Gut bauen mit KI

Ein Schritt-für-Schritt-Leitfaden für Software, Agenten und Orchestrierungen — in der Reihenfolge, die bessere Ergebnisse bringt

Rehm KI Consulting

Version 1.0 · Erste Ausgabe · Juli 2026

Warum die Reihenfolge die verborgene Variable ist

Wenn ein KI-gestützter Build schiefgeht, ist der erste Reflex, das Modell verantwortlich zu machen — es war nicht klug genug, es hat halluziniert, es hat schludrigen Code geschrieben. Das ist meist nicht die eigentliche Ursache. Die eigentliche Ursache ist fast immer die **Reihenfolge**: Die Arbeit wurde in der falschen Abfolge gemacht. Es wurde programmiert, bevor das Problem umrissen war. Das Modell sollte sich erinnern, statt das Material zu bekommen. Dasselbe, das das System gebaut hat, war auch das Einzige, das es geprüft hat. Echte Nutzer waren die ersten Tester.

Stimmt die Reihenfolge, liefert ein gewöhnliches Modell gute Arbeit. Stimmt sie nicht, liefert das beste Modell der Welt ein selbstsicheres Durcheinander.

Dieser Leitfaden legt diese Reihenfolge dar. Er gilt für drei Arten von Arbeit, die sich weniger unterscheiden, als sie scheinen:

- **Reine Software** — eine normale Anwendung oder ein Dienst.
- **KI-Agenten** — Systeme, in denen ein Modell über Werkzeuge Handlungen ausführt, mit einer gewissen Autonomie.
- **Orchestrierungen und Routinen** — mehrere Agenten oder Schritte, zu einer Pipeline zusammengesetzt, bei der die Koordination das Produkt ist.

Die Schritte sind für alle drei gleich. Wo ein Agent oder eine Orchestrierung etwas Zusätzliches braucht, sagt es der jeweilige Schritt.

Drei Gedanken liegen allem Folgenden zugrunde, und es lohnt sich, sie klar zu benennen, bevor die Schritte beginnen. **Umreißen, bevor Sie bauen** — die gestellte Anfrage ist selten das eigentliche Problem. **Verankern, nicht abrufen** — KI ist vertrauenswürdig, wenn das Material vor ihr liegt, und unzuverlässig, wenn sie ins Gedächtnis greift. **Trennen Sie den Ausführenden vom Prüfer** — nichts, das eine Sache geschaffen hat, sollte der alleinige Richter über ihre Güte sein. Fast jeder Schritt unten ist einer dieser drei Gedanken, angewandt auf einen anderen Moment.

Schritt 0 — Das eigentliche Problem umreißen

So geht's: Bringen Sie die Idee auf ein einziges, in sich stimmiges Ideendokument. Halten Sie darin das Ziel fest, Ihre Absichten und — das ist der Teil, den man überspringt — konkrete Beispiele dafür, wie **Erfolg** aussieht und wie **Misserfolg** aussieht.

Warum es wichtig ist: Die gestellte Anfrage ist ein Ausgangspunkt, keine Spezifikation. Menschen beschreiben Symptome, nicht Ursachen; sie beschreiben eine angenommene Lösung, nicht den zugrunde liegenden Bedarf. Bauen Sie genau das, was verlangt wurde, bauen Sie oft das Falsche — nur schön. Die Misserfolgsbeispiele zählen so viel wie die Erfolgsbeispiele, weil sie die Grenze ziehen, die das System nicht überschreiten darf.

So sieht gut aus: Eine Seite, die eine fremde Person lesen und korrekt erklären könnte, was Sie bauen, für wen, und woran Sie beide den Erfolg erkennen.

Beginnen Sie mit der einfachsten nützlichen Version. Widerstehen Sie dem Drang, gleich das ganze ehrgeizige System zu umreißen. Wählen Sie das kleinste Problem, das echten Wert liefert, bringen Sie das richtig gut zum Laufen und ergänzen Sie Fähigkeiten in späteren Iterationen (Schritt 6). Eine einfache Sache, die funktioniert, schlägt eine komplexe, die nur halb funktioniert — und die einfache Version zeigt Ihnen, was die komplexe wirklich braucht.

Häufiger Fehler: Vage Erfolgskriterien („mach es gut", „die Nutzer sollen es mögen"). Ist Erfolg nicht konkret, können Sie ihn nicht testen und nicht wissen, wann Sie fertig sind.

- **Für Agenten:** Umreißen Sie, wofür der Agent zuständig ist und, ebenso wichtig, was er niemals tun darf. „Erfolg" und „Misserfolg" werden zu konkreten Verhaltensweisen — den Handlungen, die Sie wollen, und denen, die eine Katastrophe wären.
- **Für Orchestrierungen:** Umreißen Sie zuerst das Gesamtergebnis, bevor Sie zerlegen. Eine Pipeline aus einzeln korrekten Schritten, die in der Summe nicht das Ziel ergeben, ist ein klassischer und teurer Fehler.

Schritt 1 — Das Kontextpaket zusammenstellen

So geht's: Sammeln Sie in echten Dokumenten alles, woraus die KI bauen soll: das Ideendokument, das Ziel, Ihre Absichten, eine Funktionsliste, die Erfolgs- und Misserfolgsbeispiele, Ihre Werte und Rahmenbedingungen sowie Ihre Programmierstandards. Erstellen Sie daraus ein kurzes Designdokument.

Warum es wichtig ist: Dies ist der größte Hebel für Qualität und die stärkste Verteidigung gegen Halluzinationen. Ein Sprachmodell sagt plausiblen Text voraus; fehlt ihm ein Fakt, erfindet es einen plausiblen — mit derselben selbstsicheren Stimme, die es für Gewusstes verwendet. Die Lösung ist nicht, auf sein Erinnern zu hoffen — sie besteht darin, die Wahrheit vor es zu legen. Ein Build, der in einem von Ihnen gelieferten Kontextpaket verankert ist, arbeitet aus Ihrer Realität, nicht aus der Näherung des Modells.

So sieht gut aus: Ein Ordner mit klaren Dokumenten. Nichts Tragendes existiert nur in Ihrem Kopf oder in einem Chat-Verlauf, den Sie nächste Woche verlieren.

Häufiger Fehler: Entscheidende Rahmenbedingungen (eine Lizenzpflicht, eine Sicherheitsregel, ein Datenformat), die „offensichtlich" waren und daher nie aufgeschrieben wurden — sodass das Modell sie nie kannte.

- **Für Agenten:** Das Kontextpaket wird zum System-Prompt des Agenten, zu seinen Werkzeugdefinitionen, seinen Leitplanken und einer Reihe durchgearbeiteter Beispiele. Seien Sie explizit über verfügbare Werkzeuge und festgelegte Grenzen.
- **Für Orchestrierungen:** Definieren Sie den gemeinsamen Kontext und die Verträge zwischen den Komponenten — was jede über die anderen annehmen darf. Mehrdeutigkeit zwischen Agenten ist die Stelle, an der Orchestrierungen verrotten.

Schritt 2 — Die Spezifikation schreiben

So geht's: Lassen Sie ein starkes Reasoning-Modell aus dem Kontextpaket eine schriftliche Spezifikation entwerfen. Dann lesen Sie sie, korrigieren sie und geben sie frei. Die menschliche Freigabe ist nicht optional.

Warum es wichtig ist: Die Spezifikation ist der Vertrag zwischen Absicht und Umsetzung. Ein Fehler in der Spezifikation ist jetzt billig zu beheben und teuer, sobald darauf aufgebaut wurde. Die Spezifikation zu prüfen ist die wirkungsvollste halbe Stunde des ganzen Projekts.

So sieht gut aus: Eine Spezifikation, die Sie persönlich gelesen haben und der Sie wirklich zustimmen — präzise genug, um Mehrdeutigkeit auszuräumen, aber nicht so starr, dass sie jede Umsetzungsentscheidung diktiert und bessere ausschließt.

Häufiger Fehler: Die Spezifikation überfliegen, weil sie „richtig aussieht". Eine Spezifikation, die richtig aussieht und subtil falsch ist, ist schlimmer als keine, weil ihr nun alle vertrauen.

- **Für Agenten:** Spezifizieren Sie die Rolle, die Ein- und Ausgaben, die Werkzeuge und die Abbruchbedingungen — wann der Agent zurückgeben, eskalieren oder ablehnen soll.
- **Für Orchestrierungen:** Spezifizieren Sie jede Übergabe. Die Nachrichten zwischen den Schritten sind die eigentliche Schnittstelle; spezifizieren Sie sie so sorgfältig wie eine öffentliche API.

Schritt 3 — Einen detaillierten Bauplan schreiben

So geht's: Erstellen Sie aus der freigegebenen Spezifikation einen detaillierten, geordneten Bauplan — detailliert genug, dass ein kleineres, günstigeres Modell ihn bis zur Fertigstellung ausführen könnte, ohne das Reasoning zu brauchen, das ihn hervorgebracht hat.

Warum es wichtig ist: Ein starker Plan erlaubt es, den Großteil der Arbeit an wirtschaftliche Modelle zu delegieren und die Kosten vernünftig zu halten (siehe „Das richtige Modell für die Aufgabe"). Er erzwingt außerdem Modularität: Um den Plan zu schreiben, müssen Sie die Nahtstellen finden, an denen sich das System natürlich trennt, und diese Nahtstellen werden zu sauberen Schnittstellen statt zu verworrenen Abhängigkeiten.

So sieht gut aus: Einzelne, geordnete Aufgaben mit klaren Schnittstellen dazwischen — jede entweder direkt baubar oder direkt prüfbar.

Häufiger Fehler: Ein Plan, der in Wahrheit ein verkappter Monolith ist — Schritte so verflochten, dass sich nichts isoliert bauen oder testen lässt.

- **Für Agenten & Orchestrierungen:** Definieren Sie die Nachrichtenverträge, das Routing (wer ruft wen, und wann) und die Fehlerbehandlung vor dem Bauen. Legen Sie im Voraus fest, was passiert, wenn ein Schritt eine Zeitüberschreitung hat, Unsinn zurückgibt oder in eine Schleife läuft. Bauen Sie modular, mit einem Kommunikationsbus zwischen den Komponenten als Standard, sofern kein konkreter Grund dagegen spricht.

Schritt 4 — Mit Disziplin bauen

So geht's: Führen Sie den Plan aus. Bauen Sie antifragil und von der ersten Zeile an sicher, nicht als späteren Durchgang. Arbeiten Sie testgetrieben, in Modulen. Führen Sie Tests fortlaufend während des Bauens aus und verlangen Sie, dass jeder Test lokal besteht, bevor gepusht wird.

Warum es wichtig ist: Sicherheit und Widerstandsfähigkeit, am Ende nachgerüstet, passen nie richtig — die Annahmen sind bereits eingebacken. Antifragil heißt, das System ist so ausgelegt, dass, wenn ein Teil ausfällt, der Rest weiterarbeitet; der Fehler wird isoliert, statt zu kaskadieren. Das ist der Unterschied zwischen einem schlechten Nachmittag und einer Katastrophe, und für alles, was als Dienst läuft, ist es nicht optional.

So sieht gut aus: Ein modularer Build, Tests lokal grün, und nachweisbar kontrollierter Abbau — Sie können eine Komponente abschalten und zusehen, wie der Rest hält.

Häufiger Fehler: „Sicherheit und Tests machen wir später." Später kommt nicht, und wenn doch, ist es eine Neuentwicklung.

- **Für Agenten:** Leitplanken, Eingabevalidierung, gekapselter Werkzeugzugriff sowie vernünftige Zeitüberschreitungen und Wiederholungen. Nehmen Sie an, dass das Modell manchmal schlechte Ausgaben erzeugt, und bauen Sie so, dass schlechte Ausgaben keinen Schaden anrichten können.
- **Für Orchestrierungen:** Isolieren Sie Fehlerdomänen, damit ein ausfallender Agent die Pipeline nicht vergiftet. Machen Sie Schritte wo möglich idempotent und fügen Sie Schutzschalter (Circuit Breaker) hinzu, damit eine hängende oder schleifende Komponente gestoppt wird, statt weiterzulaufen.

Schritt 5 — Unabhängige Prüfung und Härtung

So geht's: Lassen Sie **zwei getrennte** KI-Sitzungen die Arbeit adversarial kritisieren. Sie kritisieren nur — sie beheben nichts. Ihre Befunde gehen zurück an den ursprünglichen Ersteller zur Umsetzung. Führen Sie dann Penetrationstests durch, mehrfach, nicht einmal.

Warum es wichtig ist: Man lässt den Buchhalter nicht seine eigenen Bücher prüfen, aus demselben Grund, aus dem der Agent, der den Code geschrieben hat, nicht der Einzige sein sollte, der ihn beurteilt: Der Interessenkonflikt ist strukturell, keine Frage der guten Absicht. Ein Prüfer mit frischem Kontext und ohne Anteil an der Arbeit sieht, wofür der Autor konstitutionell blind ist. Kritik und Behebung getrennt zu halten bewahrt diese Unabhängigkeit — in dem Moment, in dem ein Kritiker zu beheben beginnt, erwirbt er einen Anteil.

Nicht verhandelbar. Drei Dinge sind hier keine Ermessensfragen. Erstens läuft alles durch eine CI/CD-Pipeline — jede Änderung, jedes Mal, automatisch gebaut und getestet. Zweitens bestehen alle Tests, immer; ein roter Build wird nie gemergt und nie „zum späteren Beheben" ausgeliefert. Drittens sind Penetrationstests Pflicht, kein Luxus für Zeiten, in denen man Muße hat — ein ungetestetes System ist ein nicht vertrauenswürdiges System.

So sieht gut aus: Konkrete, umsetzbare Kritik, die auch umgesetzt wird; ein Ausführender und ein Prüfer, die nie derselbe Agent sind; und Penetrationstests, die wiederholt wurden, bis sie keine wesentlichen Probleme mehr zutage fördern.

Häufiger Fehler: Ein einziger Prüfdurchgang oder ein „Prüfer", der in Wahrheit dieselbe Sitzung mit anderem Hut ist. Unabhängigkeit, die Sie nicht tatsächlich erzwungen haben, ist Unabhängigkeit, die Sie nicht haben.

- **Für Agenten & Orchestrierungen:** Testen Sie die Prompts und den Ablauf adversarial — Prompt Injection, Jailbreaks, fehlerhafte Eingaben, feindselige Werkzeugantworten. Führen Sie ein Red-Teaming der gesamten Orchestrierung durch, nicht nur einzelner Agenten; die interessanten Fehler leben in den Übergaben.

Schritt 6 — Menschliches Testen, dann iterieren

So geht's: Führen Sie den ersten menschlichen Test durch, wenn das System vollständig läuft — nicht früher. Von dort aus entwickeln Sie iterativ auf echtem Feedback weiter.

Warum es wichtig ist: Ein halbfertiges System zu testen verwendet menschliche Aufmerksamkeit auf Probleme, die der Build von selbst gelöst hätte. Menschliches Urteil ist knapp und wertvoll; richten Sie es auf eine echte, laufende Sache, wo das Feedback echt ist.

So sieht gut aus: Ein laufendes System vor einem Menschen, das eine kurze, priorisierte Liste echter Verbesserungen hervorbringt — kein Gerangel, es zum Starten zu bringen.

Häufiger Fehler: Echte Nutzer als ersten Integrationstest zu verwenden. Sie werden feststellen, dass es kaputt ist, nicht dass es unvollkommen ist, und Sie werden Vertrauen verbrannt haben, um etwas zu lernen, das ein Rauchtest Ihnen gesagt hätte.

Das richtige Modell für die Aufgabe

Geben Sie das schwierigste Denken — Umreißen, Spezifikation, Planung und Kritik — dem stärksten Modell, auf das Sie Zugriff haben. Geben Sie die Routineausführung dem günstigsten Modell, das die Aufgabe zuverlässig erledigt. Für alles ein teures Modell zu verwenden verschwendet Geld; für das schwierige Reasoning ein billiges zu verwenden verschwendet alles, weil sich die Fehler in jeden nachgelagerten Schritt fortpflanzen.

Eine praktische Klassifizierung, **ehrlich als Arbeitsstand benannt, der schnell veraltet** — Modellnamen und Versionen ändern sich ständig, und jede konkrete Rangfolge sollte vor dem Verlassen darauf neu geprüft werden:

- **Spitzenklasse — schwierigstes Reasoning:** Umreißen, Spezifikation, Bauplan, adversariale Kritik.
- **Leistungsfähige Mittelklasse und wirtschaftliche Modelle — Routineausführung:** Umsetzung gut spezifizierter Aufgaben aus einem detaillierten Plan.

Die Disziplin ist nicht die konkrete Liste; sie ist die Gewohnheit, Leistungsfähigkeit an Aufgabenschwierigkeit anzupassen und regelmäßig neu zu benchmarken, statt einer Rangfolge zu vertrauen, die sechs Monate alt ist.

Trennen Sie den Ausführenden vom Prüfer

Dies ist das Herz guter Orchestrierung und bekommt daher eine eigene Behandlung. Das Prinzip ist das des Buchhalters: Wer die Arbeit macht, sollte nicht der Einzige sein, der sie prüft. Wenn Sie entscheiden, wie Arbeit über Modelle oder Agenten aufgeteilt und zugewiesen wird, prüfen Sie jede Aufgabe an vier Fragen:

1. **Größe** — wie groß ist die Aufgabe in Tokens für eine einzelne Sitzung? Aufgaben, die nicht sauber in ein Kontextfenster passen, müssen aufgeteilt oder anders strukturiert werden.
2. **Unabhängigkeit** — lässt sich die Aufgabe aufteilen, ohne an Wirksamkeit zu verlieren? Manches zerlegt sich sauber; manches verliert seinen Sinn, wenn man es zerschneidet. Teilen Sie entlang natürlicher Nahtstellen, nicht willkürlicher.
3. **Trennung der Zuständigkeiten** — würde ein Agent, der beide Hälften macht, einen Interessenkonflikt erzeugen? Wenn ja, trennen Sie sie, selbst um den Preis etwas geringerer Effizienz.

4. **Prüfbarkeit** — ist das Prüfen der Arbeit viel billiger als das Tun? Wenn ja, setzen Sie mehrere unabhängige Prüf-Agenten ein. Billige, parallel laufende Verifikation kauft viel Verlässlichkeit für wenig Geld.

Der letzte Punkt ist die stille Superkraft der Orchestrierung: Verifikation ist oft weit billiger als Generierung, sodass Sie es sich leisten können, dieselbe Ausgabe mehrfach zu prüfen und zu fangen, was jede einzelne Prüfung übersehen würde.

Die Anti-Muster, an einem Ort

- **Das gestellte Problem lösen statt des eigentlichen.** Erst umreißen.
- **Das Modell erinnern lassen, statt ihm das Material zu geben.** Alles verankern.
- **Code schreiben, bevor Spezifikation und Plan existieren.** Die Reihenfolge ist das ganze Spiel.
- **Der verkappte Monolith** — ein „Plan", dessen Schritte sich nicht isoliert bauen oder testen lassen.
- **Sicherheit und Tests „später".** Später ist eine Neuentwicklung.
- **Selbstprüfung** — der Ersteller bewertet seine eigene Arbeit.
- **Ein Penetrationstest.** Wiederholen, bis es still wird.
- **Nutzer als erste Tester.** Menschliches Testen beginnt an einem vollständig laufenden System.
- **Ein Modell für alles** — entweder Routinearbeit überbezahlt oder das schwierige Reasoning unterversorgt.

Das ehrliche Fazit

Nichts davon ist exotisch. Jeder Schritt ist offensichtlich, sobald er ausgesprochen ist. Die Schwierigkeit ist, dass unter dem Druck, einfach loszubauen, die Standardreflexe zurückkehren — das Umreißen überspringen, dem Gedächtnis des Modells vertrauen, es sich selbst prüfen lassen, an Nutzer ausliefern, „um zu sehen, was passiert". Das gut zu machen ist vor allem die Disziplin, der Reihenfolge zu folgen, auch wenn Vorpreschen schneller wirkt. Es ist nicht schneller. Es fühlt sich nur so an — bis Schritt 6.

Dies ist ein allgemeiner Überblicksleitfaden zum guten Bauen mit KI. Er bleibt bewusst auf der Ebene von Prinzip und Reihenfolge und kann die Besonderheiten eines einzelnen Projekts nicht abdecken. Für detaillierte, auf Ihre Situation zugeschnittene Beratung nehmen Sie Kontakt auf.

Über Rehm KI Consulting

Rehm KI Consulting hilft Organisationen, mit KI so zu bauen, wie gebaut werden sollte — mit echter ingenieurmäßiger Disziplin dahinter, nicht mit einer als Produkt verkleideten Demo. Der Ansatz gründet in Jahrzehnten der Hardware- und Softwareentwicklung, vom Chip-Design und Netzwerkprozessoren bei Intel bis zu Jahren des Bauens von Webanwendungen, heute darauf ausgerichtet, Unternehmen bei der KI-Einführung zu helfen — ehrlich, verlässlich und auf Dauer gebaut.

Die Haltung ist einfach: klare Antworten darauf, wo KI wirklich hilft und wo nicht. Kein Hype, keine Magie.

Rehm KI Consulting · Christopher Rehm · Bayern, Deutschland · contact@christopherrehm.de

In diesem Leitfaden genannte Modellnamen, Werkzeuge und Versions-Rangfolgen ändern sich schnell — behandeln Sie sie als zu überprüfende Beispiele, nicht als feststehende Fakten.