

Vom KI-Prototyp zur Produktion: Was zwischen Demo und Betrieb liegt

Von Thomas Faß · 9. Oktober 2026 · aktualisiert 16. Oktober 2026

Einen mit KI gebauten Prototyp in Produktion zu bringen heißt, die Stellen nachzuholen, die beim schnellen Bauen liegen bleiben: Rechte und Authentifizierung, Umgang mit Zugangsdaten, Datenmodell, Tests, Deploy-Pfad, Betrieb und Wartbarkeit. Ein Prototyp beweist, dass eine Idee funktioniert. Ob er im Betrieb trägt, entscheidet sich an diesen Stellen. Wie groß die Lücke typischerweise ist, zeigt der Veracode GenAI Code Security Report 2026: Im Durchschnitt über mehr als 100 Sprachmodelle besteht nur 56 Prozent des erzeugten Codes die Sicherheitsprüfung, praktisch unverändert gegenüber dem Vorjahr, obwohl der Code fast immer syntaktisch korrekt ist.

Das Wichtigste in Kürze

- Ein Prototyp beweist die Idee, nicht den Betrieb. Was in der Demo funktioniert, sagt wenig über Sicherheit, Datenmodell, Last und Wartbarkeit.
- Lauffähig heißt nicht sicher. KI-generierter Code ist laut Veracode fast immer syntaktisch korrekt, besteht aber im Durchschnitt nur zu 56 Prozent die Sicherheitsprüfung (Veracode GenAI Code Security Report, 28.07.2026, über 100 Modelle).
- Größere Modelle lösen das Problem nicht. Die Sicherheitsquote ist gegenüber 2025 praktisch unverändert, die Modellgröße hat laut Veracode keinen erkennbaren Einfluss.
- Die Lücken zeigen sich im Betrieb. Bei 170 von 1.645 geprüften Lovable-Projekten waren Datenbanktabellen ohne ausreichende Zugriffsregeln von außen lesbar (Sicherheitsforscher Matt Palmer, CVE-2025-48757, Mai 2025).
- Wer mit KI-Assistent programmiert, hält unsicheren Code eher für sicher. In einer Nutzerstudie der Stanford University schrieben Teilnehmende mit KI-Assistent deutlich unsichereren Code und hielten ihn häufiger für sicher (Perry et al., ACM, 2023).
- Zwölf Punkte entscheiden über die Produktionsreife, von Rechten und Zugangsdaten über Tests und Deploy-Pfad bis zu Übergabe und Eigentum.
- Ausbauen oder neu bauen ist eine Sachentscheidung. Sie hängt an Datenmodell, Architektur und Testbarkeit, weniger am Werkzeug, mit dem der Prototyp entstanden ist.

Warum gehen so viele KI-Prototypen nicht in den Betrieb?

Ein Prototyp hat eine Aufgabe: zeigen, dass eine Idee funktioniert. Mit KI-Agenten und Prompt-zu-App-Plattformen entsteht so ein Prototyp heute in Stunden statt Wochen. Was dabei entsteht, ist auf die Demo optimiert. Die Oberfläche reagiert, die Daten erscheinen, der Ablauf stimmt.

Im Betrieb kommen andere Fragen dazu. Wer darf welche Daten sehen? Wo liegen die Zugangsdaten? Was passiert, wenn hundert Nutzer gleichzeitig kommen? Wie wird eine Änderung ausgeliefert, ohne dass Daten verloren gehen? Wer bemerkt einen Fehler um drei Uhr nachts? Keine dieser Fragen ist in einer Demo sichtbar. Deshalb bleiben sie beim schnellen Bauen typischerweise liegen.

Teil 1 dieser Reihe hat drei Stufen unterschieden: KI-gestützte Entwicklung, KI-native Entwicklung und Vibe Coding (siehe [Was ist KI-native Softwareentwicklung?](#)). Viele Prototypen entstehen in der dritten Stufe, ohne festen Prüfpfad. Das ist für einen Machbarkeitsnachweis legitim. Problematisch wird es an dem Punkt, an dem aus dem Prototyp ein System werden soll, mit echten Nutzern, echten Daten und echter Verantwortung.

Lauffähig ist nicht dasselbe wie sicher

Die größte systematische Messung zur Sicherheit von KI-generiertem Code stammt vom Sicherheitsanbieter Veracode. Für den Report 2025 ließ Veracode mehr als 100 Sprachmodelle 80 Programmieraufgaben lösen und prüfte das Ergebnis mit statischer Analyse. In 45 Prozent der Fälle enthielt der erzeugte Code Schwachstellen aus den OWASP Top 10, den häufigsten Sicherheitslücken in Webanwendungen. Bei Java lag die Fehlerquote über 70 Prozent, bei Python, C# und JavaScript zwischen 38 und 45 Prozent. Gegen Cross-Site-Scripting war der Code in 86 Prozent der Fälle nicht abgesichert, gegen Log-Injection in 88 Prozent (Veracode, Pressemitteilung vom 30.07.2025).

Ein Jahr später hat Veracode die Messung wiederholt. Über vier Messzeitpunkte und mehr als 100 Modelle liegt die durchschnittliche Sicherheitsquote bei 56 Prozent, praktisch unverändert gegenüber dem Vorjahr. Die Syntax-Quote liegt dagegen bei rund 100 Prozent: Der Code läuft fast immer. Größere Modelle schneiden nicht erkennbar besser ab als kleinere (Veracode GenAI Code Security Report 2026, Pressemitteilung vom 28.07.2026).

Zwei Einschränkungen gehören dazu. Veracode ist ein Anbieter von Sicherheitsprüfungen und hat damit ein Interesse an diesem Befund. Und getestet wurden reine Modelle mit standardisierten Aufgaben ohne sicherheitsspezifische Anweisungen, keine Agenten mit Prüfstufen in einer echten Arbeitsumgebung. Für die Frage dieses Artikels ist genau das die relevante Lage: Ein Prototyp, der ohne Prüfpfad entsteht, bekommt den Code so, wie das Modell ihn liefert.

Das Gefühl von Sicherheit

Eine zweite Beobachtung macht die Lage heikler. In einer Nutzerstudie der Stanford University schrieben Teilnehmende mit Zugang zu einem KI-Assistenten deutlich unsichereren Code als Teilnehmende ohne Assistenten. Gleichzeitig hielten sie ihren Code häufiger für sicher. Wer der KI weniger vertraute und die Anweisungen stärker anpasste, lieferte sichereren Code (Perry, Srivastava, Kumar, Boneh: „Do Users Write More Insecure Code with AI Assistants?“, ACM, 2023).

Die Studie nutzte ein Modell von 2022 und ist damit technisch überholt. Der Befund zur Selbsteinschätzung passt aber zu dem, was die METR-Studie von 2025 für die Geschwindigkeit gemessen hat: Die Beteiligten schätzen die Wirkung der KI systematisch günstiger ein, als sie gemessen wird. Die Studienlage dazu wertet der Fachartikel [KI-Produktivität in der Softwareentwicklung](#) aus. Für einen Prototyp heißt das: Das Gefühl „läuft doch“ ist kein Prüfergebnis.

Was zeigen reale Vorfälle?

Drei öffentlich dokumentierte Fälle zeigen, wo die Lücken im Betrieb aufbrechen. Sie stehen hier als Beispiele für Muster, nicht als Urteil über einzelne Plattformen.

Fehlende Zugriffsregeln: Lovable, 2025. Der Sicherheitsforscher Matt Palmer prüfte im März 2025 die 1.645 Projekte, die auf der Präsentationsseite der Plattform Lovable gelistet waren. Bei 170 Projekten, rund 10 Prozent, fand er 303 Datenbank-Endpunkte ohne ausreichende Row-Level-Security, also ohne Regeln, wer welche Zeilen einer Tabelle lesen oder schreiben darf. Sichtbar waren unter anderem E-Mail-Adressen, Telefonnummern, Zahlungsdaten und API-Schlüssel. Geprüft hatte er nur die Startseiten, die tatsächliche Zahl kann höher liegen. Die Schwachstelle ist als CVE-2025-48757 registriert. Lovable bestreitet, dass es sich um eine Schwachstelle der Plattform handelt, und verweist darauf, dass jeder Kunde für den Schutz der Daten seiner Anwendung verantwortlich ist (NVD-Eintrag vom 29.05.2025; Stellungnahme Matt Palmer, 29.05.2025).

Keine Trennung der Umgebungen: Replit, 2025. Der Unternehmer Jason Lemkin berichtete im Juli 2025, dass ein KI-Agent der Plattform Replit während einer ausdrücklich angeordneten Änderungssperre die Produktionsdatenbank seines Projekts gelöscht habe. Zuvor habe der Agent Fehler mit erfundenen Daten überdeckt. Lemkin forderte danach eine saubere Trennung von Vorschau-, Test- und Produktionsumgebung (The Register, 21.07.2025).

Öffentlich erreichbare Anwendungen: Erhebung 2026. Die israelische Sicherheitsfirma RedAccess fand nach eigenen Angaben rund 380.000 öffentlich zugängliche Anwendungen, die mit Plattformen wie Lovable, Base44, Replit und Netlify gebaut wurden. Etwa 5.000 davon enthielten sensible Unternehmensdaten, darunter medizinische und finanzielle Informationen. Die Methodik ist nicht im Detail offengelegt, mehrere Plattformen widersprechen der Darstellung in Teilen und verweisen auf die Freigabe-Einstellungen der Nutzer (Axios, 07.05.2026).

Allen drei Fällen ist gemeinsam: Die Anwendungen funktionierten. Was fehlte, waren Zugriffsregeln, getrennte Umgebungen und eine bewusste Entscheidung darüber, was öffentlich sein darf. Das sind die Punkte, die beim Übergang in den Betrieb nachgeholt werden müssen.

Was fehlt einem Prototyp typischerweise?

Zwischen einem funktionierenden Prototyp und einem System, das im Betrieb trägt, liegen zwölf Punkte. Unsere [Checkliste zur Produktionsreife](#) fasst sie kompakt zusammen. Hier stehen sie ausführlicher: was typischerweise fehlt, warum es in der Demo nicht auffällt und woran man es prüft.

1. Auth und Rechte. Prototypen haben oft eine Anmeldung, aber keine durchgängige Rechteprüfung. Ob ein Nutzer fremde Datensätze abrufen kann, indem er eine Kennung in der Adresse ändert, zeigt keine Demo. Prüfen: Jede Abfrage auf Server- oder Datenbankebene gegen die Frage „Darf genau dieser Nutzer genau diesen Datensatz sehen?“ testen, nicht nur die Oberfläche.

2. Secret-Hygiene. API-Schlüssel und Datenbank-Zugangsdaten landen im Prototyp leicht im Quellcode, im Frontend-Bundle oder in der Versionsgeschichte. Prüfen: Repository einschließlich Historie auf

Zugangsdaten scannen, alle gefundenen Schlüssel rotieren, Secrets in eine Verwaltung auslagern. Ein gelöschter Schlüssel ist in der Git-Historie weiterhin vorhanden.

3. Datenmodell und Migrationen. Das Datenmodell eines Prototyps entsteht oft im Gespräch mit der KI und wird nebenbei angepasst. Im Betrieb braucht jede Änderung eine Migration, die vorhandene Daten mitnimmt. Prüfen: Lässt sich die Datenbank aus dem Repository vollständig neu aufbauen? Gibt es versionierte Migrationen?

4. Eingabevalidierung und Injection-Schutz. Hier liegen die Schwachstellen, die Veracode am häufigsten gemessen hat: Cross-Site-Scripting, SQL- und Log-Injection. Prüfen: statische Analyse auf die OWASP Top 10, Eingaben serverseitig validieren, Ausgaben kodieren.

5. Fehlerbehandlung. Ein Prototyp kennt den Erfolgsfall. Was bei einem Zeitüberlauf, einer leeren Antwort oder einem fehlerhaften Datensatz passiert, ist oft nicht definiert. Prüfen: gezielt Fehlerfälle auslösen. Zeigt die Anwendung dem Nutzer interne Fehlermeldungen oder Stacktraces?

6. Tests. KI-Agenten schreiben auf Anfrage Tests, aber in einem Prototyp fragt selten jemand. Prüfen: Gibt es automatisierte Tests für die geschäftskritischen Abläufe, und laufen sie bei jeder Änderung?

7. Deploy-Pfad. Viele Prototypen laufen in der Umgebung der Plattform, auf der sie entstanden sind. Der Replit-Fall zeigt, was passiert, wenn Entwicklung und Produktion nicht getrennt sind. Prüfen: Gibt es getrennte Umgebungen, einen reproduzierbaren Build und eine Möglichkeit, eine fehlerhafte Version zurückzunehmen?

8. Skalierung und Last. Mit drei Testnutzern fällt eine Abfrage, die für jede Zeile einer Liste die Datenbank erneut befragt, nicht auf. Mit dreitausend Nutzern legt sie das System lahm. Prüfen: Lasttest mit realistischen Datenmengen, langsame Abfragen identifizieren.

9. Monitoring und Logging. Im Prototyp schaut der Entwickler selbst zu. Im Betrieb muss ein System melden, wenn etwas schiefgeht. Prüfen: Gibt es Fehler-Monitoring, Alarmierung und Protokolle, die keine personenbezogenen Daten oder Zugangsdaten enthalten?

10. DSGVO und Datenpfad. Prompt-zu-App-Plattformen und KI-Dienste verarbeiten Daten oft bei Anbietern außerhalb der EU. Im Betrieb mit echten Kundendaten wird daraus eine Datenschutzfrage. Prüfen: Wo liegen die Daten, welche Dienste sehen sie, gibt es Verträge zur Auftragsverarbeitung? Teil 4 dieser Reihe am 23. Oktober behandelt die Architekturfragen dahinter.

11. Wartbarkeit. KI-generierter Code ist schnell geschrieben und manchmal schwer zu pflegen, etwa wenn dieselbe Logik an mehreren Stellen kopiert wurde. Prüfen: Lässt sich eine typische Änderung an einer Stelle vornehmen? Gibt es eine nachvollziehbare Struktur?

12. Übergabe und Eigentum. Wem gehören Repository, Plattform-Konto, Domain und Datenbank? Bei Prototypen, die von einer einzelnen Person oder einem externen Dienstleister gebaut wurden, ist das oft ungeklärt. Prüfen: Liegen alle Konten beim Unternehmen? Gibt es eine Dokumentation, mit der ein anderes Team weiterarbeiten kann?

Die zwölf Punkte sind keine Rangfolge. In der Praxis sind die ersten beiden am dringendsten, weil sie bei einem öffentlich erreichbaren Prototyp sofort ausgenutzt werden können.

Welche Fehlerbilder tauchen immer wieder auf?

Aus den zwölf Punkten lassen sich fünf typische Fehlerbilder ableiten, die in Prototypen besonders häufig sind:

- **Zugangsdaten im Frontend.** Ein Schlüssel für einen KI-Dienst oder eine Datenbank steht im Code, der an den Browser ausgeliefert wird. Jeder Besucher kann ihn auslesen und auf Kosten des Unternehmens nutzen.
- **Autorisierung nur in der Oberfläche.** Die Anwendung blendet fremde Daten aus, die Schnittstelle dahinter liefert sie trotzdem. Genau dieses Muster lag den Lovable-Funden zugrunde.
- **Schema ohne Migrationen.** Tabellen wurden direkt in der Datenbank angelegt und geändert. Die nächste Strukturänderung im Betrieb gefährdet vorhandene Daten.
- **Eine Umgebung für alles.** Entwicklung, Test und Produktion teilen sich dieselbe Datenbank. Ein fehlerhafter Testlauf trifft echte Daten.
- **Abfragen ohne Rücksicht auf Datenmenge.** Listen laden für jeden Eintrag nachträglich weitere Daten. Bei Testdaten unsichtbar, bei echten Datenmengen ein Lastproblem.

Ist KI-generierter Code schwerer zu warten?

Für die Wartbarkeit gibt es Hinweise, aber keinen kausalen Beleg. Das Analyseunternehmen GitClear hat 211 Millionen geänderte Codezeilen aus den Jahren 2020 bis 2024 ausgewertet. Der Anteil kopierter Zeilen stieg von 8,3 Prozent im Jahr 2021 auf 12,3 Prozent im Jahr 2024. Der Anteil verschobener Zeilen, ein Indikator für Umbau und Aufräumen, sank im selben Zeitraum von 25 Prozent auf unter 10 Prozent. 2024 wurde nach GitClears Auswertung erstmals mehr Code kopiert als umgebaut (GitClear, AI Copilot Code Quality, Auswertung 2025).

Die Entwicklung fällt zeitlich mit der Verbreitung von KI-Assistenten zusammen. Ob sie dadurch verursacht ist, zeigt die Auswertung nicht. GitClear ist zudem Anbieter von Code-Analysewerkzeugen. Für einen Prototyp ist der Befund trotzdem ein brauchbarer Prüfpunkt: Wo dieselbe Logik mehrfach im Code steht, muss jede künftige Änderung an mehreren Stellen erfolgen, und jede vergessene Stelle wird zum Fehler.

Ausbauen oder neu bauen?

Die wichtigste Entscheidung beim Übergang in den Betrieb ist, ob der Prototyp ausgebaut oder als Spezifikation genutzt und neu gebaut wird. Beides kann richtig sein. Die Kriterien:

Kriterium	spricht für Ausbauen	spricht für Neubau
Datenmodell	sauber, normalisiert, mit Migrationen	gewachsen, widersprüchlich, ohne Migrationen

Kriterium	spricht für Ausbauen	spricht für Neubau
Architektur	klare Trennung von Oberfläche, Logik und Daten	Logik in der Oberfläche, keine Schichten
Technologie	passt zur Zielumgebung und zum Team	an eine Plattform gebunden, nicht exportierbar
Testbarkeit	Kernabläufe lassen sich automatisiert testen	Tests nur über die Oberfläche möglich
Sicherheitsbefund	punktueller Lücken, behebbar	Rechteprüfung fehlt grundsätzlich
Umfang	überschaubar, gut lesbar	groß, mit vielen Kopien derselben Logik

Die Tabelle ist eine Entscheidungshilfe, keine Formel. Eine einzelne Zeile in der rechten Spalte erzwingt keinen Neubau.

Zwei Hinweise helfen bei der Entscheidung. Erstens ist auch ein Neubau keine verlorene Arbeit: Der Prototyp hat Anforderungen, Abläufe und Oberfläche geklärt. Er wird damit zur besten Spezifikation, die ein Team haben kann. Zweitens ist ein Neubau mit KI-Agenten heute deutlich günstiger als früher. Die Abwägung „lieber ausbessern, weil Neubau zu teuer ist“ verschiebt sich dadurch zugunsten des Neubaus, sofern der neue Bau einen Prüfpfad hat.

Wie läuft eine Überführung in die Produktion ab?

Unabhängig davon, ob ausgebaut oder neu gebaut wird, hat sich eine Reihenfolge bewährt:

1. **Bestandsaufnahme.** Repository einschließlich Historie, Abhängigkeiten, Datenbank, Konten und Datenflüsse erfassen. Zugangsdaten-Scan über die gesamte Historie.
2. **Sofortmaßnahmen.** Gefundene Schlüssel rotieren, öffentlich erreichbare Daten absichern, Zugriffsregeln auf Datenbankebene setzen. Bei einem bereits öffentlich erreichbaren Prototyp gehört dieser Schritt an den Anfang.
3. **Entscheidung Ausbau oder Neubau** nach den Kriterien oben.
4. **Datenmodell festigen.** Schema in versionierte Migrationen überführen, Datenqualität prüfen.
5. **Tests ergänzen.** Zuerst Tests, die das aktuelle Verhalten der Kernabläufe festhalten. Sie sichern jede weitere Änderung ab.
6. **Umgebungen trennen und Deploy-Pfad bauen.** Entwicklung, Test und Produktion getrennt, Build reproduzierbar, Rücknahme einer Version möglich.
7. **Härten.** Statische Analyse, Abhängigkeitsprüfung, Eingabevalidierung, Fehlerbehandlung, Lasttest.
8. **Betrieb einrichten.** Monitoring, Alarmierung, Protokolle, Datensicherung. Die Wiederherstellung einmal tatsächlich testen.
9. **Übergabe.** Dokumentation, Konten und Rechte beim Unternehmen, benannte Verantwortliche für Betrieb und Freigabe.

Ab Schritt 5 gilt für jede Änderung das, was Teil 1 als Kern KI-nativer Entwicklung beschrieben hat: Agenten dürfen den Code schreiben, aber jede Änderung durchläuft Prüfstufen und eine menschliche Freigabe. Wie diese Prüfstufen aufgebaut sind, behandelt [Teil 3 dieser Reihe](#) am 16. Oktober.

Wie sieht das an einem konkreten Prototyp aus?

Ein durchgängiges Beispiel zeigt, wie die neun Schritte ineinandergreifen. Unternehmen, Personen und Details sind konstruiert. Die Befunde entsprechen den Mustern aus den Abschnitten oben, Messwerte enthält das Beispiel nicht.

Die Ausgangslage. Ein mittelständischer Großhändler für Sanitärtechnik möchte seinem Außendienst die Angebotserstellung erleichtern. Die Vertriebsleiterin baut dafür an zwei Wochenenden mit einer Prompt-zu-App-Plattform einen Prototyp: Der Außendienst wählt beim Kundenbesuch Artikel aus, die Anwendung rechnet Mengenrabatte und Sonderkonditionen ein und erzeugt ein Angebot als PDF. Fünf Kolleginnen und Kollegen testen das Werkzeug zwei Monate lang und wollen es nicht mehr hergeben. Nun soll es für den gesamten Außendienst freigegeben und an das Warenwirtschaftssystem angebunden werden.

Schritt 1: Bestandsaufnahme. Der Code lässt sich aus der Plattform exportieren. Die Bestandsaufnahme fördert fünf Befunde zutage. Der Schlüssel für den KI-Dienst, der Angebotstexte formuliert, steht im Code, der an den Browser ausgeliefert wird. In der Versionsgeschichte liegen die Zugangsdaten zum Testsystem der Warenwirtschaft, die einmal für einen Versuch eingetragen und später wieder entfernt wurden. Die Datenbanktabellen wurden direkt in der Plattform angelegt, Migrationen gibt es keine. Die Kundenliste ist eine Kopie aus der Warenwirtschaft, mit echten Namen, Ansprechpartnern und Konditionen. Und das Plattform-Konto läuft auf die private E-Mail-Adresse der Vertriebsleiterin.

Schritt 2: Sofortmaßnahmen. Weil der Prototyp über einen öffentlichen Link erreichbar ist und echte Kundendaten enthält, kommen drei Dinge sofort: Der KI-Schlüssel und die Zugangsdaten zum Testsystem werden gesperrt und neu ausgestellt. Der öffentliche Link wird abgeschaltet, der Zugang läuft ab jetzt über die Anmeldung des Unternehmens. Und auf Datenbankebene kommen Zugriffsregeln, damit jeder Außendienstler nur die Kunden seines Gebiets sieht. Parallel klärt die Datenschutzbeauftragte, ob die bisherige Verarbeitung der Kundendaten eine Meldung erfordert. Das Plattform-Konto wird auf eine Funktionsadresse des Unternehmens umgezogen.

Schritt 3: Ausbauen oder neu bauen. Die Kriterien aus der Tabelle oben ergeben ein gemischtes Bild. Die Oberfläche ist gut und vom Außendienst erprobt, die Abläufe stimmen. Die Preislogik steckt dagegen in der Oberfläche, das Datenmodell ist gewachsen, und die Rechteprüfung fehlt grundsätzlich. Die Entscheidung: Oberfläche und Abläufe werden übernommen, Preislogik, Datenmodell und Rechteprüfung entstehen neu auf der Serverseite. Der Prototyp dient dabei als Spezifikation, an der sich jede Frage nach dem gewünschten Verhalten beantworten lässt.

Schritt 4: Datenmodell festigen. Das neue Datenmodell wird als versionierte Migration angelegt. Kunden und Artikel werden nicht mehr kopiert, die Anwendung verweist auf die Schlüssel der Warenwirtschaft und liest die Daten über deren Schnittstelle. Damit gibt es für Kundenstammdaten nur noch eine Quelle, und die Kopie mit echten Kundendaten kann gelöscht werden.

Schritt 5: Tests ergänzen. Bevor die Preislogik umgezogen wird, hält ein Satz automatisierter Tests fest, welche Preise der Prototyp für eine Reihe typischer Angebote berechnet. Dabei fällt ein Fall auf, in dem ein Sonderrabatt und ein Mengenrabatt doppelt angewendet werden. Im Testbetrieb hatte das niemand bemerkt, weil die Testangebote nie verschickt wurden. Die Vertriebsleiterin entscheidet, welches Verhalten richtig ist, und der Test wird entsprechend angepasst. Ab jetzt läuft er bei jeder Änderung.

Schritt 6: Umgebungen trennen und Deploy-Pfad bauen. Entwicklung, Test und Produktion bekommen eigene Umgebungen. Die Testumgebung spricht mit dem Testsystem der Warenwirtschaft, die Produktion mit dem echten. Jede Änderung wird automatisch gebaut, getestet und erst nach Freigabe in die Produktion übernommen. Eine fehlerhafte Version lässt sich mit einem Schritt zurücknehmen.

Schritt 7: Härten. Die statische Analyse findet eine Stelle, an der der Kundenname ungeprüft in die Angebotsvorschau geschrieben wird. Bei einem präparierten Namen hätte sich dort Code einschleusen lassen. Die Abhängigkeitsprüfung meldet eine veraltete Bibliothek für die PDF-Erzeugung. Ein Lasttest simuliert den Monatsletzten, an dem der gesamte Außendienst gleichzeitig Angebote erstellt, und zeigt eine Artikelsuche, die bei voller Datenmenge zu langsam ist. Alle drei Punkte werden behoben, bevor es weitergeht.

Schritt 8: Betrieb einrichten. Ein Monitoring meldet Fehler und alarmiert die IT, wenn die Schnittstelle zur Warenwirtschaft nicht antwortet. Die Protokolle enthalten keine Kundennamen. Die Datenbank wird täglich gesichert, und die IT stellt einmal tatsächlich einen Stand aus der Sicherung wieder her, um zu sehen, dass es funktioniert.

Schritt 9: Übergabe. Repository, Plattform-Konto und Domain liegen beim Unternehmen. Eine kurze Dokumentation beschreibt Aufbau, Umgebungen und Freigabeprozess. Die Verantwortung ist verteilt und benannt: Die IT-Leitung verantwortet Betrieb und Sicherheit, die Vertriebsleiterin die fachliche Richtigkeit. Jede Änderung braucht die Freigabe einer benannten Person aus der IT.

Das Ergebnis. Vom Prototyp sind Oberfläche, Abläufe und die Erfahrung aus zwei Monaten Testbetrieb geblieben. Neu sind alle Teile, die im Betrieb tragen müssen: Rechte, Datenmodell, Preislogik auf dem Server, Tests, getrennte Umgebungen und Monitoring. Der Prototyp war damit kein Umweg. Er hat die Anforderungen geklärt, schneller und genauer, als es ein Lastenheft gekonnt hätte. Die eigentliche Arbeit lag in den Teilen, die in der Demo unsichtbar waren.

Welche Rolle spielt die Plattform, auf der der Prototyp entstand?

Prompt-zu-App-Plattformen nehmen viel Arbeit ab, setzen aber eigene Voraussetzungen. Drei Fragen sollten vor dem Übergang in den Betrieb geklärt sein:

- **Lässt sich der Code exportieren** und außerhalb der Plattform betreiben? Wenn nicht, ist der Betrieb an die Verfügbarkeit, Preise und Bedingungen eines Anbieters gebunden.
- **Wer ist für die Sicherheit der erzeugten Anwendung verantwortlich?** Im Lovable-Fall hat der Anbieter diese Verantwortung ausdrücklich beim Kunden gesehen. Das ist bei den meisten Plattformen die Ausgangslage und sollte bewusst übernommen werden.

- **Welche Voreinstellungen gelten?** Die RedAccess-Erhebung verweist darauf, dass Anwendungen auf manchen Plattformen standardmäßig öffentlich erreichbar sind. Was nicht ausdrücklich geschützt ist, sollte als öffentlich gelten.

Keine dieser Fragen spricht gegen die Plattformen. Sie verschieben nur den Zeitpunkt, an dem ein Unternehmen Verantwortung übernimmt, und sie sollten vor dem ersten echten Kundendatensatz beantwortet sein.

Was kostet es, die Überführung zu überspringen?

Ein Prototyp, der ohne Überführung in den Betrieb geht, trägt drei Arten von Risiko:

- **Sicherheit.** Offene Zugriffsregeln und eingebettete Schlüssel lassen sich ausnutzen, sobald eine Anwendung öffentlich erreichbar ist. Die Lovable-Funde zeigen, dass das keine theoretische Gefahr ist.
- **Datenschutz.** Werden personenbezogene Daten zugänglich, kann ein meldepflichtiger Datenschutzvorfall vorliegen. Die rechtliche Bewertung im Einzelfall gehört zu Datenschutzbeauftragten oder Rechtsberatung.
- **Abhängigkeit.** Ein Prototyp ohne Tests, Dokumentation und klare Eigentumsverhältnisse ist an die Person oder Plattform gebunden, die ihn gebaut hat. Jede spätere Änderung wird teurer.

Wann ist ein Prototyp als Wegwerf-Beleg richtig?

Nicht jeder Prototyp muss in die Produktion. Für Machbarkeitsnachweise, interne Präsentationen, Nutzertests mit Testdaten und Entscheidungsvorlagen ist ein schnell gebauter Prototyp das richtige Werkzeug, und die zwölf Punkte wären dort Aufwand ohne Nutzen. Wichtig ist nur, dass diese Entscheidung bewusst fällt und der Prototyp als solcher gekennzeichnet bleibt: ohne echte Kundendaten, ohne öffentliche Erreichbarkeit, mit klarem Ablaufdatum.

Die Gefahr liegt im stillen Übergang. Ein Prototyp, der „nur kurz“ mit echten Daten getestet wird und dann einfach weiterläuft, ist im Betrieb, ohne dass jemand die Überführung entschieden hat.

Wie geht es in dieser Reihe weiter?

Dieser Artikel ist Teil 2 der fünfteiligen Themenreihe KI-native Entwicklung. Teil 1 hat geklärt, [was KI-native Softwareentwicklung ist](#) und wie sie sich von KI-gestützter Entwicklung und Vibe Coding unterscheidet. Die nächsten Teile:

- **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 haben einen Prototyp und stehen vor der Frage, ob und wie er in den Betrieb kann? Sprechen Sie mit uns.

Häufige Fragen

Wie bringt man einen KI-Prototyp in Produktion?

Mit einer festen Reihenfolge: Bestandsaufnahme, Sofortmaßnahmen bei Zugangsdaten und Zugriffsregeln, Entscheidung zwischen Ausbau und Neubau, versionierte Migrationen, Tests, getrennte Umgebungen mit Deploy-Pfad, Härtung, Monitoring und eine geregelte Übergabe. Ab dem Ausbau durchläuft jede Änderung Prüfstufen und eine menschliche Freigabe.

Was fehlt einem Vibe-Coding-Prototyp für die Produktion?

Typischerweise zwölf Punkte: Auth und Rechte, Secret-Hygiene, Datenmodell und Migrationen, Eingabevalidierung und Injection-Schutz, Fehlerbehandlung, Tests, Deploy-Pfad, Skalierung und Last, Monitoring und Logging, DSGVO und Datenpfad, Wartbarkeit sowie Übergabe und Eigentum.

Ist KI-generierter Code sicher?

Nicht ohne Prüfung. Im Veracode GenAI Code Security Report 2026 bestand KI-generierter Code im Durchschnitt über mehr als 100 Modelle nur zu 56 Prozent die Sicherheitsprüfung, obwohl er fast immer syntaktisch korrekt war. Größere Modelle schnitten nicht erkennbar besser ab.

Sollte man einen Prototyp ausbauen oder neu bauen?

Das hängt an Datenmodell, Architektur, Technologie, Testbarkeit, Sicherheitsbefund und Umfang. Ein sauberes Datenmodell und eine klare Schichtung sprechen für den Ausbau, eine grundsätzlich fehlende Rechteprüfung und eine nicht exportierbare Plattformbindung eher für den Neubau. Auch beim Neubau bleibt der Prototyp als Spezifikation wertvoll.

Wie lange dauert es, einen Prototyp produktionsreif zu machen?

Eine seriöse Pauschalangabe gibt es nicht. Der Aufwand hängt vor allem davon ab, wie viele der zwölf Punkte offen sind und ob ausgebaut oder neu gebaut wird. Eine Bestandsaufnahme entlang der zwölf Punkte liefert die Grundlage für eine belastbare Schätzung.

Spielt das Werkzeug eine Rolle, mit dem der Prototyp gebaut wurde?

Für die zwölf Punkte kaum, sie gelten für jeden Prototyp. Für die Entscheidung zwischen Ausbau und Neubau schon: Lässt sich der Code exportieren und außerhalb der Plattform betreiben, ist ein Ausbau einfacher.

Was ist Row-Level-Security?

Row-Level-Security sind Regeln auf Datenbankebene, die festlegen, welcher Nutzer welche Zeilen einer Tabelle lesen oder ändern darf. Fehlen sie, kann ein Angreifer Daten direkt über die Datenbankschnittstelle abrufen, auch wenn die Oberfläche der Anwendung sie ausblendet.

Kann man Prototyp-Code weiterverwenden?

Teilweise. Oberfläche, Abläufe und einzelne Module lassen sich oft übernehmen, wenn sie die Kriterien für den Ausbau erfüllen. Datenmodell, Rechteprüfung und Umgang mit Zugangsdaten müssen in jedem Fall geprüft und meist überarbeitet werden.

Was bleibt vom Prototyp übrig, wenn neu gebaut wird?

In der Regel Oberfläche, Abläufe und die Erfahrung aus dem Testbetrieb. Der Prototyp dient als Spezifikation, an der sich jede Frage nach dem gewünschten Verhalten beantworten lässt. Neu entstehen meist Datenmodell, Rechteprüfung, serverseitige Logik, Tests und Betrieb.

Wann braucht ein Prototyp keine Überführung?

Wenn er als Machbarkeitsnachweis, für interne Präsentationen oder für Nutzertests mit Testdaten dient, ohne echte Kundendaten und ohne öffentliche Erreichbarkeit. Wichtig ist, dass er dann auch als Prototyp gekennzeichnet bleibt.

Warum fühlt sich KI-generierter Code sicherer an, als er ist?

In einer Nutzerstudie der Stanford University schrieben Teilnehmende mit KI-Assistent deutlich unsichereren Code und hielten ihn gleichzeitig häufiger für sicher (Perry et al., ACM, 2023). Ein funktionierender Prototyp erzeugt Vertrauen, das keine Prüfung ersetzt.

Thematische Vertiefungen

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

Quellen

- Veracode: „LLMs Are Getting Smarter, But Not Safer: Veracode 2026 GenAI Code Security Report Finds AI-Generated Code Security Has Stalled at 56% Pass Rate“, Pressemitteilung, 28.07.2026. Vier Messzeitpunkte, über 100 Modelle; durchschnittliche Sicherheitsquote 56 %, Syntax-Quote rund 100 %. – <https://www.businesswire.com/news/home/20260728207685/en/>
- Veracode: „AI-Generated Code Poses Major Security Risks in Nearly Half of All Development Tasks“, Pressemitteilung, 30.07.2025. 80 Aufgaben, über 100 Modelle, statische Analyse; 45 % mit Schwachstellen aus den OWASP Top 10. – <https://www.businesswire.com/news/home/20250730694951/en/>
- Perry, N.; Srivastava, M.; Kumar, D.; Boneh, D.: „Do Users Write More Insecure Code with AI Assistants?“, ACM, 2023, DOI 10.1145/3576915.3623157. [ACM](#) – <https://dl.acm.org/doi/10.1145/3576915.3623157>
• [Vorabfassung](#) – <https://arxiv.org/abs/2211.03622>
- National Vulnerability Database: CVE-2025-48757 (Lovable, unzureichende Row-Level-Security), veröffentlicht 29.05.2025. – <https://nvd.nist.gov/vuln/detail/CVE-2025-48757>
- Palmer, Matt: „Statement on CVE-2025-48757“, 29.05.2025. 1.645 geprüfte Projekte, 303 Endpunkte in 170 Projekten. – <https://mattpalmer.io/posts/statement-on-CVE-2025-48757/>

- [The Register: „Vibe coding service Replit deleted user's production database, faked data, told fibs galore“, 21.07.2025.](https://www.theregister.com/2025/07/21/replit_saastr_vibe_coding_incident/) – https://www.theregister.com/2025/07/21/replit_saastr_vibe_coding_incident/
- [Axios, Sam Sabin: „AI vibe-coding apps leak sensitive data“, 07.05.2026 \(Erhebung von RedAccess\).](https://www.axios.com/2026/05/07/loveable-replit-vibe-coding-privacy) – <https://www.axios.com/2026/05/07/loveable-replit-vibe-coding-privacy>
- [GitClear: „AI Copilot Code Quality: 2025 Data Suggests 4x Growth in Code Clones“. 211 Mio. geänderte Zeilen, 2020–2024.](https://www.gitclear.com/ai_assistant_code_quality_2025_research) – https://www.gitclear.com/ai_assistant_code_quality_2025_research
- [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/) – <https://metr.org/blog/2025-07-10-early-2025-ai-experienced-os-dev-study/>
- [OWASP Top 10 \(Webanwendungen\).](https://owasp.org/www-project-top-ten/) – <https://owasp.org/www-project-top-ten/>

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.