W H I T E P A P E R

Ein Prozess zur KI-gestützten Softwareentwicklung

Werte, Rollen, Takte und Praxis

Manfred Wolff

23. Mai 2026

Inhaltsverzeichnis

1. Vorwort und Motivation

Die Softwareentwicklung hat in den letzten Jahren eine Verschiebung erlebt, die sich nicht in der Sprache klassischer agiler Methoden beschreiben lässt. Generative Sprachmodelle und agenten-basierte Werkzeuge haben die Frage, wer im Team eigentlich Code schreibt, neu aufgeworfen. Eine entwickelnde Person, die mit einem solchen Werkzeug arbeitet, ist nicht mehr nur Autor:in ihres Codes; sie ist gleichzeitig erste Review-Instanz, verantwortet die Architektur und trägt die Endverantwortung. Das ist keine kleine Anpassung des Tagesgeschäfts, sondern eine andere Art zu arbeiten.

Scrum, wie es 2001 als Antwort auf den Wasserfallprozess entstand, hat darauf keine direkte Antwort. Seine Ceremonies setzen ein vollständig menschliches Team voraus. Seine Sprints sind in Wochen gerechnet, nicht in Stunden. Sein Backlog ist statisch im Vergleich zu dem, was ein Sprachmodell innerhalb eines Vormittags an Vorschlägen produzieren kann. Und vor allem fehlt Scrum das Konzept einer Persistenz zwischen Sessions: die Annahme, dass ein Teammitglied morgens weiß, was gestern gesagt wurde, gilt für die KI nicht ohne Weiteres.

Trotzdem wäre es ein Fehler, das Agile Manifest pauschal für überholt zu erklären. Vieles bleibt richtig. Individuen und Interaktionen über Prozesse und Werkzeuge, gerade weil das Werkzeug jetzt eine eigene Stimme hat. Funktionierende Software über umfassende Dokumentation, gerade weil Dokumentation leicht generiert wird. Kundennähe über Vertragsverhandlung, gerade weil schneller Output die Versuchung erhöht, an der Kundschaft vorbeizuliefern. Und Reaktion auf Veränderung über das Befolgen eines Plans, gerade weil die Werkzeug-Möglichkeiten sich monatlich verschieben.

Dieses Papier schlägt einen Prozess vor, der diese Werte erhält, sie aber für KI-augmentierte Entwicklung neu instrumentiert. Er entlehnt aus mehreren aktuellen Strömungen: dem *Manifesto for Enterprise Agility 2026*[1], dem V-Bounce-Modell der AI-Native-SDLC-Forschung[2], der Agentsway-Methodik[3] und der XP2025-Roadmap[4]. Er bündelt diese Strömungen mit konkreter Praxis-Erfahrung aus einem laufenden Projekt zu einem kohärenten Vorgehen, das ein Team an einem Tag einführen kann.

Zielgruppe

Dieser Text richtet sich an drei Lesergruppen. Erstens an KI-augmentierte Entwicklungsteams, die ihre tägliche Praxis explizit machen wollen, statt sie aus Tradition fortzuschreiben. Zweitens an Einzel-Entwickelnde, die mit einem KI-Pair arbeiten und nach einer Struktur suchen, die ihre Verantwortung nicht verwässert. Drittens an Tech-Leads und Engineering Manager:innen, die ihre Organisation auf

[1] PMI und Agile Alliance: *Manifesto for Enterprise Agility*, 2026.
[2] „The AI-Native Software Development Lifecycle: A Theoretical and Practical New Methodology", arXiv-Preprint 2408.03416, 2024. arxiv.org/abs/2408.03416
[3] „Agentsway: Software Development Methodology for AI Agents-based Teams", arXiv-Preprint 2510.23664, 2025. arxiv.org/abs/2510.23664
[4] „AI and Agile Software Development: A Research Roadmap from the XP2025 Workshop", arXiv-Preprint 2508.20563, 2025. arxiv.org/abs/2508.20563

einen Modus umstellen wollen, in dem die KI als ernstzunehmendes Werkzeug eingebunden ist, ohne dass sie zur Quelle der Wahrheit wird.

Der Text ist als verbindlicher Vorschlag formuliert, nicht als Wunschliste. Wo eine Regel steht, ist sie als Default gemeint, von dem nur mit guter Begründung abgewichen werden sollte. Das ist nicht Bürokratie um ihrer selbst willen, es ist die Anerkennung, dass eine sehr schnelle Werkzeug-Maschine sehr klare Leitplanken braucht.

2. Stand der Diskussion

Wer einen Prozess vorschlägt, sollte zeigen, an welche Diskussion er anschließt. Dieser Abschnitt skizziert die fünf Stränge, aus denen der hier vorgestellte Prozess seine Bausteine bezieht.

2.1 Manifesto for Enterprise Agility (2026)

Das Project Management Institute und die Agile Alliance haben 2026 gemeinsam ein *Manifesto for Enterprise Agility* veröffentlicht[5]. Es verschiebt den Fokus weg von der Team-Ebene und hin zur organisatorischen Fähigkeit. Drei Punkte sind für unsere Frage relevant: erstens die Betonung von gemeinsamer Ausrichtung, also einer geteilten Richtung über Team-Grenzen hinweg; zweitens die Idee gemeinsame Ergebnisse, die mehr zählen als individuelle Story-Punkte; drittens die explizite Aufnahme von menschlich getragener Anpassungsfähigkeit als Wert. Letzteres ist im KI-Kontext nicht trivial: es ist eine Erinnerung, dass die Anpassungsleistung beim Menschen bleibt, auch wenn das Werkzeug schneller adaptiert.

2.2 AI-Native SDLC und V-Bounce-Modell

Das V-Bounce-Modell[6] ist eine Variante des klassischen V-Modells, in der die Implementierungsphase durch die KI dramatisch verkürzt wird. Was übrig bleibt, und damit aufgewertet wird, sind die Phasen davor und danach: Anforderungsklärung, Architektur, Validierung. Die Rolle des Menschen verschiebt sich vom primären Implementieren zum primären Validieren. Die KI wird zur Implementations-Engine. Dieser Begriff ist im Folgenden konsistent verwendet, weil er die Beziehung zwischen Menschen und Werkzeug am treffendsten beschreibt: ein leistungsstarker Motor, der eine Hand am Steuer braucht.

2.3 Agentsway

Agentsway[7] ist eine 2025 vorgeschlagene Methodik für Teams, in denen mehrere KI-Agenten parallel arbeiten. Ihre wichtigste konzeptionelle Neuerung ist die Rolle des *Human Orchestrator*. Dieser Mensch koordiniert die spezialisierten Agenten (Planning Agent, Coding Agent, Testing Agent, Fine-tuning Agent), behält die Strategie in der Hand und trifft die Entscheidungen, die die Agenten nicht

[5] PMI und Agile Alliance: *Manifesto for Enterprise Agility*, 2026.
[6] „The AI-Native Software Development Lifecycle: A Theoretical and Practical New Methodology", arXiv-Preprint 2408.03416, 2024. arxiv.org/abs/2408.03416
[7] „Agentsway: Software Development Methodology for AI Agents-based Teams", arXiv-Preprint 2510.23664, 2025. arxiv.org/abs/2510.23664

treffen dürfen. Auch wenn der hier vorgeschlagene Prozess weniger auf Multi-Agenten-Setups zielt, ist die Idee der menschlichen Orchestrierung zentral.

2.4 XP2025-Workshop Research Roadmap

Ein Workshop auf der XP2025[8] hat eine Forschungsagenda für AI-augmented Agile formuliert. Drei konkrete Vorschläge prägen den hier formulierten Prozess: erstens die Idee von KI-Assistenzrollen (AI Product Owner Assistant, AI Quality Assurance Partner, AI Pair Programming Partner), die als ergänzende Werkzeuge benutzt werden können, ohne dass sie als Teammitglieder verstanden werden müssen. Zweitens der Vorschlag einer eigenständigen KI-Retrospektive, die getrennt von der klassischen Team-Retrospektive die Mensch-KI-Zusammenarbeit reflektiert. Drittens das Konzept von Prompt-Bibliotheken als lebende Dokumentation, die im Repository gepflegt werden.

2.5 GitHub Continuous AI

GitHubs Konzept der Continuous AI[9] beschreibt Hintergrund-Loops, in denen Agenten täglich kleine Pull Requests erzeugen, für Triage, Dokumentationspflege, kleine Qualitätsverbesserungen. Diese Idee ist die Ergänzung des hier vorgeschlagenen Prozesses, nicht sein Ersatz. Continuous-AI-Loops eignen sich für klar abgegrenzte Aufgaben, die kein menschliches Urteil im Moment der Entstehung brauchen. Alles, was Architektur-Entscheidungen, Sicherheitsabwägungen oder Stakeholder-Kommunikation berührt, gehört in die menschlich getaktete Iteration.

2.6 Persistente Kontext-Dateien

Ein Punkt, der in allen genannten Quellen anklingt und in diesem Prozess explizit gemacht wird, ist die Bedeutung persistenter Kontext-Dateien. Ein Sprachmodell vergisst zwischen Sessions, und auch innerhalb einer Session ist sein Gedächtnis begrenzt. Die Lösung ist nicht, das Modell zu zwingen, mehr zu erinnern, sondern die Erinnerung aus dem Modell herauszuziehen: Konventionen, Regeln, frühere Entscheidungen werden in Dateien geschrieben, die in jeder neuen Session wieder geladen werden. Diese Dateien sind das Gedächtnis des Teams, nicht das Gedächtnis der KI.

3. Es geht nur agil

Wer heute eine Software entwickelt und dabei mit einem Sprachmodell arbeitet, hat keine echte Wahl bei der Methode. Klassische, schwergewichtige Vorgehensmodelle scheitern in der KI-augmentierten Welt schneller und gründlicher als sie es je in der rein menschlichen Welt getan haben. Wasserfall, V-Modell mit langen Implementierungsphasen, Phasen-Gate-Modelle mit halbjährlichen Release-Zyklen, diese Ansätze waren schon vor der KI brüchig. Im KI-Zeitalter sind sie unhaltbar.

[8] „AI and Agile Software Development: A Research Roadmap from the XP2025 Workshop", arXiv-Preprint 2508.20563, 2025. arxiv.org/abs/2508.20563

[9] „Continuous AI in practice: What developers can automate today with agentic CI", GitHub Blog, 2026. github.blog/.../continuous-ai-in-practice

Der Grund ist nicht ideologisch, sondern praktisch. Schwergewichtige Methoden setzen auf Plan-Stabilität: man entscheidet einmal, dokumentiert ausführlich, implementiert lange. Diese Annahme funktioniert nur, wenn sich die Welt zwischen Plan und Auslieferung nicht zu schnell ändert. In einer Welt, in der Sprachmodelle ihre Fähigkeiten monatlich erweitern und in der eine sinnvolle Aufgabe oft in Stunden statt Tagen umsetzbar ist, ist Plan-Stabilität eine Illusion. Wer sich daran hält, baut Software, die schon beim Release veraltet ist.

3.1 Warum die Notwendigkeit zur Agilität wächst

Drei Verschiebungen erzwingen Agilität. Erstens: die Implementierung wird billig. Wenn ein Sprachmodell in zwei Stunden produziert, wofür ein Team früher zwei Tage brauchte, dann ist Implementierung nicht mehr die knappe Ressource. Knapp sind nun das Verstehen, das Reviewen, das Entscheiden. Diese Tätigkeiten sind ihrer Natur nach iterativ, sie funktionieren in kurzen Schleifen mit Feedback, nicht in langen Phasen mit Übergaben.

Zweitens: die Werkzeuglage ist dynamisch. Was heute mit dem einen Modell schwer ist, ist morgen mit einem neuen Modell trivial. Wer einen Sechs-Monats-Plan aufstellt, plant gegen eine Werkzeug-Generation, die zur Hälfte der Laufzeit überholt sein wird. Reaktion auf Veränderung ist keine Option, sondern eine Eigenschaft des Mediums.

Drittens: die Erwartungen der Auftraggebenden haben sich angepasst. Wer einen sichtbaren Prototyp innerhalb von Tagen erwarten kann, wird Wochen oder Monate nicht mehr tolerieren. Diese Erwartung ist nicht unrealistisch, sie ist eine direkte Folge dessen, was möglich geworden ist. Sie verlangt Takte, die in klassischen Modellen nicht vorgesehen waren.

3.2 Was vom Manifest 2001 trägt

Das Agile Manifest von 2001 formuliert vier Wertepaare und zwölf Prinzipien. Beide haben in der KI-Ära nichts an Gültigkeit verloren, im Gegenteil, ihre Notwendigkeit ist gewachsen. Individuen und Interaktionen über Prozesse und Werkzeuge: gerade weil das Werkzeug jetzt sehr stark geworden ist, ist die Frage, wie Menschen miteinander und mit dem Werkzeug interagieren, wichtiger denn je. Funktionierende Software über umfassende Dokumentation: gerade weil die KI Dokumentation auf Knopfdruck produziert, ist es wichtig, sie nicht zum Selbstzweck werden zu lassen. Zusammenarbeit mit der Kundschaft über Vertragsverhandlung: gerade weil schneller Output das Risiko erhöht, an der Kundschaft vorbei zu liefern. Reagieren auf Veränderung über das Befolgen eines Plans: gerade weil sich die Möglichkeiten monatlich verschieben.

Dieser Text übernimmt das Manifest deshalb nicht in geänderter Form. Es braucht keine Umformulierung; es braucht eine Begründung, warum es im KI-Zeitalter eher leichter als schwerer einzuhalten ist. Das ist das Thema des folgenden Kapitels.

3.3 Was sich verschiebt, ohne den Kern zu treffen

Einiges am ursprünglichen Manifest fühlt sich beim Wiederlesen historisch an. Die Annahme, dass alle Teammitglieder im selben Raum sitzen sollten, fünfzes Prinzip, passt nicht in eine Welt mit Remote-Arbeit und KI-Werkzeugen. Die Schätzung, dass funktionierende Software in Wochen bis Monaten

ausgeliefert werden soll, drittes Prinzip, wirkt langsam, wenn die KI-Takt Tage zulässt. Diese Verschiebungen verlangen aber keine Ersetzung der Prinzipien, sondern eine Neukalibrierung ihrer Skala.

Der hier vorgeschlagene Prozess hält an den zwölf Prinzipien fest und beschreibt im nächsten Kapitel, wie jedes einzelne im KI-augmentierten Modus konkret aussieht, was es weiter zu leisten hat, was leichter wird, und an welcher Stelle eine neue Vorsicht nötig ist.

4. Die zwölf Prinzipien im KI-Zeitalter

Das Agile Manifest hängt an zwölf Prinzipien[10]. Sie sind 2001 als praktische Konkretisierung der vier Wertepaare entstanden und haben sich über zwei Jahrzehnte gehalten. Dieses Kapitel formuliert keine neuen Prinzipien und beansprucht auch keine Aktualisierung des Manifests. Es nimmt die zwölf originalen Sätze, einen nach dem anderen, und fragt: was verlangen sie weiterhin, was wird in der KI-augmentierten Praxis bei ihrer Umsetzung leichter, und wo ist eine neue Vorsicht nötig? Das Ergebnis: die Prinzipien sind nicht nur weiterhin gültig, ihre Umsetzung ist heute oft deutlich leichter als 2001. Wer sie schon damals ernst genommen hat, hat im KI-Zeitalter einen guten Startpunkt.

4.1 Frühe und kontinuierliche Auslieferung wertvoller Software

Das erste Prinzip stellt Kundenwert als oberstes Ziel über alles. Es verlangt früh und kontinuierlich auszuliefern, nicht spät und auf einen Schlag. Das gilt unverändert: die Zufriedenheit der auftraggebenden Person entsteht aus Stücken, die sie sehen, anfassen, testen kann, nicht aus Planungs-Slideshows.

In der KI-Ära wird die Umsetzung dieses Prinzips deutlich leichter. Was früher Sprint-Takte mit zweiwöchigen Releases bedeutete, kann heute tägliche Auslieferung sein. Ein Issue, das morgens geklärt wird, kann nachmittags auf dem Test-Server stehen. Die Maschinerie aus Build-Pipeline und Continuous Delivery, die schon vor der KI existierte, bekommt nun einen Implementierungsmotor an die Spitze gestellt, der das Tempo annimmt, das die Pipeline ohnehin verkraftet hätte. Die Voraussetzung dafür ist Disziplin: kontinuierliche Auslieferung darf nicht zu unverstandener Auslieferung werden. Push und Deploy bleiben menschlich freigegebene Schritte, kein Automatismus.

4.2 Anforderungsänderungen sind willkommen, auch spät

Das zweite Prinzip macht aus dem klassischen Risiko, späten Änderungen, einen Wettbewerbsvorteil. Es verlangt, agile Prozesse so zu bauen, dass Veränderungen nicht zur Krise werden. Auch das gilt weiterhin: wer spät umdenkt, hat oft einen guten Grund, meistens, weil sich erst jetzt zeigt, was wirklich gebraucht wird.

Mit der KI wird die Reaktion auf späte Änderungen substantiell günstiger. Ein Refactoring, das früher zwei Tage Konzentration verlangte, kann mit gut formulierten Issues und einem zweiten Modell als Review in wenigen Stunden vollzogen sein. Eine Änderung an einem Domain-Modell, die sich durch

[10]Beck, Kent et al.: *Manifesto for Agile Software Development*, 2001. agilemanifesto.org

drei Schichten zieht, ist mit KI-Unterstützung in einem überschaubaren Aufwand zu machen. Die Voraussetzung ist, dass die Änderung disziplinarisch eingespeist wird: das Issue wird angepasst, nicht der Chat-Verlauf; die neue Akzeptanzbedingung wird ergänzt, nicht stillschweigend angenommen. Späte Änderungen, die im Chat liegen bleiben und nicht ins Issue wandern, sind im KI-Modus so gefährlich wie immer.

4.3 Funktionierende Software wird häufig ausgeliefert

Das dritte Prinzip nennt ursprünglich Wochen bis Monate als Takt, mit Vorzug für die kürzeren Skalen. Der Kern ist: häufig liefern, damit Feedback früh ist und Korrekturen klein. Dieser Kern gilt weiterhin.

Was sich verschiebt, ist die Skala. Daily-Iterationen sind die neue Norm, Vier-Stunden-Iterationen sind für klar abgegrenzte Aufgaben realistisch. Was 2001 als ambitionierte Takt galt, ist heute ein Default. Eine ganze Klasse von Verzögerungen, Build-Zeiten, manuelle Tests, Hand-Offs zwischen Rollen, ist durch Automatisierung und KI-Unterstützung in den Hintergrund gerückt. Das Risiko ist die andere Seite: häufige Auslieferung ohne Disziplin produziert eine Folge halbgarer Releases. Test-First, manuelle UI-Verifikation, Security-Gates sind kein optionaler Zusatzaufwand, sondern die Bedingung, unter der häufige Auslieferung verantwortbar ist.

4.4 Fachseite und Entwicklung arbeiten täglich zusammen

Das vierte Prinzip verlangt tägliche Zusammenarbeit zwischen Business und Entwicklung. Das galt 2001 für viele Organisationen schon als ambitioniert; in vielen Unternehmen ist es bis heute eher Wunschvorstellung als Realität.

Die KI macht diese Zusammenarbeit konkreter. Wenn die Fachseite eine Anforderung diktiert und ein:e Senior-Developer:in sie in ein Issue überführt, kann die KI noch am selben Tag einen ersten Prototyp liefern. Die Diskussion zwischen Fach und Entwicklung springt damit aus der abstrakten in die konkrete Phase: nicht „könnte man das so meinen?" sondern „so sieht es aus, passt das?". Die menschliche Diskussion wird damit nicht ersetzt, sondern verstärkt. Die KI ist Beschleuniger, nicht Ersatz.

4.5 Motivierte Individuen mit Umgebung und Vertrauen

Das fünfte Prinzip ist ein Bekenntnis zum Menschen: Projekte werden um motivierte Menschen herum gebaut, die das Vertrauen bekommen, ihre Arbeit zu tun. Vertrauen ist hier nicht naiv, es ist die einzige tragfähige Basis, weil Mikromanagement in Wissensarbeit nicht funktioniert.

In der KI-Ära steigt der Anspruch an die Menschen, nicht der Bedarf an Mikromanagement. Wer mit Sprachmodellen arbeitet, muss kritisch denken, plausible Ergebnisse von korrekten unterscheiden, Architektur-Entscheidungen verantworten. Diese Tätigkeiten sind anspruchsvoller als das, was sie ersetzen, und sie verlangen Motivation, Konzentration und ein Umfeld, das beides ermöglicht. Eine Organisation, die das fünfte Prinzip 2001 verletzt hat, wird im KI-Zeitalter besonders schnell merken, dass demotivierte Menschen mit mächtigen Werkzeugen schneller Schaden anrichten.

4.6 Direkte Kommunikation als effektivste Methode

Das sechste Prinzip behauptet, dass Auge-zu-Auge-Gespräche die effektivste Form der Informationsübertragung in einem Entwicklungsteam sind. Es ist das Prinzip, dem die KI-Ära am sichtbarsten widerspricht, und das gleichzeitig nicht widerlegt, sondern neu eingeordnet wird.

Zwischen Menschen bleibt direkte Kommunikation überlegen. Im Daily-Standup, im Stakeholder-Review, in der KI-Retrospektive, das menschliche Gespräch trägt weiterhin am meisten Information pro Minute. Was sich verschiebt, ist der Anteil der Kommunikation, der zwischen Mensch und KI stattfindet. Diese kann nicht von Angesicht zu Angesicht sein; sie ist notwendig schriftlich. Daraus folgt eine pragmatische Konsequenz: alles, was die KI wissen muss, gehört in ein Issue, in KI-Konventionsdateien oder in Memory-Notizen. Das Gespräch zwischen Menschen darf dadurch nicht verarmen. Die Versuchung, alles in Schriftform zu pressen, weil die KI das gut liest, wäre eine Verletzung dieses Prinzips.

4.7 Funktionierende Software als wichtigstes Fortschrittsmaß

Das siebte Prinzip ist möglicherweise das wichtigste im KI-Zeitalter. Es verwirft alle Stellvertretermessgrößen, Velocity-Punkte, Lines-of-Code, Anzahl der PRs, und sagt: das einzige Maß ist funktionierende Software.

Die KI macht diese Frage drängender als je zuvor. Ein Modell produziert in einer Stunde mehr Code als eine entwickelnde Person an einem Tag. Wenn das Maß die Menge wäre, sähe alles großartig aus. Das Maß ist aber, ob diese Menge etwas Funktionierendes ergibt. Funktionierend heißt: Tests grün, Akzeptanzkriterien erfüllt, manuelle Verifikation bestanden, Stakeholder zufrieden. Wer im KI-Modus mit anderen Metriken arbeitet, baut Velocity-Charts, die im Burnout enden. Die einzige tragfähige Metrik ist die alte: läuft es?

4.8 Nachhaltige Geschwindigkeit

Das achte Prinzip verlangt eine Geschwindigkeit, die unendlich lange durchgehalten werden kann. Es richtet sich gegen Death-Marches und Endspurts, die in klassischen Sprints häufig waren.

Die KI verändert die Dynamik dieses Prinzips. Auf der einen Seite ist nachhaltiges Tempo leichter, weil viele lästige Tätigkeiten (Boilerplate, Tests-Schreiben für offensichtliche Fälle, Dokumentation) der KI übergeben werden können. Auf der anderen Seite ist die Versuchung größer, weil die KI nie müde wird. Wer sich von der KI tempotreiben lässt, läuft schneller in den Burnout als ohne sie. Die menschliche Review-Kapazität wird zur knappen Ressource. Eine Daily-Iteration, in der die KI zwölf Issues abarbeitet und ein Mensch alle zwölf reviewen soll, ist nicht nachhaltig. nachhaltiges Tempo heißt im KI-Modus: die Takt wird vom langsamsten menschlichen Schritt bestimmt, nicht vom schnellsten Werkzeug-Schritt.

4.9 Ständige Aufmerksamkeit auf technische Exzellenz

Das neunte Prinzip verlangt anhaltende Konzentration auf technische Qualität und gutes Design. Wer das schleifen lässt, baut technische Schuld auf, die die spätere Agilität auffrisst.

Die KI verstärkt dieses Prinzip in beide Richtungen. Wer mit Architektur-Verständnis und Senior-Erfahrung arbeitet, kann mit KI sehr saubere Lösungen sehr schnell erzeugen. Wer ohne Architektur-Verständnis arbeitet, baut mit KI sehr schnelle Stapel schlechten Codes. Senior-Developer-Rolle und Qualitätsexperte werden nicht überflüssig, im Gegenteil, ihre Bedeutung wächst. Plus: KI-generierter Code hat oft eine plausible Form, aber subtile Inkonsistenzen, die kein Test sofort aufdeckt. Continuous Refactoring, ArchUnit-Regeln und ein zweites Modell beim Review sind die operativen Mittel, mit denen dieses Prinzip im KI-Modus gehalten wird.

4.10 Einfachheit, die Kunst, nicht zu tun

Das zehnte Prinzip lobt das Maximieren der nicht gemachten Arbeit. Es ist das einzige Prinzip, das explizit gegen Übereifer warnt, und damit eines der schwersten zu leben.

Mit der KI wird Einfachheit schwerer einzuhalten, weil das Hinzufügen so billig wird. Ein zusätzliches Feature, eine zusätzliche Option, eine zusätzliche Validierung, alles kostet die KI fast nichts an Aufwand. Die Disziplin liegt darin, das Generieren zu unterlassen. Issues setzen Scope-Grenzen; die KI darf nichts hinzufügen, was nicht im Issue steht. Senior-Developer prüfen, ob ein KI-Vorschlag das eigentliche Problem löst oder Nebenkriegsschauplätze produziert. Das alte „you ain't gonna need it" ist im KI-Zeitalter ein zentrales Korrektiv geworden, gerade weil die Versuchung größer ist denn je.

4.11 Selbstorganisierende Teams

Das elfte Prinzip behauptet, dass die besten Architekturen, Anforderungen und Designs aus selbstorganisierenden Teams entstehen. Es ist ein Bekenntnis gegen Top-Down-Anweisung und für die Reife der Beteiligten.

Selbstorganisation funktioniert zwischen Menschen. Die KI ist nicht Teil dieser Selbstorganisation. Sie wählt nicht aus, an welchem Issue sie arbeitet; sie definiert nicht die Architektur; sie verhandelt nicht mit Stakeholder:innen. Sie ist Werkzeug. Was sich verschiebt, ist die Form der Selbstorganisation: weniger Stand-ups, mehr asynchrone Koordination über Issue und Board; weniger Pair-Programming, mehr Mensch-KI-Pair mit menschlichem Review. Die Senior-Developer-Rolle wird zur Architektur-Ankerin, weil die KI alleine keine kohärente Architektur erzeugt. Selbstorganisation darf nicht heißen, dass die KI ein gleichberechtigtes Teammitglied wird, sonst verliert das Prinzip seinen Sinn.

4.12 Regelmäßige Reflexion und Anpassung

Das zwölfte Prinzip verlangt, dass das Team in regelmäßigen Abständen reflektiert, wie es effektiver werden kann, und sein Verhalten entsprechend anpasst. Die Retrospektive ist seine Veranstaltung.

Im KI-Modus tritt zur klassischen Team-Retrospektive die KI-Retrospektive als eigenständiges Format hinzu (siehe Kapitel 7.5). Sie ist nicht Tagesordnungspunkt der allgemeinen Retro, sondern ein eigenes Event. Drei Fragen führen sie: Wo hat die Mensch-KI-Zusammenarbeit gehakt? Welche Memory-

Einträge sind veraltet? Welche Workflow-Regel braucht eine Schärfung? Die Antworten produzieren konkrete Änderungen an KI-Konventionsdateien, Memory-Files und der Prompt-Bibliothek. Das Prinzip wird damit stärker, weil es eine zweite Reflexionsebene bekommt. Wer in der KI-Ära nicht regelmäßig reflektiert, sammelt Memory-Drift an: alte Regeln, neue Realität, wachsende Reibung.

4.13 Zwischenbilanz

Alle zwölf Prinzipien gelten unverändert. Bei keinem ist die Umsetzung im KI-Modus schwerer geworden, bei den meisten ist sie leichter, bei einigen verschiebt sich die Skala oder die Form. Das Manifest schreibt sich nicht neu, es wird im KI-Zeitalter nur deutlicher lesbar. Wer es 2001 ernst genommen hat, ist heute gut gerüstet. Wer es als Lippenbekenntnis behandelte, wird seine Versäumnisse jetzt deutlicher merken: die KI verstärkt sowohl gute als auch schlechte Gewohnheiten. Diese Verstärkung ist der eigentliche Hebel der KI-augmentierten Entwicklung.

5. Rollen

Ein KI-augmentiertes Team hat drei zentrale menschliche Rollen. Daneben wird die KI als Werkzeug eingesetzt, das eigene Charakteristika hat, aber keine Rollenposition im klassischen Sinne. Dieser Abschnitt beschreibt beide Seiten und macht die Grenze deutlich.

5.1 Product Owner

Der Product Owner verantwortet das Was. Er definiert Anforderungen, priorisiert sie, formuliert Issues oder lässt sie nach Diktat formulieren. Er entscheidet, welches Issue als nächstes umgesetzt wird und welches warten muss. Er gibt das GO zur Implementierung und die Trigger-Phrasen für Push und Merge. Wenn das Team ein Kanban-Board nutzt, ist er es, der Issues nach Ready zieht, niemals die KI.

In klassischem Scrum würde diese Rolle weitere Aufgaben tragen: Sprint Planning, Stakeholder-Management, Release-Strategie. Diese Aufgaben verschwinden im KI-augmentierten Modus nicht; sie werden nur möglicherweise auf engere Takte verteilt. Ein PO in diesem Modus kann pro Tag mehrere Mikro-Plannings durchführen und mehrere Releases freigeben, weil die KI die operative Last reduziert.

5.2 Senior Developer

Die Senior-Developer-Rolle verantwortet das Wie. Sie prüft Architektur-Vorschläge der KI, bevor sie in Code übergehen. Sie hat ein Veto auf jedes Issue, das technisch unausgegoren ist. Sie entscheidet, wann ein Refactoring nötig ist und wann ein vorhandenes Muster gerade ausreicht. Sie sorgt dafür, dass die KI nicht in eine eigene Designwelt abdriftet, die zwar funktionierenden, aber inkonsistenten Code produziert.

Diese Rolle ist im KI-Zeitalter eher anspruchsvoller als entspannter geworden. Wer zehn Jahre Erfahrung mit einem Tech-Stack hat, kann mit der KI in Stunden Dinge produzieren, für die früher Tage nötig gewesen wären. Wer drei Monate Erfahrung hat, kann mit derselben KI Dinge produzieren, die plausibel aussehen, aber subtil falsch sind. In dieser Welt Senior-Developer:in zu sein, bedeutet, das Plausible vom Korrekten unterscheiden zu können.

5.3 Qualitätsexperte

Der Qualitätsexperte verantwortet die Verifizierung. Er führt die Code-Reviews durch ein zweites Modell anderer Bauart als das implementierende — beispielsweise Claude Code als Implementations-Werkzeug und OpenAI Codex als Review-Instanz, oder umgekehrt; auch Kombinationen mit Cursor, GitHub Copilot oder einem lokalen Modell sind möglich. Wichtig ist nicht das spezifische Werkzeug, sondern die unterschiedliche Bauart.. Er prüft Coverage- und Mutation-Scores. Er führt Security-Reviews durch und stellt sicher, dass Secrets nicht ins Repository wandern, dass SQL-Queries parametrisiert sind, dass DTOs validiert werden.

Diese Rolle muss nicht hauptamtlich besetzt sein. In einem kleinen Team kann sie die Senior-Developer-Rolle mitübernehmen. Wichtig ist, dass sie explizit existiert und nicht stillschweigend bei einer generalistisch arbeitenden Person landet. Die KI ist gut darin, Codes-Smells zu übersehen, die ihr selbst beim Generieren unterlaufen sind.

5.4 Die KI als Werkzeug

Die KI ist in diesem Prozess konsequent als Werkzeug behandelt. Sie hat keine Stimme im Daily-Standup. Sie wird nicht in der Team-Retrospektive nach ihrer Meinung gefragt. Sie tritt nicht in Sprint-Reviews vor Stakeholder. Sie hat keinen Namen am Whiteboard.

Was sie hat, ist eine sehr klare Aufgabenliste. Sie schreibt Code. Sie schreibt Tests. Sie schreibt Issues, wenn der Product Owner sie diktiert. Sie pflegt Memory-Dateien. Sie liest die KI-Konventionsdateien und wendet sie an. Die folgende Tabelle macht das explizit:

Die KI darf	**Die KI darf NICHT**
Code, Tests und Dokumentation schreiben Issues nach Diktat formulieren Pläne erstellen und zur Diskussion stellen Refactorings durchführen, sobald freigegeben Lokale Builds und Test-Suites ausführen Code-Reviews von zweiten Modellen einlesen Memory-Dateien aktualisieren und konsolidieren Konventionen aus den KI-Konventionsdateien anwenden	Pushen ohne explizite Trigger-Phrase PRs nach production ohne Trigger erstellen Eigenständig Issues nach Ready ziehen Plan-Approval als GO interpretieren Tests skippen oder Hooks mit --no-verify umgehen Eigenmächtig Backlog-Issues priorisieren Verantwortung für Release-Entscheidungen tragen Stakeholder-Kommunikation selbständig führen

Symbolisch wird die Mitarbeit der KI in Commit-Messages durch eine Co-Authored-By-Zeile kenntlich gemacht. Diese Zeile dokumentiert Transparenz; sie verleiht keinen Teammitglieds-Status. Sie ist eher das, was in einem Wissenschaftspaper eine Acknowledgement-Sektion ist: Anerkennung dafür, dass das Werkzeug substantiell beigetragen hat, ohne dass das Werkzeug deshalb Co-Autor mit gleicher Stimme wird.

5.5 Optionale KI-Assistenzrollen

Die XP2025-Roadmap[11] schlägt drei KI-Assistenzrollen vor: AI Product Owner Assistant, AI Quality Assurance Partner, AI Pair Programming Partner. Diese können im hier vorgeschlagenen Prozess als spezialisierte Werkzeuge eingesetzt werden, beispielsweise als Hilfsmittel beim Backlog-Refinement oder bei der Test-Generierung. Wichtig ist, dass sie als Werkzeuge sichtbar bleiben. Sobald ein AI PO Assistant beginnt, eigenständig Tickets zu schreiben, ohne dass ein Mensch sie freigibt, wird aus dem Werkzeug eine Rolle, und die Verantwortungskette bricht.

6. Artefakte

Der Prozess hängt an wenigen, aber sorgfältig definierten Artefakten. Sie sind die materielle Seite der Werte und Prinzipien.

6.1 Das Issue als Single Source of Truth

Ein Issue im hier vorgeschlagenen Format hat vier Abschnitte. Erstens den Kontext: warum diese Aufgabe gemacht wird, was vorher fehlt, welche Vorgeschichte dazugehört. Zweitens die Aufgabe: was konkret zu tun ist, welche Dateien betroffen sind, welche Tests vorher geschrieben werden müssen, falls TDD greift. Drittens das Akzeptanzkriterium: wie verifiziert wird, dass die Aufgabe erledigt ist; konkret, messbar oder ausführbar. Viertens die Abhängigkeiten: welche anderen Issues zuerst fertig sein müssen, oder „Keine".

Issues sind kleinteilig. Ein Issue ist ein logischer Schritt, der eigenständig getestet werden kann. Wer ein Issue schreibt, das nur in einem Drei-Tage-Marathon gelöst werden kann, sollte es in Sub-Issues schneiden. Kleinteilige Issues sind nicht nur leichter umzusetzen, sondern auch leichter zu reviewen und im Fehlerfall leichter zurückzurollen.

6.2 Das Kanban-Board

Das Kanban-Board mit fünf Spalten ist der zentrale Statusträger. Es macht für alle Beteiligten, Menschen wie KI, sichtbar, wo ein Issue gerade steht.

Spalte	Bedeutung	Wer schiebt
Backlog	Idee oder Issue mit offenen Designfragen; nicht freigegeben.	Beide
Ready	Vollständig groomt, Akzeptanzkriterien stehen, gilt als generelles GO.	Nur Mensch
In progress	Aktuelle Arbeit. Ein Issue zur Zeit.	KI beim Start
In review	Lokal fertig, Tests grün, nicht gepusht.	KI beim Abschluss

[11] „AI and Agile Software Development: A Research Roadmap from the XP2025 Workshop", arXiv-Preprint 2508.20563, 2025. arxiv.org/abs/2508.20563

Spalte	Bedeutung	Wer schiebt
Done	Mensch hat geprüft, Push erfolgt.	Nur Mensch

Die Ready-Spalte ist der Kern des Boards. Ein Issue in Ready ist mit allem Notwendigen ausgestattet, um sofort umgesetzt zu werden. Die Bewegung von Backlog nach Ready ist eine bewusste Entscheidung des Menschen, nie eine stillschweigende Verschiebung der KI. Innerhalb der Ready-Spalte gilt eine klare Reihenfolge: aufsteigend nach Issue-Nummer wird abgearbeitet. Diese Konvention spiegelt die Erstellungsreihenfolge und macht Abhängigkeiten von älteren zu neueren Issues transparent.

Die Done-Spalte ist ein UI-Signal, kein Push-Trigger. Auch wenn ein Issue formal Done erreicht hat, bleibt der Push an die explizite Trigger-Phrase gebunden. Diese Trennung verhindert, dass Board-Mechanik eine Sicherheitsbarriere ersetzt.

6.3 Persistente Kontext-Dateien

Vier Klassen von Dateien tragen den persistenten Kontext eines Teams:

- Die zentrale KI-Konventionsdatei im Repo-Root beschreibt die projekt-spezifischen Regeln und Konventionen — bei Claude Code heißt sie typischerweise CLAUDE.md, bei OpenAI Codex AGENTS.md, bei Cursor liegen vergleichbare Regeln in .cursor/rules. Das Konzept ist werkzeug-agnostisch, der Dateiname Werkzeug-Sache. Wer hier liest, weiß, was in diesem Projekt anders ist als in anderen.
- Eine Workflow-Konventionsdatei (im Beispiel CLAUDE-workflow.md) beschreibt den Prozess selbst. Diese Datei ist bewusst projekt-unabhängig gehalten, damit sie in andere Projekte kopiert und dort minimal angepasst werden kann.
- Tech-spezifische Konventionsdateien (im Beispiel CLAUDE-java.md, CLAUDE-react.md, CLAUDE-python.md) sammeln Konventionen pro Sprache und Framework: Testpyramiden, Code-Style, Architektur-Muster.
- Eine Security-Konventionsdatei (im Beispiel CLAUDE-security.md) bündelt die Security-Checks vor Commit und Push.
- Persönliche Memory-Dateien im Workspace der entwickelnden Person ergänzen das um individuelle Vorlieben, bevorzugte Tools, Sprachpräferenz, persönliche Routinen.

Alle diese Dateien werden gepflegt wie Code. Änderungen daran sind Commits. Sie sind Teil des Reviews. Sie werden mit dem Repo versioniert. Wer sie als Schmierzettel behandelt, verliert ihren Wert.

6.4 Prompt-Bibliothek

Wiederverwendbare Anweisungen an die KI werden im Repository gepflegt, als kleine Dateien oder als Sektionen in den KI-Konventionsdateien. Ein gut gepflegter Prompt für die Erstellung eines neuen Issues, ein Prompt für ein Code-Review, ein Prompt für die Konsolidierung von Memory-Dateien: solche Bausteine sparen Sitzungen ein und sorgen für Konsistenz.

Die Prompt-Bibliothek ist nicht trivial zu pflegen. Sie veraltet schnell, weil sich die Modelle weiterentwickeln und manche Formulierung obsolet wird. Eine regelmäßige Inspektion, etwa monatlich, gehört zur Hygiene.

6.5 PR- und Commit-Konventionen

Jeder Commit hat einen aussagekräftigen Subject, eine Body-Begründung, einen Bezug zum Issue (typischerweise „Closes #N") und eine Co-Authored-By-Zeile, wenn die KI substanziell beteiligt war. Diese Konvention ist nicht Selbstzweck, sie macht den Verlauf nachvollziehbar, auch nach Monaten.

7. Takt und Events

Ein Prozess lebt von seinen Wiederholungen. Die folgenden fünf Events bilden das Mindestgerüst eines KI-augmentierten Teams. Sie ersetzen die Scrum-Events nicht eins zu eins, aber sie greifen die Funktionen auf, die in Scrum durch Sprint Planning, Daily, Review und Retrospektive geleistet werden.

Event	Takt	Teilnehmer	Zweck
Daily-Standup	Täglich, 15 Min	Menschen	Was steht heute an, was ist reviewbereit, welche Releases?
Issue-Grooming	Kontinuierlich	PO, Senior	Backlog klären, Issues nach Ready ziehen
Implementations-Iteration	Stunden bis 1 Tag	KI + Mensch	Ready-Issues sequenziell umsetzen
Stakeholder-Review	Wöchentlich	Team + Stakeholder	Stand zeigen, Feedback einsammeln
KI-Retrospektive	Alle 1-2 Wochen	Menschen	Mensch-KI-Zusammenarbeit reflektieren

7.1 Daily-Standup

Das Daily-Standup ist eine kurze, menschliche Veranstaltung. Fünfzehn Minuten reichen. Drei Fragen werden beantwortet: Welche Issues wandern heute von Ready nach In progress? Welche Issues stehen in In review zur Freigabe an? Welche Push- oder Merge-Entscheidungen sind heute zu treffen?

Die KI nimmt am Daily nicht teil. Das ist keine Bevormundung, es ist eine pragmatische Entscheidung. Das Daily dient der Koordination zwischen Menschen, die Verantwortung tragen. Die KI hat keine eigene Tageslage, kein Gefühl für Stakeholder-Druck, keinen Kalender. Was sie zu wissen braucht, steht in den Issues, im Board, in den CLAUDE-Dateien.

7.2 Issue-Grooming

Issue-Grooming ist kein eigenes Meeting, sondern eine kontinuierliche Tätigkeit. Mindestens einmal pro Tag, typischerweise als Teil der Morgens-Routine, geht der Product Owner durch das Backlog. Welche Issues haben offene Designfragen, die heute geklärt werden können? Welche sind reif, in Ready zu wandern? Welche sind veraltet und sollten geschlossen werden?

Die KI kann beim Grooming sehr nützlich sein: Issues nach Diktat formulieren, Akzeptanzkriterien vorschlagen, Designfragen aufdecken, die nicht offensichtlich sind. Was sie nicht tut, ist die Bewegung

nach Ready. Diese Bewegung ist der Moment, in dem ein Issue zum verbindlichen Auftrag wird, und das bleibt menschliche Verantwortung.

7.3 Implementations-Iteration

Die Implementations-Iteration ist das zentrale Arbeits-Event. Sie beginnt mit einer Trigger-Phrase wie „implementiere alles in Ready", dem pauschalen GO. Von dort arbeitet die KI Issue für Issue durch, in der Reihenfolge der aufsteigenden Issue-Nummern. Jedes Issue durchläuft die fünf Schritte: Bewegung nach In progress, Implementierung gegen das Issue, lokaler Commit ohne Push, Bewegung nach In review, dann das nächste Issue.

Ist Ready leer, meldet die KI Vollzug. Sie zieht keine Backlog-Issues eigenmächtig nach vorne. Sie wählt nicht aus, was sie als nächstes interessant findet. Sie wartet auf die nächste Iteration oder die nächste Anweisung.

Eine Implementations-Iteration kann mehrere Stunden bis einen ganzen Tag dauern. Ihr Output sind Commits, die auf einem Worktree-Branch liegen, plus Issues in der In-review-Spalte. Nichts ist gepusht, nichts in Produktion.

7.4 Stakeholder-Review

Das Stakeholder-Review ist ein wöchentliches Ereignis, in dem das Team der auftraggebenden Person zeigt, was entstanden ist. Es kann auf dem Test-Server stattfinden, auf dem die main-Branch deployed ist. Es ist die Stelle, an der „funktionierende Software über umfassende Dokumentation" konkret wird: nicht ein Slide-Deck wird gezeigt, sondern eine laufende Applikation, in der die letzten Tage Arbeit greifbar sind.

Aus dem Review entstehen typischerweise neue Backlog-Einträge, manchmal auch Korrekturen bestehender Issues. Die KI kann diese Anpassungen unmittelbar mitformulieren, sobald das Review beendet ist.

7.5 KI-Retrospektive

Die KI-Retrospektive ist die wichtigste Neuerung im Eventkanon. Sie ist nicht Teil der klassischen Team-Retrospektive, sondern ein eigenes Format alle ein bis zwei Wochen. Drei Fragen stehen im Zentrum: Wo hat die Mensch-KI-Zusammenarbeit gehakt? Welche Memory-Einträge sind veraltet und sollten konsolidiert werden? Welche Workflow-Regel braucht eine Schärfung, weil ein Anti-Pattern sich wiederholt hat?

Output der KI-Retrospektive sind konkrete Änderungen an den KI-Konventionsdateien, den Memory-Dateien oder dem Prompt-Bibliotheks-Bestand.. Es ist die einzige Veranstaltung, in der der Prozess sich selbst überarbeitet.

8. Iterationsmodelle

Die Wahl der Iterationslänge ist keine triviale Designentscheidung. Sie bestimmt, wie schnell ein Team auf Stakeholder-Feedback reagieren kann, wie viel Arbeit zwischen zwei Reviews liegt und wie häufig Fehler in Produktion landen können.

8.1 Die Tages-Iteration als Default

Eine Tages-Iteration ist ein realistischer Default für die meisten KI-augmentierten Teams. Morgens findet das Grooming statt: was wird heute gearbeitet, was wandert in Ready. Mittags läuft die Implementations-Iteration: die KI arbeitet die Ready-Issues ab, der Mensch reviewt zwischendurch. Abends erfolgt der Push und gegebenenfalls der Merge nach production.

Diese Takt hat zwei Vorteile. Sie ist überschaubar, ein Mensch kann am Abend ehrlich beurteilen, was am Morgen versprochen wurde. Und sie produziert täglich Output auf dem Test-Server, was Stakeholder-Vertrauen aufbaut. Wer mit weniger Takt arbeitet, läuft Gefahr, dass die KI im Hintergrund Dinge baut, die niemand mehr nachvollziehen kann.

8.2 Die Vier-Stunden-Iteration

Schnellere Iterationen, etwa alle vier Stunden, sind möglich, aber nicht für alle Arbeitsarten geeignet. Sie passen zu klar abgegrenzten Mikro-Features, etwa kleinen UI-Verbesserungen oder Bugfixes, bei denen Stakeholder-Feedback unmittelbar einfließen soll. Sie passen, wenn das Team in einem engen Austausch mit der auftraggebenden Person steht und mehrfach täglich eine neue Version sehen will.

Sie passen nicht für Architektur-Arbeit. Ein neues Modul, ein größeres Refactoring, eine Schemaerweiterung in der Datenbank, solche Aufgaben brauchen Denkzeit, in der nicht alle vier Stunden ein Zwischenstand erzwungen wird. Wer hier in zu enge Takte zwingt, bekommt eine Folge halbgarer Pushes, von denen jeder einzeln den Test-Server destabilisiert.

Eine pragmatische Regel: Solange die Iteration ein abgeschlossenes, deploybares Issue produziert, ist sie sinnvoll. Sobald das Issue gezwungen wird, in zwei Iterationen zerteilt zu werden, weil die Takt drängt, ist die Takt falsch.

8.3 Das Continuous-AI-Modell

Das Continuous-AI-Modell, wie es etwa GitHub propagiert, läuft parallel zur menschlich getakteten Iteration. Hintergrund-Agenten arbeiten kontinuierlich an Triage, Dokumentationspflege, kleinen Qualitätsverbesserungen. Sie öffnen kleine Pull Requests, die ein Mensch reviewt und merged. Sie laufen, während das Team schläft.

Continuous AI ist eine sehr gute Ergänzung, aber kein Ersatz für die menschliche Iteration. Was die Hintergrund-Agenten produzieren, muss reviewbar sein und muss durch dieselbe menschliche Freigabe gehen wie alle anderen Pushes. Wer Continuous AI ohne diese Disziplin einsetzt, läuft Gefahr, dass das Repository sich in eine Richtung entwickelt, die niemand bewusst gewählt hat.

8.4 Sprint vs. Flow

Eine letzte Designfrage ist die zwischen Sprint und Flow. Klassisches Scrum arbeitet in Sprints, also festen Zeitperioden mit definiertem Anfang und Ende. Kanban arbeitet im Flow: Issues fließen kontinuierlich, ohne Zeitfenster zu respektieren.

Für KI-augmentierte Entwicklung ist der Flow meist passender. Issues sind kleinteilig, ihre Bearbeitung dauert Stunden statt Tage. Ein Zwei-Wochen-Sprint, der mit dreißig Issues geplant wird, ist eher Bürokratie als Hilfsmittel. Ein Kanban-Board mit Ready-Reihenfolge ist die natürliche Form. Sprints behalten ihren Sinn dort, wo eine externe Takt sie erzwingt, etwa eine zweiwöchige Release-Vorgabe oder eine fixierte Stakeholder-Review-Takt.

9. Der Prozess in neun Schritten

Was bisher in Werten, Prinzipien und Artefakten beschrieben wurde, lässt sich als konkrete neun-Schritt-Anleitung formulieren. Jeder Schritt hat einen klaren Aktor und einen klaren Output.

Schritt 1, Anforderungsdefinition

Die auftraggebende Person formuliert oder diktiert eine Anforderung. Wenn ein Satz abbricht oder mehrdeutig bleibt, wird nachgefragt, nicht geraten. Output: eine knappe, verständliche Beschreibung dessen, was gebaut werden soll.

Schritt 2, Plan

Die KI erstellt einen Plan. Dieser benennt Ziel, betroffene Bereiche, architektonische Entscheidungen, betroffene Dateien und die geplante Verifizierung. Annahmen werden explizit gemacht. Der Plan wird zur Diskussion gestellt, nicht zur Implementierung. Output: ein lesbarer Plan, an dem sich entscheiden lässt, ob er sinnvoll ist.

Schritt 3, Plan zu Issues

Der Plan wird in ein oder mehrere GitHub-Issues überführt. Jedes Issue ist kleinteilig und steht für sich. Das Issue ist ab jetzt die Quelle der Wahrheit. Output: ein oder mehrere Issues im Backlog.

Schritt 4, GO

Die auftraggebende Person gibt das GO. Bei Kanban-Nutzung geschieht das durch Verschieben des Issues nach Ready. Eine Plan-Akzeptanz allein reicht nicht, das GO ist ein separater, expliziter Schritt. Output: ein freigegebenes Issue.

Schritt 5, Implementierung

Die KI implementiert gegen das Issue, nicht gegen den Chat. Sie bewegt das Issue nach In progress, schreibt Code und Tests, committet lokal, bewegt das Issue nach In review. Output: lokaler Commit mit Bezug auf das Issue.

Schritt 6, Lokale Prüfung

Tests laufen grün, Unit, Slice, Integration. Builds laufen grün, Backend und Frontend. Bei UI-Änderungen wird der Dev-Server gestartet und das Feature im Browser durchgeklickt. Coverage- und Mutationsschwellen werden erreicht. Output: eine grüne Pflicht-Check-Liste.

Schritt 7, Code-Review durch zweites Modell

Der Qualitätsexperte (oder der ihm zugeordnete Mensch) startet ein Review durch ein zweites Modell. Die Befunde werden im Issue oder im PR kommentiert. Output: ein dokumentierter zweiter Blick.

Schritt 8, Push

Die auftraggebende Person sagt „push main". Die KI pusht, den aktuellen Commit-Batch, nicht mehr. Eine frühere Push-Freigabe in derselben Session gilt nicht für neue Commits. Output: ein neuer Stand auf origin/main.

Schritt 9, Deployment und Pull Request

Der Test-Server zieht main automatisch. Die auftraggebende Person prüft auf dem Server. Wenn die Prüfung erfolgreich ist, wird ein PR von main nach production erstellt, entweder direkt oder nach einer expliziten Trigger-Phrase „merge production" durch die KI. Der PR wird durch die auftraggebende Person gemerged. Output: ein Release in Produktion.

10. Qualität

Geschwindigkeit ohne Qualität ist in der KI-augmentierten Entwicklung kein Vorteil, sondern eine Falle. Wer das Tempo der KI nutzt, ohne die Qualitätsprüfung mitzuwachsen, baut sehr schnell sehr viel sehr Fehlerhaftes. Die folgenden fünf Bereiche sind Mindeststandard.

10.1 Test-Driven Development bleibt zentral

Die alte Frage „Test zuerst oder Code zuerst?" wird in der KI-Ära aus pragmatischen Gründen klar zugunsten des Tests beantwortet. Wer der KI sagt „schreib mir Funktion X", bekommt eine plausibel aussehende Funktion X, die in 80 Prozent der Fälle ihren Job tut. Wer der KI sagt „schreib mir einen Test, der das Verhalten X spezifiziert, und dann eine Funktion, die diesen Test grün macht", bekommt eine Funktion, deren Output-Vertrag explizit ist.

Test-First mit KI ist nicht langsamer, es ist die einzige Möglichkeit, das KI-Tempo verantwortbar zu nutzen. Tests sind die Stelle, an der das Modell an die Realität gebunden wird.

10.2 Zwei-Modelle-Review

Ein Modell ist seiner eigenen Lösung gegenüber unkritisch. Es hat die Lösung soeben selbst formuliert; es würde dieselben Vereinfachungen jetzt nochmal vornehmen. Ein zweites Modell, mit anderer Architektur, anderem Trainingskorpus, anderen Eigenheiten, bringt einen frischen Blick. Es findet andere Fehler. Es macht andere Vorschläge.

Praktisch heißt das: wenn das implementierende Modell Claude ist, reviewt OpenAI Codex (oder ein vergleichbares Tool). Wenn das implementierende Modell GPT ist, reviewt Claude. Der Effekt ist messbar, die Fehlerrate sinkt deutlich.

10.3 Pflicht-Checks vor jedem Push

Vor jedem Push laufen automatisierte Pflicht-Checks. Sie sind im Build verdrahtet, nicht als optionaler Zusatzschritt. Ein Push, bei dem die Checks nicht grün sind, ist kein Push, sondern ein abgebrochener Vorgang.

- Code-Coverage (Linien und Branches): hundert Prozent oder begründeter Ausschluss.
- Mutation Score: idealerweise hundert Prozent.
- ArchUnit-Tests: grün, die Architektur-Regeln werden eingehalten.
- Statische Analyse: keine Findings von SpotBugs, PMD, ErrorProne, Checkstyle.
- Spotless: Code-Style einheitlich.
- Frontend-Build: tsc und Vite-Build erfolgreich.

10.4 Security-Gates

Sicherheit ist in einer KI-augmentierten Welt besonders fragil, weil die KI viele plausible, aber unsichere Muster vorschlägt. Sie generiert SQL-String-Konkatenation, weil die Trainingsdaten voll davon sind. Sie schlägt Debug-Logging mit sensiblen Daten vor, weil Hello-World-Tutorials so aussehen. Sie übersieht Validierungen, weil sie es eilig hat. Vor jedem Push wird daher geprüft:

- Keine Secrets im Diff, keine API-Keys, keine Passwörter, keine privaten Zertifikate.
- Alle SQL- und JPQL-Queries verwenden Parameter-Bindung.
- Controller-DTOs sind mit @Valid annotiert.
- Debug-Ausgaben enthalten keine sensiblen Daten.
- Actuator-Endpunkte sind authentifiziert, falls überhaupt aktiv.

10.5 Manuelle UI-Verifikation

Die KI kann viel, sie kann nicht im Browser klicken. Bei jeder Frontend-Änderung gehört eine manuelle Verifikation in das Schritt-6-Pflichtset. Der Dev-Server wird gestartet, das Feature wird durchgeklickt, der Golden Path und mindestens ein Edge Case werden geprüft.

Wenn diese Verifikation nicht möglich ist, kein Docker-Daemon, kein lokaler Build erreichbar, wird das im Abschlussbericht explizit vermerkt. Es ist keine Schande, einen Schritt nicht ausführen zu können; es ist eine Schande, dies zu verschweigen.

11. Anti-Patterns

Ein Prozess wird erst dadurch konkret, dass er die typischen Verletzungen benennt. Die folgenden acht Anti-Patterns sind real beobachtbar. Wer sich beim Arbeiten dabei ertappt, einen dieser Gedanken zu denken, sollte stoppen.

11.1 „Plan-Approval reicht, ich fange einfach an"

Die häufigste Verletzung. Ein Plan wird akzeptiert; die KI interpretiert das als Implementierungsfreigabe und beginnt zu coden. Plan-Approval ist Schritt 2 des Prozesses, GO ist Schritt 4. Sie sind nicht dasselbe. Die Trennung ist die zentrale Sicherheitsschwelle, wer sie aufweicht, kann den ganzen Rest auch streichen.

11.2 „Schnell pushen, ist eh trivial"

Ein kleiner Bugfix, ein CSS-Tweak, ein Doku-Update. Die Versuchung, schnell zu pushen, weil die Änderung „eh trivial" ist, ist menschlich. Sie ist auch falsch. Jeder Push verändert den Test-Server. Auch eine CSS-Änderung kann ein Layout zerschießen, das ein Stakeholder gleich sehen sollte. Jeder Commit-Batch braucht seine eigene Push-Freigabe.

11.3 „Tests folgen später"

Sie folgen nicht. Was später erledigt werden soll, wird nicht erledigt. Wer Test-First aufschiebt, hat im Endeffekt Test-Never. Wer Geschwindigkeit gewinnen will, muss früh investieren, der Test ist die billigste Versicherung gegen Regressionen.

11.4 „Coverage bei 98 Prozent reicht"

Coverage ist ein Stellvertreterwert. Sie zeigt, ob ein Test über jede Code-Zeile gelaufen ist, nicht, ob der Test diesen Code wirklich geprüft hat. Mutation Testing schließt diese Lücke. 98 Prozent Coverage mit 70 Prozent Mutation Score ist schlechter als 95 Prozent Coverage mit 98 Prozent Mutation Score. 100 Prozent oder begründeter Ausschluss ist die einzige sinnvolle Linie.

11.5 „Die KI macht das schon richtig"

Vertrauen ohne Verifikation ist die teuerste Form der Naivität. Die KI ist sehr gut darin, plausibel zu klingen. Sie ist nicht gut darin, ihre eigenen Fehler zu erkennen. Verifikation ist Menschen-Pflicht. Wer dieses Pflicht nicht trägt, sollte nicht mit KI arbeiten.

11.6 Memory-Drift

Memory-Dateien veralten. Eine Konvention, die vor sechs Monaten richtig war, ist heute vielleicht obsolet. Eine Regel, die mehrfach gebrochen wurde, gehört entweder konsolidiert oder gestrichen. Wer Memory-Dateien nur ergänzt, ohne sie zu pflegen, baut eine wachsende Schuldlast auf. Die KI-Retrospektive ist die Stelle, an der das geprüft wird.

11.7 KI-Halluzinationen als verifizierte Wahrheit

Die KI erfindet plausibel klingende Bibliotheksnamen, Methodensignaturen, Stack-Overflow-Antworten. Wer eine solche Halluzination ohne Verifikation in den Code übernimmt, baut Phantomschulden ein, die irgendwann mit Interesse zurückgezahlt werden. Jede Behauptung der KI, die einen externen Bezug hat, gehört geprüft.

11.8 Überautomatisierung ohne Stop-Punkte

Continuous AI ist sehr verlockend, Agenten arbeiten nachts, das Repository entwickelt sich von selbst weiter. Was fehlt, sind Stop-Punkte. Wer einen Hintergrund-Agenten startet, der für jedes neue Issue eigenständig einen Branch öffnet, Code schreibt, Tests grün macht und einen PR aufmacht, verliert die menschliche Kontrolle. Stop-Punkte sind keine Bequemlichkeitseinschränkung, sondern Strukturelement.

12. Erste Schritte für neue Teams

Wer den Prozess in einem neuen Team einführen will, kann das an einem Vormittag tun. Die folgende Bootstrapping-Checkliste ist die Minimalausstattung. Vieles davon ist nicht spektakulär, aber es macht den Unterschied zwischen einem Team, das mit KI arbeitet, und einem Team, das von der KI bearbeitet wird.

12.1 Dateien im Repo-Root

- Die zentrale KI-Konventionsdatei (im Beispiel-Setup CLAUDE.md) als Projekt-Doku, was tut das Projekt, welche Standards gelten..
- Eine Workflow-Konventionsdatei (im Beispiel CLAUDE-workflow.md) mit der hier beschriebenen 9-Schritt-Anleitung und der Kanban-Sektion.
- Tech-spezifische Konventionsdateien je nach Stack (im Beispiel CLAUDE-java.md, CLAUDE-react.md, CLAUDE-python.md).
- Eine Security-Konventionsdatei (im Beispiel CLAUDE-security.md) mit der Checkliste vor Commit.

Diese Dateien sind kopierbar zwischen Projekten. Wer sie einmal sorgfältig erstellt hat, kann sie in das nächste Projekt übernehmen und nur die projekt-spezifischen Details anpassen.

12.2 Kanban-Board

Ein GitHub-Project mit fünf Spalten: Backlog, Ready, In progress, In review, Done. Felder für Priorität, Schätzung und Owner sind optional, helfen aber bei der Übersicht. Wichtig ist die explizite Konvention: Ready-Bewegungen macht der Mensch, In-progress- und In-review-Bewegungen macht die KI, Done macht der Mensch.

12.3 Trigger-Phrasen vereinbaren

Das Team einigt sich auf konkrete Trigger-Phrasen. „GO" oder „implementiere das" für die Implementierungsfreigabe. „Push main" für den Push auf origin/main. „Merge production" für den

PR-Trigger nach Produktion. Diese Phrasen werden in der Workflow-Konventionsdatei festgehalten und sind für alle Beteiligten verbindlich.

12.4 Erstes Issue: Setup

Das erste Issue in einem neuen Projekt sollte das Setup der genannten Konventionen sein. Klingt rekursiv, ist es auch. Es ist gleichzeitig der erste Praxis-Test des Prozesses: Plan, Issue, GO, Implementierung, Review, Push. Wer hier durchläuft, kennt den Prozess vom ersten Tag an.

12.5 Pflicht-Checks im Build verdrahten

Coverage-Thresholds, Mutation-Schwellen, ArchUnit-Tests, Static Analysis werden im Maven- oder npm-Build als verbindliche Gates eingerichtet. Ein Build, der nicht grün ist, blockiert den Push. Diese Verdrahtung ist die eine Investition, die am Anfang Zeit kostet und über Monate Zeit spart.

12.6 Zweite KI für Reviews wählen

Es wird festgelegt, welches zweite Modell die Reviews macht. Wenn das implementierende Modell Claude ist, wird typischerweise OpenAI Codex oder ein vergleichbares Tool gewählt. Wichtig ist nicht das spezifische Tool, sondern dass es ein anderes ist. Das Setup wird in einer kleinen Anleitung dokumentiert, damit ein neues Teammitglied am ersten Tag damit arbeiten kann.

13. Ausblick

Der hier vorgeschlagene Prozess ist eine Momentaufnahme. Die Werkzeuge ändern sich monatlich; die Best Practices werden sich entsprechend ändern. Drei Entwicklungen sind absehbar.

Erstens: mehr Autonomie der KI-Agenten. Was heute als Implementations-Engine fungiert, wird morgen in Mehr-Agenten-Aufstellungen arbeiten, wie sie Agentsway[12] beschreibt. Ein Planning-Agent, ein Coding-Agent, ein Testing-Agent, ein Documentation-Agent, orchestriert von einer menschlichen orchestrierenden Person. Der hier formulierte Prozess ist auch in einer solchen Welt anwendbar; er muss nur in einigen Rollen anders besetzt werden.

Zweitens: Continuous AI wird zum Standard. Was GitHub heute[13] als Vision formuliert, wird in zwei Jahren Selbstverständlichkeit sein. Repositories werden Hintergrund-Loops haben, in denen kleine Verbesserungen kontinuierlich entstehen. Das macht die Trennung zwischen menschlich getakteter Iteration und KI-getakteter Hintergrundarbeit umso wichtiger.

Drittens: die offenen Forschungsfragen aus der XP2025-Roadmap[14] werden weiter unbeantwortet bleiben. Wie balanciert man KI-Autonomie und menschliche Oversight in Echtzeit? Welche

[12] „Agentsway: Software Development Methodology for AI Agents-based Teams", arXiv-Preprint 2510.23664, 2025. arxiv.org/abs/2510.23664

[13] „Continuous AI in practice: What developers can automate today with agentic CI", GitHub Blog, 2026. github.blog/.../continuous-ai-in-practice

[14] „AI and Agile Software Development: A Research Roadmap from the XP2025 Workshop", arXiv-Preprint 2508.20563, 2025. arxiv.org/abs/2508.20563

Governance-Frameworks tragen ethische Anforderungen? Wie verändert sich die Kultur eines Teams, das täglich mit Sprachmodellen arbeitet? Welche Schulung brauchen agile Praktiker, um sinnvolle KI-Partnerschaften aufzubauen? Diese Fragen werden nicht durch Methoden gelöst, sondern durch Erfahrung. Der hier vorgeschlagene Prozess ist ein Vorschlag, sie zu sammeln.

Die Grundregel bleibt: je autonomer die KI, desto strenger müssen die menschlichen Kontrollpunkte sein. Wer das vergisst, baut Software, die niemand verantwortet, und niemand verantwortete Software hat eine Lebenserwartung, die in Wochen zu rechnen ist.

14. Quellenverzeichnis

Die folgenden Quellen sind über die Fußnoten im Text referenziert. Zugriffsdaten: Mai 2026.

- [1] PMI und Agile Alliance: *Manifesto for Enterprise Agility*, 2026. Diskussion in „Agile Software Development in 2026: Principles, AI Impact, and Why It Still Dominates Modern Delivery" und „New Agile Principles in AI Era".
- [2] „The AI-Native Software Development Lifecycle: A Theoretical and Practical New Methodology", arXiv-Preprint 2408.03416. https://arxiv.org/abs/2408.03416
- [3] „Agentsway: Software Development Methodology for AI Agents-based Teams", arXiv-Preprint 2510.23664. https://arxiv.org/pdf/2510.23664
- [4] „AI and Agile Software Development: A Research Roadmap from the XP2025 Workshop", arXiv-Preprint 2508.20563. https://arxiv.org/pdf/2508.20563
- [5] „Continuous AI in practice: What developers can automate today with agentic CI", GitHub Blog. github.blog/ai-and-ml/generative-ai/continuous-ai-in-practice
- [6] Beck, Kent et al.: *Manifesto for Agile Software Development*, 2001. https://agilemanifesto.org

13. Über die Entstehung dieses Texts

Dieses Whitepaper ist nach derselben Methodik entstanden, die es beschreibt: in Ko-Produktion mit einem KI-Werkzeug, konkret mit Claude (Anthropic). Ich habe Struktur, Kernthesen und alle inhaltlichen Entscheidungen verantwortet, Claude hat die ersten Formulierungen erzeugt und ich habe sie redigiert, präzisiert und an mehreren Stellen ganz verworfen. Die Kontrolle über das Ergebnis bleibt damit beim Autor: das KI-Werkzeug arbeitet zu, aber es signiert nicht mit.

Diese Arbeitsweise habe ich an anderer Stelle ausführlicher beschrieben:

https://blog.mwolff.org/warum-ich-meine-rohtexte-nie-mehr-ohne-ki-bearbeite-und-trotzdem-die-kontrolle-behalte/".