Erfolgreiches Vibe Coding in der Anwendung

Was ist Vibe Coding: Programmieren im Dialog mit künstlicher Intelligenz

Softwareentwicklung galt lange als hochspezialisierte Tätigkeit, die tiefgehende Kenntnisse von Programmiersprachen, Systemarchitekturen und technischen Standards voraussetzt. Entwickler übersetzten Anforderungen in komplexe Code-Strukturen, testeten diese manuell und optimierten sie Schritt für Schritt. Dieser klassische Ansatz prägt bis heute große Teile der IT-Branche. Doch mit dem Aufkommen leistungsfähiger KI-Systeme entsteht derzeit ein neues Entwicklungsmodell: das sogenannte Vibe Coding.

Der Begriff beschreibt eine Form der Softwareentwicklung, bei der Programmcode nicht mehr ausschließlich manuell geschrieben wird, sondern im Dialog zwischen Mensch und künstlicher Intelligenz entsteht. Der Anwender formuliert dabei weniger technische Lösungen, sondern beschreibt zunächst Ziele, Anforderungen und gewünschte Funktionen in natürlicher Sprache. Die KI interpretiert diese Vorgaben und generiert daraus Programmcode, Datenmodelle oder komplette Analyseprozesse.

Vibe Coding verändert damit die Rolle des Entwicklers grundlegend. Statt jede Codezeile selbst zu erstellen, übernimmt der Mensch zunehmend die Funktion eines Architekten und Konzeptgestalters. Er definiert Problemstellungen, Bewertungslogiken oder Geschäftsprozesse, während die KI die technische Umsetzung unterstützt oder teilweise automatisiert.

Der Begriff „Vibe“ verweist dabei auf die Idee, dass Software nicht mehr ausschließlich aus formalen Anweisungen entsteht, sondern aus einer Mischung aus Zielvorstellungen, Intuition und fachlicher Logik. Der Entwickler beschreibt die „Richtung“ oder „Stimmung“ der Anwendung, während die KI daraus konkrete technische Lösungen ableitet. Dieser dialogbasierte Entwicklungsprozess ähnelt weniger klassischer Programmierung als vielmehr einer Zusammenarbeit zwischen Fachanwender und technischem Assistenten.

Besonders sichtbar wird dieser Wandel in datengetriebenen Anwendungen. In vielen Analyse- oder Automatisierungsprojekten liegt der entscheidende Mehrwert nicht im Programmcode selbst, sondern in der fachlichen Bewertungslogik. Anwender definieren beispielsweise Kennzahlensysteme, Risikomodelle oder Entscheidungsstrukturen, während KI-Systeme diese Vorgaben in Softwareprozesse übersetzen. Dadurch verschiebt sich der Schwerpunkt der Softwareentwicklung von technischer Implementierung hin zu konzeptioneller Modellierung.

Vibe Coding steht damit exemplarisch für eine breitere Transformation der digitalen Arbeitswelt. Softwareentwicklung wird zunehmend zugänglicher für Fachanwender, während klassische Entwickler stärker strategische und qualitätssichernde Aufgaben übernehmen. Gleichzeitig entstehen neue Anforderungen an Kontrolle, Validierung und Sicherheitsprüfung, da KI-generierter Code überzeugend wirken kann, ohne zwangsläufig fehlerfrei zu sein.

Trotz dieser Herausforderungen zeigt sich bereits heute, dass Vibe Coding Entwicklungsprozesse erheblich beschleunigen und neue Formen der Zusammenarbeit ermöglichen kann. Software entsteht nicht mehr ausschließlich durch Programmierung im traditionellen Sinn, sondern durch einen kontinuierlichen Dialog zwischen menschlicher Expertise und maschineller Umsetzungskompetenz.

Wie strukturierte Prompts aus Ideen funktionierende Software machen

Der entscheidende Erfolgsfaktor beim Vibe Coding ist nicht die KI selbst, sondern die Qualität der Kommunikation zwischen Mensch und Maschine. Während klassische Programmierung auf formalen Programmiersprachen basiert, entsteht Software beim Vibe Coding aus einem dialogbasierten Entwicklungsprozess. Der Nutzer beschreibt Ziele, Anforderungen und Rahmenbedingungen, während die KI daraus Code und Softwarearchitektur generiert.

In der Praxis zeigt sich jedoch schnell, dass unscharfe oder unvollständige Anweisungen häufig zu unzureichenden Ergebnissen führen. Wer Vibe Coding erfolgreich nutzen möchte, sollte deshalb lernen, Anforderungen strukturiert und nachvollziehbar zu formulieren. Eine klare Promptstruktur wirkt dabei wie ein Bauplan für die spätere Software.

Der erste Schritt: Das Problem verständlich formulieren

Viele Anwender beginnen Vibe Coding mit einer rein technischen Fragestellung. In der Praxis führt dieser Ansatz oft zu fragmentiertem Code ohne klare Zielrichtung. Erfolgreicher ist es, zunächst das eigentliche Problem oder den gewünschten Nutzen zu beschreiben.

Dabei sollte man sich bewusst fragen, welche Aufgabe die Software lösen soll und welchen Mehrwert sie für den Nutzer erzeugen soll. Eine solche Zielbeschreibung zwingt dazu, fachliche Anforderungen von technischen Details zu trennen. Gleichzeitig erhält die KI ein klares Verständnis des Anwendungskontextes.

Ein typisches Beispiel aus datengetriebenen Analyseprojekten wäre die Beschreibung eines Tools, das Unternehmensdaten aus verschiedenen Quellen zusammenführt, Kennzahlen berechnet und daraus strukturierte Entscheidungsgrundlagen ableitet. Entscheidend ist dabei nicht die Wahl der Programmiersprache, sondern die Klarheit über den Zweck der Anwendung.

Der zweite Schritt: Den Nutzungskontext definieren

Nachdem das Ziel beschrieben wurde, sollte der Einsatzbereich möglichst konkret dargestellt werden. Dieser Kontext hilft der KI, Prioritäten zu setzen und passende Lösungsansätze zu entwickeln.

Besonders hilfreich ist es, die Zielgruppe der Anwendung, typische Nutzungsszenarien und mögliche Datenquellen zu beschreiben. Software für wissenschaftliche Analysen stellt beispielsweise andere Anforderungen als ein Tool für betriebswirtschaftliche Entscheidungsprozesse oder automatisierte Berichterstellung.

Gerade bei komplexeren Analyseprojekten zeigt sich, dass KI-Systeme deutlich bessere Ergebnisse liefern, wenn sie den fachlichen Hintergrund verstehen. Anwendungen, die Daten analysieren oder bewerten sollen, profitieren besonders von einer klar formulierten fachlichen Logik.

Der dritte Schritt: Funktionen und Arbeitsschritte festlegen

Erst nach Zielbeschreibung und Kontextdefinition sollten konkrete Funktionsanforderungen formuliert werden. Diese sollten möglichst klar und in logisch nachvollziehbarer Reihenfolge beschrieben werden.

Hilfreich ist es, die geplanten Funktionen entlang eines Arbeitsprozesses zu strukturieren. Eine typische Struktur könnte beispielsweise vorsehen, dass Daten eingelesen, geprüft, verarbeitet, ausgewertet und schließlich visualisiert werden. Durch diese prozessorientierte Beschreibung kann die KI leichter modulare Softwarearchitekturen entwickeln.

In datengetriebenen Analyseprojekten lassen sich häufig auch Bewertungs- oder Scoring-Modelle integrieren. Hier definiert der Anwender beispielsweise, welche Kennzahlen relevant sind, wie Risiken bewertet werden und welche Kriterien zur Entscheidungsfindung herangezogen werden sollen. Die KI übernimmt anschließend die technische Umsetzung dieser Logik.

Der vierte Schritt: Technische Rahmenbedingungen vorgeben

Obwohl Vibe Coding stark auf natürliche Sprache setzt, bleiben technische Vorgaben weiterhin wichtig. Dazu gehören Programmiersprachen, Bibliotheken, Entwicklungsumgebungen oder Integrationsanforderungen.

Besonders bei Projekten, die später erweitert oder in bestehende Systeme integriert werden sollen, empfiehlt es sich, modulare Strukturen und dokumentierten Code einzufordern. Dies erleichtert Wartung, Erweiterung und Zusammenarbeit mit anderen Entwicklern.

Der fünfte Schritt: Qualitäts- und Prüfanforderungen definieren

Ein häufiger Fehler beim Vibe Coding besteht darin, sich ausschließlich auf funktionierende Ergebnisse zu konzentrieren. Nachhaltige Softwareentwicklung erfordert jedoch zusätzliche Qualitätskontrollen.

Dazu gehören Plausibilitätsprüfungen von Eingabedaten, Fehlermeldungen bei inkonsistenten Datensätzen und automatisierte Testfälle zur Überprüfung von Berechnungslogiken. Besonders in Analyse- und Bewertungsanwendungen kann dieser Schritt entscheidend sein, um Fehlinterpretationen oder falsche Entscheidungsgrundlagen zu vermeiden.

Der sechste Schritt: Den Entwicklungsdialog bewusst offen halten

Der Entwicklungsdialog beginnt häufig mit einem ersten funktionierenden Prototypen. Dieser erfüllt zwar grundlegende Anforderungen, enthält aber oft noch ineffiziente Algorithmen, unvollständige Datenprüfungen oder unklare Architekturentscheidungen. Statt den Code sofort produktiv einzusetzen, empfiehlt es sich, die KI aktiv in einen Verbesserungsprozess einzubeziehen.

Dabei können Anwender gezielt Rückfragen stellen oder Optimierungen anfordern. Typische Folgeanweisungen sind beispielsweise die Verbesserung der Performance, die Erweiterung von Fehlermeldungen oder die Anpassung der Software an neue Datentypen. Ebenso sinnvoll ist es, die KI um alternative Lösungsansätze zu bitten. Dadurch entstehen häufig robustere und besser skalierbare Softwarestrukturen.

Wichtig ist es, die KI regelmäßig um Erläuterungen des generierten Codes zu bitten. Dieser Schritt dient nicht nur der Qualitätssicherung, sondern auch dem Aufbau eines besseren Verständnisses für die zugrunde liegende Logik. Anwender können dadurch fundierter entscheiden, welche Teile des Codes übernommen, angepasst oder verworfen werden sollten.

Ein weiterer hilfreicher Bestandteil des Entwicklungsdialogs ist die Integration realer Testdaten. Erst durch den Einsatz praxisnaher Datensätze lassen sich viele Schwachstellen erkennen, die in theoretischen Tests verborgen bleiben. KI-Systeme können anschließend gezielt auf diese Probleme reagieren und den Code entsprechend optimieren.

In komplexeren Analyseprojekten zeigt sich häufig, dass neue fachliche Anforderungen erst während der Nutzung entstehen. Bewertungsmodelle müssen angepasst, Kennzahlen erweitert oder Entscheidungsregeln verfeinert werden. Vibe Coding ermöglicht es, solche Anpassungen ohne vollständige Neuentwicklung umzusetzen. Stattdessen wird der bestehende Code schrittweise weiterentwickelt.

Langfristig entsteht auf diese Weise ein lernender Entwicklungsprozess. Prompts, Codeversionen und Qualitätsprüfungen entwickeln sich parallel weiter. Software wird dadurch nicht mehr als statisches Produkt verstanden, sondern als dynamisches System, das sich kontinuierlich an neue Anforderungen anpassen lässt.

Beispiel für einen strukturierten Vibe-Coding-Prompt

Eine bewährte Formulierung kann folgendermaßen aussehen:

„Ich möchte ein Analyse-Notebook entwickeln, das strukturierte Unternehmensdaten aus Excel-Dateien einliest und daraus automatisch betriebswirtschaftliche Kennzahlen berechnet. Ziel ist die Erstellung einer nachvollziehbaren Entscheidungsgrundlage für Analyse- und Bewertungszwecke. Das Programm soll Eingabedaten prüfen, Kennzahlen berechnen, Ergebnisse visualisieren und optional ein Bewertungssystem integrieren. Bitte nutze Python mit Pandas[1] und Visualisierungsbibliotheken. Der Code soll modular aufgebaut, kommentiert

[1] *Pandas* ist eine weit verbreitete Open-Source-Programmbibliothek für die Programmiersprache Python. Sie wird vor allem zur Verarbeitung, Analyse und Strukturierung von Daten verwendet. Mit Pandas lassen sich beispielsweise Excel-Tabellen einlesen, Datensätze filtern, Kennzahlen berechnen oder Zeitreihen analysieren.

und testbar sein. Erkläre mir den Aufbau des Codes verständlich und schlage Verbesserungen vor.“[2]

Warum strukturierte Prompts besonders bei komplexen Analyseprojekten wichtig sind

In vielen datengetriebenen Anwendungen liegt der eigentliche Mehrwert nicht im Programmcode selbst, sondern in der zugrunde liegenden Bewertungslogik. Moderne Analyseprojekte zeigen, dass Softwareentwicklung zunehmend aus fachlicher Modellierung entsteht. Der Anwender definiert Entscheidungsstrukturen, während die KI diese in technische Abläufe übersetzt.

Dieser Ansatz ermöglicht es, komplexe Analyse-Workflows modular aufzubauen, Daten automatisiert auszuwerten und Entscheidungsprozesse transparenter zu gestalten. Solche Anwendungen verdeutlichen exemplarisch, wie Vibe Coding Fachwissen und Softwareentwicklung miteinander verbindet.

Typische Fehler beim Einsatz von Vibe Coding

Trotz der großen Potenziale entstehen in der Praxis häufig Probleme durch unklare Anforderungen oder fehlende Qualitätskontrollen. Besonders riskant ist es, KI-generierten Code ungeprüft zu übernehmen. Viele Fehler entstehen nicht durch falsche Programmierung, sondern durch missverständliche Zieldefinitionen.

Auch überladene Prompts können problematisch sein. Komplexe Projekte lassen sich meist besser in mehreren Entwicklungsschritten umsetzen. Eine modulare Vorgehensweise erleichtert dabei die Fehlersuche und verbessert die Wartbarkeit der Software.

Empfehlung für den praktischen Einstieg

Wer Vibe Coding nutzen möchte, sollte mit überschaubaren Projekten beginnen und die Struktur seiner Prompts schrittweise verfeinern. Mit zunehmender Erfahrung entwickelt sich meist ein eigener Kommunikationsstil mit KI-Systemen. Dieser Lernprozess ist ein zentraler Bestandteil moderner Softwareentwicklung.

Gleichzeitig zeigt die Praxis, dass erfolgreiche Vibe-Coding-Projekte häufig dort entstehen, wo fachliche Expertise und technische Umsetzung eng miteinander verzahnt werden. Anwendungen, die Daten analysieren, bewerten oder Entscheidungsprozesse unterstützen, profitieren besonders von dieser Methode.

---

[2] Dies beschreibt in Ansätzen mein Due Diligence – Projekt, das sich dieser Codierung und des beschriebenen Prozesses bedient.

Praxisbox: So lässt man sich KI-Code verständlich erklären

Eine der größten Stärken moderner KI-Systeme liegt darin, dass sie nicht nur Code erzeugen, sondern auch komplexe Zusammenhänge verständlich erklären können. Gerade Anwender ohne tiefgehende Programmiererfahrung profitieren davon, wenn sie KI gezielt in einen sogenannten Instruktormodus versetzen.

Der Instruktormodus bedeutet, dass die KI nicht nur Ergebnisse liefert, sondern Schritt für Schritt erläutert, wie Code funktioniert, warum bestimmte Lösungswege gewählt wurden und welche Alternativen möglich wären. Dadurch wird KI zu einem Lern- und Entwicklungspartner und nicht nur zu einem automatischen Codegenerator.

In der Praxis ist dieser Ansatz besonders hilfreich, wenn man neue Analyse- oder Automatisierungsprojekte entwickelt. Anwender können auf diese Weise die Funktionsweise ihrer Software nachvollziehen und langfristig besser anpassen oder erweitern.

Beispiel-Prompt im Instruktormodus

„Erkläre mir bitte den erzeugten Code so, als hätte ich nur grundlegende Programmierkenntnisse. Beschreibe Schritt für Schritt:

- Welche Funktion jeder Codeabschnitt erfüllt
- Warum diese Lösung gewählt wurde
- Welche Alternativen möglich wären
- Welche Risiken oder Schwachstellen existieren
- Wie ich den Code später selbst anpassen kann
  Nutze einfache Sprache und verzichte auf unnötige Fachbegriffe.“

Wann der Instruktormodus besonders sinnvoll ist

Der Instruktormodus eignet sich besonders bei:

- Entwicklung neuer Analysemodelle
- Automatisierung betrieblicher Prozesse
- Aufbau modularer Softwarelösungen
- Übernahme fremder oder KI-generierter Codes
- Schulungs- und Lernprojekten

Gerade in datengetriebenen Anwendungen zeigt sich, dass ein tieferes Verständnis der Berechnungslogik oft wichtiger ist als die reine technische Umsetzung.

Didaktischer Zusatznutzen

Viele Anwender berichten, dass sie durch den Instruktormodus nicht nur Software schneller entwickeln, sondern gleichzeitig ihre Programmierkenntnisse erweitern. KI wird dadurch zu einem interaktiven Lehrsystem, das komplexe Zusammenhänge anschaulich erklärt.

Praxisbox: Qualitätscheck für KI-generierten Code

Prüfliste für sichere und professionelle Anwendungen

KI kann in kürzester Zeit funktionsfähigen Code erzeugen. Dennoch ersetzt dies keine systematische Qualitätsprüfung. Gerade in Anwendungen mit finanziellen, strategischen oder sicherheitsrelevanten Auswirkungen ist eine strukturierte Kontrolle unverzichtbar.

Der folgende Qualitätscheck hat sich in der Praxis als besonders hilfreich erwiesen.

---

1. Plausibilitätsprüfung der Ergebnisse

Zunächst sollte überprüft werden, ob die Ergebnisse logisch und fachlich nachvollziehbar sind. Stimmen Kennzahlen mit bekannten Referenzwerten überein? Reagiert das Programm korrekt auf Extremwerte oder unvollständige Daten?

Gerade bei Bewertungs- oder Analysemodellen können kleine Fehler in der Berechnungslogik erhebliche Auswirkungen auf Entscheidungen haben.

---

2. Überprüfung der Datenverarbeitung

Ein häufiger Fehler entsteht durch unzureichende Datenvalidierung. Software sollte erkennen können, wenn Eingabedaten fehlen, fehlerhaft formatiert sind oder nicht den erwarteten Strukturen entsprechen.

Empfehlenswert sind automatische Prüfmechanismen, die fehlerhafte Datensätze markieren oder alternative Berechnungswege vorschlagen.

---

3. Nachvollziehbarkeit der Berechnungslogik

Code sollte nicht nur funktionieren, sondern auch verständlich dokumentiert sein. Jeder zentrale Berechnungsschritt sollte kommentiert werden. Anwender müssen nachvollziehen können, wie Ergebnisse entstehen.

Diese Transparenz ist besonders wichtig in Analyseprojekten, bei denen Software als Entscheidungsgrundlage dient.

---

4. Sicherheitsprüfung

KI-generierter Code kann ungewollte Sicherheitslücken enthalten. Dazu gehören unsichere Datenzugriffe, ungeschützte Schnittstellen oder fehlende Zugriffskontrollen. Vor allem Anwendungen mit sensiblen Daten sollten deshalb einer zusätzlichen Sicherheitsprüfung unterzogen werden.

---

5. Performance und Skalierbarkeit

Code, der mit kleinen Datensätzen funktioniert, kann bei größeren Datenmengen schnell an Leistungsgrenzen stoßen. Eine Performanceanalyse hilft dabei, Engpässe frühzeitig zu erkennen und zu beheben.

---

6. Modularität und Wartbarkeit

Nachhaltige Software sollte modular aufgebaut sein. Funktionen sollten klar getrennt und erweiterbar bleiben. Dies erleichtert spätere Anpassungen und reduziert Wartungskosten.

---

7. Dokumentation und Versionskontrolle

Gerade bei KI-gestützten Projekten ist es wichtig, Änderungen nachvollziehbar zu dokumentieren. Versionskontrollsysteme und strukturierte Entwicklungsprotokolle helfen, Fehlerquellen zu identifizieren und Entwicklungsfortschritte zu sichern.

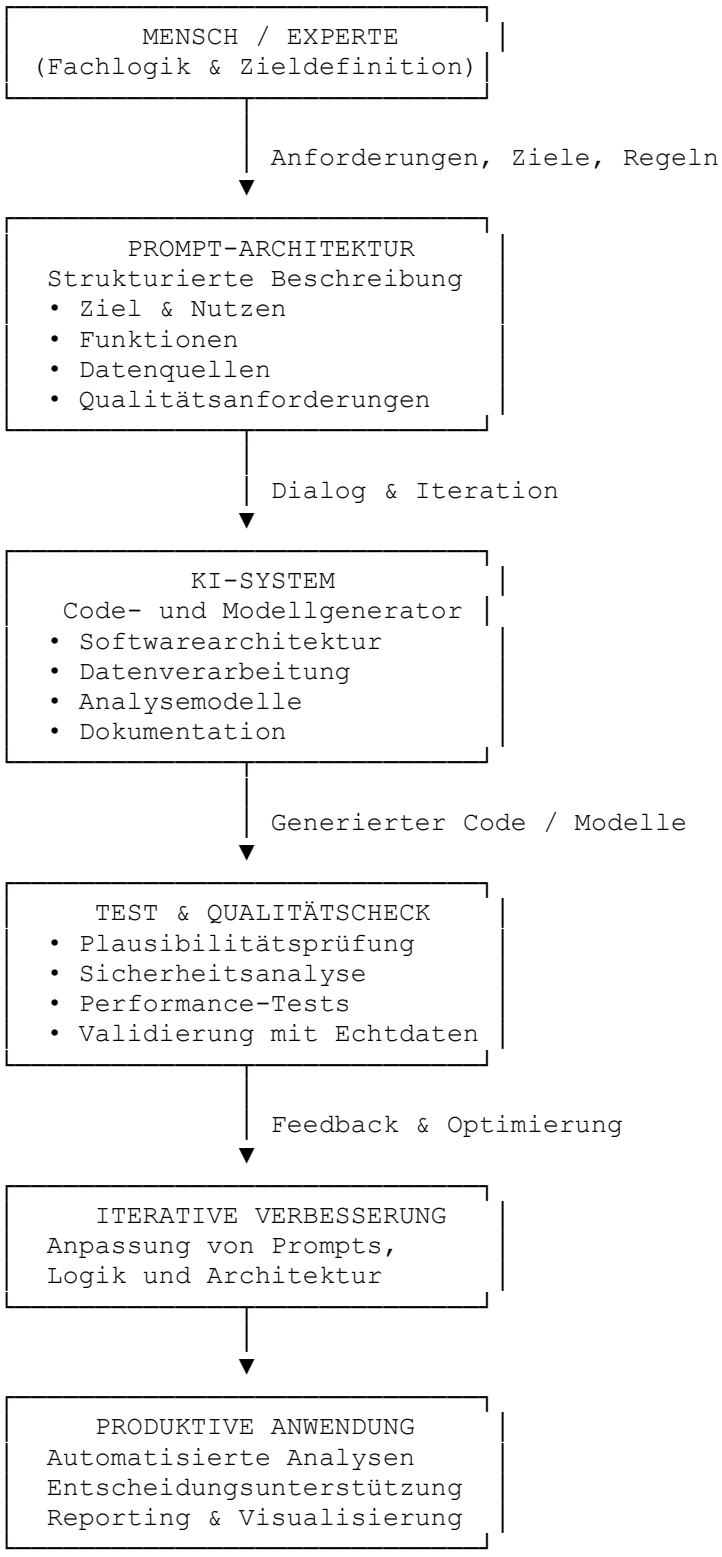
Warum Qualitätsprüfungen beim Vibe Coding besonders wichtig sind

Vibe Coding beschleunigt Entwicklungsprozesse erheblich. Gleichzeitig steigt jedoch das Risiko, dass überzeugend wirkender Code ungeprüft übernommen wird. KI kann sehr plausible Lösungen erzeugen, die dennoch fachlich oder technisch fehlerhaft sind.

Professionelle Anwendungen entstehen deshalb immer aus der Kombination von KI-Unterstützung und menschlicher Qualitätskontrolle.

Visuelle Übersicht

Vibe Coding Workflow – Zusammenarbeit zwischen Mensch, KI und Qualitätssicherung



So lesen Sie den Workflow

Der dargestellte Prozess zeigt, dass Vibe Coding kein linearer Entwicklungsweg ist, sondern ein Kreislauf aus Planung, Umsetzung und Qualitätskontrolle.

1. Der Mensch bleibt der strategische Mittelpunkt

Der Entwicklungsprozess beginnt immer mit fachlicher Zieldefinition. Der Anwender entscheidet, welches Problem gelöst werden soll und welche Bewertungslogiken oder Entscheidungsregeln relevant sind.

Dieser Schritt ist entscheidend, weil KI zwar technische Lösungen generieren kann, aber keine eigenständigen fachlichen Zielsetzungen entwickelt.

2. Die Prompt-Architektur übersetzt Fachwissen in Softwarelogik

Prompts fungieren als Schnittstelle zwischen menschlicher Problemdefinition und technischer Umsetzung. Je strukturierter Anforderungen formuliert werden, desto stabiler und nachvollziehbarer wird die Softwarearchitektur.

In komplexeren Analyseprojekten entsteht hier oft bereits die eigentliche Modellstruktur der Anwendung.

3. KI fungiert als Entwicklungs- und Umsetzungspartner

Die KI übernimmt die technische Umsetzung. Dazu gehören Programmierung, Datenverarbeitung, Visualisierung und teilweise sogar Dokumentation.

Dabei entstehen häufig mehrere Lösungsvorschläge, die anschließend bewertet und optimiert werden können.

4. Qualitätssicherung verhindert Fehlentscheidungen

Der Qualitätscheck bildet eine zentrale Kontrollinstanz. Gerade bei datengetriebenen Anwendungen ist es entscheidend, dass Berechnungsmodelle fachlich plausibel und technisch stabil sind.

In professionellen Anwendungen erfolgt diese Prüfung meist anhand realer Datensätze und Vergleichsmodelle.

---

5. Iteration ist der Kern von Vibe Coding

Vibe Coding lebt von kontinuierlicher Verbesserung. Neue Anforderungen, erkannte Fehler oder optimierte Bewertungslogiken fließen direkt in neue Promptversionen ein. Dadurch entwickelt sich Software Schritt für Schritt weiter.

---

6. Ergebnis: Automatisierte und skalierbare Anwendungen

Am Ende entstehen Anwendungen, die komplexe Analysen automatisieren, Entscheidungsprozesse unterstützen oder Geschäftsprozesse effizienter gestalten können.

Quellen:

Grundlagen:

https://www.geeksforgeeks.org/techtips/what-is-vibe-coding/

1. arXiv – LLMs als Programmierassistenten
   https://arxiv.org/abs/2302.06590
   Empirische Studie zu Produktivitätsgewinnen durch KI-gestütztes Programmieren.
2. OpenAI – Agenten, Tool Use & Function Calling
   https://platform.openai.com/docs/guides/agents
   Technische Grundlage für iterative Workflows.
3. Anthropic – Human-AI Collaboration & Claude
   https://www.anthropic.com/research
   Forschung zu Mensch-KI-Zusammenarbeit.
4. Harvard Business Review – KI verändert Wissensarbeit (ersetzt)
   https://hbr.org/2023/11/how-generative-ai-will-transform-knowledge-work
   Analyse des Wandels in Wissensarbeit durch GenAI.
5. McKinsey & Company – Generative AI in Software Engineering
   https://www.mckinsey.com/capabilities/technology-and-digital/our-insights/unleashing-developer-productivity-with-generative-ai
   Produktivitätsanalysen für Entwickler.