

Von 6 Monaten auf 1 Woche: Professionelles Engineering im KI-Zeitalter

Wie ein systematischer Framework die Spielregeln
der Softwareentwicklung neu definiert.



Die Ausgangsfrage: Was kostet ein professionelles Desktop-Produkt?



Definieren wir ein konkretes Projekt:
Eine hochwertige Desktop-Anwendung.

Technologie-Stack: Electron, TypeScript, SQLite.

Umfang: ≈ 13.000 Lines of Code (LOC).

Anspruch: Nicht nur Code, sondern professionelles Engineering.

- ✓ Migrations-Logik für die Datenbank.
- ✓ Hohe Testabdeckung (≈ 90% Coverage, Unit & E2E).
- ✓ Performance- & Security-Tests.
- ✓ Saubere CI-Pipeline und Qualitätssicherung.

Die Experten-Schätzung: Wo die Zeit wirklich hingeht

Eine realistische Aufwandsspanne für einen einzelnen, sehr guten Senior-Entwickler.

1. Produkt-/Domänen-Denkarbeit

10–25 Personentage

(Datenmodell, Flows, Edge Cases, UX-Entscheidungen)



3. Tests auf hohem Niveau

15–35 Personentage

(Unit-Tests, E2E-Setup, Erreichen von 90% Coverage)



5. Qualitätssicherung / DevEx

10–20 Personentage

(CI-Pipeline, Linting, Build/Packaging)



2. Kern-Implementierung

20–45 Personentage

(Architektur, State/IPC, Persistenz, Migrationslogik)

4. Performance & Security

5–15 Personentage

(Profiling, Bottlenecks, Hardening)

Gesamtschätzung

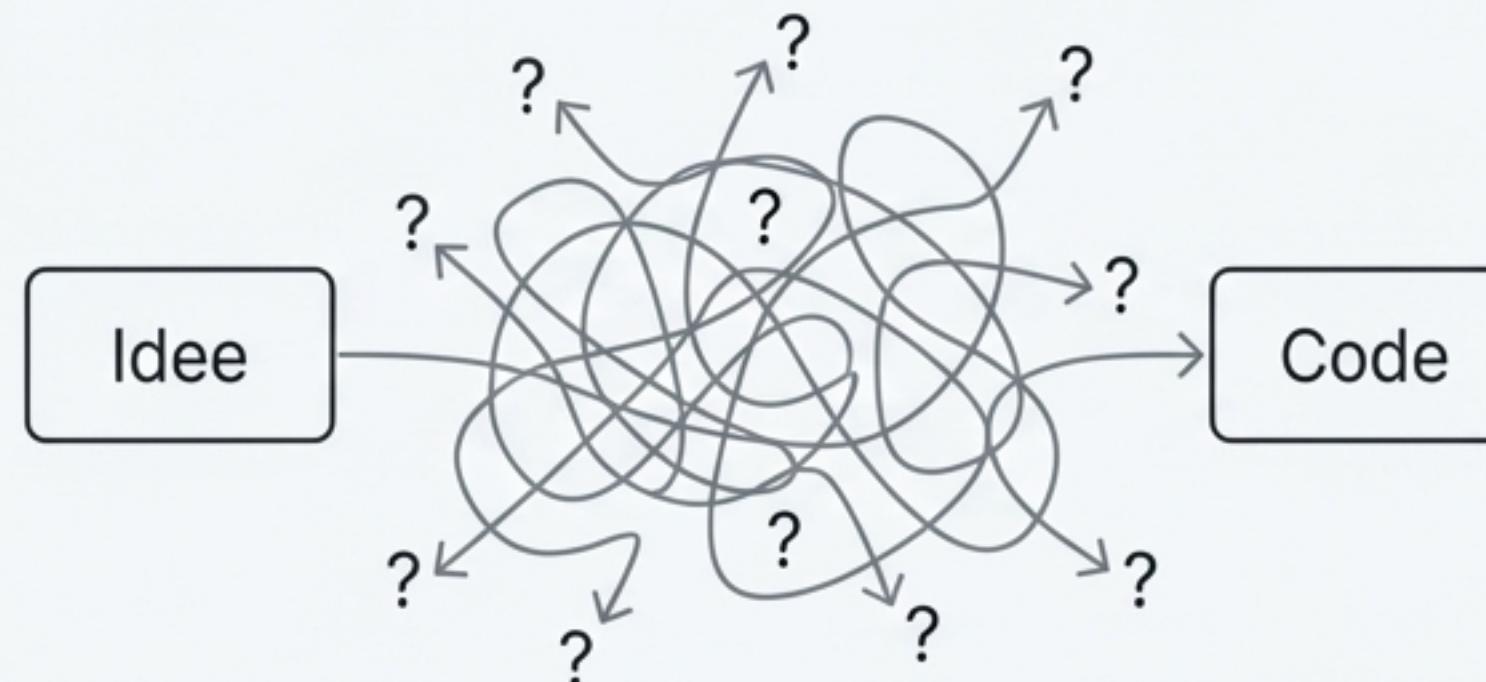
60–120 Personentage

entspricht 3–6 Monaten Vollzeit

**“DAS PROJEKT WURDE VON EINER KI
INNERHALB VON EINER WOCHE ERSTELLT.”**

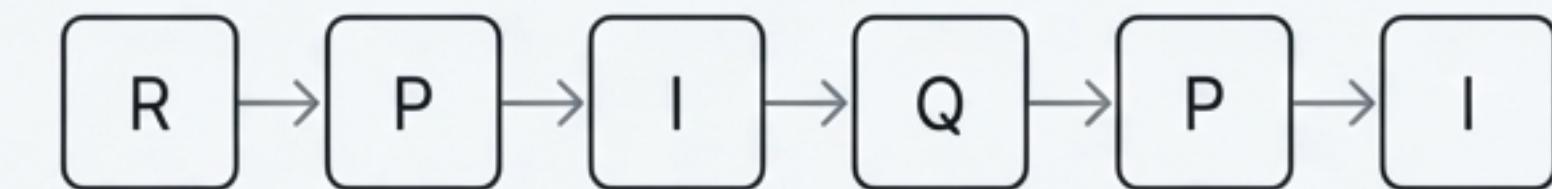
Das Dilemma der KI-Entwicklung: Mächtig, aber oft chaotisch

Das Problem (Söhne Kraftig)



Die Realität ohne Framework: Unstrukturierte Entwicklung, mangelnde Nachvollziehbarkeit, ineffiziente Iterationen und Scope Creep durch "halluzinierende" Agenten.

Die Lösung (Söhne Kraftig)



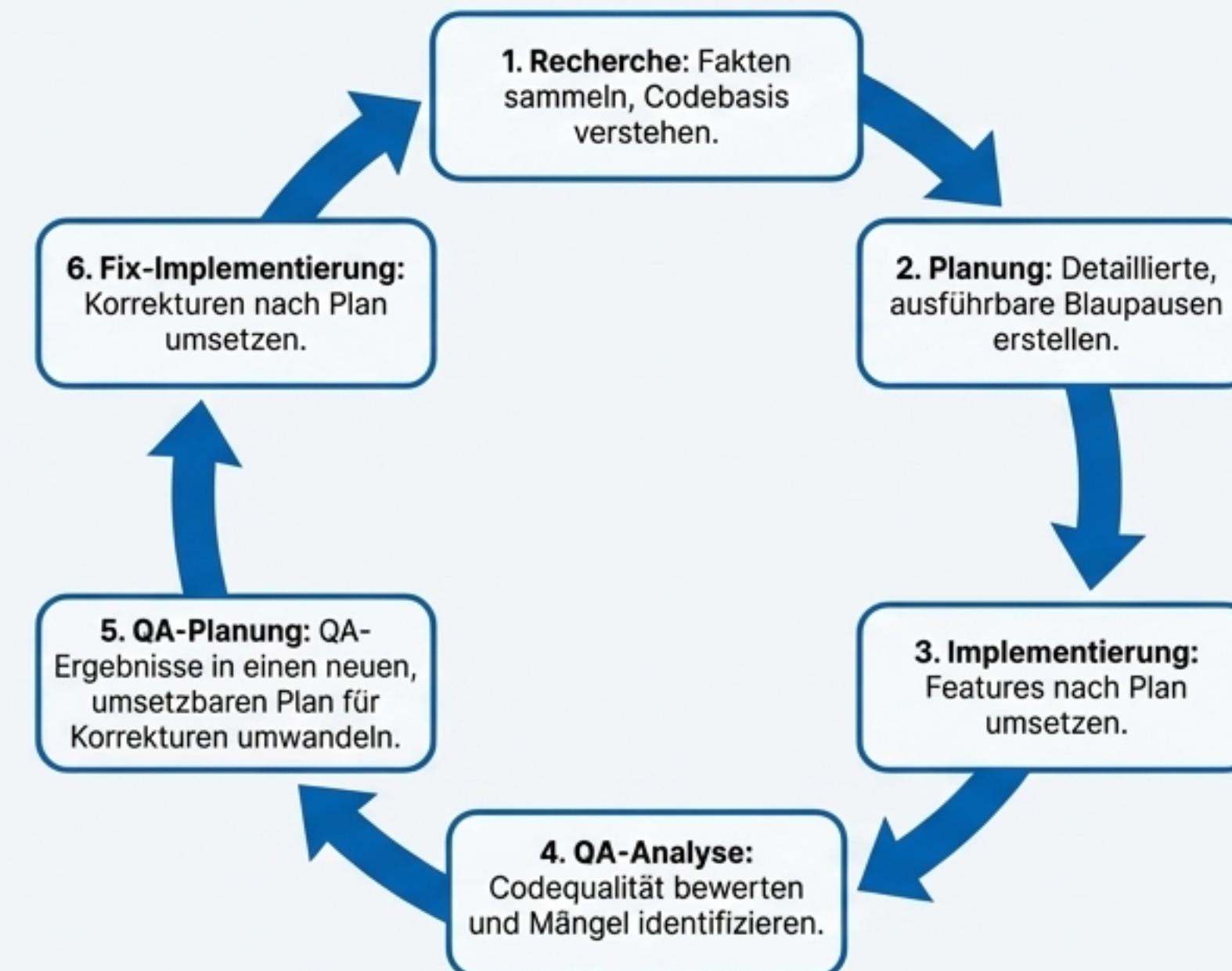
RPIQPI (Research, Plan, Implement, QA, Plan, Implement) ersetzt Chaos durch einen systematischen, phasengesteuerten Prozess.

Trennung der Belange

Evidenzbasierte Planung

Phasengesteuerte Umsetzung

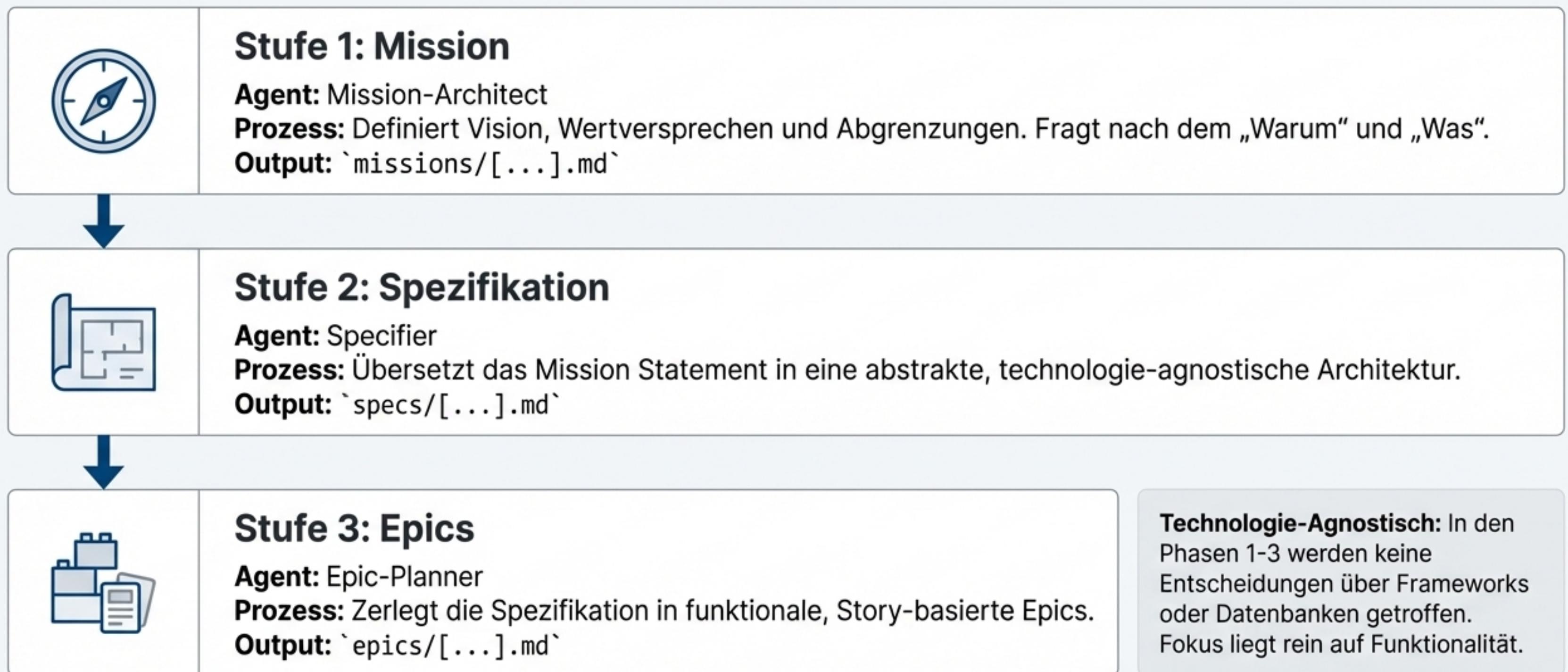
Der RPIQPI-Lebenszyklus: Ein geschlossener Kreislauf für kontinuierliche Verbesserung



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 Iteration.

Der Bauplan: Von der Vision zum umsetzbaren Epic

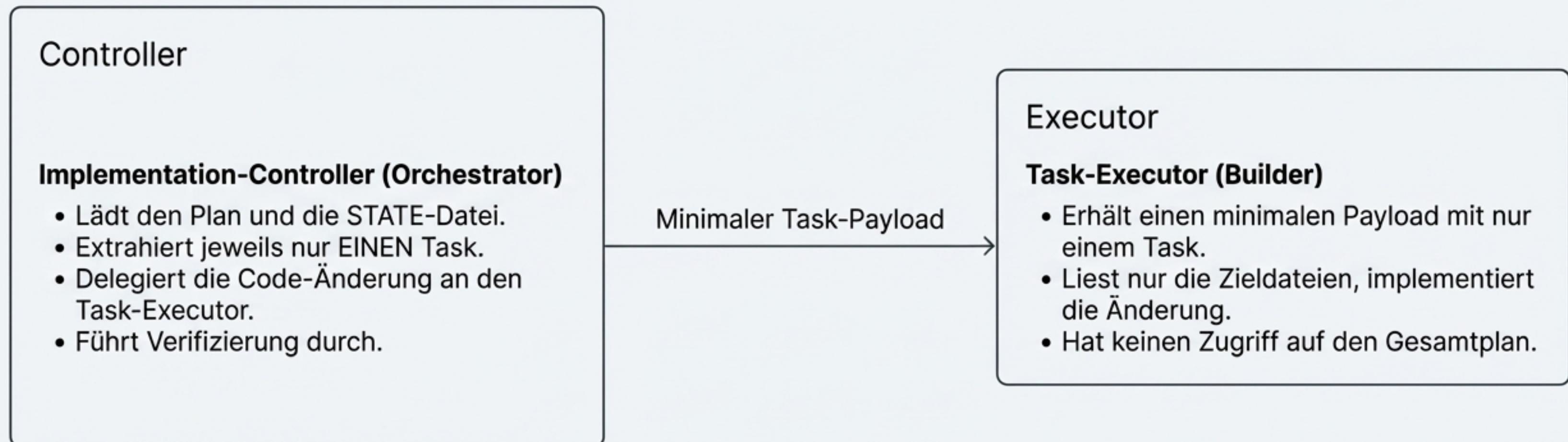
Der Greenfield-Workflow



Effiziente Umsetzung: Die Controller/Executor-Architektur

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

Die RPIQPI-Lösung: Eine Zwei-Agenten-Architektur zur radikalen Kontextreduktion.



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 Task entspricht einem Git-Commit dedenker Git-Commit ('PLAN-XXX: <description>'). Die Git-Historie wird zum Audit-Trail.

~66%

Reduktion des LLM-Kontexts pro Task

RPIQPI Executor-Kontext

Einzelner Task-Payload (~50 Zeilen)

Zieldateien (~200 Zeilen)

Gesamtkontext: ~250 Zeilen



Pausier- und Fortsetzbar: Der Workflow kann jederzeit das Workflow und unterbrochen und nahtlos wieder aufgenommen werden.

Qualität als System: Der integrierte QA-Workflow

In RFiQPI ist Qualitätssicherung kein manueller, isolierter Schritt am Ende des Prozesses. Es ist ein integrierter, agentengesteuerter Kreislauf.

👉 Söhne Kraftig

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

- Purpole: Für schnelles Feedback während der Entwicklung.
- Prozess: Führt automatisierte Checks aus (ruff, pyright, tsc, eslint, etc.).
- Resultilt: Eine sofortige, umsetzbare Aufgabenliste.

👉 Söhne Kraftig

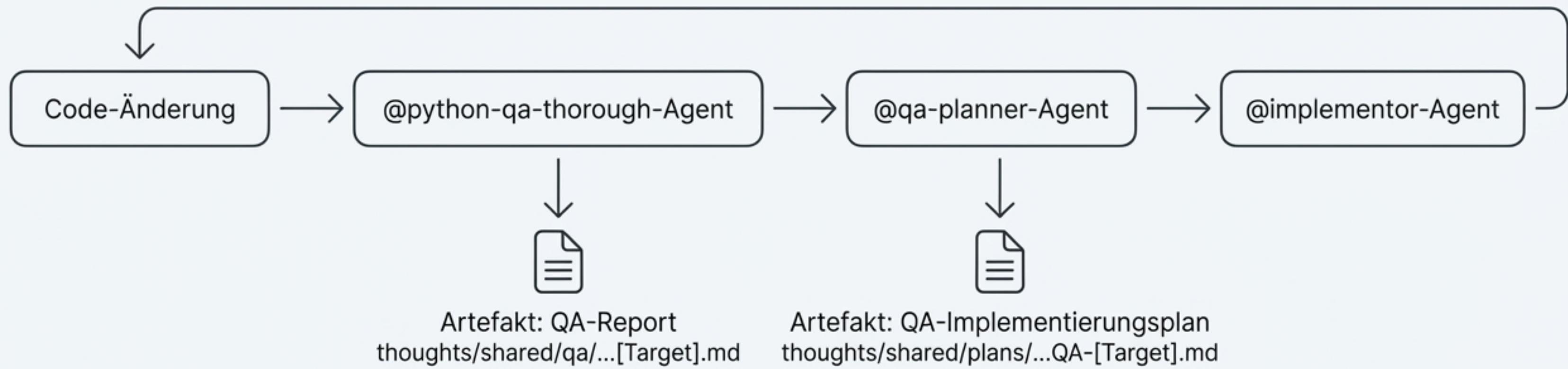
👉 @python-qa-thorough / @typescript-qa-thorough

- Purpole: Für umfassende Reviews.
- Prozess: Kombiniert automatisierte Tools mit manueller Analyse von Lesbarkeit, Wartbarkeit und Testbarkeit.
- Resultilt: 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)

Hauptagenten, mit denen Sie direkt interagieren. Sie steuern übergeordnete Workflows, treffen strategische Entscheidungen und delegieren Aufgaben.

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

Sub-Agenten (Spezialisten)

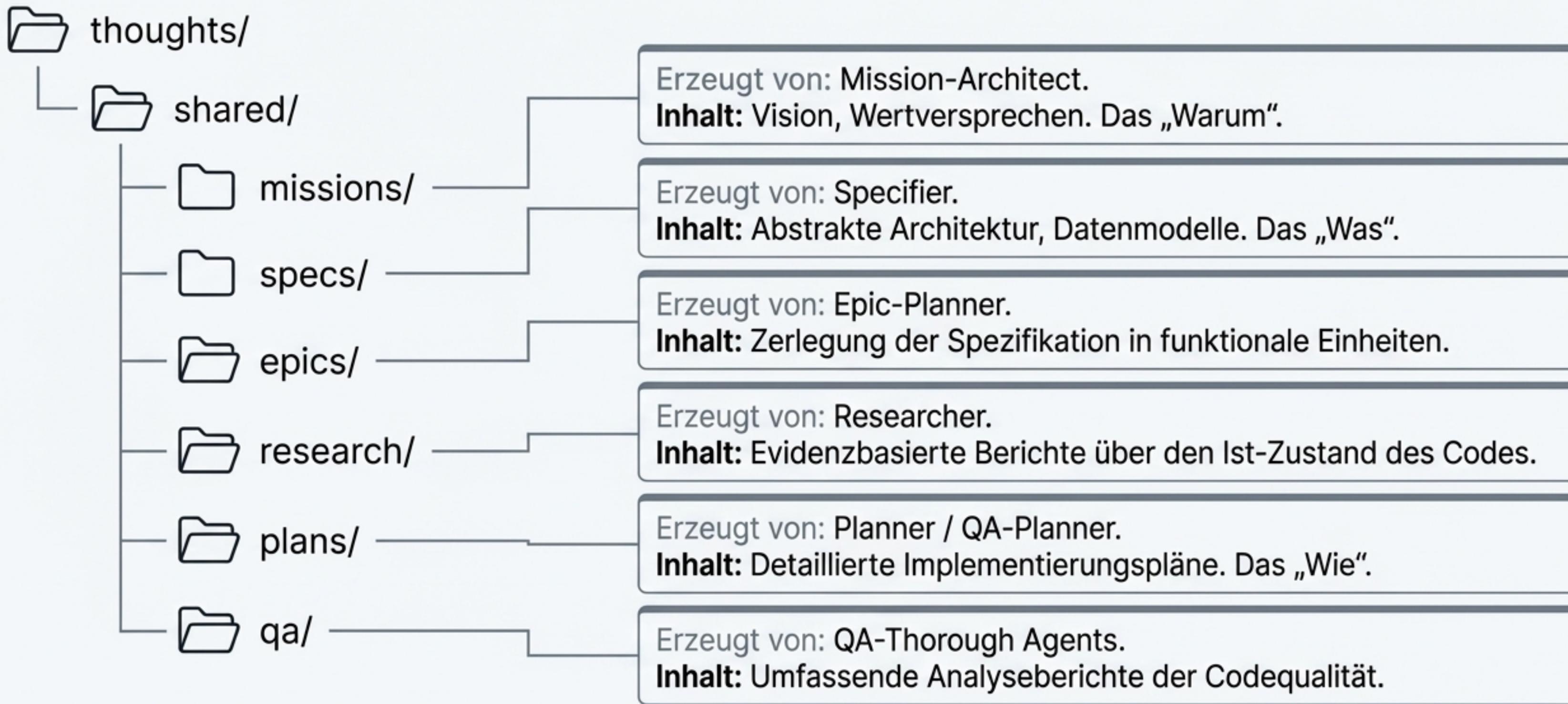
Spezialisierte Arbeiter, die von den Primär-Agenten aufgerufen werden. Sie interagieren normalerweise nicht direkt mit dem Benutzer.

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

Der Mehrwert: Wie RPIQPI die Kernprobleme löst

Herausforderung	RPIQPI-Lösung
Chaotische, unstrukturierte Entwicklung	Phasengesteuerte Workflows: Jeder Schritt hat einen definierten Zweck (Mission → Spec → Plan → Implement).
Nicht nachvollziehbare Änderungen & Kontextverlust	Git als Evidenz & Artefakt-Kette: Jede Anforderung ist von der Mission bis zum Code rückverfolgbar.
Ineffiziente LLM-Nutzung & hohe Kosten	Controller/Executor-Architektur: ~66% Kontextreduktion, Fehlerisolierung und gezielte Wiederholungslogik.
Mangelnde Qualitätssicherung	Integrierter QA-Kreislauf: Systematische Umwandlung von QA-Befunden in ausführbare Pläne.
Unklare Vision und Scope Creep	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



Built with OpenCode - The open source AI coding agent.