SPEAKER

Stefan Diefenbacher

Gesellschafter Profile GmbH · Winterthur · Wirtschaftsinformatiker & Software Entwickler.

ROLLE	**Gesellschafter Profile GmbH** Winterthur · profile.ch
HINTERGRUND	**Wirtschaftsinformatiker** Software Entwickler
OMNIS	**8 Jahre Entwicklung**
AKTUELL	**AI Business Specialist** in Ausbildung

// jetzt_fokus > learn(agentic_engineering) > build(omnis_context_layer) > ship(rag + mcp + json) // frage > wie wird omnis agentenfähig?

KONTAKT

web	www.profile.ch
github	github.com/SDI76
linkedin	linkedin.com/in/stefan-diefenbacher

KI · ZWISCHEN HYPE UND UMBRUCH

Wie agentische Entwicklung Software verändert — und was das für Omnis bedeutet.

WARUM JETZT

Omnis war immer ein Produktivitätswerkzeug.

Agentische Entwicklung ist keine Gegenbewegung dazu, aber die nächste Frage, die sich stellt.

Wie wird ein schnelles Entwicklungswerkzeug selbst agentenfähig?

Omnis ist visuell, integriert, schnell und sicher. Business-Anwendungen entstehen mit wenig Reibung. Die Frage ist wie wir RAD-Stärke mit Kontext, Tools, Tests und Freigaben verbinden.

```
// omnis.scan() rad = true business_ctx = high ai_ready = not_yet // frage >
verbinde(rad, ai, context, tools, tests)
```

VON COPILOT ZU AGENTEN

Die Arbeit verschiebt sich in den Workflow.

Der Fortschritt liegt nicht nur in den Modellen, sondern darin, wie nah KI an Codebase, Tools, Tests und echte Entwicklungsprozesse kommt.

Autocomplete

Copilot schlägt die nächste Zeile vor. KI sitzt neben dem Cursor.

Antwort

Chat klärt Fragen, kennt aber kein Projekt. KI sitzt im Chat.

Kontext

Projektwissen, Templates, Arbeitsräume. KI hilft präziser.

Handeln

Dateien lesen, Tools nutzen, Tests ausführen. KI arbeitet im Workflow.

Control shifts from typing to steering.

VIBE CODING

Beeindruckend für Prototypen. Riskant als Produktionsmodell.

01 / IDEE

Ein Gedanke, ein Prompt, eine schnelle Demo.

Niemand muss die Architektur kennen, um zu starten.

02 / PROMPT

Das Modell improvisiert eine mögliche Lösung.

Plausibel klingender Code, der oft funktioniert — manchmal.

03 / DEMO

Es läuft, aber Risiken bleiben unsichtbar.

Niemand versteht den Code. Wartbarkeit ist eine offene Frage.

Schnell ist nicht dasselbe wie wartbar und sicher.

FAILURE MODES

Was KI heute nicht zuverlässig kann.

Stark bei klaren, begrenzten Aufgaben. Gefährlich dort, wo Kontext fehlt, Ergebnisse plausibel klingen und niemand systematisch prüft.

01 **Kontextarmut** · falsche Annahmen über Architektur, Datenmodell und Nebenwirkungen.

02 **Legacy-Wissen** · implizite Regeln, alte Patterns und Fachlogik bleiben unsichtbar.

03 **Fehlende Tests** · kein schnelles Signal, ob die Änderung wirklich funktioniert.

04 **Plausible Fehler** · Outputs wirken sauber, sind aber fachlich oder technisch falsch.

05 **Verantwortung** · Freigabe, Datenschutz und Risiko bleiben beim Menschen.

Je größer System und Risiko, desto wichtiger Kontext, Tests, Review und Grenzen.

HYPE SIGNAL ANALYZER

Die produktive Zone liegt nicht an den Extremen.

Diskussionen pendeln zwischen Euphorie und Ablehnung. Engineering passiert rechts davon — begrenzt, prüfbar, nützlich.

HYPE	ERNÜCHTERUNG	PRODUKTIVE NUTZUNG
„Ein Prompt ersetzt ein Entwicklerteam."	„KI macht nur kaputten Code."	KI spart Zeit bei *klaren, prüfbaren* Aufgaben.
Unrealistisch · überschätzt Autonomie	Defensiv · ignoriert reale Use Cases	Begrenzt · kalibriert · überprüft

> hype.detect() → level = unstable · > use_case.check() → calibration required

USE CASE MAP

Wo KI in der Entwicklung heute stark ist.

PRODUCE **Erzeugen** Code-Generierung · Boilerplate · API-Integration	IMPROVE **Verbessern** Refactoring · Unit-Tests · Dokumentation
UNDERSTAND **Verstehen** Codeverständnis · Architekturvorschläge · RAG über Codebasen	OPERATE **Betreiben** Agentic Workflows · DevOps · CI/CD Automatisierung

Stark, wenn die Aufgabe heruntergebrochen und das Ergebnis prüfbar ist.

AGENTIC WORKFLOW

Entwicklung verschiebt sich zu begrenzten Aufgabenketten.

Nicht nur Vorschläge — klar abgegrenzte Arbeitspakete mit Tools, Tests und Review.

01	02	03	04	05	06
Anforderung	**Architektur**	**Code**	**Tests**	**Build**	**PR**
analysieren	planen	erstellen	schreiben	prüfen	erstellen

```
// task > fix failing checkout tests > read files > edit code > run tests > report diff for review
```

OUTPUT

Developer reviews diff, logs, tests und intent — nicht jede einzelne Zeile.

ABSTRAKTIONSSPRUNG

Die nächste Abstraktionsschicht.

Wir hören nicht auf zu programmieren. Wir verschieben die Arbeit — weg von einzelnen Befehlen, hin zur präzisen Steuerung einer Maschine, die Code erzeugt.

EBENE 01	EBENE 02	EBENE 03 / JETZT
Assembler	**Hochsprachen**	**Agentic Engineering**
Ich sage der Maschine fast jeden Schritt.	Ich beschreibe Absicht in lesbaren Strukturen.	Ich programmiere den Code-schreibenden Prozess.

Der neue Programmierer schreibt weniger Code direkt — aber programmiert die Bedingungen, unter denen guter Code entsteht.

CONTEXT PIPELINE

Kontext ist die neue Produktivitätsgrenze.

KI-Systeme sind nur so gut wie der Kontext, den sie nutzen können.

CODE	DOCS	ARCH	AGENTS
Lesbarkeit	**Aktualität**	**Entscheide**	**Verteilung**
verständliche Struktur, sprechende Namen.	auffindbar, gepflegt, einheitlich.	explizit dokumentiert, nicht implizit.	strategische Aufgabenzuordnung.

Garbage in → plausible garbage out.

TEAM OS

Custom Agents und Skills machen KI teamfähig.

Nicht ein universeller Agent — sondern wiederholbare Rollen, Rechte und Workflows.

PLAN

Liest nur. Klärt Anforderungen, entwirft Vorgehen.

tools: search, web

BUILD

Setzt den Plan um, editiert Dateien, lässt Tests laufen.

tools: edit, terminal

REVIEW

Prüft Qualität, Security, Risiken und Tests.

tools: diff, search

HANDOFFS

Planung → Implementierung → Review

SKILLS

Spezialisierte Fähigkeiten werden bei Bedarf geladen: Testing, Debugging, Deployment.

QUALITY GATE

Feedback-Loops statt blindem Vertrauen.

KI braucht Prüfmechanismen: Tests, Reviews, Analyse, Laufzeitfeedback und Freigabe.

01

Implement

Agent setzt um, begrenzt auf eine klare Aufgabe.

02

Verify

Unit-Tests, Build, statische Analyse laufen automatisch.

03

Fix

Bei Signalfehlern korrigiert der Agent oder eskaliert.

04

Review

Mensch prüft Diff, Logs, Intent und Risiken.

05

Approve

Freigabe als bewusste, dokumentierte Handlung.

Vertrauen entsteht durch Prüfung, nicht durch Glauben.

DEVELOPER ROLE

Die Rolle des Entwicklers verschiebt sich.

Entwickler verschwinden nicht. Die wertvollsten Tätigkeiten verschieben sich.

WENIGER	MEHR
Codeproduktion	Spezifikation
Repetitive Standardlogik	Architekturdenken
Boilerplate	Review & QS
Copy-Paste-Prompting	Tooling & Kontextgrenzen

> role.update() coder → engineer → orchestrator

PRODUCTIVITY ADVANTAGE

Omnis hat bereits einen Produktivitätsvorteil.

KI sollte diesen Vorteil erweitern — nicht ersetzen.

OMNIS RAD

visuell

schnell

integriert

Businesslogik, UI und Daten in einem Werkzeug.

+ KI LAYER

Analyse

Dokumentation

Tests & Review-Hilfe

Automatisierung entlang bestehender Stärke.

extend(omnis) · replace = false · amplify = true

AGENT ACCESS

Die KI-Herausforderung für Omnis.

KI kann Omnis nicht nativ zuverlässig — Modelle haben Omnis-Syntax und Patterns kaum gelernt. Agenten brauchen Export, Dokumentation und sichere Schnittstellen.

STATE	BRIDGE 01	BRIDGE 02	BRIDGE 03
Black Box	**JSON Export**	**RAG / Docs**	**API / MCP**
visuell, nicht modell-nativ — KI kann es nicht lesen.	lesbarer Code gibt Modellen Kontext.	Kontext und Wissen über Syntax und Patterns.	sicheres, begrenztes Handeln zurück ins Studio.

Agentenfähigkeit = lesen + verstehen + validieren + begrenzt handeln.

MARKTDRUCK · OUTLOOK

Time-to-Market verschiebt sich.

KI-native Teams können in sehr kurzer Zeit brauchbare Produktvarianten bauen. Dadurch gerät nicht nur Entwicklungsarbeit unter Druck, sondern das bisherige Tempo des Marktes.

KLASSISCHER PROJEKTTAKT

Analyse

Anforderungen klären

Design

UI & Datenfluss abstimmen

Entwicklung

bauen, testen, ausliefern

→ Wochen / Monate

KI-NATIVE LOOP

> spec / prompt > prototype > design variants > release candidate

→ Tage / Wochen

Bestehende Kunden bleiben vermutlich. Die offene Frage ist, ob neue Projekte Omnis noch evaluieren.

RISIKO-HEATMAP

Welche Omnis-Leistungen geraten unter Druck?

Am stärksten betroffen: standardisierte, sichtbare und schnell imitierbare Leistungen.

BEREICH	RISIKO	RELEVANZ
Standard-UI, CRUD, einfache Workflows	● Hoch	Gut beschreibbar, schnell generierbar, leicht vergleichbar.
Reports, Dashboards, kleinere Erweiterungen	● Hoch	Kurze Spezifikationen reichen oft für brauchbare Versionen.
Webbasierte Kundenportale	● Hoch+	Frontend, Design und Iteration werden besonders stark beschleunigt.
Fatclient oder Hybrid-Clients	○ Mittel	Desktop heißt heute oft: Web-UI, lokale Dateien, Backend-Zugriff.

Schutzfaktoren: Domänenwissen, installierte Basis, Vertrauen, Datenverständnis, sichere Integration.

OMNIS KI GAP

Omnis Dev ist nicht KI-ready. Noch nicht.

Kein Modell kann nativ zuverlässig Omnis-Code schreiben. Und Omnis Studio bringt heute keine eigene KI-Integration mit.

REALITÄT 01

Syntax und Commands sind nicht modell-nativ.

REALITÄT 02

Code liegt im Studio, nicht im Agenten-Workflow. Tests nur in Studio möglich.

REALITÄT 03

Referenzen, Guides und Kontext sind verteilt.

STRATEGIE

Wir bauen eine externe Kontextschicht , mit der VS-Code-Agenten Omnis-Arbeit heute schon lernen können, und machen Omnis für das Modell verstehbar.

TRANSLATION LAYER

Die Übersetzungsschicht für Omnis.

MCP, RAG und JSON-Pipeline machen Omnis-Code für VS-Code-Agenten lesbar und bearbeitbar.

SOURCE	MCP	JSON	RAG
Omnis Studio	**Schnittstelle**	**Pipeline**	**Knowledge**
Klassen, Methoden, Variablen, bestehender Code.	Code-Schnittstelle zu Omnis · Aktionen zurück.	Skripte schreiben Omnis-kompatiblen JSON-Export.	Syntax, Commands, Guides, Framework, Businesslogik.

```
// agent.context { structure: MCP, knowledge: RAG , export: JSON }
```

OUTPUT

VS-Code-Agenten verstehen Kontext und erzeugen *kontrollierte* Änderungen.

LEARNING LAYER

Kein Workaround. Ein Experimentierfeld.

Alles, was wir extern mit VS Code aufbauen, wird später wertvoll — wenn Omnis native KI-Integration nachzieht.

HEUTE LERNEN WIR	MORGEN ÜBERTRAGEN WIR
Code verstehen	**Tasks** Analyse, Review & Änderung wiederholbar.
Dokumentation	**Skills** Omnis-Code, SQL und Doku bündeln.
Diagramme erstellen	**Agenten** strategic_agents.md als Rollenkarte.
Architektur	**Guardrails** Regeln für Kontext, Freigaben, Änderungen.
Fehler finden	
Einfache Änderungen	

Nicht warten — sondern vorbereitet sein.

OPERATING MODEL

Tasks starten Arbeit. Skills machen sie wiederholbar.

Task = Auftrag · Agent = Rolle · Skill = Methode · AGENTS.md = Projektregeln.

ROUTING · AGENTS.md

```
if task.agent == omnis-docs-agent : read(agent_specific.md) use(omnis.skills)


// agent_specific.md beschreibt Denken und Prioritäten eines Agenten.
```

DREI EBENEN

AGENTS.md = Regeln agent_specific.md = Rolle Skills = Methode

BEISPIEL-SKILLS · OMNIS-REPO

- **omnis-code-documentation**
  Methoden, Klassen, Workflows
- **omnis-naming-review**
  Variablen, Tasks, Libraries, Klassen
- **omnis-variable-model**
  Scopes, Lists/Rows, Object & Field refs
- **omnis-agent-context-pack**
  welche Dateien zuerst lesen
- **omnis-review-checklist**
  Doku, Typen, Globals, Effekte, Tests

CONTEXT LAYER · 01 / 02

RAG findet Wissen.

Seit 2020 · Lewis et al., NeurIPS · Wissen aus Dokumenten, Code und Tickets in die Antwort holen — ohne das Modell neu zu trainieren.

WAS

Index

Omnis-Code, Framework, Domänenwissen, Omnis Referenz, Fachwissen

Retrieval

semantisch, hybrid oder gefiltert.

Nutzen

aktueller Kontext ohne Modelltraining.

KOMMUNIKATION

// rag.flow > Frage > Retriever / Index > passende Chunks > Prompt-Kontext > Antwort

RAG macht Omnis-Wissen auffindbar.

CONTEXT LAYER · 02 / 02

MCP verbindet Werkzeuge.

Seit 25.11.2024 · Anthropic Open Standard · ein einheitliches Protokoll für Tools, Datenquellen und Aktionen — statt N Einmal-Connectoren.

WAS

Server

exponieren Tools, Resources, Prompts.

Protokoll

Kommunikation per JSON-RPC 2.0.

Nutzen

weniger Einmal-Connectoren, mehr Standard.

KOMMUNIKATION

// mcp.flow > Agent / Host > MCP Client > JSON-RPC > MCP Server > Git · DB · Files · Studio · APIs

MCP macht Omnis-Systeme KI bedienbar.

DEMO BOUNDARY

Was die Demo zeigen soll.

Die Demo ist kein Autopilot-Versprechen. Sie zeigt einen begrenzten, einfachen Workflow: Omnis-Artefakte werden lesbar, analysierbar und anschlussfähig.

ZEIGT

› KI analysiert Omnis-Code im JSON-Export und dokumentiert ihn.

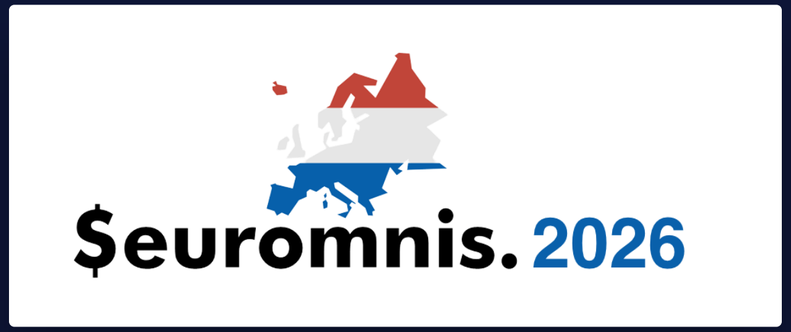
› Geänderte Klassen werden über MCP zurück in die Omnis IDE geschrieben.

› Agent greift per RAG auf Omnis-Doku und Syntaxwissen zu.

// demo.assert > scope = **bounded** > output = **reviewable** > autonomy = **limited** // ok: useful != magic

Evidence, not magic.

→ FORTSETZUNG FOLGT



Mehr im Detail — auf der Bühne.

An der EuroOmnis 2026 gehe ich tiefer auf agentische Workflows und konkrete Ansätze zur Omnis-Integration ein — und zeige, wie wir das bei **Profile** einführen.

WAS EUCH DORT ERWARTET

01 **Agentic Engineering im Detail**

Vom Prompt zur kontrollierten Aufgabenkette

02 **Ansätze zur Omnis-Integration**

MCP-Brücke, RAG-Layer — wie Omnis-Code für Agenten editierbar wird.

03 **Wie wir das bei Profile umsetzen**

Live-Beispiele, Lessons Learned und der Weg vom Experiment zum Workflow.

SEE YOU AT EUROMNIS 2026 _