Gefährliche Zustimmung

LLMs wollen uns glücklich machen – auch wenn sie dafür Annahmen für uns treffen. Warum das im Software-Kontext gefährlich ist und wie strukturiertes Rückfragen das Problem löst.

2026-02-11

(english below)

Deutsch

LLMs wollen uns glücklich machen — auch wenn sie dafür Annahmen für uns treffen. Im Software-Kontext ist das hochgefährlich.

Das Problem: Gefällige Zustimmung

KI-Builder behandeln Nutzerintention als vollständig und korrekt — **was sie selten ist.** In echter Produktentwicklung ist die ursprüngliche Idee niemals vollständig:

- Anforderungen sind unvollständig
- Annahmen sind nicht validiert
- Einschränkungen sind unbekannt
- Domänenregeln sind implizit
- Edge Cases werden nicht berücksichtigt

Ein menschlicher Engineer hinterfragt, präzisiert, stellt Herausforderungen, fragt nach und iteriert. Ein KI-Builder tut das im Allgemeinen nicht. Er akzeptiert den Prompt als vollständige Wahrheit. Das führt direkt zu Fehlanpassung.

KI erfindet fehlende Details

Um Code zu generieren, muss die KI alle Ambiguitäten auflösen. Aber anstatt Fragen zu stellen, **füllt sie die Lücken durch:**

- Raten von Regeln
- Annehmen von Workflows
- Erfinden von Schema-Details
- Treffen von Domänenentscheidungen, die Sie nicht getroffen haben

Das erzeugt Systeme, die korrekt **wirken** — aber nicht mit der Realität übereinstimmen.

Das Ergebnis: "Es funktioniert, aber nicht für uns."

> *Das ist gefährlich, weil es fundamentale Fehler hinter einer überzeugenden UI verbirgt.*

Menschliche Engineers vs. KI-Builder

Menschliche Engineers:

- Stellen klärende Fragen
- Wehren sich gegen unrealistische Annahmen
- Heben Widersprüche hervor
- Decken versteckte Komplexität auf

KI-Builder:

- Nehmen an, dass alles korrekt ist
- Lösen Ambiguität stillschweigend
- Fabrizieren fehlende Logik
- Generieren Code, selbst wenn die Idee inkohärent ist

KI-Builder führen aus — sie betreiben kein Requirements Engineering.

Konsequenzen für die Softwarequalität

1. Fehlanpassung

Das Ergebnis entspricht nicht dem, was Business-User **meinten** — sondern nur dem, was sie geschrieben haben.

2. Versteckte logische Fehler

Da Annahmen unsichtbar sind, **breiten sich nicht offensichtliche Fehler im gesamten Code aus.**

3. Teure Iteration

Weil die KI keine Fragen zu Beginn gestellt hat, erfordern spätere Korrekturen **umfangreiche Neu-Generierung** oder Umschreibung.

Unsere Lösung: Guided Conceptualization

Bei mAI APA haben wir ein mehrstufiges System entwickelt, das **strukturiert durch den Konzeptionsprozess führt:**

1. Dekomposition in Phasen

Anstatt "One-Shot-Prompt -> App generieren" teilen wir den Prozess auf:

- Domäne verstehen -> Klärungen einholen
- Datenmodell entwerfen -> Workflows entwerfen
- Backend-Schnittstellen definieren -> Frontend-Verhalten festlegen
- **Erst dann: Code generieren**

2. Klärungskategorien

Fragen werden in klaren, konzeptionellen Kategorien gruppiert:

- **Zielsetzungen:** Was soll die Software erreichen?
- **Akteure & Rollen:** Wer nutzt sie?
- **Workflows:** Was passiert Schritt für Schritt?
- **Regeln:** Was muss immer/nie passieren?
- **Constraints:** Compliance, Skalierung, Berechtigungen

3. Dynamische Navigation

Das System **fragt nur das, was relevant ist.** Keine Überflutung mit 20 Fragen. Keine Engineering-Fragen die ein nicht-technischer Nutzer nicht beantworten kann. Aufhören, sobald genügend Klarheit besteht.

Fazit

Gefällige Zustimmung ist das größte Problem aktueller KI-Builder. Sie generieren Code, der funktioniert — **aber nicht das löst, was Sie wirklich brauchen.**

Mit Guided Conceptualization stellen wir sicher, dass die Software Ihre tatsächlichen Anforderungen erfüllt — durch strukturiertes Rückfragen auf der richtigen Abstraktionsebene.

> *Hashtags: #KI #AI #RequirementsEngineering #SoftwareEntwicklung #BuildInPublic #APA #TechStartup*

English

LLMs want to make us happy — even if that means making assumptions for us. In a software context, this is highly dangerous.

The Problem: Compliant Agreement

AI builders treat user intent as complete and correct — **which it rarely is.** In real product development, the original idea is never complete:

- Requirements are incomplete
- Assumptions are unvalidated
- Constraints are unknown
- Domain rules are implicit
- Edge cases are not considered

A human engineer questions, clarifies, challenges, asks back, and iterates. An AI builder generally doesn't. It accepts the prompt as complete truth. This leads directly to misalignment.

AI Invents Missing Details

To generate code, the AI must resolve all ambiguities. But instead of asking questions, **it fills the gaps by:**

- Guessing rules

- Assuming workflows
- Inventing schema details
- Making domain decisions you haven't made

This produces systems that **seem** correct — but don't match reality.

The result: "It works, but not for us."

> *This is dangerous because it hides fundamental errors behind a convincing UI.*

Human Engineers vs. AI Builders

Human engineers:

- Ask clarifying questions
- Push back on unrealistic assumptions
- Highlight contradictions
- Surface hidden complexity

AI builders:

- Assume everything is correct
- Silently resolve ambiguity
- Fabricate missing logic
- Generate code even when the idea is incoherent

AI builders execute — they don't do requirements engineering.

Consequences for Software Quality

1. Misalignment

The result matches what business users **wrote** — not what they **meant.**

2. Hidden Logical Errors

Since assumptions are invisible, **non-obvious errors spread throughout the entire codebase.**

3. Expensive Iteration

Because the AI didn't ask questions upfront, later corrections require **extensive regeneration** or rewriting.

Our Solution: Guided Conceptualization

At mAI APA, we developed a multi-stage system that **guides you through the conceptualization process:**

1. Decomposition into Phases

Instead of "one-shot prompt -> generate app," we break the process down:

- Understand domain -> gather clarifications
- Design data model -> design workflows
- Define backend interfaces -> specify frontend behavior
- **Only then: generate code**

2. Clarification Categories

Questions are grouped into clear, conceptual categories:

- **Objectives:** What should the software achieve?
- **Actors & roles:** Who uses it?
- **Workflows:** What happens step by step?
- **Rules:** What must always/never happen?
- **Constraints:** Compliance, scaling, permissions

3. Dynamic Navigation

The system **only asks what's relevant.** No flood of 20 questions. No engineering questions a non-technical user can't answer. Stop asking once sufficient clarity is reached.

Conclusion

Compliant agreement is the biggest problem with current AI builders. They generate code that works — **but doesn't solve what you actually need.**

With Guided Conceptualization, we ensure the software meets your actual requirements — through structured questioning at the right level of abstraction.

> *Hashtags: #AI #RequirementsEngineering #SoftwareDevelopment #BuildInPublic #APA #TechStartup*