Bitkom AK Software Engineering & Software Architektur
Datum: 3. März 2026 | **Ort:** Webkonferenz

Vibe Coding: Zwischen Prompt und Produktivcode

1. Vibe Coding in Practice: Context Engineering and Driving AI Toward Truth

Jan Bronicki, Software Engineer, Microsoft

Einleitung:

Der Vortrag beleuchtete aus persönlicher Erfahrung, wie KI-gestützte Softwareentwicklung primär als Context Engineering verstanden werden sollte. Die zentrale These: LLMs liefern probabilistische Annäherungen, keine garantierte Korrektheit – Ingenieure müssen das Ergebnis aktiv in Richtung objektiver Wahrheit steuern.

Key Takeaways:

- **LLMs optimieren Wahrscheinlichkeit, nicht Wahrheit:** Sie generieren statistisch plausible Outputs basierend auf Trainingsdaten (GitHub, Stack Overflow, Dokumentation), verstehen aber nicht die tatsächliche Architektur oder Wahrheit eines Systems.
- **Context Engineering statt Context Overload:** Quality over Quantity beim Kontext – zu viel Kontext kann die Präzision sogar verschlechtern. Die Qualität der Inputs und Constraints bestimmt, wie nah das Ergebnis an der Realität liegt.
- **Verification Loops sind essenziell:** Unit Tests, gute Dokumentation und CI/CD schaffen Boundaries für KI-Systeme. Tests und Feedback-Schleifen verwandeln Annäherungen in zuverlässige Ergebnisse.
- **Skill-Shift in der Entwicklung:** Weniger differenzierend werden Typing Speed, API-Memorisierung und Syntax Recall. Wichtiger werden Architectural Reasoning, das Definieren von Invarianten, Trade-off-Bewertung und präzises Context Engineering.
- **Judgment bleibt der Flaschenhals:** KI verstärkt starke Ingenieure, aber Verantwortlichkeit und Korrektheit bleiben menschliche Aufgaben. Die wichtigste Fähigkeit ist nicht Keybindings, sondern den KI-Assistenten richtig zu leiten.

2. Vom Prompt zum Produkt: Wie KI die Softwareentwicklung verändert

Tobias Wagner, Fellow, MaibornWolff

Einleitung:

Anhand einer Live-Demo und konkreter Projekterfahrungen zeigte der Vortrag, wie KI die Softwareentwicklung von der Idee zum Prototyp dramatisch beschleunigt. Dabei wurde die Unterscheidung zwischen »Vibe Coding« und »Agentic Engineering« herausgearbeitet und die organisatorischen Konsequenzen beleuchtet.

Key Takeaways:

- **Vibe Coding vs. Agentic Engineering:** Vibe Coding liefert schnelle Ergebnisse ohne tiefes Verständnis des Codes, führt aber oft zu schlechten Design-Entscheidungen. Agentic Engineering hingegen nutzt KI-Tools professionell mit Fokus auf Architektur, Qualitätskontrollen und automatisierte Tests.
- **Praxisbeispiel:** Entwicklung einer Access-97-Haushaltsbuch-Migration zu einer modernen Electron-App in nur 3 Tagen statt geschätzt 1 Monat. Die App umfasste Dashboard, Datenverwaltung, Import von Altdaten, Reporting und Abfrage-Editor.
- **Benchmark-Entwicklung:** Aktuelle Modelle können bereits Aufgaben lösen, für die Menschen im Schnitt mehrere Stunden bis Tage benötigen – Tendenz stark steigend.
- **Veränderte Team-Strukturen:** Teams werden kleiner, Full-Stack-Entwickler werden wertvoller als Spezialisten. Die Kommunikation wird zum Flaschenhals, nicht mehr die Implementierung. Scrum-Prozesse müssen angepasst werden: kürzere Iterationszyklen und direktere Kommunikation.
- **Organisatorische Herausforderungen:** Weiterbildung und Ausbildung, Infrastruktur und Datensouveränität, Kostenkontrolle, Sicherheit und Datenschutz sowie Reduzierung von Bürokratie sind zentrale Themen für Unternehmen.
- **Ausbildungsprogramm bei MaibornWolff:** Agentic Engineering Project Workshops (seit August 2025) und Agentic Coding Foundation School (seit Februar 2026) mit Rollout an 8 Standorten.

3. Agentic Development in der Praxis: AI Agents mit Instructions, MCP und Skills im Produktivprojekt

Luis Deppisch, Senior Consultant, MHP Management- und IT-Beratung

Einleitung:

Der Vortrag lieferte einen praxisnahen Erfahrungsbericht aus einem realen Softwareprojekt, in dem der Entwicklungsprozess durch einen AI Agent unterstützt wird. Über Instructions und Skills werden die Ergebnisse gezielt optimiert, während MCP den Zugriff auf Kontextwissen aus Confluence und Jira ermöglicht.

Key Takeaways:

- **Drei Ausbaustufen der KI-Nutzung:** Stufe 1 (Ask Mode ohne Kontext) liefert Halluzinationen und falsche Klassen. Stufe 2 (Agent Mode) verbessert die Ergebnisse durch iterative Ausführung. Stufe 3 (Agent Mode + Instructions + MCP) liefert korrekte Validierung mit nur noch 30 – 45 Minuten Nacharbeit.
- **Instructions als »Rules of the Game«:** Definieren Coding Standards, Architekturspezifikationen, Reihenfolge und Do's and Don'ts in Markdown-Dateien (.github/copilot-instructions). Der Agent bezieht diese bei seinen Entscheidungen ein.
- **MCP (Model Context Protocol):** Ermöglicht Agents den Zugriff auf externe Systeme wie Jira, Confluence, APIs und Datenbanken. Mittels MCP kann auf aktuelle Versionen von Frameworks zugegriffen werden (z. B. Context7).
- **Skills für spezialisierte Fähigkeiten:** Skills erweitern Agenten um bestimmte Fähigkeiten wie z. B. technische Reviews. Sequential Thinking verbessert die Schritt-für-Schritt-Ausführung.

Best Practices: Prompt-Commands nutzen (/fix, /explain, /plan), verschiedene Modelle testen, Agent Mode für MCP aktivieren, Instructions schlank halten und auf gute Dokumentation im Projekt achten.

4. From AI Coding to AI Engineering: Why Design-Driven Development is the next Productivity Leap

Dr. Gerald Gassner, CEO, Knowis AG

Einleitung:

KI-Coding-Assistenten sind heute Standard und liefern lokale Produktivitätsgewinne. Bei der Skalierung entstehen jedoch Probleme wie inkonsistente Ergebnisse, architektonischer Drift und hoher Koordinationsaufwand. Der Vortrag argumentierte, dass Organisationen von prompt-getriebenem Codieren zu design-getriebener Entwicklung wechseln müssen.

Key Takeaways:

- **Das Problem:** Ohne klar definierten, teamübergreifend abgestimmten Kontext liefern KI-Tools inkonsistente Ergebnisse. Fehlender Kontext führt zu Scope-Drift, Nacharbeit und reduziertem Effizienzgewinn.
- **Shared Design Intent als Lösung:** Kollaborative, multi-team, multi-level Erarbeitung eines konsistenten Design Intents. Shift Left: Mehr Zeit im Design und in der Abstimmung vor der Implementierung investieren.
- **Design Graph:** Vernetzte, semantisch eindeutige Repräsentation aller Design-Entscheidungen, die sowohl für Menschen als auch AI-Agents zugänglich ist. Implementierungsvorschriften werden direkt im Modell definiert (Shift Down).

- **Drei-Phasen-Workflow (Idea-to-Code):** 1. Architect Phase (Strukturelle Dekomposition, C4-Notation), 2. Design Phase (Datenmodelle, Ablaufdiagramme, Spezifikationen), 3. Code Phase (AI-Agents greifen direkt auf Design Graph zu).
- **Ausblick: Erwartung für 2026:** Remote Agents werden Design-Änderungen automatisch implementieren, Tests anpassen und Merge Requests erstellen.

Zusammenfassung

Die Sitzung verdeutlichte, dass »Vibe Coding« als Einstieg in die KI-gestützte Entwicklung dient, für den produktiven Einsatz jedoch ein Paradigmenwechsel hin zu Agentic Engineering und Design-Driven Development erforderlich ist. Die vier Vorträge zeigten einen klaren Reifegradpfad: Von der grundlegenden Erkenntnis, dass LLMs Context Engineering und Verification Loops benötigen (Bronicki), über die praktischen Auswirkungen auf Teamstrukturen und Rollen (Wagner), hin zu konkreten Werkzeugen wie Instructions, MCP und Skills für den Projektalltag (Deppisch), bis zur unternehmensweiten Vision eines Design-Driven Development mit geteiltem Design Intent (Gassner). Die zentrale Erkenntnis: Der Mensch bleibt der entscheidende Faktor – nicht als Code-Schreiber, sondern als Architekt, Reviewer und Qualitätsgarant der KI-gestützten Softwareentwicklung.

Felix Ansmann
Bereichsleiter Software & IT-Services
T 030 27576-098
f.ansmann@bitkom.org