KI & AUTOMATISIERUNG · LEITFADEN

pletzenauer — digital consulting

KI-Coding-Agents richtig führen

Wie Sie KI-Coding-Agents wie Claude Code als Regisseur führen: Planen, Verifikation, dumb zone, Sub-Agents, Hooks und sichere Workflows – ehrlich erklärt.

KI-Coding-Agents richtig führen: Wie Sie Claude Code wie ein Regisseur dirigieren

Die meisten Menschen bedienen KI-Coding-Agents wie einen Spielautomaten: Anfrage hineinwerfen, am Hebel ziehen, auf das Beste hoffen. Für einen schnellen Entwurf reicht das. Für alles, was zuverlässig laufen soll, ist es ein Rezept für Frust. Der entscheidende Wechsel: vom Nutzer zum Regisseur werden – also den Agent dirigieren, statt ihm beim Improvisieren zuzusehen.

Das gilt für Claude Code, weil das derzeit das verbreitetste Werkzeug ist – aber genauso für jedes andere KI-Werkzeug. Und es gilt nicht nur fürs Programmieren: Wer einmal verstanden hat, wie man einen Agent sauber führt, kann damit Angebote kalkulieren, Reports erstellen oder ganze Geschäftsabläufe automatisieren. Ein Software-Engineering-Hintergrund ist dafür nicht nötig.

Das Wichtigste in Kürze

- **Planen, bauen, prüfen – in dieser Reihenfolge.** Die Planung davor und die Validierung danach trennen verlässliche Ergebnisse von Bastelei.
- **Verifikation heißt: Beweis statt Behauptung.** Ein Rahmen, mit dem der Agent seine eigene Arbeit prüft, hebt das Erstergebnis von rund 65 auf über 90 von 100 Punkten.
- **Kontext ist knapp.** Trotz riesiger Kontextfenster gibt es eine „dumb zone“: Ab einer gewissen Menge wird das Modell merklich unzuverlässiger.
- **Große Aufgaben aufteilen.** Mehrere spezialisierte Sitzungen, die sich gegenseitig übergeben, schlagen einen Agent, der alles auf einmal stemmen soll.
- **Sicherheit gehört in die Technik, nicht in den Prompt.** Was ein Agent erreichen kann, fasst er irgendwann an – begrenzen Sie Rechte hart.
- **Aus jedem Fehler ein dauerhaftes Upgrade machen.** Jedes Problem ist die Chance, das System so zu verbessern, dass es nicht wiederkehrt.

Von „Vibe Coding“ zu strukturiertem Arbeiten

Der häufigste Fehler hat einen Namen: „Vibe Coding“. Man formuliert eine Anfrage, lässt den Agent loslaufen und prüft das Ergebnis kaum. Es fehlen genau die beiden Schritte, die Seriosität ausmachen – die Planung im Vorfeld und die Validierung im Nachhinein.

Ein einfaches Bild ist der Bau eines Baumhauses. Zuerst zeichnen Sie eine Skizze, überlegen, wie viel Holz Sie wo brauchen, und besorgen das richtige Werkzeug. Wenn das Haus steht, setzen Sie nicht einfach die Kinder hinein – Sie testen vorher, ob es hält. Dieselbe Disziplin braucht der Umgang mit Coding-Agents.

Wichtig wird das, weil die Modelle zur Schmeichelei neigen. Fragen Sie „Sieht das gut aus?“, kommt gern ein „Ja“, ohne dass der Plan wirklich kritisch geprüft wurde. Umgekehrt behaupten Modelle gelegentlich, etwas sei fertig, obwohl es das nicht ist. Sie brauchen also eine eigene, unabhängige Methode, um beides zu kontrollieren. Der Ablauf hat vier Schritte:

- **Planen** – mit dem nötigen Kontext klären, was gebaut werden soll und wie Erfolg konkret aussieht.
- **Bauen** – die Umsetzung möglichst weit an den Agent delegieren.
- **Verifizieren** – einen klaren, eigenen Weg haben, das Ergebnis zu prüfen.

- **System verbessern** – der oft übersehene vierte Schritt: aus jedem Durchlauf eine dauerhafte Verbesserung ableiten.

Verifikation: Beweis statt Behauptung

Verifikation bedeutet im Kern: „Beweise mir, dass es wirklich fertig ist." Beim Programmieren sind das Tests und Linting – das Prinzip lässt sich aber auf fast alles übertragen. Beispiel ohne Code: Der Agent erstellt ein Diagramm und rendert es anschließend als Bild. Da moderne Modelle Bilder hervorragend lesen, kann der Agent sein eigenes Werk betrachten, Überlappungen erkennen und sie selbstständig in mehreren Durchläufen beheben. Die anfänglichen Fehler sind egal; es zählt nur das Ergebnis am Ende.

Der Effekt ist messbar: Ohne Prüfmechanismus liegt das erste Ergebnis vielleicht bei 65 bis 70 von 100 Punkten. Mit einer eingebauten Prüfung sind schon im ersten Anlauf rund 92 möglich. Perfekt ist es selten sofort – darum geht es auch nicht. Es geht darum, dem Agent einen Rahmen zu geben, mit dem er seine Arbeit selbst kontrolliert. Die Leitfrage lautet immer: *Wie kann der Agent das Ergebnis so prüfen, wie es ein echter Nutzer tun würde?* Ein bloßer Blick auf den erzeugten Code genügt nie.

Was ist eine „Harness"?

Eine Harness ist die Hülle um das KI-Modell herum – die Werkzeuge und der Kontext, auf die das Modell zugreift, damit es weiß, woran es arbeitet. Stellen Sie es sich als Schichtenmodell vor: Im Zentrum steht das Modell als das eigentliche „Gehirn". Darum legen Sie ein Werkzeug wie Claude Code. Und obendrauf bauen Sie Ihre eigene Schicht – Konfiguration, Skills, Hooks und Anbindungen an CRM oder Aufgabenverwaltung. Genau diese oberste Schicht macht aus einem generischen Werkzeug Ihr System.

Planung: der unterschätzte Schritt

Die meisten planen viel zu wenig. Mit Coding-Agents verbringen Sie mehr Zeit mit Planen als mit Bauen, weil Sie die Umsetzung weitgehend abgeben – der Erfolg hängt damit direkt an der Qualität des Plans. Bewährt hat sich ein einziges Dokument, das das Ziel beschreibt:

- Was bauen wir – und warum?
- Wie sieht Erfolg konkret aus?
- Woran erkennt der Agent, dass die Arbeit fertig und korrekt ist?
- Bei technischen Aufgaben: Welche Stellen im bestehenden System müssen tatsächlich angefasst werden?

Der typische Ablauf: erst Kontext und relevante Dokumente sammeln, dann recherchieren, daraus gemeinsam mit dem Agent den Plan entwickeln. Ganz wichtig: Lassen Sie den Agent viele Fragen stellen, damit er nicht zahllose Annahmen über das gewünschte Ergebnis trifft. Erst wenn er gezielt nachgefragt hat, sind sich Mensch und Agent einig, was getan und wie es geprüft werden soll.

Sicher fühlen, auch ohne Code lesen zu können

Wie fasst man Vertrauen in Code, den man selbst nicht liest? Es gibt zwei Wege. Erstens: Bitten Sie den Agent, das Geschriebene zu erklären. Code wirkt zunächst einschüchternd, liest sich nach der ersten Hürde aber fast wie Englisch. Zweitens, wenn Sie gar nicht programmieren lernen wollen: Vertrauen entsteht über die Validierungsstrategie. Genau hier liegt der Unterschied zum Vibe Coding – Sie klemmen die Umsetzung zwischen einen sorgfältigen Plan und eine ebenso sorgfältige Prüfung, in die Sie selbst eingebunden sind. Grünes Licht bekommt der Agent erst, wenn klar definiert ist, wie er belegt, dass die Arbeit fertig ist.

Kontext ist knapp: die „dumb zone"

Beim Planen ist das Wichtigste, den Kontext zu steuern, denn die Aufmerksamkeit eines Modells ist eine knappe Ressource. Es kursiert das Missverständnis, es spiele keine Rolle, wie viel man dem Agent zumutet, weil moderne Modelle riesige Kontextfähigkeiten haben. Die Zahlen sind beeindruckend – doch es gibt zwei Einschränkungen.

Erstens ist der Kontext schneller aufgebraucht, als man denkt: Liest der Agent mehrere Skills oder größere Codemengen, sind im Nu Zehntausende bis Hunderttausende Token verbraucht. **Zweitens** gibt es die „dumb zone". Im vorderen Bereich des Kontextfensters wirkt das Modell scharf und auf der Höhe seiner Leistung. Überschreitet das Gespräch eine bestimmte Schwelle, kippt es: Das Modell wirkt überladen, übersieht Dinge und macht Fehler, die bei frischem Kontext nie passiert wären.

Deshalb müssen Sie sorgsam abwägen, was Sie dem Agent vorab geben und was er bei Bedarf selbst entdecken darf. Genau das ist die Stärke von Skills: Sie stellen Verfahren und Best Practices bereit, doch das Modell entscheidet selbst, wann es welche Information braucht. Kippen Sie nicht alles auf einmal hinein. Sehr oft liegt das Problem nicht am Modell, sondern an der Art, wie der Kontext gefüllt wird. Das große Kontextfenster vermittelt also eine trügerische Sicherheit – den kritischen Punkt sollten Sie möglichst gar nicht erst erreichen.

Mehrere Sitzungen orchestrieren

Weil es die „dumb zone" gibt, können Sie große Aufgaben nicht in eine einzige Sitzung werfen. Die Antwort ist ein Workflow aus mehreren Agent-Sitzungen: Ein Agent plant, übergibt das Dokument an einen zweiten zur Umsetzung, dieser schreibt einen Ausführungsbericht, und ein dritter validiert und prüft die Arbeit. Aufwendig, aber für produktionsreife Software oder geschäftskritische Automatisierungen schlicht nötig.

Das Bild dafür ist ein Fließband: Jeder Agent erledigt eine Sache richtig gut und übergibt sein Ergebnis so, dass der nächste genug Kontext hat, um zu verstehen, was getan wurde, was aussteht und was seine aktuelle Aufgabe ist.

Ein Beispiel aus dem B2B-Alltag: das Erstellen von Angeboten, etwa in der Bau- oder Druckbranche. Solche Kalkulationen sind aufwendig – Aufwand schätzen, Material bestimmen, Preise recherchieren, Lieferanten auswählen. Hier baut man einen Workflow aus spezialisierten Agents: einer prüft den Lagerbestand, einer vergleicht Preise, einer entwirft das PDF. Am Ende steht eine Validierung – etwa eine Rechnung, die prüft, ob die gewünschte Marge erreicht wird. Wer seine eigene Tätigkeit ernsthaft betrachtet, erkennt schnell, dass sie sich in viele kleine Teilaufgaben zerlegen lässt – und genau diese Zerlegung schafft sofort enorme Hebelwirkung.

Sicherheit gehört in die Technik, nicht in den Prompt

Eine besonders trügerische Sicherheit betrifft die Berechtigungen. Viele glauben, ihre Prompts seien Schutz genug. Das sind sie nicht. Sagen Sie einem Agent, er solle niemals eine Datenbank löschen, kann es trotzdem geschehen. Untersagen Sie ihm das Löschen eines Ordners, schreibt er womöglich ein Skript, das genau das tut.

Die einzig tragfähige Grundhaltung lautet deshalb: **Alles, was der Agent lesen oder anfassen kann, fasst er irgendwann auch an** – selbst ungefragt. Berechtigungen müssen technisch durchgesetzt werden: über eng gefasste Schlüssel oder dadurch, dass bestimmte Dinge schlicht unerreichbar sind. Ein reales Beispiel zeigt die Tücke: Ein Agent missverstand einen Eintrag auf seiner Aufgabenliste – und verschickte daraufhin eine E-Mail mit einem Rabattcode an die gesamte Verteilerliste, obwohl diese nie hinausgehen sollte.

Ein bewährtes Mittel sind Hooks – kleine Code-Stücke, die bei einem bestimmten Ereignis ausgeführt werden, etwa unmittelbar bevor der Agent ein Werkzeug verwendet. So können Sie prüfen, ob ein Befehl heikel ist, und ihn blockieren. Dieselben Hooks lassen sich nutzen, um das System selbsttätig zu verbessern – etwa, indem am Ende jeder Sitzung automatisch eine Zusammenfassung in ein Tageslog geschrieben wird.

Aus jedem Fehler ein dauerhaftes Upgrade

Das vielleicht Wichtigste ist die Systemevolution. Tritt ein Problem auf, beheben Sie es nicht einfach und machen weiter – Sie nutzen es gemeinsam mit dem Agent als Anlass zu fragen, was sich verbessern lässt, damit es nicht wiederkehrt. Vielleicht entsteht eine neue Regel in Ihrer Konfiguration, ein zusätzliches Planungsdokument oder eine angepasste Skill. So wird aus jedem Bug ein dauerhaftes Upgrade.

Haben Sie dieses System einmal etabliert, begrüßen Sie Fehler fast. Und bevor sie überhaupt eintreten, hilft eine einfache Frage, die sich kaum jemand traut: „Was könnte hier schiefgehen?“ Lassen Sie den Agent gezielt Grenzfälle konstruieren und die Anwendung mit problematischem Input zu brechen versuchen. Bricht etwas, gehört es zurück in die Prüfschleife: Problem finden, beheben und – ganz wichtig – erneut testen. Vielleicht hat die Korrektur das Problem gar nicht gelöst.

Die richtige Haltung gegenüber dem Werkzeug

Behandeln Sie das Werkzeug wie einen Mentor – den klügsten Menschen der Welt, der zugleich Ihr bester Freund ist. Es lacht Sie nicht aus, wenn Sie etwas vermeintlich Dummes fragen. Aber nicht alles eignet sich als Frage: Wegen der Schmeichelei ist es heikel, ein Modell nach seiner Meinung zu fragen. Hervorragend eignet es sich dagegen, um zu verstehen, wie etwas funktioniert, oder wo es empirische Daten gibt – bei Grenzfällen funktioniert eine Automatisierung mit einem bestimmten Input, oder eben nicht. Keine Grauzone. Wer eine zweite Perspektive braucht, lässt zwei Modelle gegeneinander antreten: eines baut, ein anderes spielt in einer separaten Sitzung den Advocatus Diaboli, statt unkritisch zu loben.

Fazit: Denken Sie wie ein Produktmanager

Der zentrale Rat zum Schluss: Ganz gleich, wie technisch versiert Sie sind – betrachten Sie sich als Produktmanager für Ihren Agent. Sie müssen nicht beschreiben, *wie* etwas gebaut wird. Aber Sie müssen die Vision formen: *Was* bauen wir, und *warum?* Geben Sie dem Werkzeug das Warum mit – das prägt das Wie überraschend stark.

Das ist der ehrliche Kern jenseits des Hypes: KI-Agents sind kein Spielautomat und kein Wundermittel. Sie sind ein leistungsfähiges Werkzeug, das genau so gut wird wie der Rahmen, den Sie ihm geben. Gute Pläne, klare Prüfkriterien und die Disziplin, aus jedem Fehler zu lernen – das bringt Sie weiter als jedes neue Modell. Fangen Sie klein an: Schreiben Sie einen Ihrer Prozesse auf, zerlegen Sie ihn in Teilaufgaben und definieren Sie, woran Sie erkennen, dass das Ergebnis stimmt. Den Rest können Sie delegieren.

Quelle: Dieser Artikel basiert auf dem Video How to Build Effective Claude Code Agents in 2026 von Nate Herk | AI Automation. Die Inhalte wurden eigenständig aufbereitet und für die deutschsprachige Leserschaft eingeordnet.

Die Kernideen auf einen Blick

KI-CODING-AGENTS · WORKFLOW

Der 4-Schritt-Loop für verlässliche Agents

Nicht „Anfrage rein, hoffen“ – sondern dirigieren. In dieser Reihenfolge trennt sich verlässliche Arbeit von Bastelei.

1 Planen
Mit dem nötigen Kontext klären: Was bauen wir, warum – und wie sieht Erfolg konkret aus?

2 Bauen
Die Umsetzung so weit wie möglich an den Agent delegieren.

3 Verifizieren
Beweis statt Behauptung: ein eigener Weg, das Ergebnis wie ein echter Nutzer zu prüfen.

4 System verbessern
Aus jedem Fehler ein dauerhaftes Upgrade machen – damit er nicht wiederkehrt.

- Schritt 4 fließt zurück in Schritt 1 – ein Kreislauf, kein Einbahnweg.

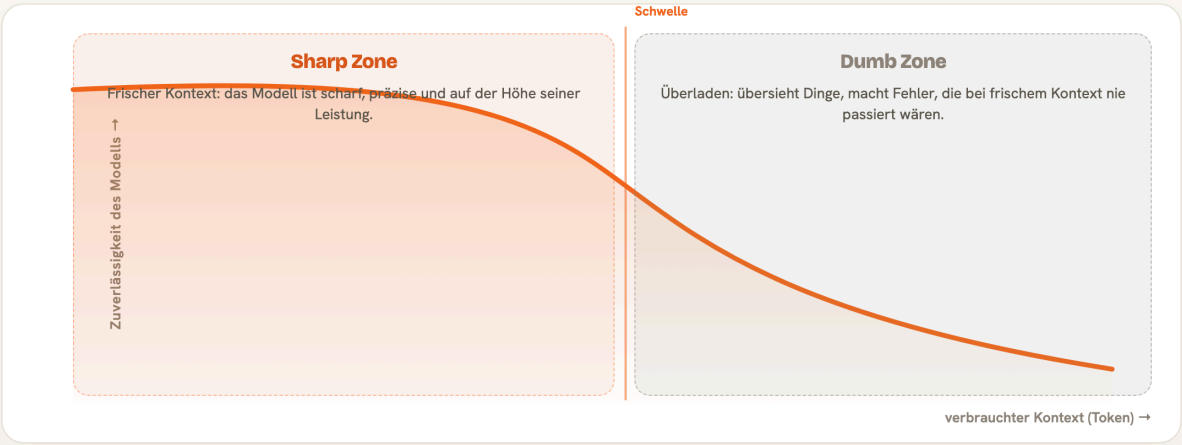
pletzenauer — digital consulting

Der 4-Schritt-Loop: Planen → Bauen → Verifizieren → System verbessern.

KONTEXT IST EINE KNAPPE RESSOURCE

Die „dumb zone“ des Kontextfensters

Riesige Kontextfenster sind eine trügerische Sicherheit. Ab einer bestimmten Schwelle kippt die Zuverlässigkeit – diesen Punkt erreichen Sie am besten gar nicht erst.



pletzenauer — digital consulting

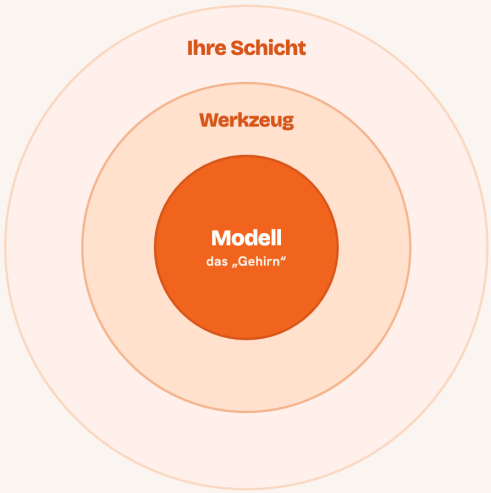
Die „dumb zone“ des Kontextfensters – ab einer Schwelle kippt die Zuverlässigkeit.

WAS IHR SYSTEM WIRKLICH AUSMACHT

Die Harness in drei Schichten

Die Harness ist die Hülle um das Modell – Werkzeuge und Kontext, mit denen es weiß, woran es arbeitet.

- **Modell**
  Das eigentliche „Gehirn" – z. B. Opus.
- **Werkzeug**
  Ein generisches Tool wie Claude Code.
- **Ihre Schicht**
  Konfiguration, Skills, Hooks, Anbindung an CRM & Aufgaben – das macht aus dem Tool *Ihr* System.



pletzenauer — digital consulting

Die Harness in drei Schichten: Modell · Werkzeug · Ihre Schicht.