KLAUS TESCHING

VIBE CODING & AGENTIC ENGINEERING

Der neue Workflow zwischen Mensch und Maschine

Klaus Tesching

2. Auflage, 2026 · ki-4-everyone.de

Impressum

Vibe Coding & Agentic Engineering Der neue Workflow zwischen Mensch und Maschine

 2. Auflage, erweiterte Fassung, Juni 2026

Autor: Klaus Tesching Kontakt: info@ki-4-everyone.de Website: https://ki-4-everyone.de

Haftungsausschluss: Die in diesem Buch enthaltenen Informationen, Prompts und Code-Beispiele wurden mit größter Sorgfalt erstellt. Der Autor übernimmt jedoch keine Gewähr für die Richtigkeit, Vollständigkeit und Aktualität der Inhalte. Die Nutzung der Inhalte erfolgt auf eigene Verantwortung. KI-generierter Code sollte vor dem produktiven Einsatz stets geprüft werden.

Markenhinweis: Claude, Claude Code und Anthropic sind Marken der Anthropic PBC. Alle weiteren genannten Marken und Produktnamen sind Eigentum ihrer jeweiligen Inhaber. Die Verwendung in diesem Buch dient ausschließlich der Beschreibung und Information und stellt keine Markenverletzung dar.

Hinweis zu KI-generierten Inhalten: Text, Illustrationen und Gestaltung dieses Werks wurden mit Unterstützung künstlicher Intelligenz erstellt. Diese Kennzeichnung erfolgt im Sinne der Transparenzpflichten für KI-generierte Inhalte (vgl. EU AI Act, Art. 50).

Bildnachweis: Titelblatt: KI-generiert (Higgsfield), Gestaltung Klaus Tesching Kapitel-Illustrationen: KI-generiert mit Recraft bzw. Soul (über Higgsfield) Technische Diagramme (Abb. 1.1–14.2): SVG, erstellt mit Claude Code Autorenfoto: KI-generiert Satz und Gestaltung: erstellt mit Claude Code (Anthropic)

Druck und Verlag: [wird ergänzt] ISBN: [wird ergänzt]

Was ist neu in dieser Auflage

Die erste Auflage dieses Buchs erschien im April 2026 unter dem Titel „VIBE Coding". Diese erweiterte Fassung der zweiten Auflage geht deutlich darüber hinaus. Wenn Sie die erste Auflage kennen, finden Sie hier den Überblick über alles, was dazugekommen ist.

Zwei neue Kapitel. Kapitel 12 „Agentic Engineering" beschreibt die Disziplin hinter dem professionellen Einsatz von Coding-Agenten: die Begriffsgeschichte von Karpathys erstem Tweet bis zur heutigen Praxis, die Trennlinie zum Vibe Coding, sieben konkrete Arbeitspraktiken und die ehrliche Studienlage. Kapitel 13 „Hermes Agent" macht diese Disziplin an einem real existierenden System greifbar: eine Tiefen-Fallstudie über einen quelloffenen, selbstverbessernden Agenten, mit zwei vollständig reproduzierbaren Praxisbeispielen vom ersten Installationsbefehl bis zur Skill, die der Agent selbst schreibt.

Die Werkzeug-Landschaft bis Juni 2026. Der neue Abschnitt 2.12 fasst zusammen, was seit der ersten Auflage passiert ist: die Agenten-Modellgenerationen von GPT-5 bis Claude Opus 4.8, Claude Code als Plattform (Skills, Plugins, Web, Agent SDK), Cursor 2.0, Spec-Driven Development, der AGENTS.md-Standard und MCP unter dem Dach der Linux Foundation.

Alle Diagramme zurück, und mehr. Sämtliche technischen Diagramme der ersten Auflage wurden für diese Fassung neu gezeichnet, ergänzt um neue Schaubilder zu Agentic Engineering und Hermes. Jedem Kapitel ist außerdem eine Illustration vorangestellt.

Mehr Nachschlagewerk. Das Glossar wuchs von 81 auf über 110 Begriffe. Alle Quellenangaben sind in der digitalen Ausgabe anklickbar, jedes Kapitel schließt mit einem eigenen Quellenverzeichnis, und sämtliche Aussagen zu Werkzeugen und Studien sind mit Abrufdatum belegt.

Durchgehend überarbeitet. Sprachliche Glättung in der gesamten Breite, korrigierte Code-Beispiele, konsistente Sie-Form und ein neues Format: A4 statt Taschenbuchformat, gesetzt für Bildschirm und Ausdruck.

Inhaltsverzeichnis

Prolog: Der Weg hierher

Über den Autor

Klaus Tesching, Jahrgang 1955, geboren im bayerischen Ingolstadt an der Donau. Aufgewachsen in einer Zeit, als Computer noch ganze Räume füllten und das Wort „Software" im normalen Sprachgebrauch nicht existierte. Wenn mir damals jemand gesagt hätte, dass ich vierzig Jahre später ein Buch darüber schreiben würde, wie man mit einer künstlichen Intelligenz programmiert, ich hätte ihn für verrückt erklärt.

Mein Weg in die Informatik begann nicht an einer Universität, sondern in einer Werkstatt. Ich lernte Maschinenbau, arbeitete mit Metall, mit Werkzeugmaschinen, mit greifbaren Dingen. Die ersten Berührungen mit Computertechnologie kamen in den 1970er Jahren: durch NC-Technik und CNC-Steuerungen. Damals programmierten wir mit Lochstreifen und Maschinencodes. Jeder Befehl hatte ein physisches Gegenstück: ein Loch im Papierstreifen, eine Bewegung des Fräskopfes. Es gab kein Undo.

Anfang der 1980er Jahre begann ich ein berufsbegleitendes Studium, eine Kooperation zwischen IBM Deutschland und der Fachhochschule für Technik Esslingen. Die Ausbildung zum Automations-Informatiker war damals etwas Neues, ein Berufsbild an der Schnittstelle zwischen klassischem Ingenieurwesen und der aufkommenden Datenverarbeitung. Hier lernte ich Assembler (Intel 8080, 8087, später 80C186 und den Math Coprocessor 80C187) und die Großrechnersprachen der IBM System/360-Architektur. Wer einmal Maschinenbefehle von Hand in Hexadezimal kodiert hat, entwickelt ein Verhältnis zu Computern, das man nur als „respektvoll" beschreiben kann.

Dann kam C. Nicht irgendeines, sondern das C von Dennis Ritchie, dem Urvater der Sprache. Ich arbeitete damit bereits unter IBM OS/2, einem Betriebssystem, das seiner Zeit voraus war und trotzdem verlor. Später folgten UNIX und Solaris, COBOL für Geschäftsanwendungen und diverse Scriptsprachen, deren Namen heute kaum noch jemand kennt.

Anfang der 1990er Jahre beschäftigte ich mich beruflich zum ersten Mal mit neuronalen Netzen, nicht als akademische Theorie, sondern als technische Umsetzung in realen Projekten. Das war dreißig Jahre bevor der Begriff „KI" zum Alltagswort wurde. Die Konzepte waren dieselben, die heute die Grundlage moderner Large Language Models bilden. Nur die Rechenleistung fehlte. Später spezialisierte ich mich auf Lotus Notes und Domino, damals das führende Groupware-System der Welt, bevor E-Mail und Collaboration in die Cloud wanderten.

Als Projektleiter arbeitete ich über viele Jahre in Großunternehmen: bei IBM, Mercedes-Benz und der EDC, bei Alcatel-Lucent, T-Systems sowie dem Landesamt für Zentrale Polizeiliche Dienste NRW. Jedes dieser Projekte lehrte mich etwas anderes über die Art, wie Menschen und Maschinen zusammenarbeiten, und wie oft sie aneinander vorbeireden. Ab 2020 begann ich, mich intensiv mit künstlicher Intelligenz zu beschäftigen: zunächst mit den technischen Grundlagen, dann zunehmend auch mit den ethischen und philosophischen Fragen, die diese Technologie aufwirft. Es war dieses Studium, das mich schließlich zu Vibe Coding führte, und zu diesem Buch.

Warum dieses Buch

Dieses Buch kommt aus der Praxis. Nicht aus einem Elfenbeinturm, nicht aus einem Marketing-Meeting, nicht aus dem Wunsch, einen Trend zu bedienen. Es kommt aus über vierzig Jahren Erfahrung in der Informationstechnik, von Lochkarten bis Large Language Models.

Ich kenne die Fragen, die sich stellen, wenn eine neue Technologie auftaucht. Ich kenne die Begeisterung und die Skepsis, die Frustration und die Durchbrüche. Die klassischen Programmiersprachen, die mich geprägt haben, C, Assembler, COBOL, sind heute fast schon historisch. Die Welt spricht heute Python, JavaScript, TypeScript, Java, Go. Die Liste wird länger, nicht kürzer.

Vibe Coding verändert die Spielregeln. Es benötigt nicht zwingend Programmierkenntnisse. Es verlangt klares Denken, präzise Sprache und die Bereitschaft, sich auf einen neuen Workflow einzulassen. Deshalb habe ich die Prompt-Beispiele am Ende dieses Buches liebevoll „OMA-Prompts" genannt. Wenn meine Prompts so einfach sind, dass auch jemand ohne jede Programmiererfahrung damit arbeiten kann, dann habe ich meinen Job gemacht. Auch Oma kann jetzt Programme erstellen.

Eine exponentielle Zeit

Künstliche Intelligenz befindet sich in einer exponentiellen Entwicklung. Large Language Models, die vor sechs Monaten erst an Anwender ausgeliefert wurden, werden heute bereits aus dem Programm genommen, weil sie schon veraltet sind. Die Zyklen werden kürzer, die Sprünge größer, das Tempo höher.

Niemand weiß, wohin das alles führt. Nicht die Forscher in den Labors, nicht die CEOs der großen Tech-Konzerne, nicht die Politiker, die Regulierung versprechen. Es ist eine spannende Zeit für jeden, der in der IT arbeitet. Aber auch eine, die Respekt verdient. Was bis vor kurzem niemand zu glauben wagte: Wir sind möglicherweise nicht mehr so weit von einer technologischen Singularität entfernt, wie wir dachten.

Dieses Buch ist ein Snapshot, ein Foto eines sich rasend schnell bewegenden Zuges. Die Werkzeuge werden sich ändern, die Modelle werden besser, die Möglichkeiten werden wachsen. Aber die Prinzipien, die dieses Buch vermittelt, bleiben: klares Denken, präzise Kommunikation, Verantwortung für das Ergebnis. Viel Spaß beim Lesen. Und beim Bauen.

Warum Anthropic? Eine persönliche Entscheidung

Die meisten Aufgaben, Prompts und Beispiele in diesem Buch arbeiten mit Produkten von Anthropic, konkret mit Claude Opus 4.6 im Browser und Claude Code mit Opus 4.6 im Terminal. Das ist eine bewusste Entscheidung, keine Werbung.

Ich habe in den vergangenen Monaten intensiv mit verschiedenen LLMs gearbeitet und experimentiert: OpenAI, Google Gemini, DeepSeek, lokale Modelle über Ollama. Alle haben ihre Stärken, aber auch so manche Macke. Die besten und konsistentesten Ergebnisse habe ich mit Anthropics Claude erzielt. Also habe ich mir den Max-Plan gekauft und arbeite täglich damit. Da ich nicht alle KI-Unternehmen gleichzeitig finanzieren kann, ist es schlüssig, sich für eine gewisse Zeit auf ein Produkt zu konzentrieren und dieses wirklich zu beherrschen.

Die gute Nachricht: Die gezeigten Prompts und Beispiele werden zum größten Teil auch mit OpenAI Codex, Google Gemini oder anderen aktuellen LLMs funktionieren. Die Prinzipien des Vibe Codings,

klare Kommunikation, präzise Prompts, iteratives Arbeiten, sind modellunabhängig. Lediglich die spezifischen Werkzeuge wie Claude Code, CLAUDE.md oder Slash-Commands sind Anthropic-eigen. Die Konzepte dahinter finden Sie aber bei jedem ernsthaften Anbieter in ähnlicher Form.

Um das klar zu sagen: Sie sind nicht gezwungen, Anthropic zu verwenden. Ich verdiene kein Geld durch die Nennung dieser Produkte. Es gibt keine Partnerschaft, keine Provision, keinen Sponsoring-Deal. Es ist schlicht das Werkzeug, mit dem ich die besten Ergebnisse erziele, nicht mehr, nicht weniger. Sollte morgen ein anderes Modell besser sein, werde ich wechseln. So ist das in der KI-Welt.

Hinweis zur Arbeitsumgebung

Alle Prompts, Beispiele und Projekte in diesem Buch wurden auf einem Windows-11-Rechner erstellt und getestet. Das gilt für Claude Code im Terminal ebenso wie für die gezeigten Pfadangaben und Systembefehle.

Claude Code läuft grundsätzlich auch unter macOS und Linux, und die Prompts selbst sind betriebssystemunabhängig: Ein guter Prompt funktioniert überall. Allerdings unterscheiden sich Terminal-Befehle, Pfadformate und einzelne Installationsschritte zwischen den Systemen. Unter macOS heißt es beispielsweise `cd ~/prompt-test` statt `cd /d E:\Prompt-Test`, und manche Systemwerkzeuge verhalten sich anders.

Ich habe die Beispiele nicht unter macOS oder Linux getestet und kann nicht garantieren, dass jeder Befehl dort identisch funktioniert. Wenn Sie auf einem Mac oder unter Linux arbeiten, passen Sie die Terminal-Befehle und Pfade an Ihr System an. Im Zweifel hilft Claude Code selbst: Fragen Sie einfach „Wie lautet dieser Befehl auf meinem Mac?" und Sie bekommen die richtige Antwort.

Klaus Tesching
Steinenbronn in Baden-Württemberg, April 2026

1 Was ist Vibe Coding?

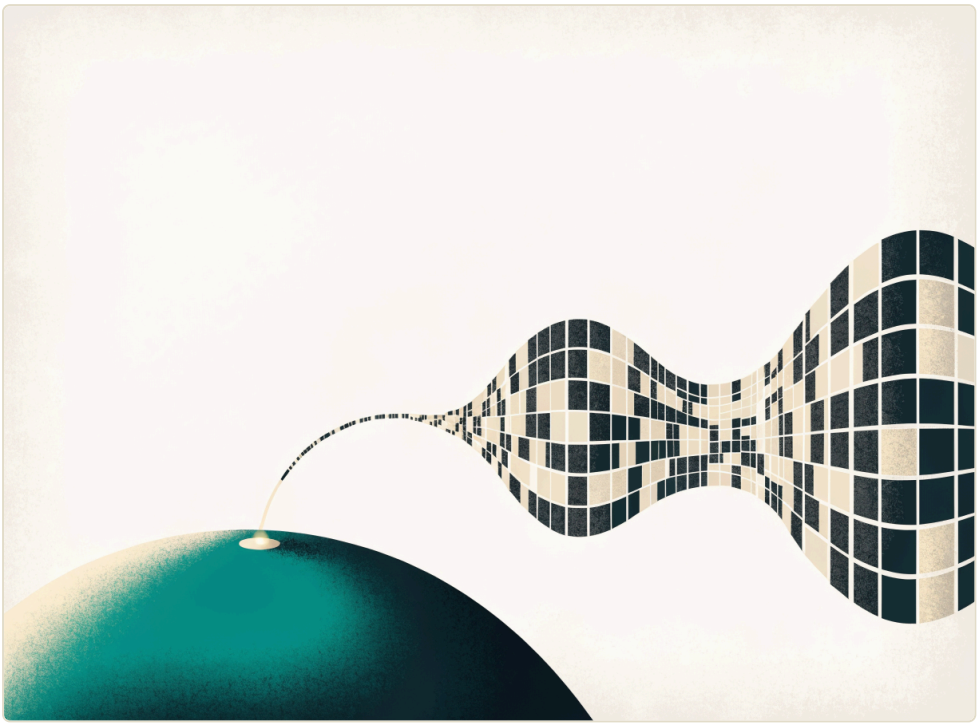

Illustration 1 Vom einzelnen Tweet zur Welle aus Code

Vom Twitter-Post zur Revolution der Softwareentwicklung

„There's a new kind of coding I call 'vibe coding', where you fully give in to the vibes, embrace exponentials, and forget that the code even exists." — Andrej Karpathy, 2. Februar 2025

1.1 Der Ursprung: Ein Tweet verändert alles

Am 2. Februar 2025 veröffentlichte Andrej Karpathy (ehemaliger Director of AI bei Tesla, Mitgründer von OpenAI und einer der einflussreichsten KI-Forscher der Welt) einen kurzen Post auf X (ehemals Twitter), der die Softwareentwicklung für immer verändern sollte.[1]

In diesem Post beschrieb Karpathy einen neuen Ansatz der Programmierung, den er „Vibe Coding" nannte. Die Kernidee: Sie beschreiben in natürlicher Sprache, was Sie bauen möchten, und überlassen der KI die eigentliche Code-Generierung. Man „gibt sich den Vibes hin", akzeptiert die Ergebnisse und vergisst, dass darunter Code existiert. Was als „Gedanken unter der Dusche" begann, so Karpathy selbst auf der 1-Jahres-Feier des Tweets, sammelte innerhalb weniger Tage über 4,5 Millionen Aufrufe und löste eine weltweite Diskussion aus. Ende 2025 wählte das Collins English Dictionary „vibe coding" zum Wort des Jahres.[2]

1.1.1 Was genau beschrieb Karpathy?

In seinem Original-Post beschrieb Karpathy seine persönliche Arbeitsweise:

- Er nutzte Cursor Composer mit Claude 3.5 Sonnet als KI-Modell
- Er sprach seine Anforderungen per Spracheingabe (SuperWhisper) ein

- Er klickte stets auf „Accept All", ohne die Code-Änderungen zu lesen
- Bei Fehlermeldungen kopierte er diese ohne Kommentar zurück an die KI
- Er bezeichnete das Ergebnis als „nicht wirklich Programmieren"

Das Besondere: Karpathy ist kein Anfänger. Er ist ein hochqualifizierter Informatiker mit Stanford-PhD und jahrelanger Erfahrung in Deep Learning. Wenn selbst jemand mit seinem Hintergrund sagt, dass er Code nicht mehr liest, ist das ein Signal, dass sich etwas Fundamentales verschoben hat.

1.2 Definition: Was Vibe Coding ist und was es nicht ist

Vibe Coding ist ein Paradigma der Softwareentwicklung, bei dem:

1. Der Mensch die Absicht beschreibt (in natürlicher Sprache)
2. Die KI den Code generiert (Implementierung)
3. Der Mensch das Ergebnis bewertet (funktioniert es?)
4. Dieser Zyklus iterativ wiederholt wird, bis das Ziel erreicht ist

Kernunterschied zur klassischen Programmierung

Klassische Programmierung: Der Mensch denkt in Code-Syntax, Datenstrukturen und Algorithmen. Vibe Coding: Der Mensch denkt in Ergebnissen, Verhalten und Erscheinungsbild. Der Code wird zum Implementierungsdetail, nicht zum Denkwerkzeug.

1.2.1 Die Abgrenzung: Drei Stufen der KI-gestützten Entwicklung

Simon Willison, einer der wichtigsten Stimmen in der KI-Entwickler-Community, hat eine hilfreiche Abgrenzung formuliert[5]:

Stufe	Beschreibung
KI-Autovervollständigung	Copilot-Stil: Die KI ergänzt einzelne Zeilen. Der Mensch schreibt den Großteil selbst.
KI-gestützte Programmierung	Chat-basiert: Die KI liefert Code-Blöcke. Der Mensch versteht und prüft den Code bewusst.
Vibe Coding	Vollständige Delegation: Der Mensch beschreibt nur das Ziel. Die KI generiert autonom. Der Mensch prüft das Ergebnis, nicht den Code.

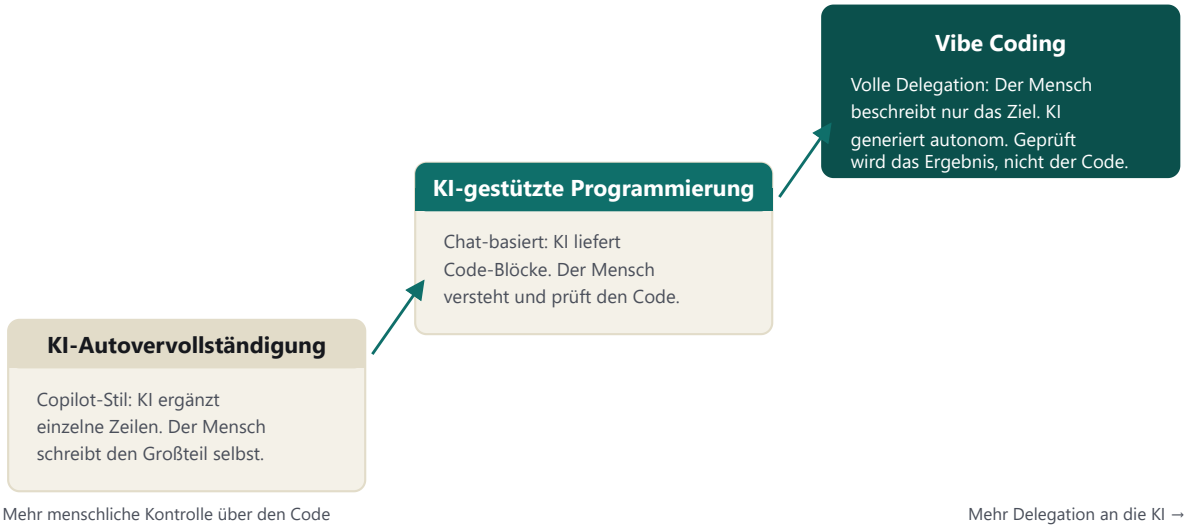

Abb. 1.2 Drei Stufen der KI-gestützten Entwicklung. Vibe Coding ist der Sprung zur vollständigen Delegation.

Vibe Coding ist kein Ersatz für alle anderen Formen. Es ist eine zusätzliche Methode, besonders für Prototypen, persönliche Projekte, kreative Experimente und schnelle MVPs geeignet. Für sicherheitskritische Systeme gelten andere Maßstäbe, dazu mehr in Kapitel 11.

1.3 Der Paradigmenwechsel: Warum jetzt?

Die Idee, Computer per natürlicher Sprache zu steuern, ist so alt wie die Informatik selbst. Was hat sich geändert, sodass es 2025 plötzlich funktioniert?

1.3.1 Die drei Säulen des Vibe Codings

Vibe Coding trägt drei tragende Säulen:

- **Sprachmodelle** (z. B. Claude, GPT, Gemini): Sie liefern die eigentliche Code-Generierung.
- **Agenten-Werkzeuge** (z. B. Claude Code, Cursor, Aider): Sie verbinden die KI mit Dateisystem, Terminal und externen Diensten.
- **Offene Standards** (MCP, Open Source): Sie sorgen dafür, dass Werkzeuge und Modelle miteinander sprechen können.

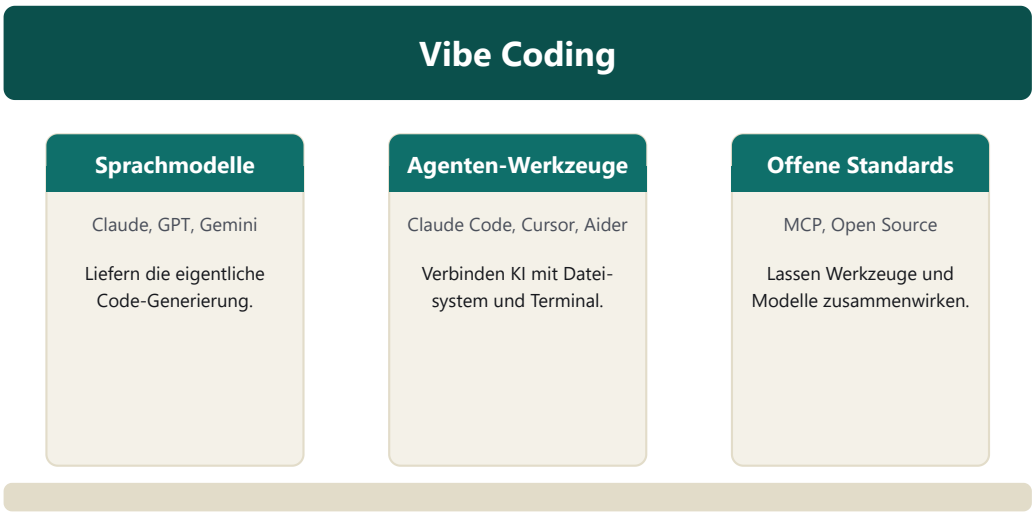

Abb. 1.3 Erst das Zusammenspiel von leistungsfähigen Modellen, agentischen Werkzeugen und offenen Standards macht Vibe Coding möglich.

1.3.1.1 Sprachmodelle mit ausreichender Codierungsfähigkeit

Zwischen 2023 und 2026 erreichten Large Language Models (LLMs, also große Sprachmodelle) eine kritische Schwelle: Sie konnten nicht nur Code generieren, sondern auch verstehen, debuggen, refaktorieren und in Kontext setzen. Claude 3.5 Sonnet konnte bereits komplexe Multi-File-Projekte bearbeiten. Die Nachfolgermodelle Claude Sonnet 4, Claude Opus 4 und Gemini 2.5 Pro haben diese Fähigkeiten weiter ausgebaut.

1.3.1.2 Werkzeuge mit Agenten-Fähigkeit

Die entscheidende Innovation war der Schritt von Chat-basierter KI zu Agenten-basierter KI. Statt nur Texte zu generieren, können moderne KI-Coding-Tools:

- Dateien lesen, erstellen und bearbeiten
- Terminal-Befehle ausführen
- Webseiten aufrufen und analysieren
- Fehler selbstständig erkennen und beheben
- Externe Dienste über MCP (Model Context Protocol) anbinden
- Mehrere Unteraufgaben parallel bearbeiten (Multi-Agent)

Das Model Context Protocol (MCP), im November 2024 von Anthropic als offener Standard eingeführt[10], war ein Wendepunkt. MCP ermöglicht es KI-Modellen, standardisiert auf externe Werkzeuge, Datenbanken, APIs und Dienste zuzugreifen, ähnlich wie USB für Hardware. Im Dezember 2025 wurde MCP an die Agentic AI Foundation (Linux Foundation) übergeben, mitgegründet von Anthropic, Block und OpenAI. Bis April 2026 existieren über 10.000 öffentliche MCP-Server, und die SDKs verzeichnen 97 Millionen monatliche Downloads.

1.3.1.3 Kostenlose und offene Werkzeuge

Vibe Coding öffnet die Softwareentwicklung, weil die Werkzeuge zunehmend kostenlos verfügbar sind:

Tool	Zugang und Kosten
Claude Code (CLI)	API-basiert (pay-per-use). Mächtigster Agenten-Modus mit MCP, Hooks, Multi-Agent.
Gemini CLI	Kostenlos (Apache 2.0). 60 Req/min, 1.000/Tag mit Google-Konto. 1M Token Kontext.
Aider	Open Source. Git-integriert, funktioniert mit jedem LLM inkl. lokaler Modelle.
OpenCode	Open Source (Go). 75+ Modelle, 140.000+ GitHub-Stars.
Cline / Roo Code	Open Source VS-Code-Extensions. Autonome Coding-Agenten.
LlamaCoder	Komplett kostenlos und Open Source. Nutzt offene Modelle.

Null-Kosten-Kombinationen für Einsteiger:

- Gemini CLI + Google-Konto = 0 € (innerhalb der Freitier-Limits)
- Aider + Ollama (lokale Modelle) = 0 € (läuft auf Ihrem Rechner)
- Roo Code (VS Code) + lokale Modelle = 0 €
- Aider + DeepSeek API = unter 5 €/Monat für intensives Coding

1.4 Zahlen und Fakten: Der Status Quo (April 2026)

Um die Dimension des Wandels zu verstehen, hier einige belastbare Signale aus Dokumentation, Wortschatz, Community und Standardisierung[1][2][5][6]:

Kennzahl	Wert
Begriff „vibe coding"	von Andrej Karpathy im Februar 2025 geprägt
Wort des Jahres	Collins wählte „vibe coding" zum Word of the Year 2025
Diskurs	Simon Willison grenzt Vibe Coding 2025 klar von verantwortungsvoller KI-Assistenz ab
MCP-Server	Über 10.000 öffentlich (April 2026)
MCP SDK Downloads	97 Mio./Monat (Python + TypeScript)

Diese Signale zeigen: Vibe Coding ist nicht mehr nur ein Meme, sondern ein reales Arbeitsmuster. Sprache, Werkzeuge und offene Standards haben sich in kurzer Zeit stabilisiert.

1.5 Der Vibe Coding Loop: So funktioniert es in der Praxis

In der Praxis folgt Vibe Coding meist einem einfachen iterativen Kreislauf: Absicht formulieren, Ergebnis erzeugen lassen, prüfen, nachschärfen. Diesen Ablauf nennen wir hier den Vibe Coding Loop:

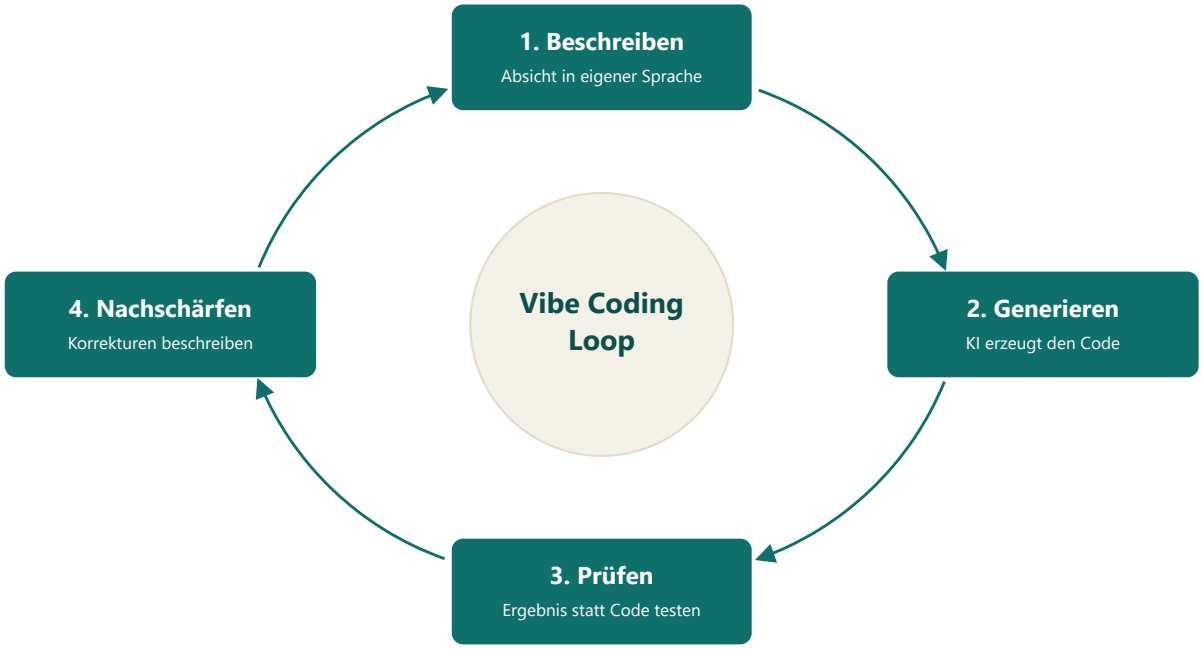

Abb. 1.1 Der Vibe Coding Loop: Beschreiben, generieren lassen, prüfen, nachschärfen, bis das Ergebnis stimmt.

1.5.1 Schritt 1: Absicht formulieren (Prompt)

Sie beschreiben in natürlicher Sprache, was Sie bauen möchten. Je präziser Ihre Beschreibung, desto besser das Ergebnis. Dies ist die wichtigste Fähigkeit im Vibe Coding, und das Thema von Kapitel 3.

1.5.2 Schritt 2: KI generiert Code

Das KI-Tool erstellt Code, Dateien, Konfigurationen. Bei Agenten-Tools wie Claude Code geschieht dies autonom: Die KI liest bestehende Dateien, versteht den Kontext und führt Änderungen durch.

1.5.3 Schritt 3: Ergebnis prüfen

Sie prüfen das Ergebnis, nicht den Code. Funktioniert die App? Sieht die Seite richtig aus? Wenn ja: weiter. Wenn nein: zurück zu Schritt 1.

1.5.4 Schritt 4: Iterieren

Sie verfeinern Ihre Anforderungen, beschreiben Korrekturen und fügen neue Features hinzu. Dieser Zyklus wiederholt sich, bis das Ergebnis Ihren Vorstellungen entspricht.

„Das Erstaunliche am Vibe Coding ist, dass zuerst das Ergebnis im Vordergrund steht, nicht die manuelle Konstruktion jeder einzelnen Codezeile."

Praxis-Beispiel: Unser Camera Studio Prompt Builder

In diesem Buch verwenden wir reale Projekte als Fallstudien. Der Camera Studio V2 Prompt Builder (eine 3.000+ Zeilen HTML/CSS/JS Web-App) wurde vollständig per Vibe Coding erstellt:

1. Beschreibung: „Erstelle einen Prompt-Builder mit Chip-Selektoren..."
2. Claude Code generierte die komplette App in einer Datei
3. Iterative Verfeinerung: „Füge Animation-Sektion hinzu", „Erweitere um Visual Styles"

4. Ergebnis: Produktionsreife Web-App mit 404 Chips, 34 Sektionen, 8 Presets

Gesamtdauer: ca. 45 Minuten für eine App, die traditionell Wochen dauern würde.

1.6 Zeitstrahl: Die Meilensteine des Vibe Codings

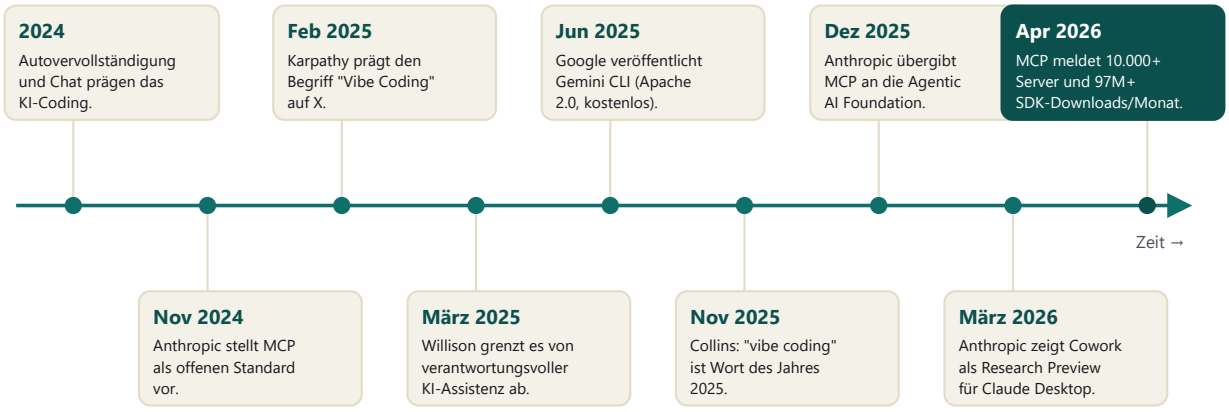

Abb. 1.4 Von der Autovervollständigung zum Ökosystem: Die Meilensteine des Vibe Codings in 18 Monaten.

Datum	Meilenstein
2024	KI-Coding ist noch stark von Autovervollständigung und Chat-Workflows geprägt.
Nov 2024	Anthropic stellt MCP (Model Context Protocol) als offenen Standard vor.
Feb 2025	Andrej Karpathy prägt den Begriff „Vibe Coding" auf X.
Mär 2025	Simon Willison grenzt Vibe Coding klar von verantwortungsvoller KI-Assistenz ab.
Jun 2025	Google veröffentlicht Gemini CLI (Apache 2.0, kostenlos).
Nov 2025	Collins Dictionary: „vibe coding" ist Wort des Jahres 2025.
Dez 2025	Anthropic übergibt MCP an die Agentic AI Foundation (Linux Foundation).
März 2026	Anthropic stellt Cowork als Research Preview für Claude Desktop vor.
Apr 2026	MCP meldet 10.000+ öffentliche Server und 97M+ monatliche SDK-Downloads.

1.7 Wer profitiert von Vibe Coding?

Vibe Coding öffnet die Softwareentwicklung für völlig neue Zielgruppen:

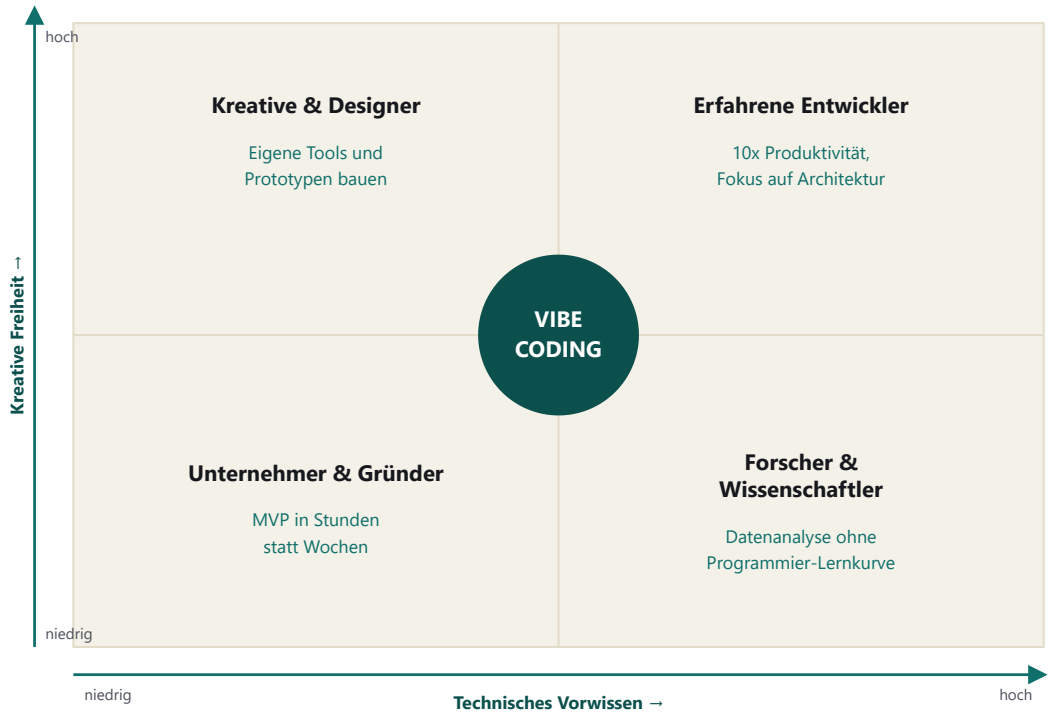

Abb. 1.5 Vibe Coding öffnet Software-Entwicklung für vier grundverschiedene Zielgruppen, jede mit eigenem Nutzen.

1.7.1 Kreative und Designer

Künstler, Musiker, Filmemacher und Designer können eigene Tools, Webseiten und Prototypen bauen, ohne Programmierkurse zu belegen. Unsere Arbeit an Video-Transitions per FFmpeg-Skripte ist ein Beispiel: Die Absicht war „saubere Schnitte", nicht „schreibe einen Python-Algorithmus für Szenen-Erkennung".

1.7.2 Unternehmer und Gründer

MVPs können in Stunden statt Wochen erstellt werden. Ein Startup kann seine Idee validieren, bevor es in ein Entwicklerteam investiert.[4]

1.7.3 Erfahrene Entwickler

Auch Profis profitieren erheblich: Boilerplate-Code, Prototyping, Dokumentation und repetitive Aufgaben werden an die KI delegiert. Die Expertise fokussiert sich auf Architektur, Sicherheit und Qualität.

1.7.4 Forscher und Wissenschaftler

Datenanalyse, Visualisierungen, Simulationen: per Vibe Coding erstellbar, ohne Python oder R lernen zu müssen.[8]

1.8 Kritische Stimmen und berechtigte Bedenken

1.8.1 Code-Qualität und technische Schulden

KI-generierter Code ist nicht immer optimal. Er kann redundant, ineffizient oder schwer wartbar sein. Simon Willison warnt: Vibe Coding funktioniert am besten für „Wegwerf-Software", also Projekte, bei denen man bereit ist, den Code komplett zu verwerfen und neu zu generieren.[5]

1.8.2 Sicherheitsrisiken

Wer Code nicht liest, kann Sicherheitslücken übersehen: SQL-Injection, unsichere API-Keys, fehlende Eingabevalidierung. Kapitel 11 behandelt dieses Thema ausführlich.

1.8.3 Abhängigkeit von KI-Diensten

Wenn der KI-Dienst ausfällt, teurer wird oder die API ändert, steht der Vibe-Coder vor einem Problem. Lösung: lokale Modelle (Ollama) als Fallback, und grundlegendes Verständnis des generierten Codes aufbauen. Genau das vermittelt dieses Buch.

1.8.4 Der Skill-Gap

Karpathy selbst räumte ein: Vibe Coding funktioniert für ihn, weil er erkennen kann, ob der Code sinnvoll ist. Dieses intuitive Verständnis kommt aus jahrzehntelanger Erfahrung. Für Anfänger ist es daher ratsam, parallel ein Grundverständnis aufzubauen, was dieses Buch Schritt für Schritt ermöglicht.

1.9 Von Vibe Coding zu Agentic Engineering

Ein Jahr nach dem ursprünglichen Tweet meldete sich Andrej Karpathy Anfang 2026 erneut zu Wort. Zum Einjährigen relativierte er den Begriff, den er selbst geprägt hatte. Vibe Coding sei eine treffende Beschreibung für einen bestimmten Moment: schnell, spielerisch, ergebnisoffen. Für ernsthafte, dauerhaft gepflegte Software brauche es jedoch mehr. Karpathy schlug dafür den Begriff **Agentic Engineering** vor, also eine reifere Form der Zusammenarbeit mit KI-Agenten.[9]

Der Unterschied ist leicht zu merken, wenn Sie auf die Rolle des Menschen schauen.

Beim Vibe Coding beschreiben Sie ein Ziel und akzeptieren das Ergebnis. Sie geben sich, wie Karpathy es formulierte, den Vibes hin. Der Code bleibt eine Black Box, also ein verschlossener Kasten, in den Sie nicht hineinschauen. Das funktioniert gut, solange das Projekt klein ist und ein Fehler keine großen Folgen hat.

Beim Agentic Engineering entwerfen Sie dagegen das System. Sie legen die Randbedingungen fest, also die Grenzen, in denen die KI arbeiten darf. Sie setzen Agenten gezielt zur Umsetzung ein, behalten aber die Kontrolle und sichern die Qualität ab. Der Mensch wird vom Beifahrer zum Ingenieur, der die Maschine baut, die den Code baut.

1.9.1 Was Agentic Engineering konkret bedeutet

Agentic Engineering ist keine neue Technik, sondern eine Haltung mit festen Tätigkeiten. Wer so arbeitet, tut der Reihe nach Folgendes:

- **Design-Spezifikation schreiben:** Bevor ein Agent loslegt, steht fest, was gebaut wird, mit welchen Schnittstellen, welchen Daten und welchen Grenzen.
- **Pläne überwachen:** Der Agent legt einen Plan vor, der Mensch prüft ihn, bevor Code entsteht.

- **Diffs prüfen:** Jede Änderung wird als Diff, also als Gegenüberstellung von Alt und Neu, gelesen und bewusst freigegeben.
- **Tests schreiben:** Automatische Tests sichern ab, dass der erzeugte Code tut, was er soll, und dass spätere Änderungen nichts zerstören.
- **Eval-Schleifen aufsetzen:** Der Mensch misst die Qualität der Agenten-Ergebnisse systematisch und bessert die Anweisungen nach, statt nur auf das Bauchgefühl zu vertrauen.
- **Rechte verwalten:** Welcher Agent darf welche Dateien ändern, welche Befehle ausführen, welche Dienste erreichen? Diese Berechtigungen werden eng gehalten.
- **Qualität sichern:** Am Ende steht eine bewusste Endkontrolle, kein blindes „Accept All".

Diese Aktivitäten ziehen sich durch das ganze Buch. Kapitel 4 zeigt die Design-Spezifikation in Form der `CLAUDE.md`, Kapitel 8 die Agenten und ihre Rechte, Kapitel 9 das Testen und Prüfen, Kapitel 11 die Grenzen, an denen Kontrolle unverzichtbar wird.

1.9.2 Wann Sie was einsetzen

Beide Arbeitsweisen haben ihren Platz. Es geht nicht um besser oder schlechter, sondern um den passenden Einsatz.

Nutzen Sie **Vibe Coding** für den Prototyp, das kreative Experiment und den schnellen Start. Wenn Sie eine Idee in einer Stunde ausprobieren wollen und der Code danach gelöscht werden darf, ist der lockere Modus genau richtig.

Nutzen Sie **Agentic Engineering** für alles, was in Produktion geht, also für Software, die echte Nutzer bedienen, die Daten verarbeiten und über Monate gepflegt werden muss. Hier zahlt sich die Disziplin aus: Spezifikation, geprüfte Diffs, Tests und klare Rechte.

In der Praxis fließen beide ineinander. Viele Projekte starten als Vibe-Coding-Experiment und wachsen in das Agentic Engineering hinein, sobald sie ernst werden. Dieses Buch begleitet genau diesen Weg: vom ersten spielerischen Prompt bis zum kontrollierten, produktionsreifen System.

1.10 Weiterführende Literatur und Quellenverzeichnis

1.10.1 Offizielle Dokumentation

- Anthropic Claude Code: docs.anthropic.com/en/docs/claude-code
- MCP Spezifikation: modelcontextprotocol.io/specification
- Google Gemini CLI: github.com/google-gemini/gemini-cli
- Claude Agent SDK: platform.claude.com/docs/en/agent-sdk/overview

1.10.2 Online-Ressourcen

- Simon Willison: simonwillison.net
- Awesome Claude Code: github.com/hesreallyhim/awesome-claude-code
- Wikipedia: en.wikipedia.org/wiki/Vibe_coding

Quellenverzeichnis

1. Andrej Karpathy, X/Twitter, 2. Feb. 2025. x.com/karpathy/status/1886192184808149383

2. Collins English Dictionary, Word of the Year 2025: „vibe coding"
3. Simon Willison, „Not all AI-assisted programming is vibe coding", 19. März 2025
4. Stack Overflow Developer Survey 2025, survey.stackoverflow.co/2025/developers
5. Anthropic, „Model Context Protocol", Nov. 2024. anthropic.com/news/model-context-protocol
6. Anthropic, „Donating the Model Context Protocol and establishing the Agentic AI Foundation", 9. Dez. 2025
7. Anthropic, „Claude Cowork", claude.com/product/cowork, 23. März 2026
8. Nature, „How scientists are using AI coding assistants", 2025 (zur Nutzung durch Forscher)
9. Andrej Karpathy, Reflexion zum Einjährigen von „Vibe Coding" und Vorschlag „Agentic Engineering", X/Twitter, Februar 2026
10. Sequoia Capital, „AI Ascent 2026", Vortrags- und Diskussionsreihe zur agentischen Softwareentwicklung
11. Fachpresse-Berichterstattung zum Übergang von Vibe Coding zu Agentic Engineering, 2026

Im nächsten Kapitel geht es um die Werkzeuge: Claude Code, Gemini CLI, Aider, Cline und mehr, mit Installationsanleitungen und konkreten Beispiel-Sessions.