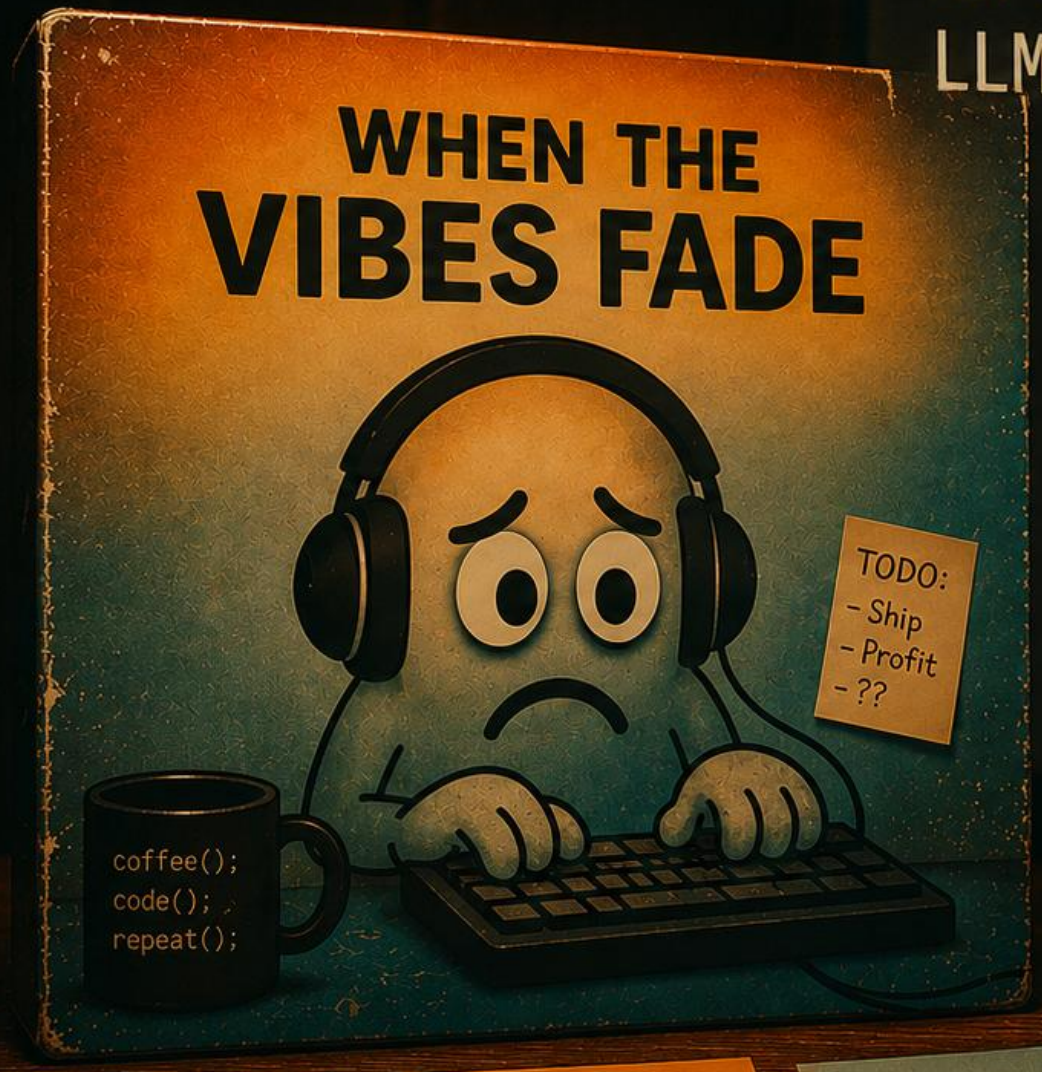
An Industry Perspective on LLM Coding Limits



- Illusion of Control
- Technical Debt
- Lack of Understanding
- Scaling Problems
- Security Risks

```
def vibe():
    prompt = "build something amazing"
    while vibes:
        code = ai.generate(prompt)
        if not error:
            ship(code)
        else:
            ignore()
    return "done"
```

vibe > plan?

It works on my machine.

Just one more prompt...

Why is nothing working?!

WHO IS SPEAKING

A position piece & question — not a solution

Where I come from

Thomas Mahringer — schorn.ai/mindruptive.com

Industrial software engineering, NOT formal methods.

I contribute empirical observations and a problem framing — not a solution.

My Question

Where can symbolic computation and formal methods connect to industrial coding workflows?

From pragmatic guardrails to formal methods.

EMPIRICAL BACKGROUND

What we actually built

> 1,5 yrs

hundreds of hours of vibe-coding, production projects

Use cases

- Self-hosted orchestration platform for local LLMs
  (air-gapped, chat, documents/knowledge (RAG), coding, transcription, cluster, MCP, SSO, …)
- **FASTCODE — a deterministic application generator**
  Abstraction layer between data schema and the vibe-coded code, 80% structurally guaranteed output.
  - crewHeaven.com workforce-housing marketplace MVP
  - **CRM**
  - **Product Information System** + complete Shopify automation & AI image generation

Tools

Cursor · RooCode · Claude Code · Antigravity · GitHub Copilot

THE PROBLEM

The senior-developer bottleneck

~75%

of a junior developer's time saved by vibe coding

~1/3

of those savings eaten by senior review, debug & fix

The senior becomes a “vibe checker”

“Manually” (through Prompts, MCP, …) checking guardrails & global coherence, ensuring the code aligns with broader architectural constraints.

➔ **Reviewer must be way more experienced than the AI-assisted developer**

➔ **AI makes good architect-developers better and bad developers worse.**

WHY IT HAPPENS

Probabilistic systems meet deterministic requirements

The line isn't easy vs. hard. It's **local vs. global scope.**

✓ Local — LLMs excel

Implement a function

Design an algorithm

Write complex generic types

Draft & refine a mapping chain

Full context in view → creative, fast,
only marginal corrections needed.

! Global — LLMs struggle

Layer separation and data structures across many files

No duplicated type declarations and code

System-wide invariants

Why?

Finite context + next-token objective ≠
system coherence over hundreds of files.

THE EVIDENCE

Nine recurring error classes

Observed repeatedly, across multiple LLMs — **not random bugs, not model-specific**.

Missing global coherence	Path of least resistance	Statistical / temporal bias	Beyond *functional* correctness
1 Structural invariants	3 Any-casting	4 Version mismatch	6 Complexity blindness
2 Type unification	9 Missing null guards		7 Arbitrary deletion
5 Code duplication			
8 Context loss (*Memento Effect*)			

1 Missing Global Coherence – Structural Invariants

What happened

A user-friendly cascading delete (dependency-tree-UX plus backend-logic).

Server-side in an atomic transaction.

LLM: separate non-transactional API calls in frontend.

Why it's hard for a token model

Local functional correctness ≠ global properties.

A locally correct block can still break the architecture.

4 Temporal Inconsistency - Version mismatch

What happened

Shopify GraphQL mutations generated in 2024 API syntax — although the system prompt said otherwise.

Older API versions dominate the training data.

Why it's hard for a token model

The model prefers the statistically more frequent patterns.
Pragmatic remedy: MCP reads the current documentation.

Beyond *functional* correctness

6 Complexity blindness

Shopify product ➔ collection mapping.

Nested loops over all products × all collections: O(n·m).

A precomputed index solves it in O(n+m).

The code is correct — it just scales badly.

Optimizes the next token, not algorithmic complexity.

7 Arbitrary deletion

On refactors, logging, debug output and tracing silently disappear.

Not property-preserving transformation.

LLM: “non-relevant” code.

The harness is just as important as the LLM

The four layers – Context Engineering

RAG & Agentic Tool Use

Bind all layers and let the model **investigate** —
grep + semantic search + AST navigation, agentically.

SPADE Repository Intelligence Graph: a deterministic map

SPADE extracts a **Repository Intelligence Graph (RIG)** — build & test topology, injected **before** the agent explores.

THE DIRECTION

A spectrum of guardrails: pragmatic → formal

Level 1 – Agentic – Retrieval-augmented generation

Enrich the context; Agentic MCP, (agentic) RAG (grep + semantic search), Graph-RAG

The model fetches what it lacks. Reduces errors (esp. 4 & 5).

Level 2 – Add deterministic information – LSP/AST/type inference · SPADE/RIG · Tomograph output

Use the language server information, like AST + type inference or SPADE Repository Intelligence Graph, to enrich context.

Level 1+2 prove nothing, still probabilistic.

Level 3 – Formal Description – DSL

A formal description like a domain-specific language adds constraints.

Level 4 – Vision – Symbolic computation

A symbolic reasoner proves what the others can't: semantic properties — equivalence, invariant preservation, complexity.

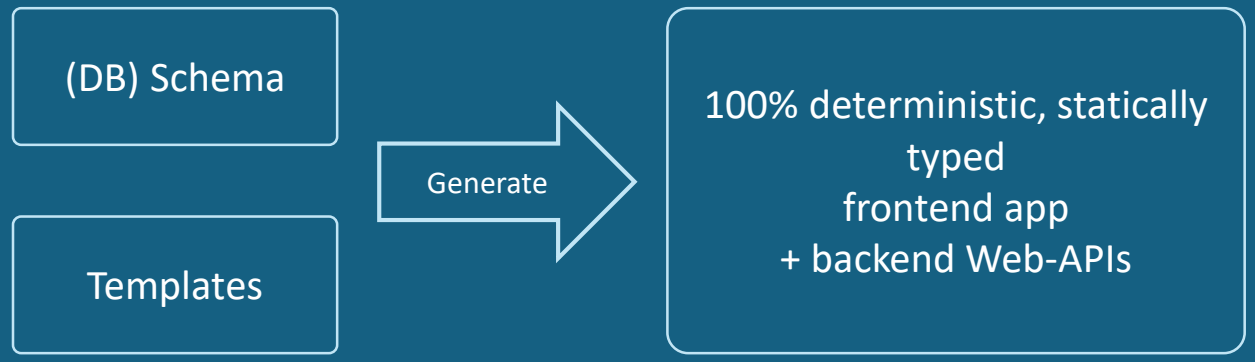
They hold for every execution. It turns "probably correct" into a guarantee.

A QUICK ASIDE FROM PRACTICE

A deterministic frame - FASTCODE

20% Vibe Coding
("Beautified" marketplace UX pages, custom-made functionality)

80% deterministically generated



Vibe coding moves up a level

Global Structural Integrity

Most Errors show at Build Time

THE VISION

From “Generate-then-check” loops to Guided Generation

1. Iterative generation: Enrich context, generate-then-check loop, “n+1” corrects “n”, guardrails 1-3

2. Constrained decoding: “Prune” non-conforming tokens, guardrails 2 & 3, loop “n” already emits correct code
   E.g., ETH LMQL + DOMINO: grammar-constrained decoding. Model produces conforming output.

3. Solution-space transformation: Reshape/shrink solution space, LLM-independent
   - Symbolic reasoning transforms the solution space, “Theorema Spirit”, Guardrail 3
   - FASTCODE, “poor man’s” transformation

Payoff beyond quality: fewer failed attempts → fewer tokens, fewer inference calls, less compute and energy.

QUESTIONS

Questions for the SCML community

? **Specs** — How to formulate specifications, contracts or rules for AI-generated enterprise code?
Generic Reasoning? Derived from Code? Formal specification/DSL?

? **When** — When can symbolic methods realistically be inserted?
Iterative context engineering, constrained decoding, and solution space transformation?

? **Payoff** — How much senior-approver time, compute and token cost could realistically be saved — even short of full verification?

Two industry hurdles

(1) practical availability & formulation of specs

(2) reasoning tools strong enough for modern (web) architectures?

CLOSING THOUGHT

Europe — quo vadis?

The race for the largest models and biggest data centers is largely lost. But raw model scale isn't everything.

1 European strengths

Scientific method, formal verification · compiler construction · symbolic computation · automated proving.

Combination of symbolic computation and probabilistic models might be a way forward.

2 Frontier-model companies are buying the market

They generated billion-dollar losses + circular financial transactions ➔ Dramatically raise prices and switch to "per-token" model.

Self-hosted models (vLLM, Ollama, llama.cpp) are a great alternative – secure & cost-controlled.

They open up the decoding loop — enforce your own constraints — "Guided decoding".

3 Educate excellent experts (mathematics, informatics) instead of "prompt monkeys".

Do not let skills erode and lose the next generation of developers and architects.

Thank You!

thomas@mindruptive.com
thomas.mahringer@schorn.ai

References

Papers & methods

- Cherny-Shahar, T. & Yehudai, A. (2026). *Repository Intelligence Graph: A Deterministic Architectural Map for LLM Code Assistants* (SPADE / RIG). arXiv:2601.10112
- Beurer-Kellner, L. et al. (2023). *Prompting Is Programming: A Query Language for Large Language Models* (LMQL). PLDI / ACM PL — doi:10.1145/3591300
- Beurer-Kellner, L. et al. (2024). *Guiding LLMs the Right Way: Fast, Non-invasive Constrained Generation.* ICML 2024
- Buchberger, B. — *Theorema* project, RISC / JKU Linz — risc.jku.at/project/theorema

Tools & platforms

- SPADE (open source) — github.com/Greenfuze/Spade
- Google *Antigravity* — developers.googleblog.com/build-with-google-antigravity…
- Cursor — *Securely indexing large codebases* & *Fast regex search* — cursor.com/blog
- Harmonic *Aristotle* (LLM + Lean proof verification) — aristotle.harmonic.fun
- Model Context Protocol (MCP) — modelcontextprotocol.io
- LangChain / LangGraph 1.0 — langchain.com
- schorn.ai in a Box (local/air-gapped LLM platform) — schorn.ai
- fal.ai / Flux (image generation) — fal.ai

This talk series

(Windows Developer Magazin, Entwickler Spezial, mindruptive.com/insights)

- Mahringer, T. — *When the Vibes Begin to Fade* (Dec 2025)
- Mahringer, T. — *Deterministic Guardrails for Vibe-Coding* (Apr 2026)
- Mahringer, T. — *The Harness, Not the LLM* (May 2026)