Großes Software-Projekt mit KI □□

Ein Selbstversuch mit KI in der professionellen Enterprise-Softwareentwicklung

In den letzten Jahren sind wir auf YouTube und in sozialen Netzwerken immer wieder mit denselben Schlagzeilen konfrontiert worden:
„Baue eine App in 10 Minuten nur mit Sprache!“
„Kein Coding mehr nötig!“
„Jeder kann jetzt Software entwickeln!“

Das klingt spektakulär – und ist für viele beeindruckend. Aber für Menschen, die tagtäglich in der professionellen Softwareentwicklung arbeiten, haben solche Aussagen meist nur begrenzten Wert. Denn die Realität in Unternehmen sieht anders aus: Dort geht es nicht um kleine Demo-Apps, sondern um **große Architekturen, komplexe Anforderungen, hohe Qualitätsansprüche, Dokumentation, Tests, DevOps, Integrationen und belastbare Systeme**, die dauerhaft funktionieren müssen.

Genau hier setzt das im Video beschriebene Experiment an. Der Autor wollte nicht wissen, ob KI eine hübsche kleine Demo erzeugen kann. Er wollte wissen:

- Kann KI **eine große Enterprise-Anwendung** entwickeln?
- Kann sie das **mit professioneller Qualität**?
- Wie schnell ist sie dabei wirklich?
- Und was bedeutet das alles für die Zukunft der Softwareentwicklung? □□

Die Antwort darauf ist so drastisch, dass sie selbst den Autor nach eigener Aussage „nicht schlafen lässt“.

https://youtu.be/eLDHrqKplVI

Ausgangspunkt: Eine reale Enterprise-Frage statt YouTube-Hype

Der Kern des Experiments ist im Grunde einfach formuliert, aber enorm anspruchsvoll:

> Was passiert, wenn man eine **große, echte, fachlich relevante Anwendung** mit allem, was in professioneller Softwareentwicklung dazugehört, von Anfang bis Ende mit maximalem KI-Einsatz neu entwickelt?

Dabei ging es ausdrücklich **nicht** um eine Spielerei. Es ging um ein reales System: die über viele Jahre gewachsene private Backoffice- und Buchhaltungsanwendung des Autors. Eine Software, die er seit rund 18 Jahren verwendet, um Rechnungen, Belege, Kundenverwaltung und weitere Geschäftsprozesse abzubilden.

Diese Anwendung war allerdings technisch stark veraltet – eine alte .NET-Framework-Anwendung mit überholter UI-Technologie. Funktional erledigte sie zwar ihren Grundzweck, aber nur in einem Umfang, der im Laufe der Jahre eher pragmatisch gewachsen als strategisch geplant war. Viele Funktionen, die eigentlich sinnvoll gewesen wären, wurden nie umgesetzt.

Das Experiment bestand also nicht nur darin, diese Anwendung neu zu bauen, sondern sie gleichzeitig massiv zu erweitern. Das Ergebnis war am Ende eine Anwendung mit etwa **dem 12- bis 15-fachen Funktionsumfang** der alten Lösung. □□

Warum dieses Experiment so relevant ist

Viele Unternehmen stehen heute an einem Wendepunkt. KI wird zwar schon genutzt, aber meist nur **punktuell**:

- zur Unterstützung beim Schreiben von Code,

- für kleinere Automatisierungen,
- für einzelne Dokumentationsaufgaben,
- für Reviews,
- für Requirements-Unterstützung,
- oder zur punktuellen Architekturvalidierung.

Was in den meisten Teams aber noch fehlt, ist ein **durchgängiger KI-unterstützter Software-Lifecycle**. Also die Frage:

> Was passiert, wenn KI nicht nur beim Coden hilft, sondern durchgängig bei Anforderungen, Architektur, Implementierung, Tests, Dokumentation und Betrieb eingebunden wird?

Genau das wurde in diesem Selbstversuch untersucht. Und das Spannende ist: Der Autor beschreibt einen entscheidenden technischen Wendepunkt Ende 2025 / Anfang 2026.

Der eigentliche Game Changer: Modelltreue gegenüber Richtlinien □□

Aus Sicht des Autors ist der große Durchbruch nicht einfach nur „bessere Codegenerierung“. Der wirkliche Wendepunkt war etwas anderes:

KI-Modelle wurden plötzlich deutlich besser darin, **Richtlinien und Vorgaben zuverlässig einzuhalten**

Frühere Modelle konnten zwar bereits brauchbaren Code generieren, aber bei größeren Projekten zeigte sich oft ein bekanntes Problem:

- Anfangs folgen sie Guidelines noch halbwegs,
- mit zunehmender Projektgröße werden sie inkonsistenter,
- Architekturvorgaben werden teilweise ignoriert,
- Stilregeln brechen auseinander,
- und die Struktur driftet von den eigentlichen Vorstellungen des Teams weg.

Mit der neuen Modellgeneration Anfang 2026 änderte sich das offenbar massiv. Der Autor beschreibt, dass Modelle ab etwa der neuen Generation – er nennt als Beispiel „Opus 4.5“ – plötzlich deutlich besser darin waren, sich **penibel an vorgegebene Guardrails, Richtlinien und Qualitätsmaßstäbe** zu halten.

Und genau das ist in professioneller Softwareentwicklung entscheidend.

Nicht der Umstand, dass KI irgendeinen Code generiert, ist revolutionär – sondern dass sie **den Code in der gewünschten Architektur, im gewünschten Stil und in der gewünschten Qualität** generieren kann.

Das ist der Unterschied zwischen einem Gimmick und einem industriell nutzbaren Werkzeug. ⚙

Das Projekt: Eine komplette Business-Plattform statt nur ein Tool

Die neue Anwendung sollte nicht einfach nur die alte Buchhaltungssoftware ersetzen. Sie sollte zu einem vollständig integrierten System werden, das mehrere Bereiche zusammenführt.

Geplante Bestandteile des Systems

1. Backoffice-System

Das Backoffice ist das Herzstück der Anwendung. Hier werden unter anderem verwaltet:

- Rechnungen
- Belege
- Kunden
- interne Verwaltungsprozesse
- Buchhaltungsrelevantes
- Kommunikation
- Geschäftslogik

Im Vergleich zur alten Anwendung sollte dieses Backoffice deutlich umfangreicher, moderner und besser integriert sein.

2. Neue Homepage

Die bestehende Website des Autors war bislang von der Backoffice-Welt entkoppelt. Eingehende Anfragen mussten manuell übertragen werden. Die neue Lösung sollte daher Website und interne Systeme direkt miteinander verbinden.

Mögliche Funktionen der neuen Website:

- Workshops buchen
- Newsletter abonnieren
- Wartelisten verwalten
- Projektanfragen stellen
- Buchungs- und Kontaktprozesse direkt ins System einspeisen

3. Customer Area

Zusätzlich wurde ein Kundenbereich vorgesehen, in dem Kunden selbstständig mit dem System interagieren können, etwa um:

- Termine zu bestätigen
- Angebote einzusehen oder zu bestätigen
- Rechnungen herunterzuladen
- Kommunikationsprozesse nachzuverfolgen

4. KI-Agent für Büroautomation

Das eigentliche langfristige Ziel des Projekts geht aber noch weiter:
Der Autor möchte den Großteil seiner ungeliebten Büroarbeit automatisieren. □□

Dafür wurde ein KI-Agent namens **Hermes** vorgesehen, der später über APIs auf das System zugreifen und viele Aufgaben autonom erledigen soll, zum Beispiel:

- Rechnungen erstellen
- Belege erfassen
- Belege automatisch verbuchen
- Teile der Kundenkommunikation übernehmen
- organisatorische Aufgaben automatisieren

Das Ziel: **80–90 % der Büroarbeit bis Ende 2026 automatisieren**.

Besonders wichtig dabei: Diese KI soll **lokal** laufen und nicht in der Cloud, weil sensible Kundendaten verarbeitet werden. Datenschutz und Vertraulichkeit sind hier also zentrale Anforderungen □□

Die Architektur: bewusst „Enterprise“ gedacht

Der Autor betont mehrfach, dass er bewusst eine **Enterprise-Architektur** gewählt hat – nicht weil er sie für sein eigenes kleines Geschäft zwingend gebraucht hätte, sondern weil er das Experiment auf ein professionelles Niveau heben wollte.

Architekturentscheidungen im Überblick

Microservices

Statt eines Monolithen wurde eine **Microservice-Architektur** aufgebaut. Fachlich wäre das für das konkrete Problem nicht zwingend nötig gewesen. Aber als Architekturbeispiel für professionelle Systeme – und auch für spätere Workshops – war es eine spannende Wahl.

API Gateway

Vor die Services wurde ein API-Gateway gesetzt, umgesetzt mit **Traefik**.

Drei Frontends / Clients

Auf dem Gateway sitzen drei Haupt-Clients:

- Homepage
- Customer UI
- Backoffice

Workflow Engine mit n8n

Unterhalb der Clients wurde eine **Workflow Engine auf Basis von n8n** integriert. Diese übernimmt die Orchestrierung von Geschäftsprozessen und die Verarbeitung von Domain-Events.

Beispielsweise können Services Domain-Events auslösen, die anschließend durch Workflows weiterverarbeitet werden.

Externe Integrationen

Zusätzlich wurden externe Systeme eingebunden, etwa:

- Office 365
- Google
- Trello
- E-Mail-Dienste
- Kalenderfunktionen
- Teams-Meetings
- Google Sheets

Damit entstehen realistische Business-Prozesse, wie sie in vielen Unternehmen täglich vorkommen.

Lokale KI-Infrastruktur

Für den autonomen KI-Agenten wurde eine **lokale KI-Lösung** auf eigener Hardware aufgebaut. Genannt wird ein Modell von Alibaba/Qwen mit 36 Milliarden Parametern. Dieses wird über einen MCP-Server in die interne Infrastruktur eingebunden, sodass der Agent definierte Systemoperationen ausführen kann.

Das ist besonders spannend, weil es zeigt, dass KI hier nicht nur als Chat-Funktion gedacht wird, sondern als **aktiver Systemakteur** in einer realen Business-Architektur.

Auch DevOps wurde ernst genommen □□

Ein häufiges Problem vieler KI-Demos ist, dass sie sich fast nur auf Code konzentrieren. In echten Projekten reicht das natürlich nicht. Deshalb wurde auch der DevOps-Bereich realitätsnah abgebildet.

Umgebungen

Es gab zwei zentrale Umgebungen:

- Testumgebung
- Produktionsumgebung

Infrastruktur

Die Infrastruktur lief nicht auf einer großen Enterprise-Serverlandschaft, sondern pragmatisch auf:

- einem Desktop-PC
- einem NAS-System mit Docker

Entwicklungssetup

Die KI-Coding-Agents hatten Zugriff auf:

- das Code-Repository
- Spezifikationen
- den sogenannten Harness
- Docker auf den lokalen Maschinen
- Logs und Deployments

Das heißt: Die Agents konnten nicht nur Code generieren, sondern ihn auch direkt deployen, starten, Logs prüfen und Fehler analysieren.

Git & Build

Verwendet wurde **Gitea/Gitty** als leichtgewichtige GitLab-Alternative. Von dort aus wurden Builds in Test- und Produktionsumgebungen verteilt.

SSH-Zugriffe

Der KI-Agent bekam weitreichende Rechte auf der Testumgebung und eingeschränkte Rechte auf Produktion. Besonders interessant: Er konnte Produktions-Logs auswerten, um Probleme zielgerichtet zu analysieren.

Das ist ein sehr realistisches Setup, weil es genau in die Richtung geht, in die moderne AI-Assisted-Development-Workflows aktuell wandern:
Nicht nur generieren, sondern beobachten, prüfen, anpassen und operativ unterstützen.

Die Qualitätssicherung: Tests als zentrales Fundament

Der Autor wollte nicht nur „funktionierenden Code“, sondern **nachweisbar belastbare Qualität**. Deshalb spielte das Thema Testing eine zentrale Rolle.

Fokus auf zwei Testarten

1. API-Tests

Die einzelnen Microservices wurden über ihre Endpunkte getestet. Dabei wurden nicht nur Responses verifiziert, sondern auch:

- ausgelöste Domain-Events
- Datenbankzustände
- korrekte Mappings und Änderungen

2. Workflow-Tests

Da n8n eine wichtige Rolle spielte, wurden auch komplette Workflows getestet – inklusive Zusammenspiel mehrerer Services und Datenbanken.

Dabei wurden echte Testdaten verwendet, um sicherzustellen, dass die orchestrierten Geschäftsprozesse korrekt funktionieren.

Mocking externer Systeme

Externe Systeme wie Office 365 oder Google konnten natürlich nicht eins zu eins in der Testumgebung laufen. Deshalb wurde **WireMock** genutzt, um externe Aufrufe abzufangen und kontrollierte Mock-Antworten zurückzugeben.

Damit entstand eine Testumgebung, in der die für den späteren KI-Agenten wichtigsten Komponenten – Backend und Workflows – realistisch und ausreichend abgesichert werden konnten.

Die eigentliche Methodik: Vom Prompting zum Voice Driven Development □□

Einer der spannendsten Teile des Videos ist die Beschreibung, **wie** sich die Arbeitsweise im Laufe des Projekts verändert hat.

Der Autor beschreibt im Grunde fünf Entwicklungsphasen.

Phase 1: Micromanagement

Am Anfang arbeitete er klassisch promptbasiert:

- Prompt schreiben
- Code generieren lassen
- Code reviewen
- nachbessern
- nächster Prompt

Das ist die Art, wie viele heute mit KI arbeiten. Sie funktioniert – aber sie ist kleinteilig, anstrengend und oft ineffizient. Gerade in großen Projekten führt sie schnell dazu, dass das Modell Code und Strukturen erzeugt, die nicht wirklich zum Gesamtbild passen.

Phase 2: Quality Driven Development

Hier kam der sogenannte **Harness** ins Spiel.

Was ist ein Harness?

Der Harness ist im Grunde ein Satz von Regeln, Vorgaben und Kontextdateien, die dem Modell erklären:

- wie entwickelt werden soll,
- welche Architekturregeln gelten,
- welche Coding-Standards gelten,
- welche Strukturentscheidungen eingehalten werden müssen,
- wie mit Umgebungen, Tests und Artefakten umzugehen ist.

Man kann sich das wie Leitplanken für das Modell vorstellen.

Zusätzlich wurden statische Analysewerkzeuge eingebunden, insbesondere:

- **NDepend** für Architekturanalyse
- **ReSharper** für Codeanalyse

Dadurch wurde nicht nur „gefühlt“ geprüft, ob der Code passt, sondern **toolgestützt**.

Der entscheidende Wechsel war:
Nicht mehr der Mensch korrigiert ständig den Code, sondern der Mensch verbessert schrittweise den **Harness**, bis das System konsistent gute Ergebnisse liefert.

Das ist ein enorm wichtiger Gedanke. □□
Der Entwickler schreibt nicht mehr nur Code – er gestaltet das **Produktionssystem für Code**.

Phase 3: Spec Driven Development

Im nächsten Schritt wurden nicht mehr nur Prompts übergeben, sondern **Spezifikationen**, also fachliche Anforderungen in Story-Form.

Das Modell lernte, wie es aus Anforderungen:

- Entwicklungsaufgaben ableitet,
- diese strukturiert zerlegt,
- dann passend umsetzt.

Das ist der Übergang von „Schreibe mal diese Funktion“ zu „Hier ist die Story – setze sie professionell um“.

Phase 4: Test Driven mit KI

Hier wurde Testing auf eine neue Stufe gehoben. Tests wurden nicht auf Basis des bereits existierenden Quellcodes generiert, sondern direkt **aus Anforderungen und Akzeptanzkriterien** abgeleitet.

Das ist ein fundamentaler Unterschied.

Viele KI-Demos zeigen:

1. Code ist da
2. KI generiert Tests dafür

Das ist nett, aber im Grunde testet man damit oft nur das bereits implementierte Verhalten.

In diesem Experiment ging es darum, dass Tests das **gewünschte Systemverhalten** verifizieren – also das, was aus den Anforderungen hervorgeht.

So wurde das System zunehmend spezifikationsgetrieben.

Phase 5: Idea Driven und Voice Driven Development

Hier wird es wirklich radikal.

Der Autor beschreibt, dass sich seine Rolle immer weiter verschoben hat:

- zuerst Entwickler,
- dann Architekt,
- dann Product Owner,
- schließlich fast nur noch Stakeholder.

Am Ende konnte er Ideen einfach **per Sprache** formulieren. Das Modell diskutierte diese Ideen mit ihm, stellte kritische Rückfragen, erinnerte an bestehende Systemregeln, rechtliche Anforderungen und Zusammenhänge im Gesamtsystem.

Ein besonders eindrucksvolles Beispiel:
Auf einer Autofahrt nach Wien diskutierte er mehrere Stunden lang per Voice-Mode mit dem Modell über ein großes Modul zur Workshop-Planung.

Das heißt: Die eigentliche „Entwicklung“ bestand am Ende nicht mehr primär darin, Code zu schreiben, sondern darin:

- Ideen zu formulieren,
- Entscheidungen zu treffen,
- Anforderungen zu schärfen,
- Prioritäten zu setzen,
- Qualitätsmaßstäbe zu definieren.

Das ist vermutlich einer der wichtigsten Gedanken des gesamten Videos.
KI verschiebt die Rolle des Entwicklers von der manuellen Umsetzung hin zur strukturierten Systemführung.

Die Ergebnisse des Projekts

Und jetzt zu den Zahlen, die das Ganze so spektakulär machen.

Umfang des Systems

Am Ende entstanden:

- **213 User Stories**
- **4.083 Akzeptanzkriterien**
- **10 Microservices**
- **420 Endpoints**
- **108 MCP-Tools**
- **135 E-Mail-Typen / E-Mail-Prozesse**
- **88 n8n-Workflows**
- **10 Datenbanken**
- **113 Datenbanktabellen**
- **1.300 Seiten Dokumentation**

Zusätzlich auf Code-Ebene:

- **2.900 API-Tests**
- **392 Workflow-Tests**
- **420.000 Zeilen Code**

Das ist kein kleines Bastelprojekt mehr. Das ist ein ernstzunehmendes Softwaresystem.

Die Qualität der Ergebnisse: Der eigentliche Schock □□

Der Autor sagt selbst: Dass KI 420.000 Zeilen Code erzeugen kann, war keine Überraschung.

Die wirkliche Überraschung war die Qualität.

Qualitätsmetriken

Testabdeckung

- **85 % Test Coverage** im relevanten Backend- und Workflow-Bereich

Die fehlenden 15 % waren bewusst nicht vollständig abgedeckt, weil manche extern getriggerten Office-/Outlook-Prozesse nicht gemockt wurden. Laut Autor wäre 100 % prinzipiell möglich gewesen, aber der Aufwand war für ihn nicht mehr sinnvoll.

Strukturelle Schulden

- **0 % strukturelle Schulden**

Das ist einer der spektakulärsten Punkte des gesamten Berichts. Gemeint ist:
Die Anwendung hält sich vollständig an die definierten Architektur- und Designvorgaben.

Dokumentation

- **100 % Entwicklerdokumentation**
- tägliche / nächtliche automatische Neuerstellung, damit sie immer synchron bleibt

Gerade der letzte Punkt ist in realen Projekten fast revolutionär. Denn eine der größten Schwächen klassischer Softwareprojekte ist, dass Dokumentation mit der Zeit veraltet. Hier wurde das Problem durch automatisierte Regeneration praktisch systematisch beseitigt. □□

Die Kosten: extrem niedrig im Verhältnis zum Ergebnis □□

Der Autor investierte über etwa ein halbes Jahr hinweg **45 volle Arbeitstage à 8 Stunden** in das Projekt – zusätzlich zu seinem normalen Beruf.

Personalkosten

Bei einem angenommenen Vollkostensatz von **500 € pro Entwickler-Tag** ergeben sich:

- **22.500 € Personalkosten**

KI-Kosten

Er nutzte zunächst Codex, später fast nur noch Claude. Insgesamt wurden dabei gigantische Mengen Tokens verarbeitet:

- 20,2 Milliarden Tokens auf Rechner 1
- 6,8 Milliarden Tokens auf Rechner 2
- insgesamt rund 27 Milliarden Tokens

Rechnet man das zu regulären Tokenpreisen, wären laut seiner Aussage etwa:

- **21.000 € bei Anthropic**
- oder nur **2.000 € bei günstigeren Modellen wie DeepSeek V4 Pro**

Tatsächlich nutzte er jedoch Abos:

- Claude Max, teilweise doppelt
- Gesamtkosten über den Zeitraum: **1.620 €**

Gesamtkosten

Damit lag das gesamte Projekt ungefähr bei:

- **25.000 €**

Für ein System dieser Größe und Qualität ist das bemerkenswert.

Der Vergleich mit menschlichen Teams □□

Um die Ergebnisse einordnen zu können, zog der Autor zwei reale Entwicklungsteams aus Kundenprojekten als Vergleich heran.

Vergleichsteams

Team 1

- 5 Entwickler
- ca. 350.000 Zeilen Code
- Umfang: ca. 24 Personenjahre

Team 2

- 4 Entwickler
- ca. 300.000 Zeilen Code
- Umfang: ca. 17 Personenjahre

Beide Teams sollten das Spektrum des Projekts – also Specs, DevOps, Architektur, Coding, Testing und Dokumentation – einschätzen.

Die Schätzungen lagen bei:

- **23 Personenjahren**
- **15 Personenjahren**

Zusätzlich ließ der Autor Claude und Codex schätzen, die auf ca. 20 Personenjahre kamen. Seine eigene Schätzung lag bei 18 Personenjahren.

Der gemittelte Wert lag am Ende bei:

- **19 Personenjahren**
- das entspricht **4.180 Arbeitstagen**
- bei 500 € pro Tag also rund **2 Millionen Euro Projektkosten**

Der Faktor: 93x schneller □□

Und damit kommt der eigentliche Paukenschlag:

- Menschliches Team: **4.180 Arbeitstage**
- KI-gestützter Aufbau: **45 Arbeitstage**

Das ergibt einen Faktor von:

93x schneller

Selbst wenn man das als zu optimistisch betrachtet und massiv nach unten korrigiert, argumentiert der Autor:

- vielleicht nur 60x?
- vielleicht nur 25x?

Selbst dann wäre der Vorsprung immer noch enorm.

Und das bei **besserer Struktur, besserer Dokumentation und besserer Testabdeckung** als in den herangezogenen Vergleichsprojekten.

Was bedeutet das für die Softwareentwicklung? □□

Hier liegt die eigentliche Brisanz des Videos.

Der Autor macht sehr deutlich:
Es geht nicht nur darum, dass KI „ein bisschen unterstützt“.
Es geht nicht nur um Produktivitätssteigerung im Bereich von 20 oder 30 Prozent.
Wenn sich solche Ergebnisse verallgemeinern lassen – auch nur annähernd –, dann reden wir über einen **massiven strukturellen Umbruch** in der Branche.

Die wichtigsten Konsequenzen

1. Geschwindigkeit wird neu definiert

Was früher Monate oder Jahre dauerte, könnte künftig in Wochen entstehen.

2. Qualität muss nicht leiden

Die alte Annahme „schneller = schlechter“ wird durch solche Experimente massiv in Frage gestellt.

3. Entwicklerrollen verschieben sich

Weniger reine Implementierung, mehr:

- Anforderungen schärfen
- Architektur definieren
- Guardrails formulieren
- Qualitätsstandards beschreiben
- Systeme führen

4. Dokumentation und Tests können erstmals dauerhaft synchron gehalten werden

Ein riesiges Problem klassischer Projekte könnte systematisch abnehmen.

5. Der Engpass ist nicht mehr das Tippen von Code

Der Engpass verschiebt sich hin zu:

- Denken
- Strukturieren
- Priorisieren
- Fachlichkeit
- Verantwortung
- Architekturverständnis

Aber: Das kann nicht einfach jeder sofort

Trotz aller Begeisterung relativiert der Autor an einer wichtigen Stelle sehr klar:

> Nein, nicht jeder kann sofort mit derselben Qualität und Geschwindigkeit solche Systeme bauen.

Warum?
Weil die Qualität am Ende **nicht zufällig** entstanden ist.

Sie entstand, weil er:

- jahrzehntelange Erfahrung in Softwareentwicklung hat,
- Architektur und Qualität tief versteht,
- weiß, wie man Standards formuliert,
- einen guten Harness aufbauen konnte,
- und die richtigen Qualitätswerkzeuge eingebunden hat.

Das ist ein ganz wichtiger Punkt. KI ersetzt hier nicht das Wissen – sie **verstärkt** es.

Oder anders gesagt:
Wenn man nicht weiß, wie gute Architektur, gute Tests, gute Anforderungen und gute Entwicklungsprozesse aussehen, wird man auch mit KI kein vergleichbares Ergebnis erzielen.

Deshalb ist seine Empfehlung sehr klar:

Grundlagen bleiben entscheidend

Wer mit KI professionell entwickeln will, sollte weiterhin verstehen:

- Softwarearchitektur
- Requirements Engineering
- Qualitätssicherung
- Testdesign
- saubere Codeprinzipien
- Systemdenken

Denn genau dieses Wissen braucht man, um der KI gute Leitplanken zu geben.

Die vielleicht wichtigste Erkenntnis des ganzen Videos

Wenn man den Kern des Experiments auf einen Satz reduzieren müsste, dann vielleicht auf diesen:

> Der größte Hebel von KI in der professionellen Softwareentwicklung liegt nicht darin, dass sie Code schreibt, sondern darin, dass sie unter guten Leitplanken **komplette Entwicklungsarbeit konsistent, schnell und qualitativ hochwertig umsetzen kann**.

Das verändert alles.

Es verändert:

- wie wir Projekte planen,
- wie Teams aufgestellt werden,
- wie Anforderungen formuliert werden,
- wie Architekturarbeit aussieht,
- wie Dokumentation entsteht,
- wie Testen funktioniert,
- wie Entwickler ihre Rolle verstehen.

Und vor allem verändert es die ökonomische Logik von Softwareentwicklung.

Wenn es möglich ist, mit einem Bruchteil der Zeit und Kosten bessere Ergebnisse zu erzielen, dann wird sich die Branche zwangsläufig daran anpassen.

Kritische Einordnung: Was man bei aller Euphorie nicht vergessen darf

So beeindruckend das Experiment ist, sollte man es natürlich auch nüchtern betrachten.

Einige wichtige Einordnungen:

1. Es ist ein Einzelfall

Das Projekt wurde von einer sehr erfahrenen Person durchgeführt, die das Domänenwissen, die Architekturkompetenz und die Qualitätsmaßstäbe selbst mitgebracht hat.

2. Schätzvergleiche sind keine realen Parallelprojekte

Der Faktor 93 basiert auf Schätzungen von Teams, nicht auf einem live parallel entwickelten Vergleichsprojekt.

3. Das Setup war sehr bewusst optimiert

Harness, statische Analyse, Teststrategie, Modellwahl, Tooling und Arbeitsweise wurden gezielt aufgebaut und iterativ verbessert.

4. Der Domänenkontext war dem Autor vertraut

Es handelte sich um seine eigene Anwendung, also um ein Gebiet mit hohem Fachverständnis.

Trotzdem bleibt das Ergebnis selbst unter konservativer Betrachtung extrem stark. Denn selbst bei deutlicher Abschwächung der Zahlen bleibt noch ein massiver Produktivitäts- und Qualitätsgewinn.

Fazit: Wir sind an einem Wendepunkt □□

Dieses Video beschreibt nicht einfach nur ein beeindruckendes KI-Projekt. Es beschreibt einen möglichen Blick in die unmittelbare Zukunft professioneller Softwareentwicklung.

Die Kernaussage ist nicht:

„Programmierer werden überflüssig.“

Die Kernaussage ist vielmehr:

„Die Art, wie professionelle Softwareentwicklung funktioniert, verändert sich fundamental.“

Der Entwickler der Zukunft ist nicht mehr nur jemand, der möglichst schnell möglichst viel Code tippen kann.
Er oder sie wird zunehmend zu jemandem, der:

- Systeme versteht,
- Qualität definiert,
- Anforderungen präzisiert,
- Architekturen beschreibt,
- KI-Arbeit orchestriert,
- und Ergebnisse verantwortet.

Das ist einerseits faszinierend.
Andererseits ist es – wie der Autor sehr offen sagt – auch beunruhigend.

Denn wenn ein einzelner, KI-unterstützter Entwickler in 45 Tagen ein System liefert, für das klassische Teams im Schnitt 19 Personenjahre ansetzen, dann ist klar:
Diese Entwicklung wird nicht folgenlos bleiben.

Aber genau deshalb ist jetzt der richtige Zeitpunkt, sich ernsthaft mit dem Thema zu beschäftigen. Nicht oberflächlich. Nicht nur mit ein paar Prompts. Sondern tief, strukturiert und professionell. □□

Die zentrale Botschaft für Entwickler und Unternehmen

Für Entwickler

- Lernt KI nicht nur als Helfer, sondern als Entwicklungsplattform zu verstehen.
- Vertieft eure Grundlagen in Architektur, Qualität und Requirements.
- Beschäftigt euch mit Guardrails, Harnesses, Tests und Automatisierung.

Für Unternehmen

- KI darf nicht nur als Side-Tool betrachtet werden.
- Wer jetzt keine Strategie entwickelt, wird später massiv aufholen müssen.
- Die eigentliche Chance liegt in der Verbindung aus KI, Struktur, Qualität und Prozessintegration.

Schlussgedanke

Was dieses Experiment so bemerkenswert macht, ist nicht nur der Faktor 93.
Es ist die Kombination aus:

- **extremer Geschwindigkeit** ⚡
- **hoher struktureller Qualität** □□
- **starker Testabdeckung** □
- **vollständiger, synchroner Dokumentation** □□
- **und einer neuen Rolle des Entwicklers** □□

Wenn sich dieser Ansatz weiter bestätigt, dann war das hier tatsächlich nicht nur ein interessantes Experiment – sondern vielleicht wirklich einer der Momente, an denen sich die Softwareentwicklung dauerhaft verändert hat.

Revision #1
Created 2026-06-25 00:32:08 UTC by art10m
Updated 2026-06-25 00:37:03 UTC by art10m