

KI-generierter Code: Qualität, Tests und Sicherheit mit Prüfstufen

Von Thomas Faß · 16. Oktober 2026

Wenn KI den Großteil des Codes schreibt, verschiebt sich Qualitätssicherung vom Schreiben zum Prüfen. Tragfähig wird das mit festen Prüfstufen: klare Vorgaben vor dem Bau, automatisierte Tests und Sicherheitsscans bei jeder Änderung, ein menschliches Review jeder Änderung, ein Architektur-Check nach jedem Abschnitt und eine benannte Freigabe vor dem Livegang. Nötig ist das, weil KI-Code fast immer läuft, aber oft nicht sicher ist: Im Veracode GenAI Code Security Report 2026 bestand er im Durchschnitt über mehr als 100 Modelle nur zu 56 Prozent die Sicherheitsprüfung. Und nur 3,1 Prozent der Entwickler haben hohes Vertrauen in die Genauigkeit von KI-Ausgaben (Stack Overflow Developer Survey 2025).

Das Wichtigste in Kürze

- Die Arbeit wandert vom Schreiben zum Prüfen. Wenn Agenten den Großteil des Codes erzeugen, entscheidet die Qualität der Prüfung über die Qualität der Software.
- KI-Code läuft fast immer, sicher ist er seltener. Im Durchschnitt über mehr als 100 Modelle besteht er nur zu 56 Prozent die Sicherheitsprüfung, bei einer Syntax-Quote von rund 100 Prozent (Veracode GenAI Code Security Report, 28.07.2026).
- Schneller ausliefern heißt nicht automatisch stabiler betreiben. KI-Einsatz hängt laut DORA positiv mit dem Auslieferungsdurchsatz zusammen und weiterhin mit höherer Instabilität (DORA, State of AI-assisted Software Development, 23.09.2025, knapp 5.000 Befragte).
- „Fast richtig“ ist der teuerste Fehler. 66 Prozent der Entwickler nennen KI-Lösungen, die fast richtig sind, als größte Frustration, 45,2 Prozent sagen, das Debuggen von KI-Code koste mehr Zeit (Stack Overflow Developer Survey 2025, 31.476 Antworten).
- KI prüft ihren eigenen Code schlecht. In einer Studie erkannten Sprachmodelle Schwachstellen in selbst erzeugtem Code kaum, die Reparatur gelang mit externen Analysewerkzeugen deutlich besser (Gong et al., arXiv, 2025).
- Fünf Prüfstufen tragen den Prozess: Vorgaben vor dem Bau, automatische Prüfungen bei jeder Änderung, menschliches Review, Architektur-Check nach jedem Abschnitt und benannte Freigabe vor dem Livegang.
- Prüfstufen kosten Zeit. Ohne Prüfkompetenz im Team bleiben sie Formalität.

Ist KI-generierter Code schlechter als menschlicher?

Die ehrliche Antwort lautet: Pauschal lässt sich das nicht sagen, und die Frage führt leicht in die falsche Richtung. Die Studien zeigen ein gemischtes Bild, und es hängt davon ab, was man misst.

Beim Funktionieren schneidet KI-Code gut ab. In drei randomisierten Feldexperimenten mit zusammen 4.867 Entwicklern bei Microsoft, Accenture und einem weiteren Großunternehmen erledigten Entwickler mit KI-Code-Vervollständigung 26,08 Prozent mehr Aufgaben. Eine negative Wirkung auf die Erfolgsquote der Builds fanden die Autoren nicht. Gleichzeitig wurde der Code 38,38 Prozent häufiger kompiliert, was die Autoren als Hinweis auf mehr Ausprobieren deuten (Cui et al., Management Science, 27.02.2026).

Bei der Sicherheit schneidet er schwach ab. Veracode hat für den Report 2025 mehr als 100 Sprachmodelle 80 Programmieraufgaben lösen lassen und das Ergebnis mit statischer Analyse geprüft. In 45 Prozent der Fälle enthielt der Code Schwachstellen aus den OWASP Top 10 (Veracode, Pressemitteilung vom 30.07.2025). Ein Jahr später lag die durchschnittliche Sicherheitsquote bei 56 Prozent, praktisch unverändert. Die Syntax-Quote lag bei rund 100 Prozent, und größere Modelle schnitten nicht erkennbar besser ab (Veracode GenAI Code Security Report 2026, 28.07.2026). Veracode verkauft Sicherheitsprüfungen und hat damit ein Interesse an diesem Befund. Getestet wurden reine Modelle ohne Prüfstufen. Genau das ist aber die Lage, in der ungeprüfter KI-Code entsteht.

Bei der Wartbarkeit gibt es Hinweise, aber keinen Beweis. Das Analyseunternehmen GitClear hat 211 Millionen geänderte Codezeilen aus den Jahren 2020 bis 2024 ausgewertet. Der Anteil kopierter Zeilen stieg von 8,3 auf 12,3 Prozent, der Anteil umgebaute Zeilen sank von 25 auf unter 10 Prozent (GitClear, Auswertung 2025). Die Entwicklung fällt mit der Verbreitung von KI-Assistenten zusammen. Ob sie dadurch verursacht ist, zeigt die Auswertung nicht.

Auf Organisationsebene steigt der Durchsatz und mit ihm die Instabilität. Der DORA-Report 2025 beschreibt einen positiven Zusammenhang zwischen KI-Einsatz und Auslieferungsdurchsatz. Der Zusammenhang mit der Instabilität, also mit fehlgeschlagenen Änderungen und Nacharbeit, bleibt negativ. DORA beschreibt KI als Verstärker: In gut organisierten Teams verstärkt sie die Stärken, in zersplitterten die Schwächen (DORA/Google Cloud, 23.09.2025).

Für die Praxis ist die Frage „besser oder schlechter als ein Mensch“ deshalb weniger hilfreich als eine andere: Unter welchen Bedingungen wird KI-Code zuverlässig gut? Die Antwort der Studienlage ist ziemlich einheitlich. Dort, wo geprüft wird. Wie sich der Produktivitätsgewinn über Aufgabentypen und Erfahrungsstufen verteilt, wertet der Fachartikel [KI-Produktivität in der Softwareentwicklung](#) im Detail aus. Seine Kernaussage gilt auch hier: Der Gewinn entsteht beim Bauen, und die Prüfung holt sich einen Teil davon zurück.

Das Vertrauensproblem

Die Entwickler selbst sehen das ähnlich. In der Stack Overflow Developer Survey 2025 gaben nur 3,1 Prozent an, der Genauigkeit von KI-Ausgaben in hohem Maß zu vertrauen. 19,6 Prozent misstrauen ihr in hohem Maß (33.244 Antworten). Wenn sie einer KI-Antwort nicht trauen, würden 75,3 Prozent noch einen Menschen fragen (29.163 Antworten).

Die größte Frustration nennt 66 Prozent der Befragten beim Namen: KI-Lösungen, die fast richtig sind, aber eben nur fast. 45,2 Prozent sagen, das Debuggen von KI-generiertem Code dauere länger (31.476 Antworten). „Fast richtig“ ist tückischer als offensichtlich falsch. Ein offensichtlicher Fehler fällt beim ersten Test auf. Ein fast richtiger Code besteht die Demo und scheitert am Sonderfall, am Monatsletzten oder am ersten Angreifer.

Wo entstehen die typischen Fehler?

Teil 2 dieser Reihe hat beschrieben, welche zwölf Punkte einem schnell gebauten Prototyp typischerweise fehlen (siehe [Vom KI-Prototyp zur Produktion](#)). Im laufenden KI-nativen Prozess verschiebt sich die Frage. Es geht weniger um Lücken eines einzelnen Prototyps und mehr um Fehlerquellen, die bei jeder Änderung wieder auftreten können. Sechs Muster fallen dabei besonders auf.

1. Eingaben und Ausgaben ohne Schutz. Die häufigsten Schwachstellen in der Veracode-Messung betreffen den Umgang mit Eingaben. Gegen Cross-Site-Scripting war der erzeugte Code in 86 Prozent der Fälle nicht abgesichert, gegen Log-Injection in 88 Prozent. Bei Java lag die Fehlerquote insgesamt über 70 Prozent, bei Python, C# und JavaScript zwischen 38 und 45 Prozent (Veracode, 30.07.2025). Solche Fehler entstehen, wenn die Aufgabe „Zeige den Kundennamen an“ lautet und niemand dazusagt, dass der Name von außen kommt.

2. Erfundene Abhängigkeiten. Sprachmodelle schlagen gelegentlich Software-Pakete vor, die es nicht gibt. In einer Studie mit 576.000 erzeugten Codebeispielen lag der Anteil erfundener Pakete im Durchschnitt bei mindestens 5,2 Prozent für kommerzielle Modelle und bei 21,7 Prozent für offene Modelle, insgesamt fanden sich 205.474 verschiedene erfundene Namen (Spracklen et al., USENIX Security Symposium 2025). Das ist mehr als ein Ärgernis. Wer einen solchen Namen kennt, kann ein gleichnamiges Paket mit Schadcode veröffentlichen, und der nächste Build installiert es.

3. Rechteprüfung an der falschen Stelle. KI-Agenten bauen Oberflächen schnell und überzeugend. Ob die Schnittstelle dahinter dieselben Regeln durchsetzt, prüft die Oberfläche nicht. Das Muster aus Teil 2, fremde Daten über eine geänderte Kennung abzurufen, entsteht auch in einem reifen Projekt bei jeder neuen Schnittstelle neu.

4. Tests, die nur den vorhandenen Code bestätigen. Wer einen Agenten bittet, Tests für den gerade erzeugten Code zu schreiben, bekommt Tests, die genau dieses Verhalten festschreiben, Fehler eingeschlossen. Die Testabdeckung steigt, die Aussagekraft bleibt gering. Brauchbar werden Tests erst, wenn sie aus der Anforderung abgeleitet sind und Grenzfälle enthalten, die im Code noch gar nicht vorkommen.

5. Kopierte Logik. Ein Agent löst eine Aufgabe oft neu. Die passende Funktion gibt es vielleicht längst, nur lag sie nicht in seinem Kontext. Die GitClear-Daten passen zu diesem Muster. Die Folge zeigt sich später: Eine Änderung der Geschäftsregel muss an drei Stellen erfolgen, und eine davon wird vergessen.

6. Stilles Abweichen von der Architektur. Ein Agent folgt dem Auftrag. Ob die Lösung zur Architektur passt, prüft er nur, wenn man es ihm vorgibt. Ohne ausdrückliche Vorgaben greift er direkt auf die Datenbank zu, wo eine Serviceschicht vorgesehen war, oder führt eine neue Bibliothek ein, wo es schon

eine gibt. Jede einzelne Abweichung ist klein. In Summe entsteht über Wochen ein System, das niemand so entworfen hat.

Allen sechs Mustern ist gemeinsam: Der Code läuft. Keines der Muster fällt in einer Demo auf, und mehrere davon bestehen sogar die eigenen Tests.

Welche Prüfstufen braucht ein KI-nativer Prozess?

Ein KI-nativer Prozess legt fest, an welchen Stellen eine Änderung angehalten und geprüft wird, bevor sie weiter darf. Diese Haltepunkte heißen in der Praxis oft Gates. Das folgende Modell beschreibt fünf Stufen als allgemeines Referenzmodell. Teams setzen es unterschiedlich um, die Reihenfolge und die Zuständigkeiten sind aber übertragbar.

Stufe 1: Vorgaben vor dem Bau. Bevor ein Agent Code schreibt, steht fest, was gebaut wird und nach welchen Regeln. Dazu gehören die Anforderung mit Akzeptanzkriterien, Architektur-Leitplanken (Schichten, erlaubte Bibliotheken, Muster für Datenzugriff und Rechteprüfung), eine Definition, wann eine Aufgabe als fertig gilt, und die Einordnung, welche Daten das Werkzeug sehen darf. Diese Vorgaben liegen im Repository, wo der Agent sie bei jeder Aufgabe mitliest. Sie verhindern einen großen Teil der Abweichungen aus Muster 6, bevor sie entstehen.

Stufe 2: Automatische Prüfungen bei jeder Änderung. Jede Änderung durchläuft eine Pipeline, bevor ein Mensch sie ansieht: Build, Formatierung und Stilregeln, automatisierte Tests mit gemessener Abdeckung, statische Sicherheitsanalyse, Suche nach Zugangsdaten im Code und Prüfung der Abhängigkeiten auf bekannte Schwachstellen und auf Existenz. Was hier scheitert, kommt gar nicht erst zum Review. Das spart die teuerste Ressource im Prozess, die Aufmerksamkeit erfahrener Menschen.

Stufe 3: Menschliches Review jeder Änderung. Eine fachkundige Person liest jede Änderung, bevor sie in den Hauptstand übernommen wird. Ihr Fokus liegt dort, wo Werkzeuge schwach sind: Stimmt die Logik mit der Anforderung überein? Wird die Rechteprüfung an der richtigen Stelle durchgesetzt? Welche Daten fließen wohin? Passen die Tests zur Anforderung oder nur zum Code? Hilfreich ist eine feste Größe für Änderungen. Kleine Änderungen lassen sich gründlich prüfen, große werden überflogen. DORA zählt das Arbeiten in kleinen Paketen zu den sieben Fähigkeiten, die den Nutzen von KI in der Entwicklung verstärken (DORA AI Capabilities Model, 2025).

Stufe 4: Architektur-Check nach jedem Abschnitt. Die Arbeit wird in Abschnitte gegliedert, etwa ein Modul oder einen fachlichen Ablauf. Nach jedem Abschnitt prüft eine erfahrene Person das Ganze: Passt der Abschnitt zur Architektur? Sind Muster und Strukturen einheitlich geblieben? Laufen Integrationstests, dynamische Sicherheitstests und, wo nötig, ein Lasttest? Der Vorteil gegenüber einer einzigen großen Prüfung am Projektende ist praktisch. Ein Befund nach dem zweiten von acht Abschnitten betrifft zwei Abschnitte. Derselbe Befund am Ende betrifft acht, und jede Korrektur zieht weitere nach sich.

Stufe 5: Benannte Freigabe vor dem Livegang. Bevor eine Version in Produktion geht, gibt eine namentlich benannte Person sie frei. Sie bestätigt, dass die vorherigen Stufen durchlaufen sind und offene Befunde bewertet wurden. Zur Freigabe gehört ein Rückweg: Eine fehlerhafte Version muss sich mit einem Schritt zurücknehmen lassen.

Die Stufen 1 bis 3 laufen bei jeder Änderung, Stufe 4 nach jedem Abschnitt, Stufe 5 vor jeder Auslieferung. Nach dem Livegang schließt der Betrieb den Kreis: Monitoring und die Kennzahlen aus dem Abschnitt unten zeigen, ob die Stufen wirken.

Warum gerade diese Verteilung?

Das Modell verteilt die Arbeit nach Stärken. Werkzeuge sind gut bei Mustern, die sich als Regel beschreiben lassen, etwa ungeschützte Eingaben, eingebettete Schlüssel oder veraltete Bibliotheken. Sie sind schwach bei der Frage, ob eine Funktion das Richtige tut. Menschen sind dort stark und werden bei Routineprüfungen müde. Ein Prozess, der die Routine an Werkzeuge gibt, hält die menschliche Aufmerksamkeit für die Fragen frei, bei denen sie gebraucht wird.

Die deutsch-französische Empfehlung von BSI und ANSSI zu KI-Programmierassistenten setzt an diesem Punkt an. KI-Assistenten seien kein Ersatz für erfahrene Entwickler, heißt es dort, und erzeugter Code solle grundsätzlich von den Entwicklern geprüft und nachvollzogen werden. Als ergänzende Maßnahmen nennen die Behörden automatisierte Funktionstests, zusätzliche Reviews durch Sicherheitsteams, automatische Schwachstellenscanner, eine Liste zugelassener Pakete und eine Software-Stückliste (SBOM). Vor der Einführung soll eine systematische Risikoanalyse stehen (BSI und ANSSI, „AI Coding Assistants“, 04.10.2024).

Welche Rolle spielen Tests, statische Analyse und Security-Scans?

Die automatischen Prüfungen in Stufe 2 und 4 lassen sich in Werkzeugkategorien ordnen. Für jede gibt es kommerzielle und offene Werkzeuge. Die Beispiele unten sind eine Auswahl ohne Wertung.

Automatisierte Tests mit Abdeckungsmessung. Unit-Tests prüfen einzelne Funktionen, Integrationstests das Zusammenspiel. Die Abdeckungsmessung zeigt, welcher Code bei den Tests überhaupt ausgeführt wurde. Werkzeuge dafür gibt es für jede Sprache, etwa coverage.py für Python, Istanbul für JavaScript oder JaCoCo für Java. Wichtig ist die Grenze der Kennzahl: Abdeckung misst, ob Code ausgeführt wurde, und sagt nichts darüber, ob ein Test das Ergebnis sinnvoll prüft. Mutationstests ergänzen das, indem sie den Code gezielt verändern und schauen, ob die Tests den Fehler bemerken.

Statische Sicherheitsanalyse (SAST). Sie untersucht den Quellcode, ohne ihn auszuführen, und findet Muster wie ungeschützte Eingaben oder unsichere Funktionen. Beispiele sind CodeQL, Semgrep und SonarQube. Für KI-generierten Code ist diese Kategorie besonders wertvoll, weil sie genau die Fehlerklassen findet, die Veracode am häufigsten gemessen hat.

Prüfung der Abhängigkeiten (SCA). Sie gleicht die eingebundenen Bibliotheken mit Datenbanken bekannter Schwachstellen ab und meldet veraltete Versionen. Beispiele sind Dependabot, OWASP Dependency-Check, Renovate und Snyk. Für KI-Code kommt eine zweite Frage dazu: Gibt es das Paket überhaupt, und stammt es aus der erwarteten Quelle? Eine Liste freigegebener Pakete oder ein interner Paketspiegel beantwortet sie zuverlässig.

Suche nach Zugangsdaten. Sie durchsucht Code und Versionsgeschichte nach Schlüsseln, Passwörtern und Tokens. Beispiele sind Gitleaks, TruffleHog und die eingebauten Funktionen großer Code-Plattformen. Sie

gehört in die Pipeline und idealerweise schon auf den Rechner, bevor eine Änderung des Repository erreicht.

Dynamische Sicherheitstests (DAST). Sie greifen die laufende Anwendung von außen an, wie es ein Angreifer täte. Beispiele sind Burp Suite und ZAP. Weil sie eine lauffähige Umgebung brauchen, gehören sie eher in Stufe 4 als in jede einzelne Änderung.

Ein Maßstab für die Prüfung. Damit Prüfungen vergleichbar werden, braucht es einen Katalog, gegen den geprüft wird. Der OWASP Application Security Verification Standard (ASVS) beschreibt Sicherheitsanforderungen an Webanwendungen in drei Stufen, von einer Basis für alle Anwendungen bis zu hohen Anforderungen für besonders schutzbedürftige Systeme. Die aktuelle Version 5.0 erschien 2025. Ein Team legt fest, welche Stufe für welches System gilt, und prüft in Stufe 4 dagegen. Unsere [Checkliste zur Produktionsreife](#) enthält die Punkte zu Tests, Eingabevalidierung und Zugangsdaten in kompakter Form.

Keine dieser Kategorien ersetzt eine andere. Statische Analyse findet keine Logikfehler, Tests finden keine Schwachstellen in Abhängigkeiten, und keines der Werkzeuge beantwortet die Frage, ob die Software das tut, was das Unternehmen braucht.

Kann KI ihren eigenen Code prüfen?

Die Versuchung liegt nahe: Wenn ein Modell Code schreibt, kann ein Modell ihn auch prüfen. Teilweise stimmt das. KI-gestützte Reviews finden Tippfehler, fehlende Fehlerbehandlung und manche Sicherheitsmuster zuverlässig und schnell.

Für die Prüfung des eigenen Codes gibt es aber einen deutlichen Befund. In einer Studie mit 4.900 geprüften Codebeispielen konnten GPT-3.5 und GPT-4 Schwachstellen in selbst erzeugtem Code nicht präzise erkennen. Unsicheren Code anderer Modelle reparierten sie in 33,2 bis 59,6 Prozent der Fälle, bei eigenem Code schnitten sie schlechter ab. Die Autoren sprechen von blinden Flecken. Mit einem Werkzeug, das die Reparatur durch externe Analyse-Engines unterstützt, stieg die Erfolgsquote auf 65,9 bis 85,5 Prozent (Gong et al., arXiv:2408.10495, Fassung vom 13.06.2025).

Zwei Einschränkungen gehören dazu. Die Studie nutzt Modelle von 2023 und 2024, neuere Modelle können besser abschneiden. Und sie ist bisher als Vorabveröffentlichung ohne Begutachtung erschienen. Die Richtung ist trotzdem plausibel und deckt sich mit dem Prinzip, das in jeder Qualitätssicherung gilt: Wer etwas erstellt hat, prüft es schlechter als jemand mit Abstand.

Für die Praxis ergeben sich drei Regeln. KI-Review ist eine sinnvolle Ergänzung in Stufe 2. Es sollte möglichst ein anderes Modell oder eine andere Konfiguration sein als die, die den Code erzeugt hat. Und es ersetzt weder die regelbasierten Werkzeuge noch das menschliche Review in Stufe 3.

Was bleibt zwingend menschliche Verantwortung?

Agenten können viel übernehmen. Einige Aufgaben bleiben trotzdem bei Menschen, weil sie Urteil, Kontext oder Verantwortung verlangen, die ein Werkzeug nicht tragen kann.

- **Anforderungen.** Was die Software tun soll, wer sie nutzt und welche Fälle sie abdecken muss, entscheidet das Unternehmen. Ein Agent kann Fragen stellen und Lücken aufzeigen. Die Antworten muss jemand geben, der das Geschäft kennt.
- **Architektur.** Schichten, Datenmodell, Schnittstellen und die Auswahl der Technologie prägen ein System über Jahre. Diese Entscheidungen gehören zu erfahrenen Menschen, die auch die Folgen für Betrieb und Wartung überblicken.
- **Rechte und Datenflüsse.** Wer welche Daten sehen darf und welche Daten welches Werkzeug sehen darf, ist eine fachliche und rechtliche Frage. Teil 4 dieser Reihe am 23. Oktober behandelt die Architekturfragen dahinter.
- **Die Freigabe.** Eine Version geht live, weil eine benannte Person das entschieden hat. Diese Entscheidung lässt sich vorbereiten und begründen, aber nicht an ein Werkzeug abgeben.
- **Die Verantwortung für Fehler.** Ein Werkzeug kann Code erzeugen, aber keine Verantwortung übernehmen. Im Unternehmen braucht jede Freigabe eine Person, die für sie einsteht.

Der Bundestag hat am 8. Oktober 2026 ein neues Produkthaftungsrecht beschlossen, das die EU-Richtlinie 2024/2853 umsetzt. Es erfasst künftig ausdrücklich auch Software und soll am 9. Dezember 2026 in Kraft treten (IHK Frankfurt am Main, Meldung „Modernisierung des Produkthaftungsrechts beschlossen“). Wer im Einzelfall haftet, ist eine Frage für die Rechtsberatung. Für den Prozess heißt es: Dokumentierte Prüfungen und Freigaben machen nachvollziehbar, wie eine Version entstanden ist.

Wie misst man, ob der Prozess funktioniert?

Prüfstufen kosten Zeit. Ob sich die Zeit lohnt, lässt sich messen, und das sollte man auch tun. Ein Gefühl reicht nicht, die METR-Studie von 2025 hat gezeigt, wie weit Selbsteinschätzung und Messung auseinanderliegen können: Die beteiligten Entwickler hielten sich für 20 Prozent schneller und waren gemessen 19 Prozent langsamer (METR, 10.07.2025). In einer Folgemessung ab August 2025 sieht METR inzwischen eher eine Beschleunigung, stuft die Daten aber selbst als unzuverlässiges Signal ein (METR, 24.02.2026).

Einen etablierten Rahmen für die Messung liefert DORA mit fünf Kennzahlen zur Auslieferungsleistung, aufgeteilt in Durchsatz und Instabilität:

- **Durchlaufzeit einer Änderung:** wie lange es dauert, bis eine Änderung von der Versionsverwaltung in der Produktion läuft.
- **Auslieferungshäufigkeit:** wie oft ausgeliefert wird.
- **Wiederherstellungszeit nach einer fehlgeschlagenen Auslieferung:** wie lange es dauert, einen Fehler im Betrieb zu beheben.
- **Fehlerquote bei Änderungen:** der Anteil der Auslieferungen, die sofort nachgebessert oder zurückgenommen werden müssen.
- **Nacharbeitsquote:** der Anteil ungeplanter Auslieferungen, die ein Vorfall im Betrieb ausgelöst hat.

(DORA, Leitfaden zu den Kennzahlen der Auslieferungsleistung)

Für einen KI-nativen Prozess sind die beiden Instabilitätswerte die wichtigsten. Genau dort zeigt der DORA-Report 2025 die Schwäche des KI-Einsatzes. Steigt der Durchsatz und bleiben Fehlerquote und Nacharbeit stabil, wirken die Prüfstufen. Steigen beide, fehlt eine Stufe oder sie wird nur formal durchlaufen.

Drei weitere Größen helfen bei der Steuerung im Team:

- **Befunde je Prüfstufe.** Wie viele Probleme findet jede Stufe? Eine Stufe, die nie etwas findet, ist entweder überflüssig oder wird nicht ernst genommen.
- **Fehler nach Release.** Wie viele Fehler meldet der Betrieb oder melden Nutzer nach einer Auslieferung, und in welcher Stufe hätten sie auffallen müssen?
- **Wartezeit im Review.** Wie lange liegt eine Änderung, bis ein Mensch sie prüft? Wächst diese Zeit, ist das Review der Engpass, und der Gewinn beim Bauen verpufft im Stau.

Für alle Kennzahlen gilt der Rat aus Teil 1: vor der Umstellung eine Ausgangsgröße erheben und den Effekt später dagegen rechnen.

Wie sieht das an einer konkreten Änderung aus?

Teil 2 hat an einem konstruierten Beispiel gezeigt, wie ein Großhändler für Sanitärtechnik den Prototyp seiner Angebots-App in den Betrieb überführt. Das Beispiel geht hier weiter. Unternehmen, Personen und Details sind konstruiert, Messwerte enthält das Beispiel nicht.

Die Anforderung. Die App läuft seit einigen Wochen im Außendienst. Nun soll eine Funktion dazukommen: Rabatte über einer bestimmten Schwelle muss die Vertriebsleitung freigeben, bevor das Angebot an den Kunden geht. Die Vertriebsleiterin beschreibt den Ablauf, die Schwellen und die Fälle, die vorkommen können, darunter Angebote mit mehreren Positionen, Rabatte, die erst durch eine Mengenänderung die Schwelle überschreiten, und Vertretungen im Urlaub.

Stufe 1: Vorgaben. Die verantwortliche Entwicklerin leitet daraus Akzeptanzkriterien ab und ergänzt die Architektur-Leitplanken im Repository: Die Freigabelogik gehört in den Server, die Prüfung der Berechtigung in dieselbe Schicht wie alle anderen Rechteprüfungen. Neue Bibliotheken braucht die Funktion nicht.

Stufe 2: Automatische Prüfungen. Der Agent baut die Funktion in drei kleinen Änderungen. In der zweiten schlägt die Abhängigkeitsprüfung an: Der Agent hat für den E-Mail-Versand der Freigabeanfrage ein Paket eingebunden, das im öffentlichen Verzeichnis nicht existiert. Die Pipeline stoppt, bevor ein Mensch die Änderung sieht. Der Agent nutzt im zweiten Anlauf die E-Mail-Funktion, die die App bereits hat.

Stufe 3: Review. Die Entwicklerin liest die dritte Änderung und findet zwei Dinge. Die Oberfläche blendet den Freigabe-Knopf für Außendienstler aus, die Schnittstelle dahinter prüft die Rolle aber nicht. Ein Außendienstler hätte seinen eigenen Rabatt über die Schnittstelle freigeben können. Und die Tests, die der Agent geschrieben hat, prüfen nur Angebote mit einer Position. Der Fall, in dem erst die Mengenänderung die Schwelle überschreitet, fehlt. Beides geht mit einem klaren Hinweis zurück an den Agenten und kommt danach erneut durch Stufe 2 und 3.

Stufe 4: Abschnitts-Check. Nach Abschluss der Funktion prüft die IT-Leitung den Abschnitt im Zusammenhang. Ein Integrationstest spielt den kompletten Ablauf mit dem Testsystem der Warenwirtschaft durch, ein dynamischer Sicherheitstest versucht, die Freigabe ohne Berechtigung auszulösen. Beides besteht. Ein Befund bleibt: Die Vertretungsregel ist fachlich noch nicht entschieden. Sie wird als offener Punkt dokumentiert und vorerst so gelöst, dass nur die Vertriebsleiterin selbst freigeben kann.

Stufe 5: Freigabe. Die IT-Leitung gibt die Version frei. Die Vertriebsleiterin bestätigt fachlich, dass die Funktion tut, was sie beschrieben hat. Die vorherige Version bleibt mit einem Schritt wiederherstellbar.

Das Ergebnis. Der Agent hat den Großteil des Codes geschrieben. Die Prüfstufen haben drei Fehler gefunden, bevor ein Kunde ein Angebot mit falscher Freigabe bekommen hat: ein erfundenes Paket, eine Rechteprüfung nur in der Oberfläche und eine Testlücke beim wichtigsten Sonderfall. Zwei davon hätte die Demo bestanden.

Wann werden Prüfstufen zur Formalität?

Prüfstufen kosten Zeit, und sie wirken nur unter Bedingungen. Drei Warnzeichen deuten darauf hin, dass ein Prozess sie nur noch formal durchläuft.

Es fehlt die Prüfkompetenz. Ein Review durch jemanden, der die Architektur nicht kennt oder Sicherheitsmuster nicht erkennt, ist eine Unterschrift. BSI und ANSSI halten genau das fest: KI-Assistenten ersetzen keine erfahrenen Entwickler, und ihr ungebremster Einsatz kann ernste Sicherheitsfolgen haben. Wer die Stufen einführt, muss auch klären, wer sie mit welcher Erfahrung besetzt.

Die Änderungen sind zu groß. Wenn ein Agent an einem Tag ein ganzes Modul erzeugt und ein Mensch es in einer Stunde prüfen soll, wird überflogen. Kleine Pakete sind deshalb eine Voraussetzung für den Prozess.

Die Stufen finden nichts mehr. Ein Prozess, in dem Review und Abschnitts-Check über Wochen keine Befunde liefern, während der Betrieb Fehler meldet, prüft an der falschen Stelle. Die Kennzahlen aus dem Abschnitt oben machen das sichtbar.

Die Zeit für die Prüfung muss ein Unternehmen bewusst einplanen. Ein Teil des Geschwindigkeitsgewinns beim Bauen fließt in die Prüfung zurück. BSI und ANSSI formulieren das als Grundsatz: Wenn Entwicklungsteams mit KI produktiver werden, muss die Qualitätssicherung mitwachsen, durch eigene KI-Unterstützung oder durch mehr Personal. Wer das nicht einplant, spart an der Prüfung und verschiebt die Kosten in den Betrieb.

Wie geht es in dieser Reihe weiter?

Dieser Artikel ist Teil 3 der fünfteiligen Themenreihe KI-native Entwicklung. Teil 1 hat geklärt, [was KI-native Softwareentwicklung ist](#), Teil 2, [was zwischen einem KI-Prototyp und dem Betrieb liegt](#). Die nächsten Teile:

- **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 wollen KI stärker in Ihre Entwicklung holen und fragen sich, wie die Prüfung dazu aussehen muss? Sprechen Sie mit uns.

Häufige Fragen

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. Mit statischer Analyse, Prüfung der Abhängigkeiten und menschlichem Review lassen sich die häufigsten Schwachstellen vor dem Livegang finden.

Ist KI-generierter Code schlechter als menschlicher Code?

Pauschal nicht. In Feldexperimenten mit 4.867 Entwicklern stieg die Zahl erledigter Aufgaben mit KI-Code-Vervollständigung um 26,08 Prozent, ohne negative Wirkung auf die Erfolgsquote der Builds (Cui et al., 2026). Bei der Sicherheit schneidet ungeprüfter KI-Code dagegen schwach ab. Entscheidend ist, ob und wie geprüft wird.

Wie funktioniert Qualitätssicherung, wenn KI den Code schreibt?

Mit festen Prüfstufen: Vorgaben vor dem Bau, automatische Prüfungen bei jeder Änderung (Tests, statische Analyse, Suche nach Zugangsdaten, Prüfung der Abhängigkeiten), menschliches Review jeder Änderung, ein Architektur-Check nach jedem Abschnitt und eine benannte Freigabe vor dem Livegang.

Kann KI ihren eigenen Code prüfen?

Nur eingeschränkt. In einer Studie erkannten GPT-3.5 und GPT-4 Schwachstellen in selbst erzeugtem Code nicht präzise und reparierten eigenen Code schlechter als fremden. Mit externen Analysewerkzeugen stieg die Reparaturquote deutlich (Gong et al., 2025). KI-Review ist eine sinnvolle Ergänzung, ersetzt aber weder regelbasierte Werkzeuge noch das menschliche Review.

Wie hoch sollte die Testabdeckung sein?

Eine allgemeingültige Zahl gibt es nicht. Viele Teams legen eine Untergrenze für neuen Code fest und messen sie bei jeder Änderung. Wichtiger als die Zahl ist, dass die Tests aus der Anforderung abgeleitet sind und Grenzfälle enthalten. Tests, die ein Agent nachträglich zum eigenen Code schreibt, bestätigen oft nur das vorhandene Verhalten. Mutationstests zeigen, ob die Tests Fehler tatsächlich bemerken.

Welche Werkzeuge braucht man für die Prüfung von KI-Code?

Fünf Kategorien: automatisierte Tests mit Abdeckungsmessung, statische Sicherheitsanalyse (SAST), Prüfung der Abhängigkeiten (SCA), Suche nach Zugangsdaten und dynamische Sicherheitstests (DAST). Für jede gibt es kommerzielle und offene Werkzeuge. Als Prüfmaßstab eignet sich der OWASP Application Security Verification Standard.

Was sind erfundene Pakete, und warum sind sie gefährlich?

Sprachmodelle schlagen manchmal Software-Pakete vor, die es nicht gibt. Im Durchschnitt waren mindestens 5,2 Prozent der vorgeschlagenen Pakete kommerzieller Modelle erfunden, bei offenen Modellen 21,7 Prozent (Spracklen et al., 2025). Angreifer können unter solchen Namen Pakete mit Schadcode veröffentlichen. Eine Liste freigegebener Pakete oder ein interner Paketspiegel schützt davor.

Wer haftet für Fehler in KI-generiertem Code?

Das hängt vom Einzelfall ab und gehört in die Rechtsberatung. Das neue Produkthaftungsrecht, das der Bundestag am 8. Oktober 2026 beschlossen hat, erfasst ausdrücklich auch Software und soll am 9. Dezember 2026 in Kraft treten. Organisatorisch gilt: Ein Werkzeug übernimmt keine Verantwortung. Jede Freigabe braucht eine Person im Unternehmen, die für sie einsteht.

Was muss in einem KI-nativen Prozess der Mensch entscheiden?

Anforderungen, Architektur, Rechte und Datenflüsse sowie die Freigabe vor dem Livegang. Diese Entscheidungen verlangen Kontext und Verantwortung, die sich vorbereiten, aber nicht an ein Werkzeug abgeben lassen.

Woran erkennt man, ob die Prüfstufen wirken?

An den DORA-Kennzahlen, vor allem an Fehlerquote bei Änderungen und Nacharbeitsquote. Steigt der Durchsatz und bleiben diese beiden Werte stabil, wirken die Prüfstufen. Ergänzend helfen Befunde je Prüfstufe, Fehler nach Release und die Wartezeit im Review.

Warum ist eine Prüfung nach jedem Abschnitt besser als eine große Prüfung am Ende?

Weil ein Befund früh weniger Arbeit betrifft. Wird ein Architekturproblem nach dem zweiten von acht Abschnitten gefunden, sind zwei Abschnitte betroffen, am Ende alle acht. Jede späte Korrektur zieht weitere Änderungen nach sich.

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. – <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. – <https://www.businesswire.com/news/home/20250730694951/en/>
- Cui, Demirer, Jaffe, Musolff, Peng, Salz: „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>

- [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) – <https://cloud.google.com/blog/products/ai-machine-learning/announcing-the-2025-dora-report> • [Zusammenfassung](#) – <https://blog.google/technology/developers/dora-report-2025/>
- [DORA: AI Capabilities Model, 2025.](https://dora.dev/ai/capabilities-model/) – <https://dora.dev/ai/capabilities-model/>
- [DORA: Leitfaden zu den Kennzahlen der Auslieferungsleistung.](https://dora.dev/guides/dora-metrics/) – <https://dora.dev/guides/dora-metrics/>
- [Stack Overflow Developer Survey 2025, Abschnitt KI.](https://survey.stackoverflow.co/2025/ai) – <https://survey.stackoverflow.co/2025/ai>
- [Gong, J. et al.: „How Well Do Large Language Models Serve as End-to-End Secure Code Agents for Python?“, arXiv:2408.10495, Fassung vom 13.06.2025.](https://arxiv.org/abs/2408.10495) – <https://arxiv.org/abs/2408.10495>
- [Spracklen, J. et al.: „We Have a Package for You! A Comprehensive Analysis of Package Hallucinations by Code Generating LLMs“, USENIX Security Symposium 2025, arXiv:2406.10279.](https://arxiv.org/abs/2406.10279) – <https://arxiv.org/abs/2406.10279>
- [GitClear: „AI Copilot Code Quality: 2025 Data Suggests 4x Growth in Code Clones“.](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/>
- [METR: „We are Changing our Developer Productivity Experiment Design“, 24.02.2026.](https://metr.org/blog/2026-02-24-uplift-update/) – <https://metr.org/blog/2026-02-24-uplift-update/>
- [BSI und ANSSI: „AI Coding Assistants“, 04.10.2024.](https://www.bsi.bund.de/SharedDocs/Downloads/EN/BSI/KI/ANSSI_BSI_AI_Coding_Assistants.pdf) – https://www.bsi.bund.de/SharedDocs/Downloads/EN/BSI/KI/ANSSI_BSI_AI_Coding_Assistants.pdf
- [OWASP: Application Security Verification Standard \(ASVS\), Version 5.0.](https://owasp.org/www-project-ASVS/) – <https://owasp.org/www-project-ASVS/>
- [IHK Frankfurt am Main: „Modernisierung des Produkthaftungsrechts beschlossen“.](https://www.frankfurt-main.ihk.de/recht/recht-und-steuern-aktuell/modernisierung-des-produkthaftungsrechts-beschlossen-7199032) – <https://www.frankfurt-main.ihk.de/recht/recht-und-steuern-aktuell/modernisierung-des-produkthaftungsrechts-beschlossen-7199032>

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.