

Was ist KI-native Softwareentwicklung? Copilot, KI-nativ, Vibe Coding

Von Thomas Faß · 2. Oktober 2026 · aktualisiert 16. Oktober 2026

KI-native Softwareentwicklung (englisch: AI-native software development) bezeichnet ein Vorgehen, bei dem der Entwicklungsprozess von Anfang an um KI-Agenten herum gebaut ist: Die KI erzeugt den Großteil des Codes, Menschen verantworten Anforderungen, Architektur, Prüfung und Freigabe. Davon zu unterscheiden sind zwei Nachbarbegriffe. KI-gestützte Entwicklung setzt einen Code-Assistenten in einen unveränderten Prozess. Vibe Coding lässt die KI Code erzeugen, den niemand prüft. Wie weit die Praxis ist, zeigt die Stack Overflow Developer Survey 2025: 23,1 Prozent der befragten Entwickler nutzen KI-Agenten mindestens wöchentlich, 13,8 Prozent setzen KI ausschließlich zur Code-Vervollständigung ein.

Das Wichtigste in Kürze

- KI-nativ beschreibt einen Prozess, kein Werkzeug. Dasselbe Werkzeug kann in einem klassischen Team als Assistent laufen oder einen KI-nativen Prozess tragen. Entscheidend ist, wer den Code erzeugt und wer ihn prüft.
- Drei Stufen lassen sich sauber trennen: KI-gestützt (Assistent im bestehenden Prozess), KI-nativ (Agenten erzeugen, Menschen prüfen und geben frei) und Vibe Coding (Agenten erzeugen, niemand prüft).
- Agenten sind im Alltag noch eine Minderheit. 23,1 Prozent der Entwickler nutzen sie mindestens wöchentlich, 37,9 Prozent planen keinen Einsatz (Stack Overflow Developer Survey 2025, 31.877 Antworten).
- Vibe Coding ist im Beruf die Ausnahme. Für 77,5 Prozent der Befragten gehört es nicht zur beruflichen Entwicklungsarbeit (gleiche Erhebung, 26.564 Antworten).
- Im deutschen Mittelstand steht die Entwicklung am Anfang. 57 Prozent der Unternehmen ab 20 Beschäftigten setzen KI ein, 11 Prozent KI-Agenten (Bitkom, 14.09.2026, 603 Unternehmen).
- Der Prüfaufwand ist der Engpass. Nur 3,1 Prozent der Entwickler haben hohes Vertrauen in die Genauigkeit von KI-Ausgaben, 66 Prozent nennen „fast richtig, aber nicht ganz“ als größte Frustration (Stack Overflow 2025).
- Erfahrung bleibt gefragt, an anderer Stelle. Mit KI-Code-Vervollständigung steigerten weniger erfahrene Entwickler ihre Ausgabe um 21 bis 40 Prozent, erfahrene um 7 bis 16 Prozent (Cui et al., Feldexperimente mit 4.867 Entwicklern, 2026). In KI-nativen Teams wandert Erfahrung in Architektur, Prüfung und Freigabe.
- Die Messlage zum Produktivitätsgewinn ist offen. Belegt sind 26 Prozent mehr erledigte Aufgaben mit Code-Vervollständigung. Für erfahrene Entwickler in eigenen Repositories schwanken die Messungen zwischen Verlangsamung (METR, 2025) und einem Signal, das die Autoren selbst als unzuverlässig einstufen (METR, 2026).

Was bedeutet „KI-nativ“ in der Softwareentwicklung?

KI-native Softwareentwicklung, im Englischen „AI-native software development“ oder „AI-native software engineering“, meint einen Entwicklungsprozess, der von Anfang an für die Arbeit mit KI-Agenten entworfen ist. Die KI erzeugt den Großteil des Codes. Menschen legen fest, was gebaut wird, entscheiden über die Architektur, definieren die Prüfregeln und geben jeden Stand frei.

Das Analystenhaus Gartner beschreibt „AI-native software engineering“ als Entwicklung, bei der KI in jede Phase des Lebenszyklus eingebettet ist, von der Gestaltung bis zur Auslieferung. Die KI übernimmt dabei nach Gartners Formulierung autonom oder teilautonom einen erheblichen Teil der Aufgaben, und die Rolle der Entwickler verschiebt sich von der Umsetzung zur Orchestrierung (Gartner, Pressemitteilung vom 01.07.2025). Gartner erwartet, dass bis 2028 90 Prozent der Softwareentwickler in Unternehmen KI-Code-Assistenten nutzen, gegenüber weniger als 14 Prozent Anfang 2024 (Analystenprognose, kein ausgewiesenes Erhebungsdesign).

Für die Praxis ist eine Folgerung wichtig: KI-nativ ist eine Eigenschaft des Prozesses. Ob ein Team KI-nativ arbeitet, lässt sich an den Rollen und Prüfpunkten ablesen. Die Lizenz für ein bestimmtes Werkzeug sagt darüber wenig. Ein Team kann denselben Agenten als Komfortfunktion nutzen oder seinen gesamten Ablauf darauf ausrichten.

Drei Merkmale tragen die Definition:

1. **Erzeugung durch Agenten.** Den Großteil des Codes schreiben KI-Agenten, die selbstständig Dateien bearbeiten, Befehle ausführen und Tests laufen lassen.
2. **Prüfung als fester Prozessschritt.** Jede Änderung durchläuft definierte Prüfstufen, bevor sie in den Hauptzweig gelangt.
3. **Menschliche Verantwortung für Ziel und Freigabe.** Anforderungen, Architektur und Freigabe liegen bei benannten Personen.

Fehlt das erste Merkmal, handelt es sich um KI-gestützte Entwicklung. Fehlen das zweite und dritte, handelt es sich um Vibe Coding.

Woher kommen die Begriffe?

Die drei Stufen sind innerhalb von fünf Jahren entstanden. Eine kurze Zeitleiste der Wegmarken, die für die Begriffe prägend waren:

- **29. Juni 2021:** GitHub startet eine technische Vorschau von Copilot und stellt das Werkzeug als „AI pair programmer“ vor, als KI-Partner beim Programmieren (GitHub-Blog). Damit beginnt die KI-gestützte Stufe in der Breite.
- **13. Februar 2023:** Eine kontrollierte Studie misst erstmals den Zeitgewinn: Entwickler mit Copilot lösen eine abgegrenzte Programmieraufgabe 55,8 Prozent schneller (Peng et al., arXiv). Gemessen wurde eine einzelne Standardaufgabe ohne Altlasten.
- **2. Februar 2025:** Andrej Karpathy prägt den Begriff „Vibe Coding“ für ein Arbeiten, bei dem man sich ganz den Vorschlägen der KI überlässt.

- **1. Juli 2025:** Gartner führt „AI-native software engineering“ unter den strategischen Trends der Softwareentwicklung.
- **4. Februar 2026:** Karpathy schlägt „agentic engineering“ als Begriff für die professionelle Arbeit mit Agenten unter menschlicher Aufsicht vor.

Die Begriffe beschreiben damit eine Bewegung in drei Schritten: von der Ergänzung einzelner Zeilen über das unkontrollierte Erzeugen ganzer Anwendungen hin zu einem Prozess, der das Erzeugen der KI überlässt und die Kontrolle neu organisiert.

Wie unterscheidet sich das von Copilot & Co. im klassischen Team?

Im klassischen Team mit KI-Assistent schreiben Menschen den Code, und die KI ergänzt Zeilen, schlägt Funktionen vor oder beantwortet Fragen. Der Prozess bleibt unverändert: dieselben Rollen, dieselben Reviews, dieselbe Planung. Diese Stufe heißt hier KI-gestützt.

Auf dieser Stufe ist der Nutzen am besten gemessen. In drei randomisierten Feldexperimenten bei Microsoft, Accenture und einem Fortune-100-Unternehmen mit zusammen 4.867 Entwicklern erledigten die Teilnehmer mit KI-Code-Vervollständigung 26,08 Prozent mehr Aufgaben, bei einem Standardfehler von 10,3 Prozent (Cui et al., Management Science, 27.02.2026). Weniger erfahrene Entwickler steigerten ihre Ausgabe um 21 bis 40 Prozent, erfahrene um 7 bis 16 Prozent (Arbeitsfassung derselben Studie, Februar 2025).

Wie verbreitet die einzelnen Stufen sind, zeigt die Frage nach KI-Agenten in der Stack Overflow Developer Survey 2025 (31.877 Antworten):

Nutzung von KI-Agenten bei der Arbeit	Anteil
täglich	14,1%
wöchentlich	9,0%
monatlich oder seltener	7,8%
nein, aber geplant	17,4%
nein, KI nur als Copilot bzw. Autovervollständigung	13,8%
nein, nicht geplant	37,9%

Quelle: Stack Overflow Developer Survey 2025, Abschnitt KI. Die sechs Antworten ergeben zusammen 100 Prozent.

Gut ein Drittel der Entwickler (30,9 Prozent) hat Agenten also schon eingesetzt, mindestens einmal pro Woche nutzen sie 23,1 Prozent. Weitere 17,4 Prozent planen den Einsatz. Die Stufe, über die dieser Artikel spricht, ist damit verbreitet genug, um sie ernst zu nehmen, und jung genug, dass sich Begriffe und Arbeitsweisen gerade erst festigen.

Wie lassen sich die drei Stufen vergleichen?

	KI-gestützt	KI-nativ	Vibe Coding
Prozess	unverändert, KI als Assistent	um Agenten herum gebaut	ohne festen Prozess
Wer erzeugt den Code?	überwiegend Menschen	überwiegend KI-Agenten	KI-Agenten
Rolle des Menschen	schreibt und prüft	spezifiziert, entscheidet über Architektur, prüft, gibt frei	beschreibt das Ziel, testet das Ergebnis oberflächlich
Prüfinstanz	bestehendes Code-Review	definierte Prüfstufen und menschliche Freigabe vor jedem Merge	keine
Typische Werkzeuge	Code-Vervollständigung und Chat im Editor	Agenten im Editor, im Terminal oder im Repository	Prompt-zu-App-Plattformen, Agenten ohne Prüfpfad
Sinnvoll für	jedes Team, geringe Umstellung	Vorhaben mit klaren Anforderungen und Prüfkompetenz im Team	Prototypen, Machbarkeitsnachweise, private Werkzeuge
Grenze	Prozess bleibt der Engpass	ohne Prüfkompetenz wird es zum Vibe Coding	nicht für Systeme im Betrieb

Die Tabelle beschreibt Idealtypen. In der Praxis arbeiten viele Teams gemischt: KI-nativ bei neuen Modulen, KI-gestützt im gewachsenen Bestand und mit Vibe Coding für interne Wegwerf-Prototypen. Das ist legitim, solange klar ist, welcher Code welche Prüfung durchlaufen hat.

Was ist Vibe Coding, und warum ist es etwas anderes?

Den Begriff „Vibe Coding“ prägte Andrej Karpathy, Mitgründer von OpenAI und früher KI-Chef bei Tesla, am 2. Februar 2025 in einem Beitrag auf X. Er beschrieb damit ein Arbeiten, bei dem man sich ganz den Vorschlägen der KI überlässt und vergisst, dass der Code überhaupt existiert. Gedacht war das ausdrücklich für Wegwerfprojekte am Wochenende.

Im Berufsalltag ist Vibe Coding selten. Auf die Frage, ob es Teil der beruflichen Entwicklungsarbeit ist, antworteten 72,2 Prozent mit „Nein“ und 5,3 Prozent mit „Nein, ausdrücklich“, zusammen 77,5 Prozent. Rund 15 Prozent bejahten in irgendeiner Form, 2,1 Prozent hatten es ausprobiert, die übrigen waren unsicher oder nicht zuzuordnen (Stack Overflow Developer Survey 2025, 26.564 Antworten).

Der Unterschied zur KI-nativen Entwicklung liegt nicht im Werkzeug. Beide lassen Agenten den Code schreiben. Beim Vibe Coding prüft niemand, was entsteht. Bei KI-nativer Entwicklung durchläuft jede Änderung festgelegte Prüfstufen, bevor ein Mensch sie freigibt. Für einen Prototyp ist Vibe Coding legitim

und schnell. Was zwischen einem solchen Prototyp und dem Betrieb fehlt, zeigt unsere [Checkliste zur Produktionsreife](#) in zwölf Punkten.

Karpathy selbst hat ein Jahr später nachgelegt. Am 4. Februar 2026 schlug er für die professionelle Variante einen eigenen Begriff vor: „agentic engineering“. Der Mensch orchestriert inzwischen meist Agenten, die den Code schreiben, und übernehmen die Aufsicht. Das sei eine eigene Disziplin mit eigener Tiefe, die man lernen könne. Der Begriff beschreibt im Kern dieselbe Stufe, die dieser Artikel KI-nativ nennt. Im deutschsprachigen Raum ist „KI-native Softwareentwicklung“ der gebräuchlichere Ausdruck, „agentische Softwareentwicklung“ wird gelegentlich synonym verwendet.

Welche Werkzeuge gibt es für KI-native Entwicklung?

Der Markt ändert sich monatlich, die Kategorien sind stabiler. Vier Werkzeugtypen lassen sich unterscheiden. Die Beispiele sind alphabetisch geordnet und stehen für die Kategorie, nicht für eine Empfehlung. Die Beschreibungen geben die Selbstdarstellung der Anbieter wieder.

1. Code-Vervollständigung und Chat im Editor. Die KI schlägt die nächsten Zeilen vor oder beantwortet Fragen zum Code, der Mensch übernimmt oder verwirft. Das ist die Werkzeugbasis der KI-gestützten Stufe. GitHub Copilot ist in dieser Form 2021 gestartet.

2. Agenten im Editor oder Terminal. Der Agent liest die Codebasis, bearbeitet mehrere Dateien, führt Befehle und Tests aus und legt das Ergebnis zur Prüfung vor. Beispiele: Claude Code von Anthropic beschreibt sich als agentisches Programmierwerkzeug, das die Codebasis liest, Dateien bearbeitet, Befehle ausführt und sich in Entwicklungswerkzeuge einbindet. Codex von OpenAI tritt als Agent für alle Orte auf, an denen programmiert wird. Der Agent von Cursor erledigt nach Anbieterangabe komplexe Programmieraufgaben selbstständig, führt Terminal-Befehle aus und bearbeitet Code. Welche Leitplanken Agenten brauchen, die im Unternehmen eigenständig handeln, behandelt der Fachartikel [KI-Agenten im Unternehmen einführen](#).

3. Agenten im Repository. Der Agent arbeitet im Hintergrund an einer Aufgabe und liefert das Ergebnis als Änderungsvorschlag im Repository. Beispiel: Der Cloud-Agent von GitHub Copilot kann laut Dokumentation selbstständig im Hintergrund Aufgaben erledigen, wie es ein menschlicher Entwickler täte.

4. Prompt-zu-App-Plattformen. Aus einer Beschreibung in natürlicher Sprache entsteht eine lauffähige Web-Anwendung. Beispiele: Bolt wirbt damit, Apps und Websites im Chat mit einer KI zu erstellen. Lovable verspricht, dass sich bauen lässt, was sich beschreiben lässt. v0 von Vercel versteht sich als KI-Assistent, um Full-Stack-Webanwendungen zu entwerfen und zu skalieren. Wie ein so entstandener Prototyp in den Betrieb kommt, behandelt [Teil 2 dieser Reihe](#).

Zwei Hinweise für die Auswahl. Erstens bestimmt der Werkzeugtyp nicht die Stufe: Ein Agent aus Kategorie 2 kann in einem KI-nativen Prozess mit Prüfstufen laufen oder ohne jede Prüfung. Zweitens hängt die Wahl im Unternehmen weniger an der Leistung des Werkzeugs als an zwei Fragen: Welche Daten darf das Werkzeug sehen, und wo laufen die Modelle? Diese Fragen behandelt Teil 4 am 23. Oktober.

Wer macht was? Die neue Arbeitsteilung

KI-Agenten übernehmen in einem KI-nativen Prozess die Ausführung. Sie lesen die bestehende Codebasis, bearbeiten Dateien, führen Befehle und Tests aus und arbeiten Aufgaben im Hintergrund ab. Für die Frage, ob ein Prozess KI-nativ ist, sind die Werkzeuge austauschbar.

Bei den Menschen bleiben vier Aufgaben, die sich nicht delegieren lassen:

1. **Anforderungen klären.** Was gebaut werden soll, entsteht im Gespräch mit Fachbereich und Nutzern. Ein Agent setzt um, was beschrieben ist. Was nicht beschrieben ist, setzt er nach eigener Annahme um.
2. **Architektur entscheiden.** Welche Systeme verbunden werden, welche Daten wohin fließen und welche Abhängigkeiten das Unternehmen eingeht, sind Urteilsfragen mit Folgen über Jahre.
3. **Prüfregeln festlegen.** Testabdeckung, statische Analyse, Sicherheitsprüfungen und Review-Kriterien bestimmen, was als fertig gilt.
4. **Freigeben und verantworten.** Jeder ausgelieferte Stand braucht eine benannte Person, die ihn freigegeben hat.

Dass die Prüfung zur Kernarbeit wird, zeigen die Vertrauenswerte. Nur 3,1 Prozent der Entwickler haben hohes Vertrauen in die Genauigkeit von KI-Ausgaben, 46 Prozent misstrauen ihr aktiv. Die häufigste Frustration sind Lösungen, die „fast richtig, aber nicht ganz“ sind: 66 Prozent nennen sie (Stack Overflow Developer Survey 2025). Im DORA-Report 2025 gaben 30 Prozent von knapp 5.000 befragten Fachleuten an, dem erzeugten Code wenig oder gar nicht zu vertrauen (DORA/Google Cloud, 23.09.2025).

Ein fast richtiger Vorschlag kostet mehr Prüfzeit als ein offensichtlich falscher. Deshalb entscheidet in einem KI-nativen Prozess die Prüfkapazität über das Tempo. Wie Prüfstufen aufgebaut sind, behandelt [Teil 3 dieser Reihe](#) am 16. Oktober.

Wie läuft eine Aufgabe in den drei Stufen ab?

Ein schematisches Beispiel macht den Unterschied greifbar. Es zeigt Abläufe, keine Messwerte. Die Aufgabe: Kunden eines Online-Portals sollen ihre Rechnungen als PDF herunterladen können.

KI-gestützt. Eine Entwicklerin liest die Anforderung, plant die Änderung und schreibt den Code im Editor. Der Assistent schlägt dabei Funktionsrumpfe, die Abfrage der Rechnungsdaten und Teile der PDF-Erzeugung vor. Sie übernimmt einen Teil, passt anderes an. Ein Kollege prüft die Änderung im üblichen Code-Review. Der Ablauf ist derselbe wie ohne KI, nur einzelne Schritte gehen schneller.

KI-nativ. Die Entwicklerin präzisiert zuerst die Anforderung: Welche Rechnungen darf welcher Nutzer sehen, welches Format, welche Fehlerfälle. Sie legt fest, in welchem Modul die Funktion liegt und welche Bibliothek für die PDF-Erzeugung zulässig ist. Ein Agent setzt die Aufgabe um, schreibt dazu Tests und führt sie aus. Die Änderung läuft durch die automatischen Prüfstufen: Tests, statische Analyse, Sicherheitsprüfung der neuen Abhängigkeit. Die Entwicklerin prüft das Ergebnis gezielt dort, wo Fehler teuer wären, etwa bei der Frage, ob ein Nutzer fremde Rechnungen abrufen kann. Erst dann gibt sie frei. Ihre Zeit fließt in Spezifikation und Prüfung, das Schreiben übernimmt der Agent.

Vibe Coding. Jemand beschreibt einer Plattform oder einem Agenten den Wunsch: „Kunden sollen ihre Rechnungen als PDF herunterladen können.“ Nach wenigen Minuten erscheint ein Knopf, der eine PDF-Datei liefert. Ob die Berechtigungen stimmen, ob andere Nutzer über eine geänderte Adresse fremde Rechnungen abrufen können und ob die verwendete Bibliothek gepflegt wird, prüft niemand. Für eine Demo reicht das. Im Betrieb wäre die fehlende Rechteprüfung ein Datenschutzvorfall.

Das Beispiel zeigt, wo der Unterschied liegt: Die eigentliche Arbeit der Menschen verschiebt sich von der Umsetzung zu Spezifikation und Prüfung. Wer diese Verschiebung nicht organisiert, landet bei gleichem Werkzeug in der dritten Stufe.

Was ändert sich für die Rollen im Team?

Die Feldexperimente von Cui et al. zeigen, wie ungleich der Effekt nach Erfahrung ausfällt: plus 21 bis 40 Prozent bei weniger erfahrenen Entwicklern, plus 7 bis 16 Prozent bei erfahrenen. Gemessen wurde zwar die KI-gestützte Stufe, die Richtung ist für die Rollenfrage trotzdem aufschlussreich. Wo die KI vor allem schreibt, profitiert am meisten, wer beim Schreiben bisher am meisten Zeit brauchte.

Für KI-native Teams ergibt sich daraus eine neue Gewichtung:

- **Erfahrene Entwicklerinnen und Entwickler** werden zu Spezifizierenden, Architekten und Prüfenden. Ihre Zeit ist in Urteilsfragen besser investiert als in die Erzeugung von Code.
- **Weniger erfahrene Teammitglieder** gewinnen beim Umsetzen am meisten, brauchen aber einen Rahmen, in dem sie lernen, erzeugten Code zu beurteilen. Ohne diesen Rahmen fehlt dem Team in einigen Jahren der Nachwuchs für die Prüfrolle.
- **Fachbereiche** rücken näher an die Entwicklung, weil präzise Anforderungen den größten Einfluss auf das Ergebnis haben.
- **Prüfkapazität wird zur Planungsgröße.** Wenn Agenten schneller erzeugen, als Menschen prüfen können, wächst der Anteil dessen, was ungeprüft durchgeht. Der DORA-Report 2025 beschreibt genau diese Kombination: steigender Durchsatz bei sinkender Auslieferungsstabilität.

Woran erkennt man, ob ein Team KI-nativ arbeitet?

Sieben Merkmale helfen bei der Einordnung, etwa im Gespräch mit einem Dienstleister oder bei der Bewertung des eigenen Teams:

1. **Anforderungen liegen schriftlich und prüfbar vor**, bevor ein Agent mit der Umsetzung beginnt.
2. **Für jede Änderung ist nachvollziehbar, ob sie von einem Agenten oder einem Menschen erzeugt wurde.**
3. **Automatische Prüfstufen sind fester Bestandteil der Pipeline:** Tests mit festgelegter Mindestabdeckung, statische Analyse, Sicherheits- und Abhängigkeitsprüfung.
4. **Vor jedem Merge steht eine menschliche Freigabe** durch eine benannte Person.
5. **Architekturentscheidungen sind dokumentiert** und werden nicht dem Agenten überlassen.
6. **Es ist geregelt, welche Daten welches Werkzeug sehen darf.**

7. **Der Gewinn wird gemessen**, gegen eine vorher erhobene Ausgangsgröße wie Durchlaufzeit oder Fehlerquote nach Freigabe.

Fehlen die Punkte 3 und 4, arbeitet ein Team im Kern mit Vibe Coding, auch wenn es moderne Werkzeuge einsetzt. Fehlt Punkt 1, arbeiten die Agenten auf Annahmen.

Für welche Vorhaben eignet sich der Ansatz, und für welche nicht?

Gute Voraussetzungen bringen Vorhaben mit klar beschriebenen Anforderungen, einem hohen Anteil wiederkehrender Bausteine und eindeutigen Abnahmekriterien mit: Neubau einer Anwendung, Erweiterungen mit klaren Schnittstellen, Testabdeckung und Dokumentation, Härtung eines bestehenden Prototyps.

Schwieriger wird es in drei Fällen:

- **Unklare Anforderungen.** Wenn sich das Ziel unterwegs verschiebt, erzeugt ein Agent schnell viel Code für eine Richtung, die später verworfen wird.
- **Fehlende Prüfkompetenz im Team.** Ohne Menschen, die Architektur und Sicherheit beurteilen können, rutscht ein KI-nativer Prozess in Vibe Coding ab, auch wenn er formal Prüfstufen hat.
- **Regulierte Kernsysteme mit sensiblen Daten.** Hier entscheidet zuerst die Frage, welches Werkzeug welche Daten sehen darf. Das ist eine Architekturentscheidung vor dem ersten Build und Thema von Teil 4 am 23. Oktober.

Im deutschen Mittelstand ist der Ansatz noch in der Anfangsphase. Laut Bitkom setzen 57 Prozent der Unternehmen ab 20 Beschäftigten KI ein, nach 36 Prozent im Vorjahr und 20 Prozent im Jahr 2024. KI-Agenten nutzen 11 Prozent, weitere 29 Prozent planen den Einsatz (Bitkom, Presseinformation vom 14.09.2026, telefonische Befragung von 603 Unternehmen, Juli und August 2026). Die Zahlen beziehen sich auf den KI-Einsatz im Unternehmen insgesamt. Für die Softwareentwicklung weist die Erhebung keinen eigenen Wert aus.

Welche Missverständnisse halten sich?

„**KI-nativ heißt, man braucht keine Entwickler mehr.**“ Die Arbeit verschiebt sich, sie entfällt nicht. Anforderungen, Architektur, Prüfung und Freigabe brauchen mehr Erfahrung als zuvor, weil die Menge des zu beurteilenden Codes steigt.

„**KI-nativ ist in jedem Fall schneller.**“ Für erfahrene Entwickler in vertrauten, gewachsenen Codebasen hat METR 2025 eine Verlangsamung gemessen, und die Folgemessung von 2026 ist nach Aussage der Autoren nicht belastbar. Der Vorsprung hängt am Aufgabentyp, an der Klarheit der Anforderungen und an der Prüfkapazität.

„**KI-nativ ist eine Werkzeugentscheidung.**“ Wer ein Agenten-Werkzeug lizenziert und den Prozess unverändert lässt, arbeitet KI-gestützt oder rutscht ins Vibe Coding. KI-nativ wird ein Team durch Rollen, Prüfstufen und Freigaberegeln.

„**Vibe Coding ist unprofessionell.**“ Für Prototypen, Machbarkeitsnachweise und interne Wegwerf-Werkzeuge ist es ein schneller und legitimer Weg. Problematisch wird es erst, wenn ein so entstandener Prototyp ohne Überarbeitung in den Betrieb geht.

Was zeigen die Studien zum Produktivitätsgewinn?

Die belastbarste Zahl stammt aus den genannten Feldexperimenten: 26,08 Prozent mehr erledigte Aufgaben mit KI-Code-Vervollständigung, mit deutlich größerem Effekt bei weniger erfahrenen Entwicklern (Cui et al., 2026). Gemessen wurde damit die KI-gestützte Stufe. Die frühere Einzelstudie mit 55,8 Prozent Zeitgewinn gilt für eine abgegrenzte Standardaufgabe und ist als Obergrenze unter günstigen Bedingungen zu lesen (Peng et al., 2023).

Für erfahrene Entwickler ist das Bild offen. In einer randomisierten Studie von METR brauchten 16 erfahrene Open-Source-Entwickler für 246 Aufgaben in ihren eigenen Repositories mit KI-Werkzeugen 19 Prozent länger. Sie hatten 24 Prozent Beschleunigung erwartet und glaubten hinterher, 20 Prozent schneller gewesen zu sein (METR, 10.07.2025). Eine Folgemessung ab August 2025 mit 57 Entwicklern und über 800 Aufgaben zeigt in den Rohdaten eher eine Beschleunigung: minus 18 Prozent Bearbeitungszeit bei den erneut teilnehmenden Entwicklern, minus 4 Prozent bei neu gewonnenen. Die Autoren stufen diese Daten selbst als unzuverlässiges Signal ein, weil viele Entwickler nicht mehr ohne KI arbeiten wollten und der Studie fernblieben (METR, 24.02.2026).

Auf Organisationsebene beschreibt der DORA-Report 2025 einen positiven Zusammenhang zwischen KI-Einsatz und Auslieferungsdurchsatz und weiterhin einen negativen mit der Auslieferungstabilität (DORA/Google Cloud, 23.09.2025).

Die hier ausgewerteten Studien messen KI-Werkzeuge überwiegend in bestehenden Prozessen. Für Unternehmen folgt daraus ein praktischer Rat: vor dem Start eine Ausgangsgröße erheben, etwa Durchlaufzeit je Vorgang oder Fehlerquote nach Freigabe, und den Gewinn später dagegen rechnen. Die Selbsteinschätzung der Beteiligten ist dafür ungeeignet, wie die METR-Studie von 2025 gezeigt hat. Eine ausführliche Auswertung der Studienlage steht im Fachartikel [KI-Produktivität in der Softwareentwicklung](#).

Wie geht es in dieser Reihe weiter?

Dieser Artikel ist Teil 1 der fünfteiligen Themenreihe KI-native Entwicklung. Jeder Teil erscheint freitags und baut auf dem vorherigen auf:

- **Teil 2, 9. Oktober:** [Warum mit KI gebaute Prototypen so oft an der Grenze zum Betrieb scheitern und was zwischen Demo und Produktion liegt.](#)
- **Teil 3, 16. Oktober:** [Qualität, Tests und Sicherheit, wenn KI den Großteil des Codes schreibt.](#)
- **Teil 4, 23. Oktober:** Souveränität im regulierten Mittelstand: welche Daten welches Werkzeug sehen darf.
- **Teil 5, 30. Oktober:** Wirtschaftlichkeit: wann sich KI-native Entwicklung lohnt und wann nicht.

Ihr nächster Schritt

Sie überlegen, wie viel KI in Ihre Softwareentwicklung gehört und wo die Prüfung bei Ihnen liegen müsste? Sprechen Sie mit uns.

Häufige Fragen

Was ist KI-native Softwareentwicklung?

KI-native Softwareentwicklung ist ein Vorgehen, bei dem der Entwicklungsprozess von Anfang an um KI-Agenten herum gebaut ist. Die KI erzeugt den Großteil des Codes, Menschen verantworten Anforderungen, Architektur, Prüfung und Freigabe.

Was heißt AI-native software development auf Deutsch?

„AI-native software development“ oder „AI-native software engineering“ wird im Deutschen meist mit „KI-native Softwareentwicklung“ übersetzt. Gemeint ist ein Entwicklungsprozess, der um KI-Agenten herum gebaut ist. Gartner führt den Begriff seit Juli 2025 unter den strategischen Trends der Softwareentwicklung.

Was ist der Unterschied zwischen KI-gestützter und KI-nativer Entwicklung?

KI-gestützte Entwicklung setzt einen Code-Assistenten in einen unveränderten Prozess, Menschen schreiben weiterhin den Großteil des Codes. Bei KI-nativer Entwicklung erzeugen Agenten den Code, und Rollen und Prüfpunkte sind darauf ausgerichtet.

Ist Vibe Coding dasselbe wie KI-native Entwicklung?

Nein. Beim Vibe Coding erzeugt die KI Code, den niemand systematisch prüft. Das eignet sich für Prototypen. KI-native Entwicklung lässt jede Änderung festgelegte Prüfstufen und eine menschliche Freigabe durchlaufen. Für 77,5 Prozent der Entwickler ist Vibe Coding kein Teil der beruflichen Arbeit (Stack Overflow Developer Survey 2025).

Was bedeutet agentic engineering?

„Agentic engineering“ ist ein Begriff, den Andrej Karpathy im Februar 2026 für die professionelle Arbeit mit KI-Agenten vorgeschlagen hat: Agenten schreiben den Code, Menschen orchestrieren sie und übernehmen die Aufsicht. Inhaltlich entspricht das weitgehend der KI-nativen Softwareentwicklung.

Was ist ein KI-Agent in der Softwareentwicklung?

Ein KI-Agent ist ein Werkzeug, das Programmieraufgaben selbstständig über mehrere Schritte bearbeitet: Es liest die Codebasis, ändert Dateien, führt Befehle und Tests aus und legt das Ergebnis zur Prüfung vor. Ein Code-Assistent schlägt dagegen einzelne Zeilen oder Funktionen vor, die ein Mensch übernimmt.

Welche Werkzeuge gibt es für KI-native Entwicklung?

Vier Kategorien lassen sich unterscheiden: Code-Vervollständigung und Chat im Editor, Agenten im Editor oder Terminal (etwa Claude Code, Codex, Cursor), Agenten im Repository (etwa der Cloud-Agent von GitHub Copilot) und Prompt-zu-App-Plattformen (etwa Bolt, Lovable, v0). Ob ein Prozess KI-nativ ist, entscheidet sich an Prüfstufen und Freigaberegeln, nicht am Werkzeug.

Ist KI-native Entwicklung dasselbe wie Low-Code?

Nein. Low-Code-Plattformen bauen Anwendungen aus vorgefertigten Bausteinen in einer Plattform des Anbieters. KI-native Entwicklung erzeugt regulären Quellcode, der in eigenen Repositories liegt und mit üblichen Werkzeugen geprüft, betrieben und weiterentwickelt wird.

Braucht man für KI-native Entwicklung noch erfahrene Entwickler?

Ja. Erfahrung verschiebt sich von der Umsetzung zu Architektur, Prüfung und Freigabe. Ohne Menschen, die erzeugten Code beurteilen können, fehlt die Prüfinstanz. Nur 3,1 Prozent der Entwickler haben hohes Vertrauen in die Genauigkeit von KI-Ausgaben (Stack Overflow Developer Survey 2025).

Wie verbreitet sind KI-Agenten in der Softwareentwicklung?

23,1 Prozent der Entwickler nutzen KI-Agenten mindestens wöchentlich, 30,9 Prozent überhaupt, 37,9 Prozent planen keinen Einsatz (Stack Overflow Developer Survey 2025, 31.877 Antworten). In deutschen Unternehmen ab 20 Beschäftigten setzen 11 Prozent KI-Agenten ein (Bitkom, September 2026), bezogen auf den Unternehmenseinsatz insgesamt.

Wie groß ist der Produktivitätsgewinn?

Für KI-Code-Vervollständigung sind 26,08 Prozent mehr erledigte Aufgaben belegt (Cui et al., Management Science, 2026). Für erfahrene Entwickler in eigenen Repositories wurde 2025 eine Verlangsamung um 19 Prozent gemessen. Eine Folgemessung von 2026 deutet eher auf Beschleunigung, gilt den Autoren aber als unzuverlässig (METR).

Woran erkennt man ein KI-natives Team?

An schriftlichen, prüfbaren Anforderungen vor der Umsetzung, an automatischen Prüfstufen in der Pipeline, an einer menschlichen Freigabe vor jedem Merge, an dokumentierten Architekturentscheidungen, an Regeln dafür, welche Daten welches Werkzeug sehen darf, und an einer Messung gegen eine vorher erhobene Ausgangsgröße.

Thematische Vertiefungen

[KI-native Softwareentwicklung für den Mittelstand](#)

Quellen

- Gartner: „Gartner Identifies the Top Strategic Trends in Software Engineering for 2025 and Beyond“, Pressemitteilung, 01.07.2025. Definition von AI-native software engineering; Prognose 90 Prozent bis 2028 (Analystenprognose, kein ausgewiesenes Erhebungsdesign). – <https://www.gartner.com/en/newsroom/press-releases/2025-07-01-gartner-identifies-the-top-strategic-trends-in-software-engineering-for-2025-and-beyond>
- Stack Overflow: „Developer Survey 2025“, Abschnitt KI. Über 49.000 Antworten aus 177 Ländern; Frage zu KI-Agenten 31.877 Antworten, Frage zu Vibe Coding 26.564 Antworten. – <https://survey.stackoverflow.co/2025/ai>

- Bitkom: „Erstmals nutzt die Mehrheit der Unternehmen KI“, Presseinformation, 14.09.2026. Telefonische Befragung von 603 Unternehmen ab 20 Beschäftigten, KW 28 bis 33/2026, repräsentativ für die Gesamtwirtschaft. – <https://www.bitkom.org/Presse/Presseinformation/Erstmals-nutzt-Mehrheit-Unternehmen-KI>
- Cui, K. Z.; Demirer, M.; Jaffe, S.; Musolff, L.; Peng, S.; Salz, T.: „The Effects of Generative AI on High-Skilled Work: Evidence from Three Field Experiments with Software Developers.“ *Management Science*, 27.02.2026. – <https://pubsonline.informs.org/doi/10.1287/mnsc.2025.00535>
- Dieselbe Studie, Arbeitsfassung Februar 2025, mit den Effekten nach Erfahrungsstufe. – https://economics.mit.edu/sites/default/files/inline-files/draft_copilot_experiments.pdf
- Peng, S.; Kalliamvakou, E.; Cihon, P.; Demirer, M.: „The Impact of AI on Developer Productivity: Evidence from GitHub Copilot.“ arXiv:2302.06590, eingereicht 13.02.2023. – <https://arxiv.org/abs/2302.06590>
- METR: „Measuring the Impact of Early-2025 AI on Experienced Open-Source Developer Productivity“, 10.07.2025. – <https://metr.org/blog/2025-07-10-early-2025-ai-experienced-os-dev-study/>
- METR: „We are Changing our Developer Productivity Experiment Design“, 24.02.2026. – <https://metr.org/blog/2026-02-24-uplift-update/>
- DORA / Google Cloud: „State of AI-assisted Software Development“, 23.09.2025. – <https://cloud.google.com/blog/products/ai-machine-learning/announcing-the-2025-dora-report>
- GitHub: „Introducing GitHub Copilot: your AI pair programmer“, Nat Friedman, 29.06.2021. – <https://github.blog/news-insights/product-news/introducing-github-copilot-ai-pair-programmer/>
- Karpathy, Andrej: Beitrag auf X vom 02.02.2025 (Prägung „vibe coding“). – <https://x.com/karpathy/status/1886192184808149383>
- Karpathy, Andrej: Beitrag auf X vom 04.02.2026 (Vorschlag „agentic engineering“). – <https://x.com/karpathy/status/2019137879310836075>
- Selbstbeschreibungen der Werkzeuge, abgerufen am 28.09. und 01.10.2026: Bolt – <https://bolt.new> • Claude Code – <https://code.claude.com/docs/en/overview> • Codex – <https://developers.openai.com/codex> • Cursor Agent – <https://cursor.com/docs/agent/overview> • GitHub Copilot Cloud Agent – <https://docs.github.com/en/copilot/concepts/agents/coding-agent/about-coding-agent> • Lovable – <https://lovable.dev> • v0 – <https://v0.app>

Über den Autor

Thomas Faß ist Gründer und CEO von InnoShore, einem IT-Dienstleister für den DACH-Mittelstand, und leitet die Regionalgruppe NRW im KI Bundesverband.