NHR FAU

FAU Friedrich-Alexander-Universität Erlangen-Nürnberg

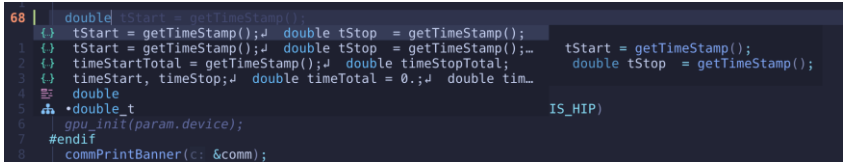
# From Hype to Help: AI Coding Agents in the Real World

Jan Eitzinger

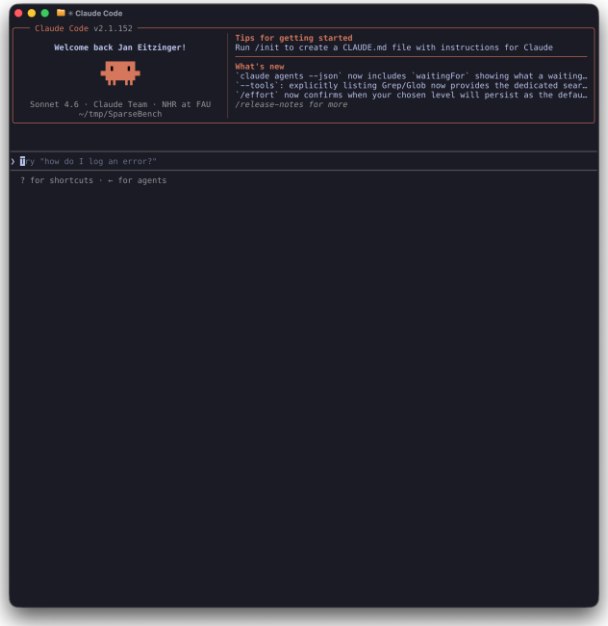
HPC Café, June 9, 2026

# Introduction

## LLMs as a completion source for IDE




## TUI AI Coding agent





## Browser chat interface

# Crash course on Nomenclature

- AI coding agent — software that takes a goal stated in plain language and then plans, writes, runs, and revises code largely on its own
- LLM (large language model) — the underlying AI model (Claude, GPT, Gemini, etc.) that predicts and generates text
- Token — the small chunks of text the model reads and writes (roughly word-pieces). Everything is measured and billed in tokens
- Context window — the agent's working memory: how much text it can "see" at once, including your code, your instructions, and the conversation so far
- Hallucination — when the model confidently states something false
- Plan mode — the agent first explores the codebase and proposes a plan for you to review before it changes anything

# Features of modern Coding Agents

**Codebase understanding**
Search and investigate code

**File creation & editing**
Across the entire project

**Tool use & shell**
Run commands, call tools

**Coding agent**

**Iterative execution**
Self-correct on errors

**Context & memory**
Manage what stays relevant

**Human-in-the-loop**
Review and approve actions

# Offerings (they get more by the day)

Full IDEs (all VSCode Clones)

- Cursor
- Windsurf  (aka Codeium)
- Google Antigravity (not anymore?)
- Kiro (AWS Team)

Terminal UI:

- Claude Code (Anthropic)
- Codex (OpenAI)
- OpenCode
- Pi (Earandil, Agent Harness)
- Many more: Qwen Code, Aider, Kimi CLI

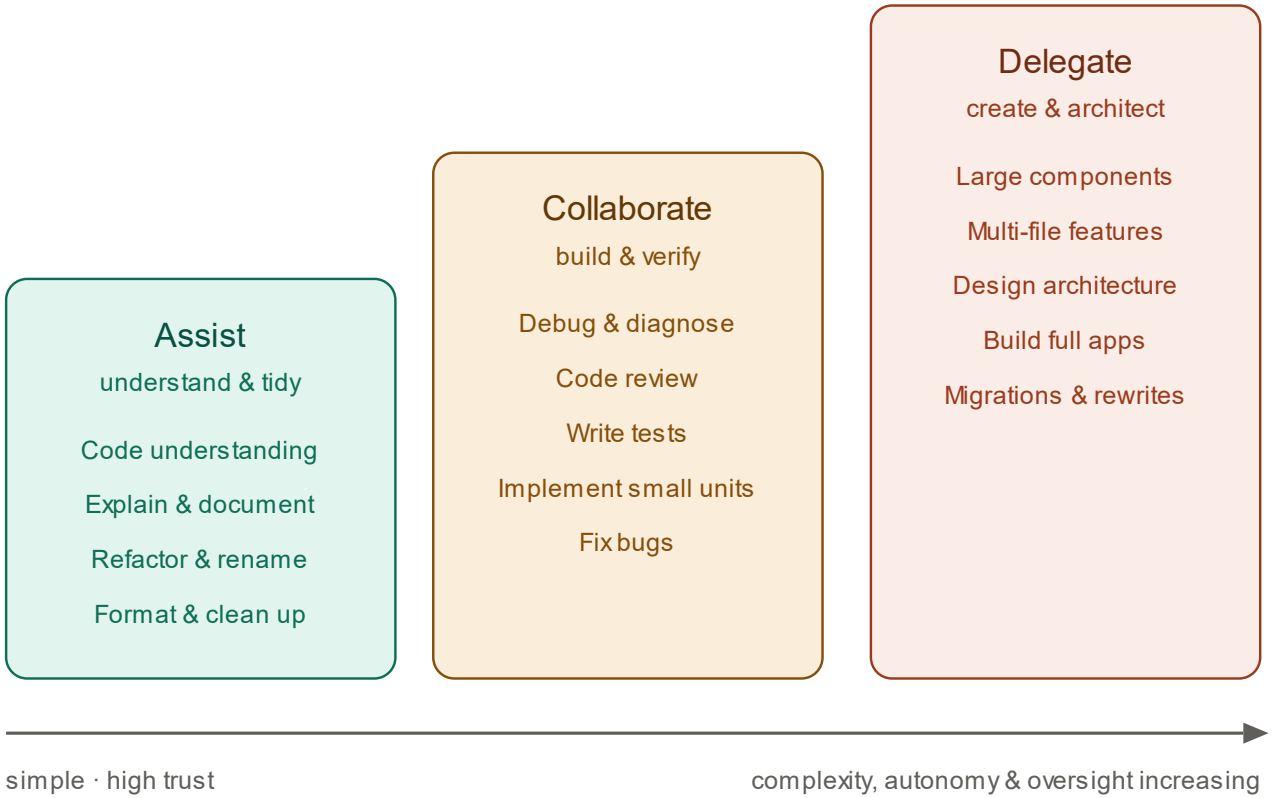





# AI coding culture war

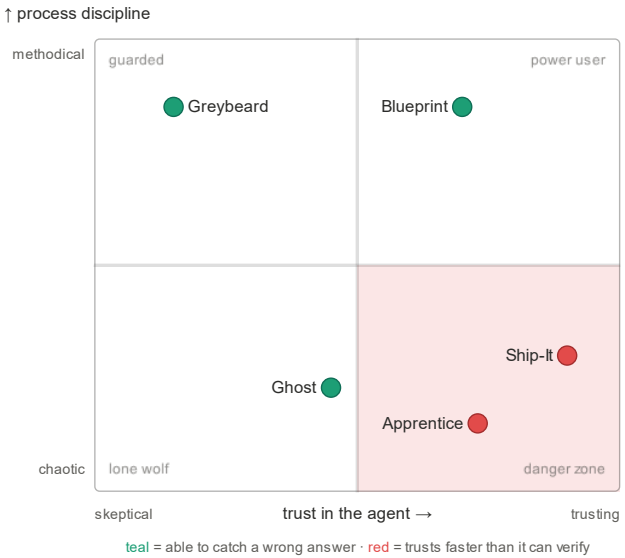

- The "AI Slop" & Burnout
  - Influx of AI-generated code pushed to open-source repositories
  - Often unvetted or low-quality patches
- Copyright and Licensing
  - legal status of purely AI-generated output is undefined
  - Companies or automated bots take copyleft-licensed code, run it through an LLM to rewrite it, and re-release the result under a more permissive license
- Productivity Placebo
  - perceived efficiency of AI coding tools and the reality
  - using AI tools can lead to slower delivery
  - burden of reviewing and debugging AI-generated code

# The task spectrum



# Compatibility with developer archetypes

- **Ghost — the virtuoso** Codes by instinct: brilliant, undisciplined, perfectionist in bursts. Good fit on his own terms — hand the agent the scaffolding and cleanup, never the inspired core.
- **Blueprint — the architect** Methodical senior who plans, specs, and documents before touching code. The most natural fit — lives in plan mode; the only risk is over-briefing and losing the speed.
- **Apprentice — the junior** Eager, fast-learning, tool-native, but without the judgment to spot a wrong answer. Highest leverage and highest risk — needs guardrails and a senior in the loop.
- **Greybeard — the skeptic** Deep-fundamentals veteran who's seen every hype cycle and distrusts the tool on principle. Under-uses it today — yet his verification instincts would make him a top user if he engaged.
- **Ship-It — the believer** Enthusiastic adopter who delegates aggressively and loves the speed. Big throughput, thin oversight — needs review discipline bolted back on.



# AI Coding agents do not change the game

- We're still stuck in the messy stored-program paradigm
- Complexity is the physics of programming

- A watertight spec — or even a full description of existing code, let alone code not yet written — is therefore nearly impossible

**And this can't be fixed !**

- Mistakes are inevitable: the AI hits the same hard limits. It's physics, after all.
- Workflows and clever context management help — but there's no silver bullet

https://people.csail.mit.edu/saman/acpss/talk-2/talk-slides.pdf

# Tips and Tricks (based on personal experience)

- Start with a clean git state
- Use plan mode for anything non-trivial
- Work iteratively, not in one shot
- One task per prompt
- Keep
- Interr
- Match the model to the task
- Give the agent context, not just commands
  - Add relevant context in CLAUDE.md / AGENTS.md
- Start a fresh context for every new task


There will be another HPC-Café with more advanced and in-depth content on effective usage

# Problems and Pitfalls

- The model will always give you an answer — whether or not one exists
- You can talk the model into almost anything
- Impossible tasks will be attempted anyway

- Underspecified context leads to unpredictable outcomes
- Hallucinated APIs, libraries, and documentation
- Confidence does not correlate with correctness

**The more skeptical you are the better!**

# The main dangers

- Convenience is a trap
- You will lose the ability to do things on your own
- Online dependency is a single point of failure
- Privacy and confidentiality are not guaranteed


What can you do?

- Local models are not optional — they are necessary
- Stay sharp deliberately. An agent is most valuable in the hands of someone who could do the work without it.

# My personal journey – A diary of ups and downs

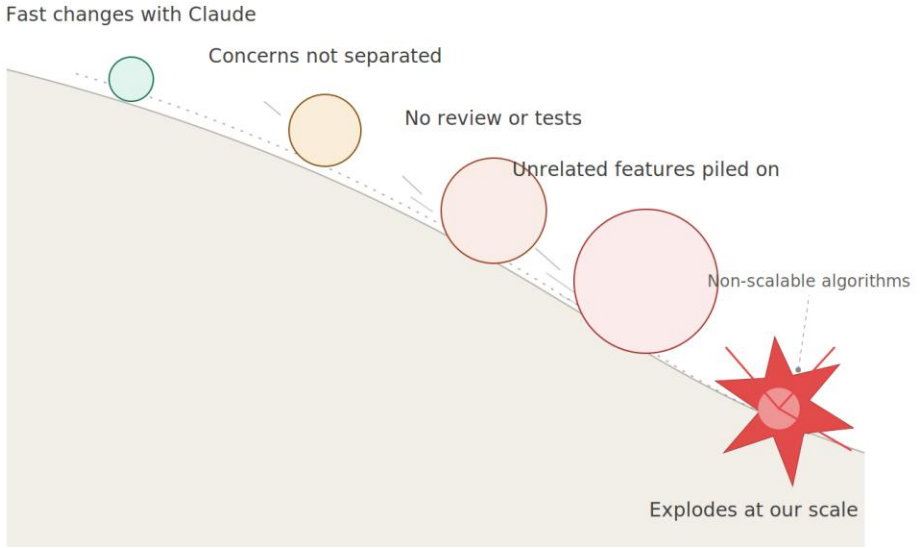
- Setting: ClusterCockpit cc-backend
  - Golang (very good tooling; simple, statically typed language)
  - Complex web backend + frontend and ClusterCockpit central hub

- First try on long delayed tasks:
  - Create new Job Archive backends (S3, sqlite blob)
  - Did not work out of the box, but issues could be easily fixed

- Extensive usage to review and document existing code
  - Initial friction with others why I (?) changed their code
  - Awkward, almost shameful undertone when you present work and the AI did the heavy lifting

# The failure

- Task: Improved implementation for in-memory metric store persistent checkpointing
  - Continuous checkpoint writing
  - Low memory usage
  - Fast checkpoint reading and writing on startup and shutdown
  - Use a standardized file format (Parquet)



# Anecdote

- Claude generated a solution for continuous checkpointing based on a Write-Ahead Logging (WAL) algorithm borrowed from DB implementations

- A colleague suggested the problem must be in this AI generated slop
- After a quick review we concluded the implementation might be buggy
- Under the impression to know that there is a bug I put Claude on it the next morning:
  - It initially said there is no bug
  - But after insisting and providing more *evidence*
  - It agreed and found and fixed a non-existing bug
- My colleague after a more thorough review reported the algorithm works just fine

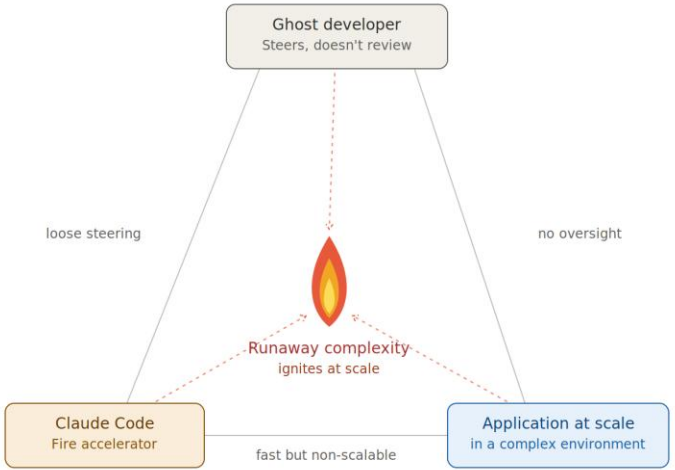
# The solution …

- High memory consumption turned out to be *several* unrelated problems with the same symptom
- Avro checkpoints were the prime suspect — they did use memory, but weren't the cause of the escalations
- Only continuous memory tracking + a profiler finally pinned down the real issues:
  - A larger DB cache blew up on queries not backed by an index
  - Parquet archiving (added with Claude) used an algorithm that made multiple in-memory copies of the data
  - In several places, Claude picked implementations that underestimated cc-backend's real scale

# The aftermath

This is a good example when complexity strikes

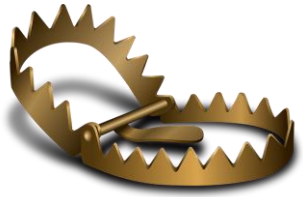
- Claude let me add and change code at high velocity
- But with no separation of concerns and no thorough review or testing, that speed let complexity accumulate
- My own under-specification lead Claude toward convenient but non-scalable algorithms — which then exploded at scale



# What did I learn (hopefully)

- In the beginning you feel enthusiastic and empowered
- The AI's confidence and capabilities let's you think you can blindly trust it

But this is a trap!



- If you proceed with unchecked changes at some point, you will be stuck in a complexity nightmare

- Always stay skeptical, always validate what the AI did
- This takes away some of the productivity gain, but there is no alternative

# Demo