SPEAKER

Stefan Diefenbacher

Partner at Profile GmbH · Winterthur · business informatics & software engineer.

ROLE	**Partner · Profile GmbH** Winterthur · profile.ch
BACKGROUND	**Business Informatics** Software engineer
OMNIS	**8 years of development**
CURRENTLY	**AI Business Specialist** in training

// current_focus > learn(agentic_engineering) > build(omnis_context_layer) > ship(rag + mcp + json) // question > how does omnis become agent-ready?

CONTACT

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

AI · BETWEEN HYPE AND SHIFT

How agentic development is changing software — and what it means for Omnis.

WARUM JETZT

Omnis has always been a productivity tool.

Agentic development isn't a counter-movement to that but the next question is.

How does a fast development tool itself become agent-ready?

Omnis is visual, integrated, fast and safe. Business applications take shape with little friction. The question is how we combine RAD strength with context, tools, tests and approvals.

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

FROM COPILOT TO AGENTS

Work is shifting into the workflow.

Progress isn't just in the models — it's in how close AI gets to the codebase, tools, tests, and real development processes.

Autocomplete

Copilot suggests the next line. AI sits next to the cursor.

Answer

Chat answers questions but has no project knowledge. AI sits in the chat.

Context

Project knowledge, templates, workspaces. AI helps more precisely.

Act

Read files, use tools, run tests. AI works inside the workflow.

Control shifts from typing to steering.

VIBE CODING

Impressive for prototypes. Risky as a production model.

01 / IDEA

A thought, a prompt, a quick demo.

No one needs to know the architecture to start.

02 / PROMPT

The model improvises a possible solution.

Plausible-sounding code that often works — sometimes.

03 / DEMO

It runs — but risks stay invisible.

No one understands the code. Maintainability is an open question.

Fast is not the same as maintainable and safe.

FAILURE MODES

What AI cannot reliably do today.

Strong on clear, bounded tasks. Dangerous where context is missing, results sound plausible, and no one systematically verifies.

01 **Context starvation** · wrong assumptions about architecture, data model and side effects.

02 **Legacy knowledge** · implicit rules, old patterns and domain logic stay invisible.

03 **Missing tests** · no fast signal whether a change actually works.

04 **Plausible errors** · outputs look clean but are technically or factually wrong.

05 **Responsibility** · approval, privacy and risk stay with humans.

The bigger the system and risk, the more important context, tests, review and limits become.

HYPE SIGNAL ANALYZER

The productive zone is not at the extremes.

Discussions swing between euphoria and rejection. Engineering happens to the right of that — bounded, verifiable, useful.

HYPE	DISILLUSIONMENT	PRODUCTIVE USE
"One prompt replaces a development team."	"AI only writes broken code."	AI saves time on *clear, verifiable* tasks.
Unrealistic · overestimates autonomy	Defensive · ignores real use cases	Bounded · calibrated · verified

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

USE CASE MAP

Where AI is strong today in development.

PRODUCE **Produce** Code generation · boilerplate · API integration	IMPROVE **Improve** Refactoring · unit tests · documentation
UNDERSTAND **Understand** Code comprehension · architecture proposals · RAG over codebases	OPERATE **Operate** Agentic workflows · DevOps · CI/CD automation

Strong when the task is broken down and the result is verifiable.

AGENTIC WORKFLOW

Development is shifting toward bounded task chains.

Not just suggestions — clearly bounded work packages with tools, tests and review.

01	02	03	04	05	06
Requirement	**Architecture**	**Code**	**Tests**	**Build**	**PR**
analyze	plan	create	write	verify	open

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

OUTPUT

Developer reviews diff, logs, tests and intent — not every single line.

ABSTRAKTIONSSPRUNG

The next abstraction layer.

We don't stop programming. We shift the work — away from individual instructions, toward precise control of a machine that generates code.

LAYER 01	LAYER 02	LAYER 03 / NOW
Assembler	**Hochsprachen**	**Agentic Engineering**
I tell the machine almost every step.	I describe intent in readable structures.	I program the code-writing process.

The new programmer writes less code directly — but programs the conditions under which good code emerges.

CONTEXT PIPELINE

Context is the new productivity ceiling.

AI systems are only as good as the context they can use.

CODE	DOCS	ARCH	AGENTS
Readable	**Current**	**Decisions**	**Routing**
clear structure, expressive names.	findable, maintained, consistent.	documented explicitly, not implicit.	strategic task assignment.

Garbage in → plausible garbage out.

TEAM OS

Custom agents and skills make AI team-capable.

Not one universal agent — but repeatable roles, permissions and workflows.

PLAN

Read-only. Clarifies requirements, drafts approach.

tools: search, web

BUILD

Executes the plan, edits files, runs tests.

tools: edit, terminal

REVIEW

Reviews quality, security, risks and tests.

tools: diff, search

HANDOFFS

Planning → Implementation → Review

SKILLS

Specialized skills load on demand: testing, debugging, deployment.

QUALITY GATE

Feedback loops instead of blind trust.

AI needs verification: tests, reviews, analysis, runtime feedback and approval.

01

Implement

Agent executes, scoped to one clear task.

02

Verify

Unit tests, build, static analysis run automatically.

03

Fix

On failure signals the agent corrects or escalates.

04

Review

Human reviews diff, logs, intent and risks.

05

Approve

Approval as a deliberate, documented act.

Trust comes from verification, not belief.

DEVELOPER ROLE

The developer's role is shifting.

Developers don't disappear. The most valuable activities shift.

LESS	MORE
Code production	Specification
Repetitive standard logic	Architectural thinking
Boilerplate	Review & QA
Copy-paste prompting	Tooling & context boundaries

> role.update() coder → engineer → orchestrator

PRODUCTIVITY ADVANTAGE

Omnis already has a productivity advantage.

AI should extend this advantage — not replace it.

OMNIS RAD

visual

fast

integrated

Business logic, UI and data in one tool.

+ AI LAYER

Analysis

Documentation

Tests & review support

Automation along existing strengths.

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

AGENT ACCESS

The AI challenge for Omnis.

AI can't handle Omnis natively — models have barely learned Omnis syntax and patterns. Agents need export, documentation and safe interfaces.

STATE	BRIDGE 01	BRIDGE 02	BRIDGE 03
Black Box	**JSON Export**	**RAG / Docs**	**API / MCP**
visual, not model-native — AI can't read it.	readable code gives models context.	context and knowledge about syntax and patterns.	safe, bounded action back into the Studio.

Agent-readiness = read + understand + validate + act with limits.

MARKET PRESSURE · OUTLOOK

Time-to-market is shifting.

AI-native teams can build usable product variants in very short time. That puts pressure not just on development work, but on the market's prior pace.

CLASSIC PROJECT CADENCE	AI-NATIVE LOOP
Analyze Clarify requirements **Design** Align UI & data flow **Build** build, test, ship → weeks / months	> spec / prompt > prototype > design variants > release candidate → days / weeks

Existing customers likely stay. The open question is whether new projects still evaluate Omnis.

RISIKO-HEATMAP

Which Omnis offerings are under pressure?

Most affected: standardized, visible and quickly imitable work.

AREA	RISK	WHY IT MATTERS
Standard UI, CRUD, simple workflows	● High	Easy to describe, fast to generate, easy to compare.
Reports, dashboards, small extensions	● Hoch	Short specs often suffice for usable versions.
Web-based customer portals	● High+	Frontend, design and iteration accelerate especially fast.
Fat client or hybrid clients	○ Medium	Today "desktop" often means: web UI, local files, backend access.

Protective factors: domain knowledge, installed base, trust, data understanding, safe integration.

OMNIS AI GAP

Omnis Dev isn't AI-ready. Not yet.

No model can write Omnis code natively and reliably. And Omnis Studio doesn't ship with its own AI integration today.

REALITY 01

Syntax and commands aren't model-native.

REALITY 02

Code lives in Studio, not in the agent workflow. Tests only possible in Studio.

REALITY 03

References, guides and context are scattered.

STRATEGY

We build an external context layer so VS Code agents can learn Omnis work today — and make Omnis legible to the model.

TRANSLATION LAYER

The translation layer for Omnis.

MCP, RAG and a JSON pipeline make Omnis code readable and editable for VS Code agents.

SOURCE	MCP	JSON	RAG
Omnis Studio	**Interface**	**Pipeline**	**Knowledge**
classes, methods, variables, existing code.	code interface to Omnis · actions back.	scripts write Omnis-compatible JSON export.	syntax, commands, guides, framework, business logic.

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

OUTPUT

VS Code agents understand context and produce *controlled* changes.

LEARNING LAYER

Not a workaround. An experimental field.

Everything we build externally with VS Code becomes valuable later — when Omnis catches up with native AI integration.

TODAY WE LEARN	TOMORROW WE TRANSFER
Understand code	**Tasks** Analysis, review & change repeatable.
Documentation	**Skills** Bundle Omnis code, SQL and docs.
Create diagrams	**Agents** strategic_agents.md as role card.
Architecture	**Guardrails** Rules for context, approvals, changes.
Find errors	
Simple changes	

Don't wait — be prepared.

OPERATING MODEL

Tasks start work. Skills make it repeatable.

Task = job · Agent = role · Skill = method · AGENTS.md = project rules.

ROUTING · AGENTS.md

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


// agent_specific.md describes an agent's thinking and priorities.
```

THREE LAYERS

AGENTS.md = rules agent_specific.md = role Skills = method

EXAMPLE SKILLS · OMNIS REPO

- **omnis-code-documentation**
  methods, classes, workflows
- **omnis-naming-review**
  variables, tasks, libraries, classes
- **omnis-variable-model**
  scopes, lists/rows, object & field refs
- **omnis-agent-context-pack**
  which files to read first
- **omnis-review-checklist**
  docs, types, globals, side effects, tests

CONTEXT LAYER · 01 / 02

RAG finds knowledge.

Since 2020 · Lewis et al., NeurIPS · Pull knowledge from documents, code and tickets into the answer — without retraining the model.

WHAT

Index

Omnis code, framework, domain knowledge, Omnis reference, expertise

Retrieval

semantic, hybrid or filtered.

Benefit

current context without model training.

COMMUNICATION

// rag.flow > Question > Retriever / Index > matching chunks > prompt context > Answer

RAG makes Omnis knowledge findable.

CONTEXT LAYER · 02 / 02

MCP connects tools.

Since 25 Nov 2024 · Anthropic open standard · one shared protocol for tools, data sources and actions — instead of N one-off connectors.

WHAT

Servers

expose tools, resources, prompts.

Protocol

communication via JSON-RPC 2.0.

Benefit

fewer one-off connectors, more standard.

COMMUNICATION

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

MCP makes Omnis systems AI operable.

DEMO BOUNDARY

What the demo is meant to show.

The demo isn't an autopilot promise. It shows a bounded, simple workflow: Omnis artifacts become readable, analyzable and connectable.

SHOWS

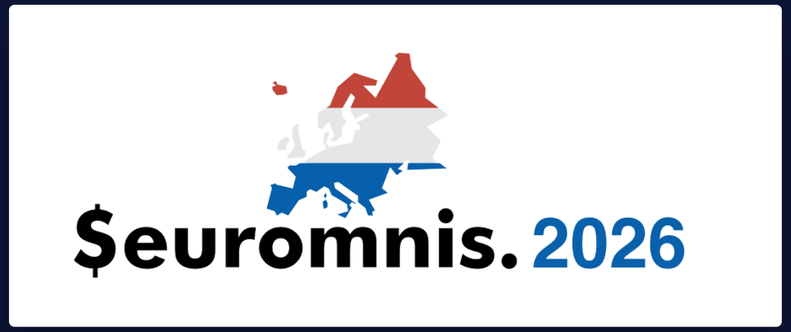
› AI analyzes Omnis code in JSON export and documents it.

› Changed classes are written back into the Omnis IDE via MCP.

› Agent accesses Omnis docs and syntax knowledge via RAG.

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

Evidence, not magic.



More detail — on stage.

At EuroOmnis 2026 I'll go deeper into agentic workflows and concrete approaches to Omnis integration — and show how we're rolling it out at **Profile**.

WHAT TO EXPECT THERE

01 **Agentic Engineering in detail**

From prompt to controlled task chain

02 **Approaches to Omnis integration**

MCP bridge, RAG layer — how Omnis code becomes editable to agents.

03 **How we're rolling this out at Profile**

Live examples, lessons learned, and the path from experiment to workflow.

SEE YOU AT EUROMNIS 2026 _