



DAS MENTER PLAYBOOK

Software selbst bauen. Mit KI an der Seite.

Ein praktischer Leitfaden für Guided Vibe Coding: wie du aus einer Idee ein funktionierendes Werkzeug machst, ohne jede Zeile selbst zu schreiben.

KLÄREN

FESTLEGEN

BAUEN

PRÜFEN

VERFEINERN

AUSROLLEN

 ÜBERBLICK

Inhalt

TEIL A • GRUNDLAGEN

1	Warum jetzt	3
2	Elf Grundsätze	5
3	Die richtigen Werkzeuge	8

TEIL B • VOM GEDANKEN ZUM PLAN

4	Sechs Etappen eines Projekts	9
5	Die Produktbeschreibung	10
6	Der Start in der Werkstatt	12

TEIL C • BAUEN UND ÄNDERN

7	Das AUFTRAG-Prinzip	14
8	Sicher ändern	16
9	Fehler methodisch finden	18

TEIL D • TECHNIK, PRÜFUNG, BETRIEB

10	Die Technik-Landkarte	20
11	Prüfen statt hoffen	22
12	In Betrieb nehmen	24
13	Zwölf Stolperfallen	25
14	Nachbauen statt abonnieren	26
15	Guided Vibe Coding mit menter	28

ANHANG

A	Vorlagen zum Kopieren	29
B	Wegweiser und Kurzreferenz	35

So arbeitest du mit diesem Playbook

Lies Teil A einmal ganz. Die Teile B bis D sind zum Nachschlagen gedacht, wenn du an der jeweiligen Stelle im Projekt stehst. Alle Vorlagen zum Kopieren findest du gesammelt im Anhang. Im Text verweisen Kürzel wie **V09** auf die passende Vorlage.

||| KAPITEL 1

Warum jetzt

Wer bisher ein eigenes Werkzeug wollte, hatte zwei Wege: selbst programmieren oder eine Agentur bezahlen. Dazwischen gibt es inzwischen einen dritten.

KI-Assistenten können heute Anforderungen ordnen, technische Optionen erklären, Code erzeugen, fremde Projekte durchleuchten, Fehlerursachen eingrenzen und Änderungen direkt im Projekt vornehmen. Für viele Menschen ohne Entwicklerausbildung ist damit eine Tür aufgegangen.

Leichter geworden ist Softwareentwicklung dadurch nicht. Verschoben hat sich vor allem, was du selbst leisten musst. Das Tippen übernimmt die Maschine. Das Denken, Entscheiden und Kontrollieren bleibt bei dir.

Du schreibst nicht mehr jede Zeile. Dafür führst du das Projekt.

Was dieses Playbook leistet

Es verspricht keinen Zauberprompt, der fertige Software ausspuckt. Es zeigt einen Arbeitsablauf, mit dem du eine Idee sortierst, Aufträge an die KI sauber formulierst, jede Änderung im Griff behältst, Fehler methodisch aufspürst und dein Werkzeug am Ende sicher in Betrieb nimmst.

Für wen es gedacht ist

- Entscheider in Clubs, Verbänden, Agenturen und Unternehmen
- Marketing-, Sponsoring- und Eventteams, die ihre Abläufe selbst verbessern wollen
- Gründer, Selbstständige und kleine Produktteams
- Technisch Interessierte ohne Programmierausbildung

Was wir mit Vibe Coding meinen

Wir benutzen den Begriff nüchtern: Vibe Coding heißt, Software mit KI zu entwerfen, zu bauen und weiterzuentwickeln, ohne den Code selbst zu verfassen. Deine Arbeit besteht darin, sechs Dinge klar zu sagen:

Was
soll entstehen?

Wozu
braucht es das?

Wie
soll es sich verhalten?

Welche Regeln
gelten dabei?

Was bleibt
unangetastet?

Woran
er kennst du, dass es stimmt?

Der typische Fehlstart

Viele beginnen mit einem einzigen großen Satz, etwa: „Programmier mir ein Sponsoring-Tool für unseren Club.“ Nach wenigen Minuten gibt es Masken, Knöpfe und Tabellen. Erst beim Ausprobieren zeigt sich: Die Abläufe passen nicht zur Realität, die Datenstruktur bildet die Partnerpakete falsch ab, und jede kleine Korrektur zieht einen großen Umbau nach sich.

Fast immer liegt das nicht an der KI, sondern an einem Auftrag, der zu viel offenlässt. Wo du nichts festlegst, entscheidet die Maschine für dich.

Deine vier Aufgaben im Projekt

01

Auftraggeber

Du legst fest, welches Problem gelöst wird und was die erste Version können muss.

02

Projektsteuerung

Du teilst große Vorhaben in überschaubare Etappen und gibst jede einzeln frei.

03

Abnahme

Du probierst aus, ob das Ergebnis im echten Alltag funktioniert.

04

Schutz des Bestands

Du achtest darauf, dass Neues nichts kaputtmacht, was schon lief.

Was du am Ende kannst

- eine Idee strukturiert zu Papier bringen
- daraus eine umsetzbare Beschreibung machen
- die passenden KI-Werkzeuge kombinieren
- bestehende Projekte von der KI erklären lassen
- Änderungen eng begrenzen
- Fehler methodisch eingrenzen
- technische Vorschläge einordnen
- vor dem Livegang gründlich prüfen

AUF DEN PUNKT

Die KI erledigt einen Großteil der Umsetzung. Die Verantwortung für Richtung und Qualität bleibt bei dir.

||| KAPITEL 2

Elf Grundsätze

Wenn ein KI-Ergebnis nicht passt, wird der Prompt meist länger. Hilfreicher ist, vorher klarer zu denken.

Ein ausführlicher, aber unscharfer Auftrag bleibt unscharf. Die folgenden Grundsätze sind in drei Gruppen sortiert: wie du denkst, wie du steuerst und wie viel du verstehen solltest.

DENKEN

1 • Vom Ergebnis her beschreiben

Frameworks und Datenbanken sind Mittel zum Zweck. Was zählt, ist das Verhalten, das ein Mensch am Ende erlebt. Beschreibe das zuerst, die Technik folgt daraus.

Werkzeug oder Wirkung?

Die Technik ist das Wie. Die Wirkung ist das Produkt.

SCHWACH

„Bau eine Next.js-App mit Postgres und einer UI-Bibliothek.“

Zählt Werkzeuge auf, sagt nichts über das Problem.

STARK

„Unser Partnermanagement soll jederzeit sehen, welche vereinbarten Leistungen pro Sponsor schon erbracht sind.“

Beschreibt die Wirkung. Die Technik lässt sich daraus ableiten.

2 • Große Vorhaben zerlegen

Eine Funktion besteht fast immer aus mehreren Bausteinen. Wer sie einzeln beauftragt, kann jeden prüfen, bevor der nächste kommt.

Aus einem Wunsch werden acht Aufträge

Partner-Leistungsnachweis

Datenstruktur

Partnerliste

Filter

Detailseite

Leistung erfassen

Nachweis anhängen

Pflichtfelder prüfen

Tests

3 • Erst den Plan, dann den Code

Steckt in einem Wunsch mehr als eine größere Änderung, bitte die KI zuerst um einen Umsetzungsplan, ausdrücklich ohne Code. Du liest den Plan, streichst oder ergänzt, und erst dann wird gebaut.

Zwei Arbeitsweisen im Vergleich

Wer erst plant, erspart sich die Kette aus Folgefehlern.

DIREKT LOSBAUEN



ERST PLANEN



STEUERN

4 • Fakten statt Rollenspiel

Sätze wie „Du bist ein erfahrener Softwarearchitekt“ bringen wenig. Viel mehr hilft, was die KI über dein Projekt wirklich wissen muss: Zweck, vorhandener Aufbau, Einschränkungen und das gewünschte Ergebnis.

5 • Grenzen ausdrücklich ziehen

Sag nicht nur, was sich ändern soll, sondern auch, was tabu ist. Der erlaubte Bereich und die Grenzen drumherum sind zusammen das wirksamste Steuerungsmittel, das du hast.

6 • Funktionierendes schützen

Nicht jeder Umbau macht etwas besser. Gerade bei kleinen Werkzeugen ist schlicht, verständlich und stabil meist mehr wert als elegant und verschachtelt.

7 • „Fertig“ messbar machen

Lege vorher fest, woran du erkennst, dass etwas erledigt ist: erwartetes Verhalten, Umgang mit Fehleingaben, Berechtigungen, Darstellung auf dem Handy, durchgelaufene Tests. Wir nennen das die Abnahmekriterien.

VERSTEHEN

8 • Die Landkarte kennen

Programmieren musst du nicht können. Einordnen solltest du aber, was Oberfläche, Serverlogik, Datenbank, Schnittstelle, Anmeldung, Versionsverwaltung, Umgebungsvariablen und Veröffentlichung jeweils bedeuten. Kapitel 10 liefert dafür das Nötigste.

9 • Begründungen einfordern

Verstehst du einen Vorschlag nicht, frag nach: Was spricht dafür, was dagegen, welches Risiko steckt darin, gibt es etwas Einfacheres? Die KI ist nicht nur Codelieferant, sondern auch Dolmetscher für Technik.

10 • Mit überzeugenden Irrtümern rechnen

KI-Assistenten erfinden gelegentlich Funktionen, die es nicht gibt, greifen zu veralteten Lösungen, übersehen Sicherheitslücken oder benennen die falsche Ursache, und das alles in sehr sicherem Ton. Für die Arbeit darfst du vertrauen. Vor der Veröffentlichung wird geprüft.

11 • Fehler als Hinweise lesen

Ein Fehler sagt dir, wo dein Verständnis oder dein Auftrag eine Lücke hatte. Der Ablauf ist immer gleich: nachstellen, Ursache benennen, kleinste Korrektur planen, umsetzen, erneut prüfen.

Die elf Grundsätze auf einen Blick

GRUPPE	GRUNDSATZ
Denken	Vom Ergebnis her beschreiben · Große Vorhaben zerlegen · Erst den Plan, dann den Code
Steuern	Fakten statt Rollenspiel · Grenzen ausdrücklich ziehen · Funktionierendes schützen · „Fertig“ messbar machen
Verstehen	Die Landkarte kennen · Begründungen einfordern · Mit überzeugenden Irrtümern rechnen · Fehler als Hinweise lesen

AUF DEN PUNKT

Wirkung vor Werkzeug. Klein schneiden. Planen, freigeben, bauen. Grenzen nennen. Nachprüfen statt glauben.

||| KAPITEL 3

Die richtigen Werkzeuge

Nicht jedes KI-Werkzeug passt zu jeder Aufgabe. Es lohnt sich, die Arbeit nach Rollen zu verteilen.

In der Praxis bewährt sich eine Aufteilung in drei Rollen. Welche Marke du dafür nimmst, ist zweitrangig. Wichtig ist, dass du weißt, in welchem Werkzeug gerade gedacht und in welchem gebaut wird.

Denkpartner

z. B. ChatGPT

Ideen sortieren, recherchieren, Beschreibungen schreiben, Begriffe erklären, Entscheidungen abwägen

Analysepartner

z. B. Claude

Lange Dokumente und große Zusammenhänge erfassen, Architektur durchdenken, Zweitmeinung geben

Werkstatt

z. B. Cursor

Im Projekt arbeiten: Dateien lesen und ändern, Tests und Builds starten, Änderungen zeigen

Der Denkpartner

Hier klärst du, was du eigentlich willst. Ein Chat-Assistent hilft dir, eine Idee zu schärfen, Alternativen zu vergleichen, eine Produktbeschreibung zu formulieren und Datenstrukturen zu skizzieren. In dieser Rolle soll die KI noch nichts programmieren.

Der Analysepartner

Wenn viel Material im Spiel ist, etwa ein langes Lastenheft, Prozessdokumentationen oder ein bestehendes System, brauchst du ein Werkzeug, das große Mengen Kontext zuverlässig verarbeitet. Diese Rolle eignet sich auch hervorragend, um einen Plan gegenlesen zu lassen, bevor du ihn freigibst.

Die Werkstatt

Gebaut wird in einer Entwicklungsumgebung mit KI-Unterstützung. Dort liegt dein Projekt, dort werden Dateien angepasst, Prüfungen ausgeführt und Änderungen sichtbar gemacht, sodass du jede einzelne nachvollziehen kannst.

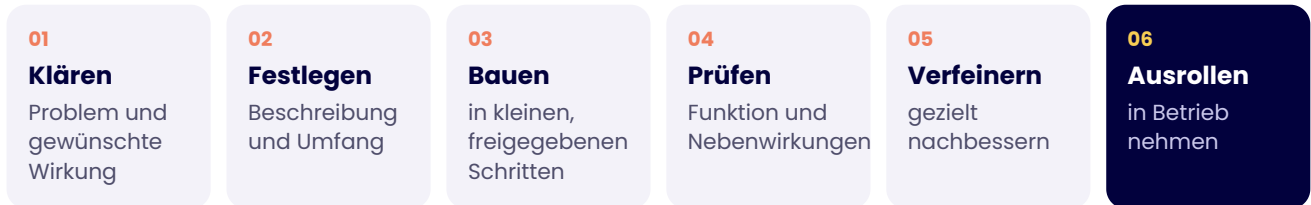
AUF DEN PUNKT

Denken und Beschreiben im Chat. Bauen und Ändern in der Werkstatt. Die Rollen zählen, nicht die Logos.

||| KAPITEL 4

Sechs Etappen eines Projekts

Jedes gelungene Projekt durchläuft dieselben Etappen. Wer zwei davon in einem Auftrag zusammenzieht, verliert die Kontrolle.



ETAPPE	LEITFRAGE	ERGEBNIS
Klären	Welcher Zustand stört, und wen betrifft er?	Problem in zwei Sätzen
Festlegen	Was muss die erste Version können, was ausdrücklich nicht?	Produktbeschreibung mit Abnahmekriterien (Kapitel 5)
Bauen	Welcher kleine Schritt kommt als Nächstes?	Geprüfter Zwischenstand, gesichert in der Versionsverwaltung
Prüfen	Tut es das Richtige, und läuft der Rest noch?	Abgehakte Prüfliste (Kapitel 11)
Verfeinern	Was zeigt der Praxistest?	Einzelne, eng begrenzte Änderungen
Ausrollen	Ist der Betrieb abgesichert?	Live-Version mit Sicherung und Rückweg (Kapitel 12)

Die Reihenfolge ist kein Formalismus. Jede Etappe liefert die Grundlage für die nächste: Ohne geklärtes Problem keine brauchbare Beschreibung, ohne Beschreibung kein sinnvoller Plan, ohne Prüfung kein sicherer Livegang.

Die Etappen Bauen, Prüfen und Verfeinern wiederholen sich in Schleifen. Jede Schleife sollte so klein sein, dass du ihr Ergebnis in wenigen Minuten selbst ausprobieren kannst.

AUF DEN PUNKT

Je mehr Etappen ein einzelner Auftrag überspringen soll, desto mehr entscheidet die KI ohne dich.

||| KAPITEL 5

Die Produktbeschreibung

„Wir bräuchten mal ein Tool für unsere Sponsoren“ ist ein guter Anfang für ein Gespräch, aber noch kein Auftrag.

Bevor gebaut wird, beantwortest du sieben Fragen. Gemeinsam ergeben sie eine Produktbeschreibung, mit der jede KI und jeder Mensch arbeiten kann.

01

Was stört?

das Problem

02

Für wen?

die Nutzer

03

Welcher Ablauf?

der Hauptweg

04

Was zuerst?

Umfang Version 1

05

Welche Daten?

Dinge und Felder

06

Welche Regeln?

Logik und Rechte

07

Wann fertig?

Abnahmekriterien

1 • Was stört?

Beschreibe den Ist-Zustand, der dich ärgert. Etwa: Erbrachte Leistungen für Sponsoren, also Bandenzeiten, Social-Media-Posts oder Hospitality-Plätze, sind über Postfächer, Ordner und Chats verstreut. Am Saisonende kann niemand vollständig belegen, was geliefert wurde.

2 • Für wen?

„Alle im Club“ ist zu ungenau. Je konkreter die Nutzer, desto leichter fallen spätere Entscheidungen. Im Beispiel: das Partnermanagement am Laptop, das Social-Media- und Eventteam am Smartphone, oft direkt am Spieltag.

3 • Welcher Ablauf?

Skizziere den wichtigsten Weg in Schritten: Leistung erbracht → Partner auswählen → Leistung mit Nachweis erfassen → Übersicht pro Partner aktualisiert → Report an den Sponsor.

4 • Was zuerst?

Sortiere alle Wünsche in drei Körbe: unverzichtbar, wünschenswert, später. Dass die KI etwas schnell bauen kann, ist kein Grund, es in die erste Version zu packen.

5 • Welche Daten?

Benenne die Dinge, mit denen das Werkzeug arbeitet, und ihre Eigenschaften, noch bevor von Tabellen die Rede ist. Etwa Partner (Name, Paket, Vertragslaufzeit) und Leistung (Art, Datum, Nachweis, erfasst von).

6 · Welche Regeln?

Halte fest, was immer gelten muss. Zum Beispiel: Eine Leistung gehört immer zu genau einem Partner mit laufendem Vertrag. Nur das Partnermanagement darf Partner und Pakete anlegen.

7 · Wann fertig?

Formuliere überprüfbare Abnahmekriterien. Zum Beispiel: Eine Leistung ist in unter 30 Sekunden erfasst. Die Übersicht zeigt nach dem Speichern sofort den neuen Stand. Auf dem Smartphone ist alles ohne Zoomen bedienbar.

BEISPIEL

Kurzbeschreibung „Partner-Leistungsnachweis“, Version 1

Problem	Erbrachte Sponsorenleistungen sind verstreut dokumentiert, Reports kosten Tage, Verlängerungen stehen auf wackliger Basis.
Nutzer	Partnermanagement (Laptop), Social-Media- und Eventteam (Smartphone)
Hauptablauf	Partner wählen → Leistung erfassen → Übersicht pro Partner
Unverzichtbar	Partnerliste, Leistung mit Nachweis-Link erfassen, Übersicht „vereinbart vs. erbracht“
Später	Fotos hochladen, PDF-Report für Sponsoren, Auswertungen über mehrere Seasons
Daten	Partner, Paket, Leistung
Regeln	Jede Leistung gehört zu genau einem aktiven Partner; Partner anlegen nur durch das Partnermanagement
Fertig, wenn	Erfassung unter 30 Sekunden, Übersicht sofort aktuell, mobil bedienbar

So nutzt du den Denkpartner dafür

Gib dem Chat-Assistenten deine Rohidee und bitte ihn ausdrücklich, noch keinen Code zu schreiben. Seine Aufgabe ist, Lücken zu finden und die sieben Fragen mit dir durchzugehen. Wo er Annahmen trifft, soll er sie kennzeichnen, damit du sie bestätigen oder korrigieren kannst.

V02 Produktbeschreibung erstellen

vollständige Vorlage im Anhang

Meine Idee: [ROHIDEE IN EIGENEN WORTEN]

Schreib noch keinen Code. Erarbeite mit mir eine Produktbeschreibung: Problem, Nutzer, Hauptablauf, Rollen, Umfang Version 1, spätere Ideen, Daten und Felder, Regeln, Anbindungen, Datenschutz, offene Fragen und Abnahmekriterien. Kennzeichne jede Annahme, die du triffst.

||| KAPITEL 6

Der Start in der Werkstatt

Wie sauber ein Projekt beginnt, entscheidet, wie gut es sich später noch steuern lässt.

Erkunden, Planen, Bauen

Bei jeder größeren Änderung in genau dieser Reihenfolge.

01

Erkunden

Anforderungen und vorhandenes Projekt verstehen

02

Planen

Vorhaben in Etappen schneiden, Risiken benennen

03

Bauen

Etappe für Etappe umsetzen und prüfen

Ein neues Projekt anlegen

Gib der Werkstatt-KI deine Produktbeschreibung und verbiete ihr ausdrücklich, schon Code zu schreiben. Erst soll sie offene Punkte markieren, einen möglichst schlichten Aufbau vorschlagen und die Arbeit in kleine, überprüfbare Etappen schneiden. Dann wartet sie auf dein Okay.

V05 Startplan für ein neues Projekt

Anhang

Wir beginnen ein neues Projekt.
Worum es geht: [KURZBESCHREIBUNG]
Nutzer: [WER] Hauptablauf: [SCHRITTE]
Umfang Version 1: [LISTE] Technische Vorgaben: [FALLS VORHANDEN]

Schreib noch keinen Code. Prüfe die Anforderungen auf Lücken, schlag den einfachsten tragfähigen Aufbau vor, nenne Abhängigkeiten und teile die Arbeit in kleine Etappen, die sich einzeln prüfen lassen.
Warte danach auf meine Freigabe.

Ein bestehendes Projekt übernehmen

Wer ein vorhandenes Projekt weiterentwickelt, ändert zuerst gar nichts. Die KI soll lesen und dir eine technische Lagekarte liefern: Zweck, eingesetzte Technik, Aufbau, Oberfläche, Serverlogik, Datenhaltung, Anmeldung, Schnittstellen, Gestaltung, Tests, Veröffentlichung und heikle Stellen. Was sie aus dem Projekt nicht sicher ablesen kann, soll sie als unbekannt markieren, statt zu raten.

V06 Bestandsaufnahme

Anhang

Lies dieses Projekt, ohne irgendetwas zu verändern. Erstelle eine kompakte Lagekarte: Zweck, Technik, Ordnerstruktur, Oberfläche, Serverlogik, Datenhaltung, Anmeldung, Schnittstellen, Gestaltung, Tests, Build, Veröffentlichung und sensible Bereiche. Was du nicht sicher aus dem Code ablesen kannst, markierst du als unbekannt.

Das Projektgedächtnis: die README

Alles, was jemand braucht, um das Projekt zu verstehen und zu starten, gehört als Datei ins Projekt selbst und nicht in einen Chatverlauf. Dazu zählen Zweck, Installation, Start der Entwicklungsumgebung, benötigte Umgebungsvariablen, Einrichtung der Datenbank, Build, Veröffentlichung und wichtige Architekturentscheidungen.

Diese Datei ist das Gedächtnis deines Projekts. Eine neue KI-Sitzung, ein Kollege oder du selbst in drei Monaten finden dort sofort den Einstieg.

Dauerhafte Spielregeln

Die meisten Werkstatt-Werkzeuge erlauben feste Projektregeln, die bei jedem Auftrag mitgelesen werden. Formuliere sie so konkret, dass man sie befolgen kann. Gute Beispiele:

- Neue Bibliotheken nur mit kurzer Begründung.
- Immer die vorhandenen Komponenten und Farben verwenden.
- Ändert sich ein Verhalten, werden die zugehörigen Tests angepasst.
- Texte in der Oberfläche auf Deutsch, in Du-Form.

Die Versionsverwaltung als Sicherheitsseil

Git hält Zwischenstände fest, zu denen du jederzeit zurückkehren kannst. Du musst kein Experte sein. Es reicht ein fester Rhythmus:



Bitte die KI nach jeder gelungenen Etappe, einen Commit mit verständlicher Beschreibung anzulegen. Geht später etwas schief, ist der Rückweg ein einziger Befehl.

AUF DEN PUNKT

Neues Projekt: erst planen lassen. Fremdes Projekt: erst lesen lassen. Und von Anfang an regelmäßig sichern.

||| KAPITEL 7

Das AUFTRAG-Prinzip

Ein guter Prompt ist ein vollständiger Arbeitsauftrag. Sieben Bausteine sorgen dafür, dass nichts fehlt.

A Absicht Was soll am Ende besser sein?	U Umfeld Was muss die KI über das Projekt wissen?	F Fokus Welche konkrete Arbeit ist jetzt dran?	T Tauglichkeit Welche Eigenschaften muss das Ergebnis haben?	R Riegel Was darf auf keinen Fall angefasst werden?	A Abgabe Was willst du danach zurückbekommen?	G Gegenprobe Wie soll das Ergebnis geprüft werden?
---	---	--	--	---	---	--

Die Bausteine im Einzelnen

BAUSTEIN	FRAGE	TYPISCHE FORMULIERUNG
Absicht	Welches Ziel steckt hinter der Änderung?	„Ziel ist, dass ...“
Umfeld	Was ist schon da, was ist festgelegt?	„Die Übersicht ist freigegeben, ...“
Fokus	Was genau soll jetzt passieren, und nur das?	„Ergänze ausschließlich ...“
Tauglichkeit	Woran misst du die Qualität?	„Muss auch auf 360 px Breite ...“
Riegel	Wo verläuft die Grenze?	„Nicht verändern: ...“
Abgabe	Was soll die KI berichten?	„Nenne danach geänderte Dateien ...“
Gegenprobe	Welche Prüfung läuft am Schluss?	„Führe Build und Tests aus ...“

Der wichtigste Baustein ist der Riegel. Die meisten Ärgernisse beim Arbeiten mit KI entstehen nicht durch das, was sie falsch macht, sondern durch das, was sie ungefragt zusätzlich ändert.

Ein Beispiel aus der Praxis

VAGE

„Mach die Partnerliste übersichtlicher.“

Was heißt übersichtlich? Was darf sich ändern? Woran erkennt man Erfolg?

VOLLSTÄNDIG

„Ausgelaufene Partnerverträge sollen sich ausblenden lassen.“

Unten ausformuliert nach dem AUFTRAG-Prinzip.

Absicht: Wer die Partnerliste öffnet, soll zuerst die laufenden Partner sehen.
 Umfeld: Die Liste und ihr Design sind freigegeben. Daten liegen bereits vor.
 Fokus: Ergänze über der Liste einen Schalter „Ausgelaufene ausblenden“.
 Tauglichkeit: Standard ist eingeschaltet. Die Wahl bleibt nach Neuladen erhalten. Funktioniert auch auf 360 px Breite.
 Riegel: Keine Änderungen an Datenmodell, Sortierung oder Gestaltung.
 Keine neue Bibliothek.
 Abgabe: Liste der geänderten Dateien und eine Zeile pro Änderung.
 Gegenprobe: Build ausführen, Schalter in beiden Stellungen testen.

Wie lang muss ein Auftrag sein?

So lang, wie das Risiko groß ist. Eine Beschriftung zu ändern braucht keine sieben Bausteine, ein Satz mit klarem Riegel genügt. Eine neue Datenstruktur oder eine Anbindung an ein fremdes System dagegen verdient den vollständigen Auftrag, oft sogar einen eigenen Planungsschritt davor.

ART DER ÄNDERUNG	EMPFOHLENE FORM
Text, Farbe, Abstand	Ein Satz plus Riegel
Neues Bedienelement	Fokus, Tauglichkeit, Riegel, Gegenprobe
Neue Funktion	Alle sieben Bausteine
Daten, Rechte, Anbindungen	Erst Plan (V07), dann alle sieben Bausteine

AUF DEN PUNKT

Ein Prompt sagt nicht nur, was passieren soll, sondern auch, was nicht passieren darf, und wie geprüft wird.

||| KAPITEL 8

Sicher ändern

Kleine Änderung, kleiner Wirkungsbereich. So bleibt ein wachsendes Projekt beherrschbar.

Der Wirkungsbereich

Je schmaler die Zone, die sich ändern darf, desto kleiner das Risiko.

GESCHÜTZT

ÄNDERUNGZONE

GESCHÜTZT

Bitte die KI vor jeder Änderung, zuerst nur die betroffenen Dateien zu lesen und den kleinstmöglichen Eingriff zu bestimmen. Sie soll benennen, welche Dateien sie anfassen will und was davon abhängt. Alles außerhalb dieser Zone bleibt unberührt: kein Aufräumen nebenbei, keine neuen Bibliotheken ohne zwingenden Grund. Zum Schluss lässt sie die passenden Prüfungen laufen und fasst zusammen.

V09 Eng begrenzte Änderung

Anhang

Änderung: [WAS GENAU]

Lies zuerst nur die Dateien, die dafür relevant sind. Bestimme den kleinstmöglichen Eingriff, nenne die betroffenen Dateien und ihre Abhängigkeiten. Nichts außerhalb dieses Bereichs anfassen, nichts nebenbei umbauen, keine neue Bibliothek ohne zwingenden Grund. Danach: passende Tests, Typprüfung, Linter oder Build ausführen und in wenigen Sätzen berichten, was geändert und was geprüft wurde.

Die Gestaltung einfrieren

Ist das Aussehen einmal abgenommen, sag das bei jedem Auftrag dazu. Farben, Schriften, Bausteine, Abstände und Muster bleiben, wie sie sind. Andernfalls interpretiert die KI das Design bei jeder Gelegenheit ein wenig neu, und nach zehn Änderungen sieht dein Werkzeug aus wie zehn verschiedene.

Aufräumen hat seinen eigenen Termin

Umbauen ohne neue Funktion, also Refactoring, kann sinnvoll sein. Heikel wird es, wenn es unbemerkt in einen ganz anderen Auftrag rutscht. Trenne deshalb streng: Die nötige Änderung kommt jetzt, die mögliche Verbesserung bekommt später einen eigenen Auftrag (V16).

Jede Änderung nachsehen

Schau nicht nur, ob das Ergebnis stimmt, sondern auch, wie viel dafür verändert wurde. Die Werkstatt zeigt dir jede Änderung als Gegenüberstellung von alt und neu. Eine kleine Aufgabe sollte eine kleine, gut lesbare Gegenüberstellung erzeugen. Ist sie überraschend groß, frag nach.

PRÜFLISTE

Sechs Blicke auf jede Änderung

- | | |
|--|--|
| ☐ Wie viele Dateien sind betroffen? | ☐ Kam eine neue Bibliothek dazu? |
| ☐ Wurde Code gelöscht? | ☐ Berührt es Anmeldung oder Rechte? |
| ☐ Ändert sich etwas an der Datenbank? | ☐ Wurden Konfiguration oder Zugangsdaten angefasst? |

Lautet eine dieser Antworten „ja“, obwohl die Aufgabe das nicht verlangt hat, ist das ein Warnsignal. Setze lieber auf den letzten Sicherungsstand zurück und formuliere den Auftrag mit einem schärferen Riegel neu.

Änderungen erklären lassen

Du musst die Gegenüberstellung nicht Zeile für Zeile lesen können. Lass sie dir in Alltagssprache übersetzen, und bitte ausdrücklich um eine kritische Haltung.

V20 Änderung erklären

Anhang

Erklär mir diese Änderung, als wäre ich kein Entwickler. Gliedere in:

1. was sich fachlich ändert, 2. welche Dateien betroffen sind,
 3. welches bisherige Verhalten anders ist, 4. neue Bibliotheken oder Einstellungen, 5. Folgen für Daten, Anmeldung und Sicherheit,
 6. was ich von Hand testen sollte, 7. was davon nicht nötig gewesen wäre.
- Sei dabei ausdrücklich kritisch.

AUF DEN PUNKT

Den Umfang einer Änderung zu prüfen ist genauso wichtig wie ihr Ergebnis.

||| KAPITEL 9

Fehler methodisch finden

„Geht nicht“ ist keine Beschreibung. Wer einen Fehler gut schildert, hat ihn schon halb gefunden.

Eine brauchbare Fehlermeldung hat fünf Teile

Soll

Was hätte passieren müssen?

Ist

Was passiert stattdessen?

Weg dorthin

Welche Schritte führen zum Fehler?

Spuren

Fehlermeldung, Protokoll, Screenshot

Umstände

Was wurde zuletzt geändert?

Mit diesen fünf Angaben kann die KI gezielt suchen, statt zu raten. Besonders der letzte Punkt spart Zeit: Die meisten Fehler entstehen durch die jüngste Änderung.

Vom Symptom zur Ursache

Zuerst verstehen, dann reparieren. Niemals Flicken auf Flicken.

01

Nachstellen

02

Ursache suchen

03

Korrektur planen

04

Minimal ändern

05

Testen

06

Bestätigen

Ursache und Symptom trennen

Das Symptom ist, was du siehst, zum Beispiel eine leere Liste. Die Ursache ist, warum das passiert, etwa ein Filter, der falsch gesetzt wird. Wer nur das Symptom behandelt, versteckt das Problem und bekommt es an anderer Stelle zurück. Verlange deshalb von der KI, dass sie die Ursache belegt, bevor sie Code ändert.

V14 Fehleranalyse

Anhang

Soll: [WAS PASSIEREN SOLLTE]

Ist: [WAS PASSIERT]

Weg dorthin: [SCHRITTE]

Spuren: [FEHLERMELDUNG, PROTOKOLL]

Umstände: [LETZTE ÄNDERUNG]

Stell den Fehler nach, verfolge den betroffenen Codepfad und benenne die wahrscheinlichste Ursache. Unterscheide Ursache und Symptom. Plane die kleinste Korrektur, ändere nichts außerhalb und prüfe danach. Ist die Ursache nicht belegt, ändere noch keinen Code.

Ein Fall aus der Praxis

Soll	Nach dem Erfassen eines Social-Media-Posts zeigt die Partnerübersicht die Leistung als erbracht.
Ist	Die Leistung bleibt als offen markiert. Erst nach Neuladen der Seite stimmt die Anzeige.
Erste Vermutung	Die Leistung wird nicht gespeichert.
Tatsächliche Ursache	Die Leistung wurde korrekt gespeichert. Die Übersicht arbeitete aber mit einer Kopie der Daten, die sie beim Öffnen geladen hatte, und erfuhr von der Änderung nichts.
Kleinste Korrektur	Nach dem Speichern die Übersicht neu laden lassen. Eine Zeile, keine neue Datenhaltung, kein Umbau.

Hätte man der ersten Vermutung geglaubt, wäre ein funktionierender Speichervorgang umgebaut worden, mit neuen Fehlern als Folge. Erst die Frage „Ist die Ursache belegt?“ führte zur richtigen, winzigen Korrektur.

Fehler nicht stapeln

Taucht nach einer Korrektur ein neuer Fehler auf, prüfe zuerst, ob er von genau dieser Korrektur stammt. Oft ist es schneller, zum letzten funktionierenden Stand zurückzukehren und den Auftrag besser zu formulieren, als eine Kette von Notlösungen zu bauen, die niemand mehr versteht.

Wenn eine Fehlermeldung rätselhaft ist

Lass sie dir zuerst übersetzen, ohne dass etwas geändert wird: Was bedeutet die Meldung, was sind übliche Auslöser, welcher davon ist in deinem Projekt am wahrscheinlichsten, und wie lässt sich das überprüfen? Die passende Vorlage ist VI5.

AUF DEN PUNKT

Erst die Ursache belegen, dann die kleinste Korrektur. Im Zweifel zurück zum letzten guten Stand.

||| KAPITEL 10

Die Technik-Landkarte

Für gute Entscheidungen brauchst du kein Informatikstudium, sondern eine Übersicht, wo was passiert.

Wie eine typische Web-Anwendung aufgebaut ist

Mensch

am Handy oder PC

Oberfläche

Seiten, Masken, Knöpfe

Serverlogik

Regeln, Rechte, Abläufe

Datenbank

dauerhafter Speicher

Schnittstellen zu anderen Systemen

Benachrichtigungen von außen

BEGRIFF	WAS DAHINTERSTECKT
Oberfläche	Alles, was Menschen sehen und bedienen: Seiten, Formulare, Tabellen, Menüs. Fachbegriff: Frontend.
Serverlogik	Alles, was man nicht dem Browser überlassen darf: Daten speichern, Rechte prüfen, Zahlungen abwickeln, andere Systeme ansprechen, Geschäftsregeln durchsetzen. Fachbegriff: Backend.
Datenbank	Bewahrt strukturierte Informationen dauerhaft auf. Wie die Daten geordnet sind, entscheidet stark darüber, wie leicht sich das Werkzeug später erweitern lässt.
Schnittstelle	Ein vereinbarter Weg, auf dem zwei Systeme Daten austauschen. Fachbegriff: API.
Webhook	Ein anderes System meldet sich von selbst, sobald dort etwas passiert, statt dass dein Werkzeug ständig nachfragen muss.
Umgebungsvariable	Eine Einstellung, die außerhalb des Codes liegt, etwa ein Zugangsschlüssel. So bleibt Geheimes aus dem Quelltext heraus.

Wer bist du, und was darfst du?

ANMELDUNG

Wer bist du?

Zum Beispiel per E-Mail und Passwort. Fachbegriff: Authentifizierung.

BERECHTIGUNG

Was darfst du?

Zum Beispiel: Darf diese Person neue Sponsoren anlegen? Fachbegriff: Autorisierung.

Zugangsdaten gehören nicht in den Code

Passwörter, Schlüssel und Tokens dürfen nie im Quelltext stehen. Wer den Code sieht, sähe sonst auch den Zugang. Stattdessen liest das Programm sie zur Laufzeit aus Umgebungsvariablen, die getrennt verwaltet werden.

SO NICHT

```
const schluessel = "sk-live-4f9a..."
const dbPasswort = "Sommer2026!"
```

Jeder mit Zugriff auf den Code kennt die Zugänge.

SO RICHTIG

```
const schluessel = process.env.API_KEY
const dbPasswort = process.env.DB_PASSWORD
```

Die Werte liegen getrennt und geschützt.

Fremde Systeme anbinden

Sobald dein Werkzeug mit einem anderen System sprechen soll, etwa mit dem CRM, der Buchhaltung oder einem Kalender, wird es heikel. Hier erfindet KI besonders gern Funktionen, die es gar nicht gibt. Lass deshalb immer zuerst die aktuelle, offizielle Dokumentation des Anbieters auswerten und einen Plan erstellen, bevor eine einzige Zeile entsteht.

Der Plan sollte beantworten: Wie meldet sich mein Werkzeug an? Welche Endpunkte und Daten werden gebraucht? Gibt es Mengengrenzen? Was passiert bei Fehlern? Gibt es Benachrichtigungen von außen? Welche Umgebungsvariablen braucht es? Und was bedeutet das für Datenschutz und Sicherheit?

V18 Schnittstelle anbinden

Anhang

Wir wollen [SYSTEM] anbinden, um [ZWECK]. Noch nichts umsetzen.
Werte zuerst die aktuelle offizielle Dokumentation und unseren Aufbau aus. Beschreibe Anmeldung, benötigte Endpunkte und Daten, Mengengrenzen, Fehlerfälle, Benachrichtigungen, nötige Umgebungsvariablen sowie Datenschutz und Sicherheit. Erstelle daraus einen Umsetzungsplan.
Verwende nur Endpunkte und Parameter, die in der Dokumentation stehen.

AUF DEN PUNKT

Oberfläche zeigt, Serverlogik entscheidet, Datenbank merkt sich. Geheimes bleibt außerhalb des Codes, und fremde Systeme werden erst nach Blick in die Dokumentation angebunden.

||| KAPITEL 11

Prüfen statt hoffen

Dass sich ein Projekt fehlerfrei bauen lässt, heißt noch lange nicht, dass es tut, was es soll.

Jede neue Funktion durchläuft sechs Prüfungen. Die ersten vier machst du selbst durch Ausprobieren, die fünfte erledigt die KI, die sechste ist ein gemeinsamer Blick auf die Daten.

01

Richtig

Tut die Funktion genau das, was beschrieben war?

02

Nebenwirkungen

Funktioniert alles Angrenzende weiterhin?

03

Fehleingaben

Was passiert bei leeren, falschen oder doppelten Eingaben?

04

Bildschirmgrößen

Klappt es auf Handy, Tablet und großem Monitor?

05

Technische Prüfungen

Tests, Typprüfung, Linter und Build laufen durch.

06

Daten

Wird korrekt gespeichert und wieder geladen?

Die KI gegen die Abnahmekriterien prüfen lassen

Bitte die KI am Ende jeder Etappe, ihr Ergebnis Punkt für Punkt mit den Abnahmekriterien abzugleichen. Wichtig ist der letzte Satz der Vorlage: Sie soll nichts als geprüft ausgeben, was sie nicht tatsächlich geprüft hat. So trennst du belastbare Aussagen von Vermutungen.

V19 Abnahme gegen Anforderungen

Anhang

Gleiche die eben umgesetzte Funktion mit den ursprünglichen Anforderungen und Abnahmekriterien ab. Führe, soweit möglich, Tests, Typprüfung, Linter, Build und die relevanten Fehlerfälle aus. Liefere vier Listen: bestätigt, nicht bestätigt, muss noch von Hand getestet werden, bekannte Risiken. Gib nichts als geprüft aus, was du nicht wirklich geprüft hast.

Die Liste „muss noch von Hand getestet werden“ ist deine Aufgabe. Plane dafür bei jeder Etappe ein paar Minuten ein. Es sind die besten Minuten des ganzen Projekts.

PRÜFLISTE

Bevor ein Zwischenstand gesichert wird

Vor jedem Commit einmal durchgehen.

- ☐ Die beauftragte Funktion arbeitet wie beschrieben.
- ☐ Es wurde nur geändert, was der Auftrag verlangt hat.
- ☐ Keine Datei wurde unerwartet angefasst.
- ☐ Keine überflüssige Bibliothek kam hinzu.
- ☐ Benachbarte Funktionen wurden ausprobiert.
- ☐ Fehleingaben werden sinnvoll abgefangen.
- ☐ Tests, Typprüfung, Linter und Build laufen, soweit vorhanden, durch.
- ☐ Es stehen keine Zugangsdaten im Code.
- ☐ Die README ist bei Bedarf auf dem neuen Stand.

PRÜFLISTE

Bevor eine Version live geht

Vor jeder Veröffentlichung einmal durchgehen.

- | | |
|---|--|
| ☐ Build für den Livebetrieb erfolgreich | ☐ Alle Umgebungsvariablen gesetzt |
| ☐ Datenbankänderungen vorbereitet | ☐ Aktuelle Datensicherung vorhanden |
| ☐ Anmeldung getestet | ☐ Berechtigungen getestet |
| ☐ Hauptabläufe durchgespielt | ☐ Darstellung auf allen Geräten geprüft |
| ☐ Protokolle sind einsehbar | ☐ Rückweg zur Vorversion bekannt |
| ☐ Keine Test-Zugänge oder Debug-Schlüssel live | ☐ Adresse und HTTPS funktionieren |

AUF DEN PUNKT

Ein erfolgreicher Build ist eine Voraussetzung, kein Beweis. Erst die sechs Prüfungen zeigen, ob eine Funktion wirklich fertig ist.

KAPITEL 12

In Betrieb nehmen

Läuft es auf deinem Rechner, ist viel geschafft. Ein verlässliches System für echte Nutzer ist es deshalb noch nicht.

Drei Umgebungen, drei Aufgaben

Entwicklung

auf dem eigenen Rechner
bauen und ausprobieren

Testumgebung

unter echten Bedingungen
prüfen, ohne echte Nutzer

Livebetrieb

echte Nutzer, echte Daten, volle
Verantwortung

THEMA	WORAUF ES ANKOMMT
Hosting	Wähle die einfachste Plattform, die deine Anforderungen zuverlässig trägt. Nicht jedes Werkzeug braucht einen eigenen Server.
Adresse und HTTPS	Live-Anwendungen laufen verschlüsselt, ganz besonders bei Anmeldung, Formularen und personenbezogenen Daten.
Datensicherung	Eine Sicherung zählt erst, wenn du einmal ausprobiert hast, dass sie sich zurückspielen lässt.
Protokolle	Meldet jemand ein Problem, zeigen die Protokolle, was im System tatsächlich passiert ist.
Überwachung	Ein Hinweis, wenn das System nicht erreichbar ist oder auffällig viele Fehler wirft.
Rückweg	Bricht eine neue Version etwas, brauchst du einen erprobten Weg zurück zur letzten funktionierenden.
Zugangsdaten	Live-Schlüssel liegen nur in der Live-Umgebung, getrennt von Test- und Entwicklungszugängen.

Startklar-Check

Bevor du veröffentlichst, lass die KI das Projekt auf Betriebsreife durchleuchten, ohne dabei selbst etwas zu veröffentlichen. Das Ergebnis ist eine Liste mit Blockern, Empfehlungen, Schritten für den Livegang, Kontrollen danach und dem Rückweg.

V24 Startklar-Check

Anhang

Prüfe das Projekt auf Betriebsreife. Veröffentliche nichts. Sieh dir Build, Hosting, Laufzeitumgebung, Datenbank und Datenbankänderungen, Umgebungsvariablen, Zugangsdaten, HTTPS, Adresse, Dateiablage, Protokolle, Sicherung, Überwachung, Anmeldung, Rechte, Schnittstellen, Fehlerbehandlung und Rückweg an. Liefere: Blocker, Empfehlungen, Schritte für den Livegang, Kontrollen danach, Weg zurück.

KAPITEL 13

Zwölf Stolperfallen

Diese Muster machen Projekte langsam und teuer. Sortiert nach der Etappe, in der sie meist auftreten.

BEIM KLÄREN UND FESTLEGEN

01 Alles in einem Auftrag

Je größer die Bitte, desto mehr Entscheidungen fällt die KI allein. Teile auf.

02 Gefühlswörter statt Kriterien

„Moderner“ oder „professioneller“ kann niemand umsetzen. Beschreibe, was man sehen oder messen kann.

03 Kein Ziel definiert

Ohne Abnahmekriterien gibt es kein Ende. Fertig heißt: Kriterien erfüllt, gesichert, nächster Auftrag.

BEIM BAUEN

04 Alles abnicken

Was technisch klingt, ist nicht automatisch richtig. Lass dir Vorschläge begründen.

05 Ohne Sicherungsstände arbeiten

Wer keinen bekannten guten Stand hat, findet nur schwer zurück. Nach jeder Etappe sichern.

06 Das Design jedes Mal neu erklären

So entsteht ein Flickenteppich. Gestaltung einmal festlegen und danach einfrieren.

07 Funktionierendes ohne Grund anfassen

Jede Änderung braucht einen konkreten Nutzen, sonst ist sie ein Risiko ohne Gegenwert.

BEIM PRÜFEN UND REPARIEREN

08 Notlösungen stapeln

Jeder Fehler wird mit mehr Code zugedeckt. Besser: Ursache finden, kleinste Korrektur, notfalls zurücksetzen.

09 Den Unterbau ständig wechseln

Datenbank, Plattform, Architektur werden ausgetauscht, sobald es hakt. Jeder Wechsel bringt neue Fehler. Nur wechseln, wenn ein belegtes Problem es verlangt.

10 Keine zweite Meinung

Bei Entscheidungen, die sich schwer zurückdrehen lassen, Alternativen vergleichen und gegenlesen lassen.

BEIM AUSROLLEN UND BETREIBEN

11 Sicherheit abgeben und vergessen

Die KI unterstützt, die Verantwortung für Daten und Zugänge bleibt bei dir.

12 Nichts aufschreiben

Wichtige Entscheidungen gehören ins Projekt, nicht in einen Chatverlauf, der morgen verschwunden ist.

||| KAPITEL 14

Nachbauen statt abonnieren

Software zur Miete, kurz SaaS, gerät durch KI unter Druck. Einzelne Funktionen gekaufter Programme lassen sich heute untersuchen und gezielt selbst nachbauen.

Ein Ansatz dafür heißt **REA**, kurz für Reverse Engineer Anything. REA ist kein fertiges Ersatzprogramm, sondern ein frei verfügbares Werkzeug für KI-Agenten wie Claude Code, Codex oder Cursor. Der Agent ruft REA auf, um eine Anwendung, eine Website oder ein Programm auf dem eigenen Rechner zu untersuchen. REA liefert Befunde mit Belegen zurück, und der Agent nutzt sie, um Abläufe zu erklären oder eine eigene Umsetzung zu schreiben und zu testen.

Die Arbeitsweise in vier Schritten

Zerlegen, verstehen, auswählen, neu bauen.

01

Untersuchen

REA analysiert die bestehende Anwendung lokal und hält fest, was es findet.

02

Verstehen

Der Agent erstellt ein Funktionsinventar: Masken, Abläufe, Daten, Regeln.

03

Auswählen

Du entscheidest, welche Funktionen du wirklich brauchst und welche nicht.

04

Neu bauen

Die Auswahl wird zur Produktbeschreibung und nach dem AUFTRAG-Prinzip umgesetzt.

Ein Beispiel aus der Praxis

≈ 1.000 €

weniger Lizenzkosten
pro Monat

In einem Kundenprojekt wurden mit diesem Ansatz mehrere Büroanwendungen nachgebaut. Die Oberflächen blieben an den bisherigen Lösungen orientiert, sodass niemand ein komplett neues Werkzeug lernen musste. Das ist ein einzelnes Beispiel und kein Versprechen für jedes Unternehmen.

Der eigentliche Gewinn: Kontrolle über die Daten

Die Daten eines Clubs oder Unternehmens müssen nicht mehr zwangsläufig über zehn verschiedene Cloud-Dienste verteilt sein. Du kannst bewusst entscheiden, welche Teile lokal oder unter eigener Kontrolle laufen und welche Dienste du weiterhin einkaufst.

Die Schattenseiten ehrlich anschauen

Wer selbst baut, übernimmt Verantwortung, die vorher beim Anbieter lag. Vier Fragen gehören deshalb vor jede Entscheidung:

FRAGE	WORAUF ES ANKOMMT
Datensicherheit	Wo liegen die Daten, wer hat Zugriff, wie werden sie gesichert? Die Prüflisten aus Kapitel 11 und 12 gelten hier besonders streng.
Ausfall	Was passiert, wenn das Werkzeug nicht läuft? Es braucht einen Notfallplan, eine erprobte Datensicherung und einen Rückweg.
Wartung	Wer kümmert sich um Updates, Fehler und Sicherheitslücken, auch in zwei Jahren noch? Ohne klare Zuständigkeit entsteht Schatten-IT.
Rechte	Darfst du die Software überhaupt untersuchen? Viele Lizenzverträge regeln oder verbieten Reverse Engineering. Funktionen neu umzusetzen ist etwas anderes, als Code, Texte oder Gestaltung zu kopieren. Klär das vorher, im Zweifel rechtlich.

REA selbst ist nach eigener Beschreibung nur für erlaubte Zwecke gedacht: Forschung, Lernen, Kompatibilität, Erhalt von Software und die Analyse von Programmen, die dir gehören oder die du untersuchen darfst. Kommerzielle Analysewerkzeuge, die REA einbinden kann, brauchen eine eigene, rechtmäßig erworbene Lizenz.

Gefühle und tatsächliche Sicherheit

Beim Blick auf Branchensoftware in Kundenprojekten zeigt sich immer wieder: Auch große Anbieter sind beim Datenschutzmanagement oder bei Ausfallplänen nicht immer so sauber aufgestellt, wie man annimmt. Trotzdem fühlt sich ein Abo oft sicherer an, weil man im Ernstfall beim Support anrufen kann. Diese Sicherheit lohnt einen zweiten Blick, in beide Richtungen.

Selbst bauen oder weiter kaufen?

NACHBAUEN PRÜFEN, WENN

nur ein kleiner Teil der Funktionen genutzt wird ·
die Lizenzkosten spürbar sind · die Daten unter
eigener Kontrolle bleiben sollen · Wartung und
Betrieb geklärt sind · die Rechtslage geklärt ist

WEITER KAUFEN, WENN

die Software gesetzliche Pflichten abdeckt, etwa
in der Buchhaltung · der Anbieter Haftung und
Updates garantiert · niemand den Betrieb
übernehmen kann · die Lizenz Analyse verbietet

AUF DEN PUNKT

Nachbauen ist eine Option, kein Automatismus. Es lohnt sich dort, wo Kosten, Datenhoheit und Betrieb zusammenpassen und die Rechte geklärt sind. Eine Vorlage für das Funktionsinventar findest du im Anhang (V28).

||| KAPITEL 15

Guided Vibe Coding mit menter

Dieses Playbook funktioniert auch allein. Mit uns musst du es nicht allein anwenden.

Wir entwickeln ein Werkzeug, in dem genau dieser Arbeitsablauf fest eingebaut ist. Die KI führt dich Schritt für Schritt: vom geklärten Problem über die Produktbeschreibung und einen freigegebenen Plan bis zum geprüften Ergebnis. Sie hält Änderungen klein, zieht Riegel und prüft gegen deine Abnahmekriterien.

Das Playbook steckt in der KI

Beschreibung, AUFTRAG-Prinzip, eng begrenzte Änderungen, Fehleranalyse und Startklar-Check laufen als geführte Schritte. Du musst keine Vorlagen abtippen.

Menschen an deiner Seite

Wo es technisch tief wird, bei Datenbanken, Schnittstellen, Sicherheit oder Betrieb, steigen unsere IT-Spezialisten direkt in dein Projekt ein.

Mitten im Unternehmenskontext

Eingebettet in das menter Digital OS: Prozesse, Werkzeuge, IT-Landschaft und Firmenwissen an einem Ort. Von dort geht es direkt in Automatisierungen oder eigene Anwendungen.

Erst die Lösung, dann der Code

Nicht jedes Problem braucht eine neue App. Im Digital OS prüft die KI zuerst, ob ein besserer Ablauf, die richtige Einstellung in einem vorhandenen System, eine Lösung vom Markt oder eine Automatisierung genügt, und umgekehrt, ob sich ein teures Abo durch einen gezielten Nachbau ablösen lässt. Gebaut wird nur, wenn Bauen die beste Antwort ist, und dann mit dem gesamten Wissen über dein Unternehmen statt bei null.

Pilotphase ab Januar 2027 mit 10 Unternehmen. Jetzt einen Platz sichern.

www.menter.digital

||| ANHANG

Vorlagen zum Kopieren

28 Vorlagen, geordnet nach der Situation, in der du gerade steckst. Ersetze alles in eckigen Klammern. Die Vorlagen sind Startpunkte, passe sie an dein Projekt an.

EINE IDEE PRÜFEN UND BESCHREIBEN

V01 Ideen-Härtetest

Hier ist meine Produktidee: [IDEE].
 Prüfe sie kritisch, nicht wohlwollend. Untersuche: Wie groß und wie häufig ist das Problem? Für wen genau? Welche Alternativen nutzen die Leute heute? Warum würde jemand dafür zahlen oder umsteigen? Welche Annahmen stecken darin, und welche ist die riskanteste? Wie ließe sich das mit möglichst wenig Bauaufwand testen? Such nicht nach Gründen, mir recht zu geben.

V02 Produktbeschreibung erstellen

Meine Idee: [ROHIDEE IN EIGENEN WORTEN]
 Schreib noch keinen Code. Erarbeite mit mir eine belastbare Produktbeschreibung mit diesen Punkten:
 Problem · Nutzer · Hauptablauf · Rollen · Umfang Version 1 · spätere Ideen · Daten und Felder · Regeln · Anbindungen an andere Systeme · Sicherheit und Datenschutz · offene Fragen · Abnahmekriterien.
 Triff keine technischen Entscheidungen, die jetzt noch nicht nötig sind.
 Kennzeichne jede Annahme.

V03 Funktionsbeschreibung

Wir wollen diese Funktion ergänzen: [FUNKTION]. Noch keinen Code.
 Beschreibe: welches Nutzerproblem sie löst, wie sie sich verhalten soll, den Bedienablauf, die nötigen Daten, Regeln, Fehlerfälle, Berechtigungen, Auswirkungen auf Bestehendes und überprüfbare Abnahmekriterien.

V04 Technikwahl

Für dieses Werkzeug suchen wir den einfachsten Aufbau, der sich gut warten lässt: [WERKZEUG]. Anforderungen: [ANFORDERUNGEN].
 Rahmen: [VORKENNTNISSE / HOSTING / BUDGET].
 Schlag höchstens drei Varianten vor und vergleiche sie nach Komplexität, laufenden Kosten, Wartung, Hosting, Wachstum, Lernaufwand und Abhängigkeit von einzelnen Anbietern.

EIN PROJEKT STARTEN ODER ÜBERNEHMEN

V05 Startplan für ein neues Projekt

Wir beginnen ein neues Projekt.
 Worum es geht: [KURZBESCHREIBUNG]
 Nutzer: [WER] Hauptablauf: [SCHRITTE]
 Umfang Version 1: [LISTE] Technische Vorgaben: [FALLS VORHANDEN]
 Schreib noch keinen Code. Prüfe die Anforderungen auf Lücken, schlag den einfachsten tragfähigen Aufbau vor, nenne Abhängigkeiten und Risiken und teile die Arbeit in kleine Etappen mit je eigenen Abnahmekriterien.
 Warte danach auf meine Freigabe.

V06 Bestandsaufnahme

Lies dieses Projekt, ohne irgendetwas zu verändern. Erstelle eine kompakte Lagekarte: Zweck, Technik, Ordnerstruktur, Datenfluss, Oberfläche, Serverlogik, Datenhaltung, Anmeldung, Schnittstellen, Gestaltung, Tests, Build, Veröffentlichung und sensible Bereiche. Was du nicht sicher aus dem Code ablesen kannst, markierst du als unbekannt, statt zu raten.

V07 Umsetzungsplan

Wir wollen Folgendes umsetzen: [VORHABEN]. Noch nichts ändern.
 Sieh dir an, wie es heute gelöst ist, und entwirf den kleinsten sinnvollen Plan. Nenne betroffene und neue Dateien, Änderungen an Daten, Schnittstellen und Oberfläche, Risiken, nötige Tests und die Reihenfolge. Kein Umbau, der für das Vorhaben nicht nötig ist.

V08 Lagebild wiederherstellen

Bevor wir weitermachen: Rekonstruiere den Stand dieses Projekts nur aus dem Code und der Dokumentation im Projekt. Was macht die Anwendung, wie ist sie aufgebaut, welche Funktionen gibt es, welche offenen Punkte sind notiert, und welche Bereiche wirken heikel? Nichts verändern.

GEZIELT ÄNDERN

V09 Eng begrenzte Änderung

Änderung: [WAS GENAU]

Lies zuerst nur die Dateien, die dafür relevant sind. Bestimme den kleinstmöglichen Eingriff, nenne die betroffenen Dateien und ihre Abhängigkeiten. Nichts außerhalb dieses Bereichs anfassen, nichts nebenbei umbauen, keine neue Bibliothek ohne zwingenden Grund. Danach: passende Prüfungen ausführen und kurz berichten, was geändert und was geprüft wurde.

V10 Oberfläche anpassen

Bereich: [WELCHER TEIL DER OBERFLÄCHE]

Gewünschtes Ergebnis: [WIE ES DANACH AUSSEHEN ODER WIRKEN SOLL]

Verwende nur die vorhandenen Bausteine und die festgelegte Gestaltung.

Nicht verändern: [TABU-BEREICHE]. Prüfe das Ergebnis auf Desktop und Smartphone.

V11 Gestaltung einfrieren

Die Gestaltung ist abgenommen und wird nicht neu interpretiert. Farben, Schriften, Bausteine, Abstände und Gewichtung bleiben exakt erhalten.

Aufgabe: [AUFGABE]. Ändere nur den Bereich, den diese Aufgabe verlangt.

V12 Funktion erweitern

Die bestehende Funktion [FUNKTION] soll um [ERWEITERUNG] wachsen.

Sieh dir zuerst an, wie sie heute gebaut ist, und nutze dieselben Muster.

Bisheriges Verhalten und bestehende Schnittstellen sollen kompatibel bleiben. Leg mir vor dem Umsetzen einen kurzen Plan vor.

V13 Folgen abschätzen

Wir planen diese Änderung: [ÄNDERUNG]. Noch nichts tun.

Welche bestehenden Funktionen, Daten, Schnittstellen oder Tests könnten dadurch beeinträchtigt werden? Ordne die Risiken nach Wahrscheinlichkeit und Tragweite und schlag vor, wie wir sie vorher absichern.

FEHLER BEHEBEN, DATEN UND ANBINDUNGEN ÄNDERN

V14 Fehleranalyse

Soll: [WAS PASSIEREN SOLLTE]

Ist: [WAS PASSIERT]

Weg dorthin: [SCHRITTE]

Spuren: [FEHLERMELDUNG, PROTOKOLL]

Umstände: [LETZTE ÄNDERUNG]

Stell den Fehler nach, verfolge den betroffenen Codepfad und benenne die wahrscheinlichste Ursache. Unterscheide Ursache und Symptom. Plane die kleinste Korrektur, ändere nichts außerhalb und prüfe danach. Ist die Ursache nicht belegt, ändere noch keinen Code.

V15 Fehlermeldung übersetzen

Was bedeutet diese Meldung: [FEHLERMELDUNG]?

Erklär sie verständlich, nenne typische Auslöser, den in unserem Projekt wahrscheinlichsten und wie wir ihn nachweisen können. Noch nichts ändern.

V16 Umbau abwägen

Wir überlegen, [BEREICH] umzubauen. Noch nichts verändern.

Welches konkrete Problem würde das lösen? Was bringt es, was riskiert es, welche Teile wären betroffen, wie aufwendig wären die Tests, welche Alternativen gibt es, und ist Abwarten im Moment vertretbar? Empfiehl den Umbau nur, wenn der Nutzen das Risiko klar übersteigt.

V17 Datenmodell ändern

Fachlich brauchen wir diese Datenänderung: [ÄNDERUNG].

Sieh dir zuerst die bestehende Datenstruktur an und erstelle einen Plan, der vorhandene Daten sicher übernimmt. Berücksichtige bestehende Einträge, Pflichtfelder, Verknüpfungen, das Nachfüllen alter Datensätze, Rückwärtskompatibilität, den Weg zurück sowie Folgen für Schnittstellen und Oberfläche. Nichts löschen oder überschreiben ohne meine Freigabe.

V18 Schnittstelle anbinden

Wir wollen [SYSTEM] anbinden, um [ZWECK]. Noch nichts umsetzen.

Werte zuerst die aktuelle offizielle Dokumentation und unseren Aufbau aus. Beschreibe Anmeldung, benötigte Endpunkte und Daten, Mengengrenzen, Fehlerfälle, Benachrichtigungen, nötige Umgebungsvariablen sowie Datenschutz und Sicherheit. Erstelle daraus einen Umsetzungsplan. Verwende nur Endpunkte und Parameter, die in der Dokumentation stehen.

PRÜFEN UND VERSTEHEN

V19 Abnahme gegen Anforderungen

Gleiche die eben umgesetzte Funktion mit den ursprünglichen Anforderungen und Abnahmekriterien ab. Führe, soweit möglich, Tests, Typprüfung, Linter, Build und die relevanten Fehlerfälle aus. Liefere vier Listen: bestätigt, nicht bestätigt, muss noch von Hand getestet werden, bekannte Risiken. Gib nichts als geprüft aus, was du nicht wirklich geprüft hast.

V20 Änderung erklären

Erklär mir diese Änderung, als wäre ich kein Entwickler. Gliedere in: 1. was sich fachlich ändert, 2. welche Dateien betroffen sind, 3. welches bisherige Verhalten anders ist, 4. neue Bibliotheken oder Einstellungen, 5. Folgen für Daten, Anmeldung und Sicherheit, 6. was ich von Hand testen sollte, 7. was davon nicht nötig gewesen wäre. Sei ausdrücklich kritisch.

V21 Kritischer Review

Sieh dir die aktuelle Änderung an wie jemand, der das Projekt jahrelang warten muss. Such nach Fehlern, Nebenwirkungen, unnötiger Komplexität, Sicherheitslücken, fehlender Fehlerbehandlung, abweichenden Mustern, fehlenden Tests und Änderungen außerhalb des Auftrags. Ordne alles in kritisch, wichtig und optional.

V22 Sicherheitsprüfung

Prüfe [BEREICH] gezielt auf Sicherheit: Anmeldung, Berechtigungen, Prüfung von Eingaben, Zugangsdaten, Datenzugriff, personenbezogene Daten, Einschleusen von Befehlen, Datei-Uploads, Fehlermeldungen und Protokolle. Sortiere die Funde nach Risiko. Noch nichts ändern.

V23 Code verständlich erklären

Erklär mir [DATEI ODER BEREICH] so, dass ich die Entscheidung dahinter verstehe, ohne selbst programmieren zu können: Aufgabe, Eingaben, Ausgaben, Abhängigkeiten, Datenfluss, Fehlerfälle und was bei Änderungen leicht kaputtgeht. Möglichst ohne Fachjargon.

VERÖFFENTLICHEN UND ÜBERGEBEN

V24 Startklar-Check

Prüfe das Projekt auf Betriebsreife. Veröffentliche nichts. Sieh dir Build, Hosting, Laufzeitumgebung, Datenbank und Datenbankänderungen, Umgebungsvariablen, Zugangsdaten, HTTPS, Adresse, Dateiablage, Protokolle, Sicherung, Überwachung, Anmeldung, Rechte, Schnittstellen, Fehlerbehandlung und Rückweg an. Liefere: Blocker, Empfehlungen, Schritte für den Livegang, Kontrollen danach, Weg zurück.

V25 Versionsfreigabe

Wir wollen Version [NEU] veröffentlichen. Vergleiche alle Änderungen seit [VORVERSION]. Achte auf Änderungen, die Bestehendes brechen, Datenbankänderungen, neue Umgebungsvariablen und Bibliotheken, Sicherheitsthemen, Nebenwirkungen, Dokumentation und Rückweg. Erstelle daraus eine Freigabe-Checkliste.

V26 Dokumentation nachziehen

Sieh dir die Änderungen seit [STAND] an. Passe nur die Dokumentation an, die dadurch wirklich veraltet ist: README, Einrichtung, Umgebungsvariablen, Aufbau, Datenbank, Veröffentlichung. Beschreibe den aktuellen Stand, nicht den Weg dorthin.

V27 Projektübergabe

Erstelle eine kompakte technische Übergabe des aktuellen Stands: Zweck, Aufbau, Funktionsumfang, wichtige Entscheidungen und ihre Gründe, bekannte Probleme, offene Aufgaben, Einrichtung, Veröffentlichung, Datenbank, Anbindungen und heikle Bereiche. Jemand ohne Kenntnis unserer bisherigen Gespräche muss damit weiterarbeiten können.

GEKAUFTE SOFTWARE ABLÖSEN

V28 Funktionsinventar einer bestehenden Software

Wir prüfen, ob wir [SOFTWARE] durch eine eigene Lösung ersetzen. Wir sind berechtigt, sie zu untersuchen: [BERECHTIGUNG].
Erstelle ein Funktionsinventar: Masken, Abläufe, Datenobjekte und Felder, Regeln, Schnittstellen und Exportwege. Markiere, welche Funktionen wir laut [NUTZUNGSDATEN ODER BEFRAGUNG] wirklich verwenden. Übernimm keinen Code, keine Texte und keine Grafiken. Liefere daraus eine Produktbeschreibung für den Nachbau und eine Liste offener Fragen zu Daten, Betrieb und Recht.

Wegweiser

Welche Vorlage passt gerade? Such deine Situation und spring zur Nummer.

DEINE SITUATION	PASSENDE VORLAGEN
Ich habe nur eine vage Idee	V01 Ideen-Härtetest
Ich weiß, was ich will	V02 Produktbeschreibung, V04 Technikwahl
Ich will eine Funktion ergänzen	V03 Funktionsbeschreibung, V07 Umsetzungsplan
Ich starte ein neues Projekt	V05 Startplan
Ich übernehme ein Projekt	V06 Bestandsaufnahme, V08 Lagebild
Ich will etwas Kleines ändern	V09 Eng begrenzte Änderung, V10, V11
Ich will etwas Größeres ändern	V13 Folgen abschätzen, V07, V12
Etwas funktioniert nicht	V14 Fehleranalyse, V15
Daten oder Anbindungen ändern sich	V17 Datenmodell, V18 Schnittstelle
Die Änderung ist fertig	V19 Abnahme, V20 Erklären, V21 Review
Ich will veröffentlichen	V22 Sicherheit, V24 Startklar, V25 Freigabe
Ich gebe das Projekt ab	V26 Dokumentation, V27 Übergabe
Ich will eine gekaufte Software ablösen	V28 Funktionsinventar, dann V02

Kurzreferenz

Verstehen

Erst lesen lassen, dann ändern

Planen

Plan freigeben, dann bauen

Klein halten

Eine Etappe, ein Auftrag

Riegel setzen

Sagen, was tabu ist

Nachsehen

Umfang jeder Änderung prüfen

Prüfen

Sechs Prüfungen, dann sichern

Du musst nicht alles selbst können.

**Entscheide klar, was entstehen soll.
Führe die KI so, dass du jederzeit weißt,
was in deinem Werkzeug passiert.**

GUIDED VIBE CODING BY MENTER

Ihr baut selbst. Wir sind dabei.

Das Playbook steckt in der KI

Unser Tool führt Schritt für Schritt durch Beschreibung, Plan, AUFTRAG und Prüfung. Keine Vorlagen zum Abtippen.

Menschen an eurer Seite

Unsere IT-Spezialisten übernehmen, wo es tief wird: Daten, Schnittstellen, Sicherheit, Betrieb.

Freigabe vor Live

Erst Testumgebung, dann Review durch uns, dann Pilot. Nichts geht ungeprüft in eure Systeme.

MENTER DIGITAL OS

Euer Unternehmen auf einen Blick.

SEHEN

Prozesse, Tools, IT-Landschaft



VERSTEHEN

Firmenwissen und KI-Analyse



ENTSCHEIDEN

Ablauf, kaufen, automatisieren oder bauen?



UMSETZEN

Automatisierung und Guided Vibe Coding

Wir ersetzen kein ERP, CRM oder die Buchhaltung, wir verbinden sie. Und gebaut wird nur, wenn Bauen die beste Antwort ist.

KONTAKT

Mario Janzen

CEO, menter Digital GmbH

TELEFON

+49 6897 9660137

WEB

www.mentor.digital

E-MAIL

mario.janzen@mentor.digital

ADRESSE

Freiheitstraße 2, 66125 Saarbrücken

ERSTGESPRÄCH

kostenlos, Pilotphase ab Januar 2027