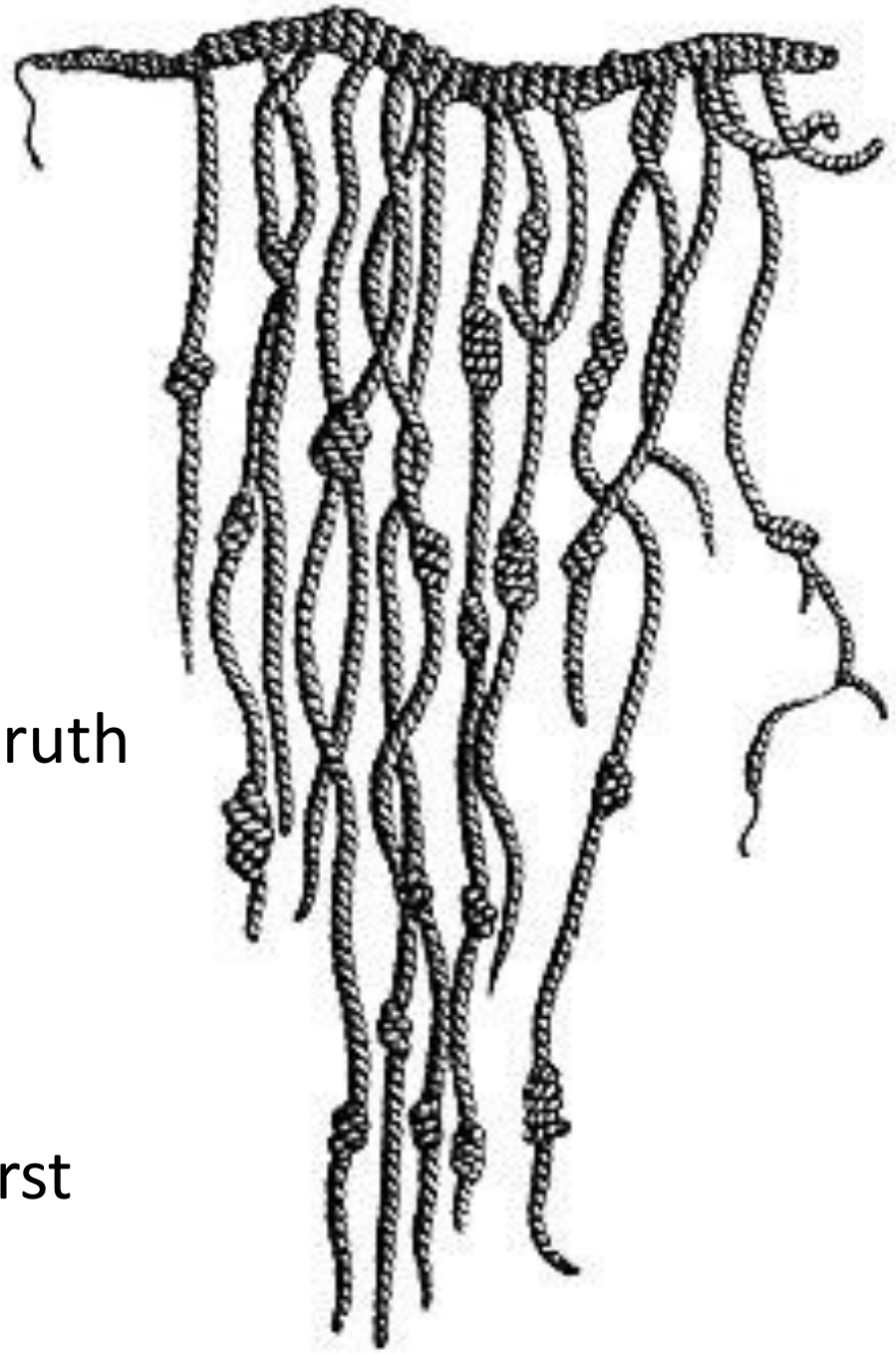
Verified Vibe Coding

Gábor Kusper
Eszterházy Károly Catholic University
kusper.gabor@uni-eszterhazy.hu

SCML – 2026
July 6-8, 2026

Outline

- Vibe Coding: definition, origin, and role shift
- Engineering risks
- Plan-Guide-Verify Principle
- Verified Vibe Coding: D / C / T / L + source of truth
  - Documentation
  - Code
  - Tests
  - Logical specification
- Four AI usage styles: D-first, C-first, T-first, L-first
- Verification loop and RISCAL example




What Is Vibe Coding?

What Is Vibe Coding?

- AI-assisted, conversational programming style
- Flow-oriented, improvisational development
- Rapid acceptance of AI-generated code
- Momentum prioritized over understanding
- A practice that existed before it was named

Carpathian Mountains

Andrej Karpathy, X twit on Feb.2.2025

There's a new kind of coding I call "**vibe coding**", **where you fully give in to the vibes, embrace exponentials, and forget that the code even exists. It's possible because the LLMs (e.g. Cursor Composer w Sonnet) are getting too good.** Also I just talk to Composer with SuperWhisper so I barely even touch the keyboard. I ask for the dumbest things like "decrease the padding on the sidebar by half" because I'm too lazy to find it. **I "Accept All" always, I don't read the diffs anymore.** When I get error messages I just copy paste them in with no comment, usually that fixes it. The code grows beyond my usual comprehension, I'd have to really read through it for a while. Sometimes the LLMs can't fix a bug so I just work around it or ask for random changes until it goes away. It's not too bad for throwaway weekend projects, but still quite amusing. **I'm building a project or webapp, but it's not really coding - I just see stuff, say stuff, run stuff, and copy paste stuff, and it mostly works.**


What Changes in
Vibe Coding?

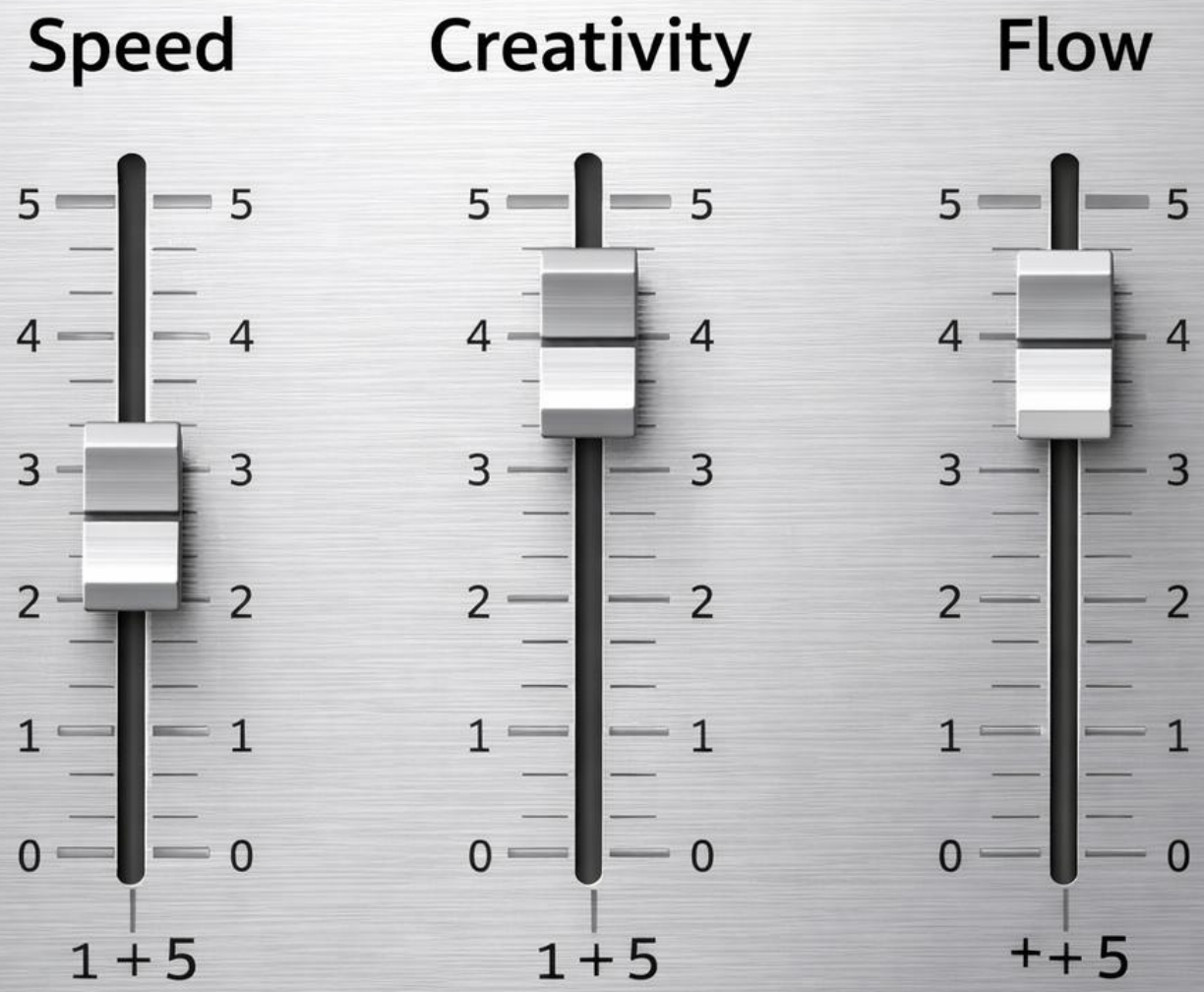
What Changes in Vibe Coding?

- Cognitive work is redistributed between human and AI
- Developer role shifts from coding to orchestration

- You should not be anymore a living **low-level** source code interpreter. You should be a **high-level** guide for a talented AI agent!
- OR: You should be both?

- Development becomes an interactive feedback loop
- BUT: Decision-making remains a human responsibility

Vibe



Engineering risks

- Vibe Coding focuses on:
  - Speed - vibe:)
  - Creativity - vibe increasing:))
  - Developer flow - vibe increasing:)))
- Critical concerns:
  - It seems to be complex. It works, but… - decreasing trust due to possible bugs
  - Did I say that it must be secure? - hidden assumptions of the user
  - Maybe the user wants me… - hidden assumptions of the AI
  - What if AI misinterprets me? - loss of developer intent
  - What if there is a change request? - decreasing maintainability

Goal of This Work

- Intentionally engaging with the ongoing debate: How to Vibe Code?
- Introducing Verification in the Loop
- Proposing several Vibe Coding style with Verification
- 4 Artifacts in the Loop:
  - D – Documentation: requirement spec., function spec., …
  - C – Code: interfaces, APIs, backend, frontend, …
  - T – Tests: unit tests, integration test, system tests, …
  - L – Logical specification: assertions, pre- and post-conditions, invariants, …
- Building on prior work by Gábor Kusper:
  - Vibe Coding in Education, ICETA-2025.
  - Vibe Coding Principles, IEE-2026.

G. Kusper: Vibe Coding Principles, IEE-2026.

1. Responsible Vibe Coding Principle
2. Tool-Aware Vibe Coding Principle
3. Specification-First Principle
4. **Plan-Guide-Verify Principle**
5. Rule of Principle-Guided Prompts
6. Definition of Done Principle

The Plan-Guide-Verify Loop

Principle / Phase	Plan	Guide	Verify
Specification-First	Define goals, constraints, and intent	Anchor prompts to specification	Check conformance to specification
Principle-Guided Prompts	Use Software Development Methodologies	Use OOP Design Principles/Patterns in prompts	Test Driven Development (TDD)
Definition of Done	Define completion criteria early	Use DoD as guidance constraint	Verify DoD

Verified Vibe Coding: D/C/T/L+source of truth

- Introducing Verification in the Loop
- Proposing several Vibe Coding style with Verification
- 4 Artifacts in the Loop:
  - D – Documentation: requirement spec., function spec., …
  - C – Code: interfaces, APIs, backend, frontend, …
  - T – Tests: unit tests, integration test, system tests, …
  - L – Logical specification: assertions, pre- and post-conditions, invariants, …
- It must be clear which artifact is the source of truth.
- The others must be verified against that one.

The source-of-truth rule

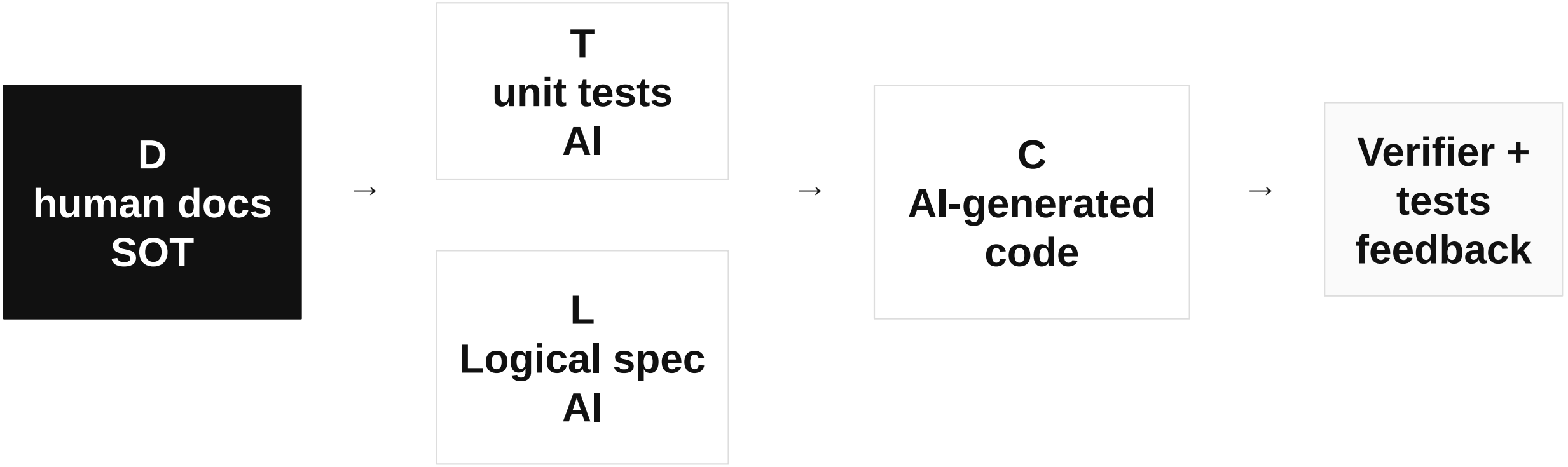
- The first question is not whether an artifact was written by a human or AI.
- The first question is which artifact expresses the intended behavior.
  - What expresses the real intent?
  - What is merely derived from it?
  - Which check does not repeat the same mistake?

The main pathes

Pattern	D	C	T	L
Documentation-first	First Level Human	AI or Human	AI or Human	AI or Human
Legacy code development	AI or Human	First Level Human	AI or Human	AI or Human
TDD – Test Driven Development	AI or Human	AI or Human	First Level Human	AI or Human
DbC – Design by Contract	AI or Human	AI or Human	AI or Human	First Level Human

Build from documentation

- D as Source of Truth (SOT)
- Human-written D defines the intent; AI derives T, L, and C from it.



The failure mode: circular validation

- The bad pattern:
- If the same AI generates code, tests, and specifications from the same prompt, then
- the artifacts may share the same mistake.

- The good pattern:
- Artifacts must be created independently.

Verification targets: from cheap to strong

- Unit tests: behavioral samples and regression signals
- Assertions: hidden assumptions made executable
- Contract: requires precondition / ensures postcondiction / invariants
- Formal tool: Dafny · JML/OpenJML · Frama-C/ACSL · RISCAL/RISCTP

Use Case: RISCAL Binary Search

- **Initial Experiment:** Implemented binary search algorithm using the RISC Algorithm Language (RISCAL) environment.
- **Development Pipeline:** Documentation -> AI-Generated Code -> Manual Testing -> Formal Verification (SMT Solvers, Theorem Proving).
- **Iterative Refinement:** Generating a functionally correct and fully executable script required multiple cycles of prompt refinement.

Use Case: RISCAL Binary Search

RISC Algorithm Language (RISCAL)

File Edit SMT TP Help

File: /home/guest/binary_search

```
1 val N: Nat;
2 val MAX_V: Nat;
3
4 type Index = Nat[N-1];
5 type Elem = Nat[MAX_V];
6 type Arr = Array[N, Elem];
7 type ResultIdx = Int[-1, N-1];
8
9 proc binarySearch(a: Arr, x: Elem): ResultIdx
10   requires forall i: Index, j: Index. i < j => a[i] <= a[j];
11   ensures (result >= 0 => a[result] = x) /\
12           (result = -1 => forall i: Index. ~(a[i] = x));
13 {
14   var l: Int[-1, N] := 0;
15   var r: Int[-1, N] := N - 1;
16   var res: ResultIdx := -1;
17   var m: Int[-1, N] := 0;
18
19   while (l <= r /\ res = -1) do
20     invariant -1 <= l /\ l <= r + 1 /\ r <= N - 1;
21     invariant res = -1 => forall i: Index. (i < l \/ i > r) => ~(a[i] = x);
22     invariant res >= 0 => a[res] = x;
23     // SMT
24     //decreases r - l;
25   {
26     m := l + (r - l) / 2;
27
28     if (a[m] = x) then {
29       res := m;
30     } else {
31       if (a[m] < x) then {
32         // l := m;
33         l := m + 1;
34       } else {
35         r := m - 1;
36       }
37     }
38   }
39
40   return res;
41 }
```

Analysis

Translation: Nondeterminism Default Value: 0 Other Values:

Execution: Silent Inputs: Per Mille: Branches: Depth:

Visualization: Trace Tree Width: 800 Height: 600

Parallelism: Multi-Threaded Threads: 4 Distributed Servers:

Operation: binarySearch(Array[ℤ],ℤ)

```
Executing binarySearch(Array[ℤ],ℤ) with all 2401 inputs.
PARALLEL execution with 4 threads (output disabled).
Execution completed for ALL inputs (36 ms, 588 checked, 1813 inadmissible).
```

Tasks

- binarySearch(Array[ℤ],ℤ)
  - Execute operation
  - Verify specification preconditions
    - Is index value legal?
  - Validate specification
    - Execute specification
    - Is precondition satisfiable?
    - Is precondition not trivial?
    - Is postcondition always satisfiable?
    - Is postcondition always not trivial?
    - Is postcondition sometimes not trivial?
    - Is result uniquely determined?
  - Verify implementation preconditions
    - Is index value legal?
    - Does operation precondition hold?
    - Is assigned value legal?
    - Is index value legal?
    - Is assigned value legal?
    - Is index value legal?
    - Is assigned value legal?
    - Is assigned value legal?
  - Verify correctness of result
    - Is result correct?
  - Verify iteration and recursion
    - Does loop invariant initially hold?
    - Does loop invariant initially hold?
    - Does loop invariant initially hold?
    - Is loop measure non-negative?
    - Is loop invariant preserved?
    - Is loop invariant preserved?
    - Is loop invariant preserved?
    - Is loop invariant preserved?
    - Is loop invariant preserved?

Thank you for your attention!

kusper.gabor@uni-eszterhazy.hu

Special Issue

Formal Methods in the Loop: Trustworthy AI Pipelines

Message from the Guest Editors

Large language model-based systems have demonstrated capabilities in applications ranging from conversational agents and code assistants to multimodal content generation. A new generation is emerging, characterized by agent-oriented AI, context-aware AI, and AI with reasoning. However, most systems still operate as black boxes, raising a question: can we trust their answers? Explainable artificial intelligence (XAI) aims to make AI systems understandable and trustworthy. Formal methods provide formal tools for specification, verification, and reasoning, enabling explanations with guarantees of correctness and reliability. This Special Issue brings together the formal methods, theoretical computer science, and AI communities. Topics include: • Formal methods and mathematical foundations of AI;
• Explainable, verifiable, and trustworthy AI;
• AI using formal methods;
• Context-aware and/or agent-based systems with formal guarantees;
• Principled and structured vibe coding approaches;
• AI ethics, accountability, and fairness;
• Critical thinking using AI, and critical use of AI;
• Human-in-the-loop AI with control frameworks;
• Digital twin technologies with formal foundations.

Guest Editors

Dr. Gábor Kusper
Dr. Benedek Nagy
Dr. Gergely Kovásznai

Deadline for manuscript submissions

28 February 2027

Machine Learning and Knowledge Extraction

an Open Access Journal by MDPI

Impact Factor 8.4
CiteScore 12.7

mdpi.com/si/282466

Machine Learning and Knowledge Extraction
Editorial Office
MDPI, Grosspeteranlage 5
4052 Basel, Switzerland
Tel: +41 61 683 77 34
make@mdpi.com

mdpi.com/journal/make