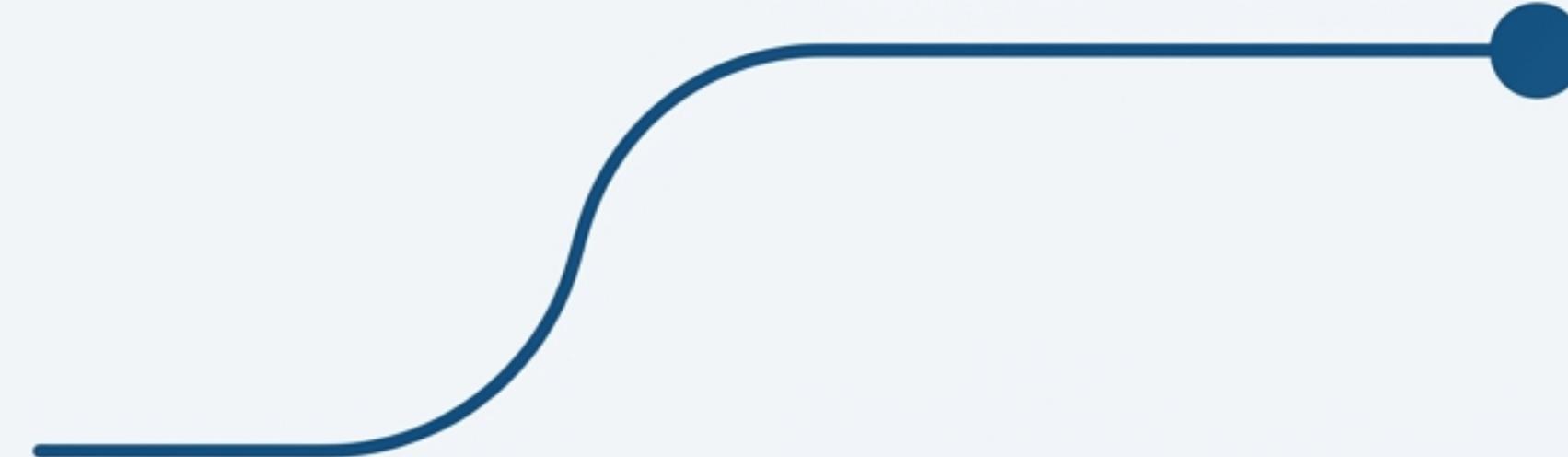


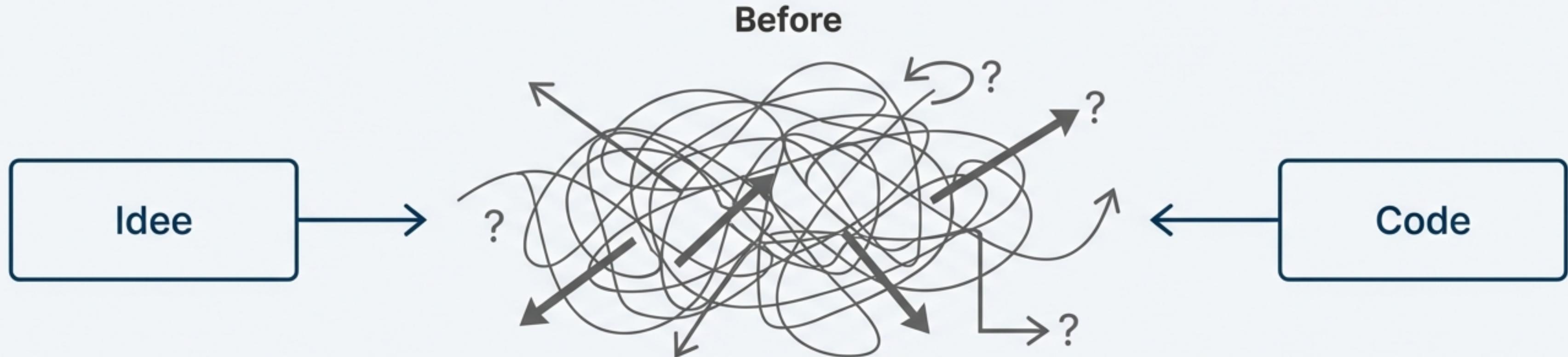


RPIQPI: Disziplinierte Softwareentwicklung mit KI-Agenten

Ein systematisches Framework für planbare, nachvollziehbare
und qualitativ hochwertige Ergebnisse.



Das Dilemma der KI-Entwicklung: Mächtig, aber oft chaotisch



Das Potenzial

KI-Agenten versprechen eine enorme Beschleunigung. Sie können Code generieren, refaktorieren und komplexe Probleme in Minuten lösen.

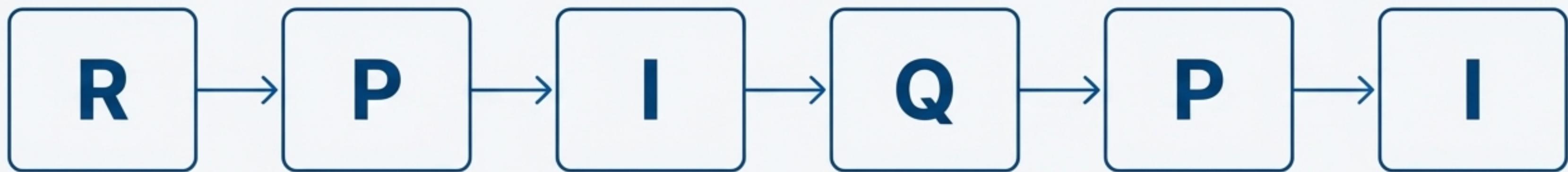
Die Realität ohne Framework

Ohne ein diszipliniertes System führt dies oft zu:

- **Unstrukturierter Entwicklung:** Agenten improvisieren, was zu inkonsistentem und schwer wartbarem Code führt.
- **Mangelnder Nachvollziehbarkeit:** Änderungen sind schwer auf ursprüngliche Anforderungen zurückzuführen. Der „Warum“-Kontext geht verloren.
- **Ineffizienten Iterationen:** Große Kontextfenster führen zu Fehlern, Wiederholungen und unvorhersehbaren Ergebnissen.
- **Scope Creep:** Agenten „halluzinieren“ Features, die nicht Teil des ursprünglichen Plans waren.

Die Lösung: RPIQPI bringt Ingenieursdisziplin in die KI-Entwicklung

RPIQPI (Research, Plan, Implement, QA, Plan, Implement) ist ein Framework, das professionelle Software-Engineering-Workflows auf die Arbeit mit KI-Agenten anwendet. Es ersetzt Chaos durch einen systematischen, phasengesteuerten Prozess.



Trennung der Belange (Separation of Concerns)

Jeder Agent hat eine klar definierte, spezialisierte Aufgabe.

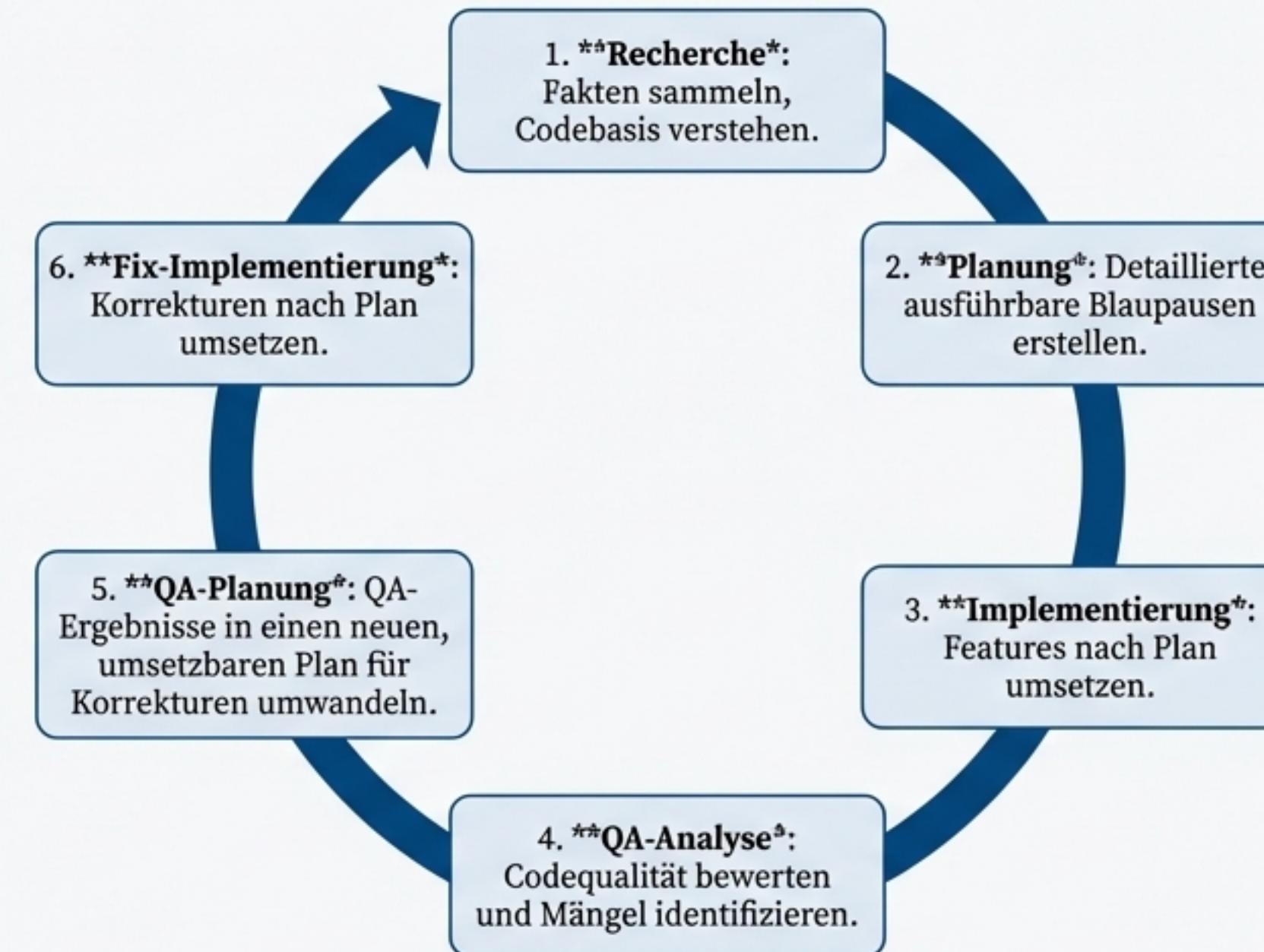
Evidenzbasierte Planung (Evidence-Based Planning)

Jede Entscheidung basiert auf Fakten aus dem Code, nicht auf Annahmen.

Phasengesteuerte Umsetzung (Phase-Gated Implementation)

Keine Implementierung ohne Plan, keine Planänderung ohne Analyse.

Der RPIQPI-Lebenszyklus: Ein geschlossener Kreislauf für kontinuierliche Verbesserung



Der RPIQPI-Zyklus stellt sicher, dass sowohl neue Features als auch Qualitätsverbesserungen dem gleichen rigorosen, planbasierten Prozess folgen. QA ist kein Endpunkt, sondern der Beginn der nächsten gezielten Verbesserungsiteration.

Der Bauplan: Von der Vision zum umsetzbaren Epic

Greenfield-Workflow

Für brandneue Projekte oder komplett neue Features in bestehenden Anwendungen stellt der Greenfield-Workflow sicher, dass eine klare, geteilte Vision existiert, bevor eine einzige Zeile Code geschrieben wird. Der Prozess zerlegt eine Idee systematisch in ihre Kernbestandteile.



- **Vision vor Code:** Stellt ein gemeinsames Verständnis sicher (Mission → Spec → Epics).



- **Technologie-agnostische Planung:** Der „Was“-Fokus wird vom „Wie“ getrennt. Die Architektur wird abstrakt definiert, bevor Technologien gewählt werden.



- **Nachvollziehbare Anforderungen:** Jede User Story lässt sich auf die ursprüngliche Vision zurückführen.



- **Klare Trennung der Belange:** Mission-Architect (Wert), Specifier (Architektur), Epic-Planner (Zerlegung).

Die Phasen des Greenfield-Workflows

Stufe 1: Mission



Agent: Mission-Architect in Source Serif Pro

Prozess: Führt einen kollaborativen Dialog mit dem Benutzer, um Vision, Wertversprechen und Abgrenzungen zu definieren. Fragt nach dem „Warum“ und „Was“.

Output:  **thoughts/shared/missions/YYYY-MM-DD-[Project-Name].md**
(Mission Statement)



Stufe 2: Spezifikation



Agent: Specifier in Source Serif Pro

Prozess: Übersetzt das Mission Statement in eine abstrakte, technologie-agnostische Architektur, konzeptionelle Datenmodelle und Schnittstellen.

Output:  **thoughts/shared/specs/YYYY-MM-DD-[Project-Name].md**
(Spezifikation)



Stufe 3: Epics



Agent: Epic-Planner in Source Serif Pro

Prozess: Zerlegt die Spezifikation in funktionale, Story-basierte Epics. Formuliert Recherchefragen für den Researcher und Akzeptanzkriterien für den Planner.

Output:  **thoughts/shared/epics/YYYY-MM-DD-[Epic-Name].md**

Technologie-Agnostisch

In den Phasen 1-3 werden keine Entscheidungen über Frameworks, Datenbanken oder Programmiersprachen getroffen. Fokus liegt rein auf der Funktionalität und Architektur.

Effiziente Umsetzung: Die Controller/Executor-Architektur

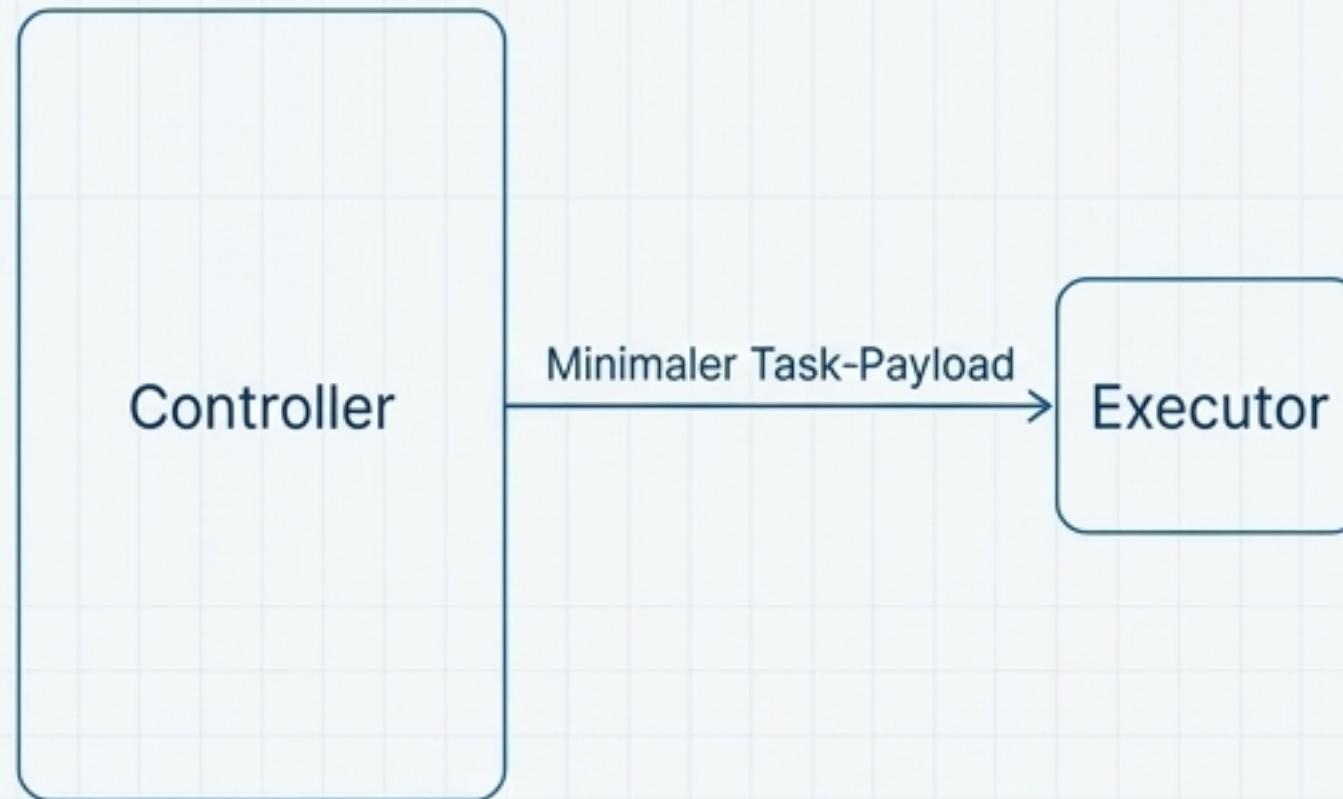
Implementierungs-Workflow

Problemstellung: Monolithische Agenten, die den gesamten Plan und alle relevanten Dateien in einem Kontextfenster halten, sind ineffizient, fehleranfällig und teuer.

Die RPIQPI-Lösung: Eine Zwei-Agenten-Architektur zur radikalen Kontextreduktion und Erhöhung der Zuverlässigkeit.

Implementation-Controller (Orchestrator)

- Lädt den Plan und die STATE-Datei.
- Extrahiert jeweils nur EINEN Task (PLAN-XXX').
- Delegiert die Code-Änderung an den Task-Executor.
- Führt Verifizierung durch, aktualisiert den Status und committet.



Task-Executor (Builder)

- Erhält einen minimalen Payload mit nur einem Task.
- Liest nur die Zielfile, implementiert die Änderung.
- Hat keinen Zugriff auf den Gesamtplan oder den Status.

Trennung der Belange

Der Controller orchestriert, der Executor implementiert. Fehler im Executor kontaminieren nicht den Kontext des Controllers.

Präzise Ausführung mit minimalem Kontext

Monolithischer Ansatz

Gesamter Plan (~500 Zeilen)

STATE-Datei (~30 Zeilen)

Zieldateien (~200 Zeilen)

Gesamtkontext: ~730 Zeilen



Fehlerisolierung: Fehler des Executors sind isoliert und können gezielt wiederholt werden.



Git als Beweismittel: Jeder erfolgreich abgeschlossene Task entspricht einem Git-Commit ('PLAN-XXX: <description>'). Die Git-Historie wird zum Audit-Trail.

RPIQPI Executor-Kontext

Einzelner Task-Payload
(~50 Zeilen)

Zieldateien (~200 Zeilen)

Gesamtkontext: ~250 Zeilen



Pausier- und Fortsetzbar: Der Workflow kann jederzeit unterbrochen und dank der STATE-Datei nahtlos wieder aufgenommen werden.

Qualität als System: Der integrierte QA-Workflow

Qualitätssicherungs-Workflow

In RPIQPI ist Qualitätssicherung kein manueller, isolierter Schritt am Ende des Prozesses. Es ist ein integrierter, agentengesteuerter Kreislauf, der Analyse nahtlos in umsetzbare Verbesserungen überführt.

👉 **@python-qa-quick / @typescript-qa-quick**

Für schnelles Feedback während der Entwicklung. Führt automatisierte Checks (ruff, pyright, bandit / tsc, eslint, knip) aus und liefert eine sofortige, umsetzbare Aufgabenliste.

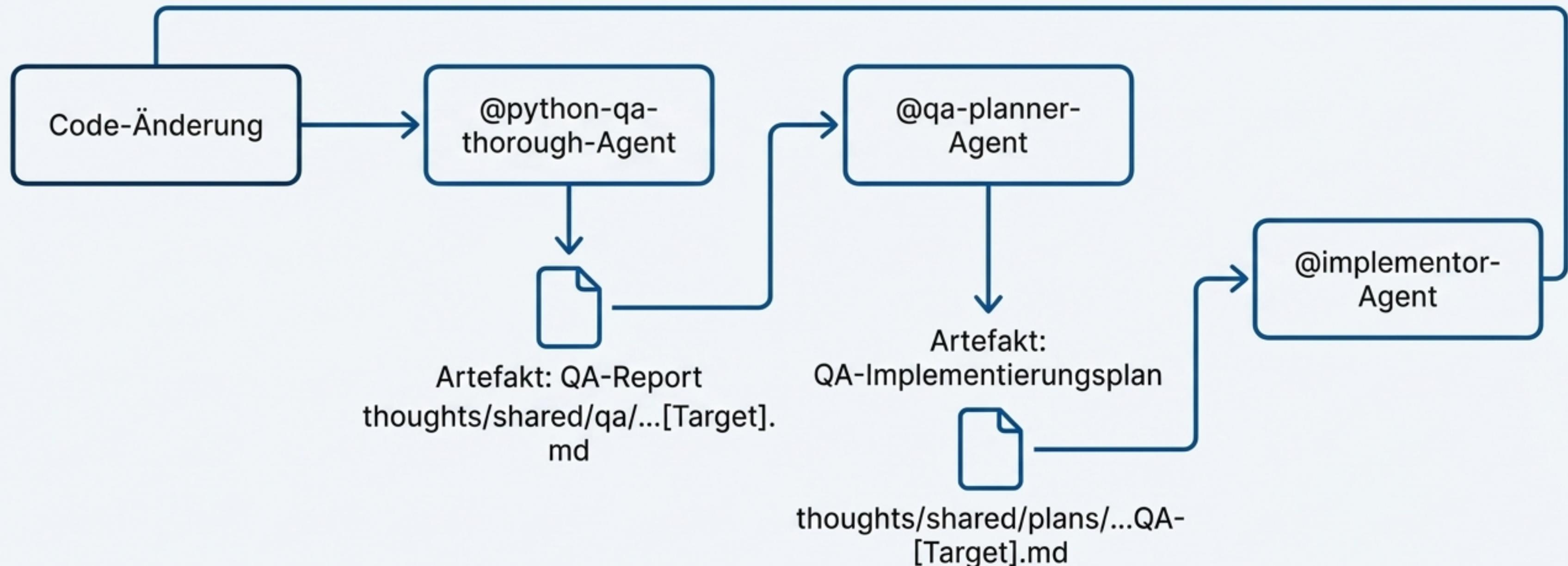
👉 **@python-qa-thorough / @typescript-qa-thorough**

Für umfassende Reviews. Kombiniert automatisierte Tools mit manueller Analyse von Lesbarkeit, Wartbarkeit und Testbarkeit. Das Ergebnis ist ein detaillierter QA-Report.

Die QA-zu-Implementierungs-Brücke

QA-Befunde werden nicht nur gemeldet, sondern systematisch in einen neuen Implementierungsplan umgewandelt.

Vom Befund zum Fix: Die QA-zu-Implementierungs-Brücke



Dieser Prozess stellt sicher, dass die Behebung von Qualitätsproblemen der gleichen Disziplin und Nachvollziehbarkeit unterliegt wie die Entwicklung neuer Features.

Das Agenten-Team: Eine Hierarchie von Spezialisten

Das RPIQPI-Framework besteht aus 12+ spezialisierten Agenten, die in zwei Kategorien fallen, um eine klare Aufgabentrennung zu gewährleisten.

Primär-Agenten (Orchestratoren)

Dies sind die Hauptagenten, mit denen Sie direkt interagieren. Sie steuern die übergeordneten Workflows, treffen strategische Entscheidungen und delegieren Aufgaben. Sie sind über die `Tab`-Taste in OpenCode umschaltbar.

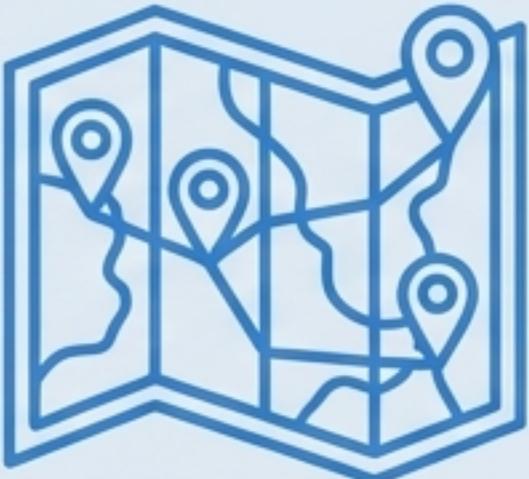
- Mission-Architect
- Specifier
- Epic-Planner
- Researcher
- Planner
- Implementor (Implementation-Controller)
- Python-QA-Quick / Thorough
- TypeScript-QA-Quick / Thorough

Sub-Agenten (Spezialisten)

Diese Agenten sind spezialisierte Arbeiter, die von den Primär-Agenten aufgerufen werden, um fokussierte Aufgaben auszuführen. Sie interagieren normalerweise nicht direkt mit dem Benutzer.

- Task-Executor
- Codebase-Locator
- Codebase-Analyzer
- Codebase-Pattern-Finder
- Web-Search-Researcher
- Thoughts-Analyzer / Locator

Die Spezialisten: Fokussierte Sub-Agenten im Detail



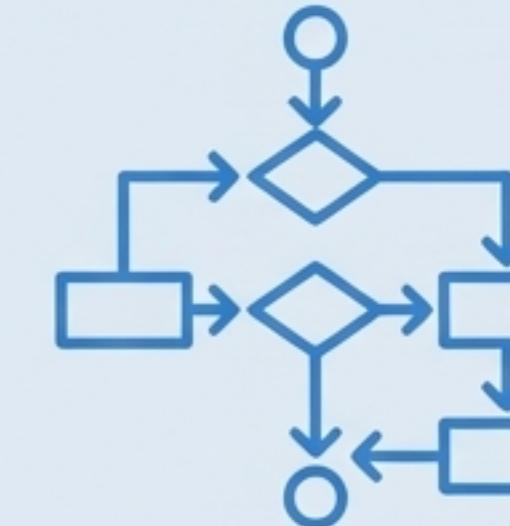
Codebase-Locator ("Der Kartograph")

Aufgabe

- Findet wo sich Code befindet.
Spezialisiert auf Dateisystem-Topologie, Pfade und Verzeichnisstrukturen.

Einschränkung

- Analysiert keine Code-Logik.



Codebase-Analyzer ("Der Logik-Tracer")

Aufgabe

- Versteht wie Code funktioniert. Verfolgt Ausführungspfade, Datenflüsse und Abhängigkeiten innerhalb von Dateien.

Einschränkung

- Kann nicht suchen; folgt nur expliziten Pfaden.



Codebase-Pattern-Finder ("Der Muster-Bibliothekar")

Aufgabe

- **Aufgabe:** Identifiziert wie etwas üblicherweise implementiert wird. Findet Nutzungsmuster, Idiome und konkrete Code-Beispiele.

Einschränkung

- **Einschränkung:** Bewertet nicht die Qualität der Muster, sondern katalogisiert sie nur.



Web-Search-Researcher ("Der externe Scout")

Aufgabe

- **Aufgabe:** Beschafft externes Wissen. Recherchiert Bibliotheken, APIs und Best Practices außerhalb der Codebase.

Einschränkung

- Hat keinen Zugriff auf den lokalen Code.

Der Mehrwert: Wie RPIQPI die Kernprobleme löst

Herausforderung

Chaotische, unstrukturierte Entwicklung

Nicht nachvollziehbare Änderungen & Kontextverlust

Ineffiziente LLM-Nutzung & hohe Kosten

Mangelnde Qualitätssicherung

Unklare Vision und Scope Creep

RPIQPI-Lösung

Phasengesteuerte Workflows: Jeder Schritt hat einen definierten Zweck und ein klares Ergebnis (Mission → Spec → Epic → Plan → Implement).

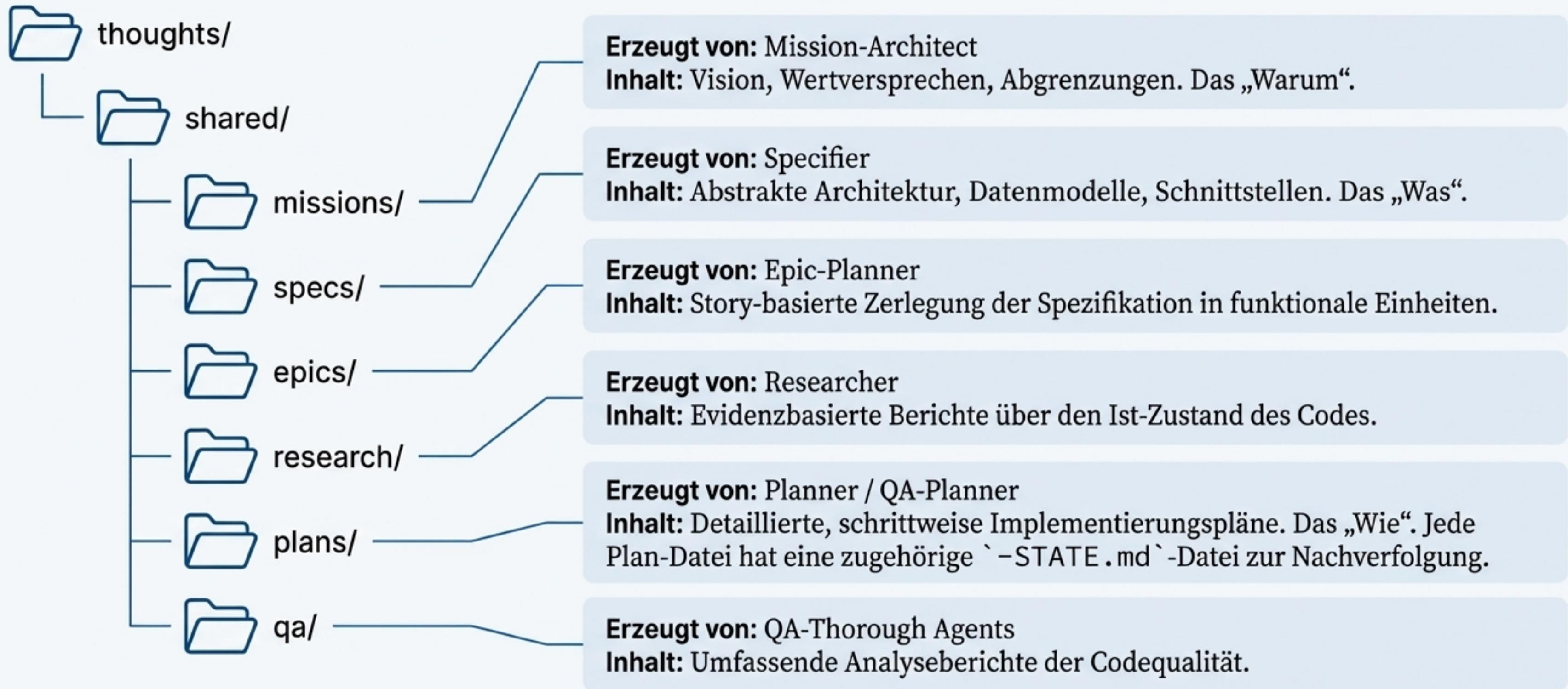
Git als Evidenz & Artefakt-Kette: Jeder Task ist ein Commit. Jede Anforderung ist von der Mission bis zum Code rückverfolgbar.

Controller/Executor-Architektur: ~66% Kontextreduktion, Fehlerisolierung und gezielte Wiederholungslogik.

Integrierter QA-Kreislauf: Systematische Umwandlung von QA-Befunden in ausführbare Pläne.

Greenfield-Workflow: Erzwingt die Klärung von Vision, Wert und Abgrenzungen vor der Implementierung.

Die Anatomie eines RPIQPI-Projekts: Artefakte und Struktur



Mehr als nur Code-Generierung: Engineering für das KI-Zeitalter.

RPIQPI ist ein Bekenntnis zu den Prinzipien, die robuste und wartbare Software seit jeher auszeichnen: Struktur, Nachvollziehbarkeit und Qualität. Es ist das Framework, um die transformative Kraft von KI-Agenten in einen zuverlässigen, professionellen Engineering-Prozess zu kanalisieren.

Planbarkeit

Durch Greenfield &
Planning Workflows

Nachvollziehbarkeit

Durch Git-as-Evidence &
Artefakt-Kette

Qualität

Durch integrierten
QA-Kreislauf

OpenCode

Built with OpenCode - The open source AI coding agent.