

Data Science Colle... · [Follow publication](#)

The AI Vibe Coding Paradox: Why Experience Matters More Than Ever

AI can now code faster than any developer. But without experienced leadership, it breaks down in record time

8 min read · 4 days ago



Simon Greenman

[Follow](#)[Listen](#)[Share](#)[More](#)

```
import react = 'require';
import express = '<g';
const pg = seach();

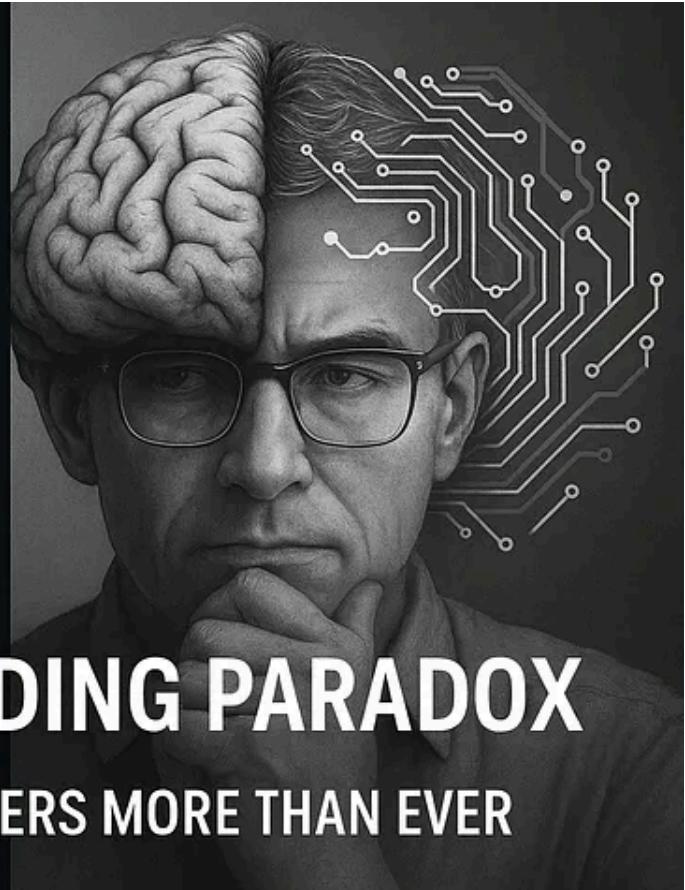
const qquery = --query; async search() ->
const onSearch() {
    setqquery; onChange(ev);

const renderCaseE: redier(impt//button')>

    aqc/'api/seart',o(arq) {
        request(tapi/Search: q) -> {
            request apI = searst.matejson(
        );
        r),use( $axp('/query', 'name.json');
        render(input('$esults');
    };

```

THE AI VIBE CODING PARADOX
WHY EXPERIENCE MATTERS MORE THAN EVER



When Machines Code but Can't Think

I'm not too modest to say I used to be a pretty good coder. In my teens, I fell down the rabbit hole of writing and publishing computer games — and failed a few school exams along the way. Thirty years later, I'm rediscovering that excitement through the new wave of vibe coding tools. All I need to do is describe what I want — “a voice-controlled to-do list app”- and within minutes, I have a working prototype.

After hundreds of hours experimenting with [Replit](#), [Claude Code](#), and [Cursor](#), I've seen both their astonishing power and their hidden fragility. These tools are revolutionary, but not in the way most people imagine. Without human judgment and experienced leadership, they often create architectures and code that collapse under their own weight.

• • •

The Promise: Instant Code Across Specialisms at Unprecedented Speed

AI coding assistants deliver spectacularly on their promise. In one project, I asked Replit to build a graphical search interface and database application using Gen AI “vector similarity.” Within twenty minutes, I had a React frontend, Express backend, PostgreSQL and Pinecone databases, and a full API layer. Nothing in my career has matched that velocity for a prototype.

What struck me most wasn't just the speed — it was the breadth. Tasks that once required specialists in different aspects of coding were now accessible to a single developer. The old divisions and hiring of frontend, backend, and database engineers are blurring. With vibe coding everyone can now become a full-stack engineer.

Or so it seems. Then the problems began.

In production, I discovered that over a third of the codebase was duplicated. One component had swollen to nine times the recommended size, crashing my environment. The same text-normalization function appeared in fifteen separate files. These weren't bugs — they were symptoms of how AI builds.

AI delivers speed — but not modularity of structure.

The Limitation: Pattern Recognition Without Understanding

The core issue that became clear as I used these tools is Gen AI can generate code, but it rarely comprehends the system.

Humans construct **mental models** when we code — maps of how data flows, how components interact, and how changes ripple through the architecture. That's **systems thinking**: perceiving the whole, not just the parts

Experienced developers don't just write code; they reason in dependencies, constraints, and feedback loops. They anticipate how optimizing one component might break another.

AI, by contrast, operates through local pattern recognition. It stitches together fragments of syntax without grasping structure or consequence.

AI doesn't understand systems. It reproduces them.

Strong systems thinkers reason dynamically. They see how problems evolve over time, not just at compile time. That combination of *analytical reasoning, empathy, and foresight* is what enables humans — and not machines — to design resilient systems.

Why Experience Still Matters

Working effectively with AI tools requires exactly what AI lacks: true architectural vision, scale awareness, and debugging instinct.

- **Architectural vision.** When deciding between dual versus unified Pinecone indexes, or whether to store color data as JSONB or arrays, those weren't coding choices — they were architectural trade-offs. The AI could implement any of them but couldn't reason about which would scale or remain maintainable.
- **Scale awareness.** When vibe coding generated a 39-MB JSON blob that crashed browsers, I immediately recognized the flaw: *this works in test data but will collapse in production.* The AI couldn't extrapolate.
- **Debugging instinct.** When normalize functions appeared in fifteen files, I saw a missing abstraction. When bugs surfaced, I didn't guess — I hypothesized: middleware order, async timing, data mismatch. Vibe debugged wherever I pointed but lacked the instinct for *why*.
- **Perceptual judgment.** When implementing color matching for a design-search system, vibe coding executed the math flawlessly — but the results looked

wrong. The model couldn't tell that "red" as an accent behaves differently from "red" as a dominant tone. Only human perception could make that call.

AI executes code brilliantly. Humans architect systems that work.

I had many situations where I was getting frustrating and insufficient answers from the vibe coding with answers such as the following.

A fairly common set of mea culpas from vibe coding tools as I questioned its coding and debugging

The Paradox: Lower Barriers, Higher Standards

Here lies the paradox: AI lowers the barrier to entry while raising the bar for mastery. Non coders and junior developers can now generate impressive full-stack demos — but when things break, they often lack the conceptual modeling skills to recover without mentorship. Experienced developers, by contrast, can multiply their impact because they know where the boundaries are. They understand that "it runs" is not the same as "it works." The most effective users of vibe coding tools aren't those who prompt well -they're those who think well.

Why This Moment Is Different

Skeptics might say software has always required both access and expertise- and they're right. What's new is how fast you can build and how fast things break.

AI has collapsed the barrier to entry: anyone can describe a system and have it built in minutes. Before AI, *starting* was the hard part. Now, *maintaining and scaling* are.

It has also decoupled code generation from comprehension, creating systems we don't fully understand. It used to take many months or years to produce bad code at scale; now it takes seconds.

Experience still matters — but its meaning has changed. In the AI era, experience means judgment, abstraction, and coherence: knowing how to guide an intelligent system that doesn't understand its own consequences.

AI doesn't change what makes good software — it changes how quickly bad software happens without experience.

The New Skill: Teaching Machines

To use these tools well, you must manage them like exceptionally capable but inexperienced developers. I've learned to use coaching language.

Instead of “add a search feature,” I say:

“Before implementing search, identify existing components and extend them if they exist.”

Instead of “fix this bug,” I pose hypotheses:

“Could this be a timing issue in React? Are we awaiting the async call properly?”

That shift reduced my debugging cycles from five iterations to one or two.

I also ask questions the AI never will:

“What happens at 100× scale? Will this still work if the file is 50 MB? or 1M records instead of 1,000?”

Sometimes, I even let the machines debate each other — two AI tools critiquing one another's output. It's pair programming between two intelligent but differently trained juniors. In the case Replit and Claude Code. AI tools don't need micromanaging- they need continual coaching and mentoring.

The Organizational Challenge: Don't Be Tempted To Hire Cheap

AI won't replace programmers per se — it will increase demand for experienced ones who can guide, structure, and validate what machines produce.

The real risk for organizations is misunderstanding this shift. Hiring less experienced developers and giving them AI assistants may look efficient, but it could build technical debt at machine speed. The initial velocity advantage quickly can become a drag coefficient on productivity.

Smart organizations are already evolving — updating hiring, workflows, and governance to reflect this new reality.

How Recruiting Must Change: Systems Thinking, Empathy and Teaching

Traditional “write this code” interviews are obsolete. Anyone can now generate passable code with Cursor, Lovable, or Claude.

The new test should be:

“Here’s code an AI wrote — what’s wrong with it, and how would you fix it?”

Hire for **systems thinking**, not syntax. Organizations that recruit for just initial speed will amplify AI’s weaknesses; those that hire for judgment will amplify its strengths.

Experienced engineers will set boundaries, establish architectural checkpoints, and teach machines effectively. The developers who thrive won’t be those replaced by AI — they’ll be those who learn to manage it.

Empathy and teaching are now part of engineering.

Organizations need to also invest in mentorship programs that help junior developers learn to work with AI effectively. Pair programming takes on new meaning when one developer is guiding both an AI tool and a junior colleague simultaneously. This creates a learning flywheel: juniors learn systems thinking while experienced developers multiply their impact.

The Opportunity for Junior Developers

If you’re early in your career, this shift is actually exciting news — but only if you approach it correctly. Don’t use AI to avoid learning fundamentals; use it to accelerate your development of systems thinking. When Claude or Cursor generates code, ask yourself: *Why did it structure it this way? What would break if this scaled?*

$100\times$? Where are the dependencies? What's wrong with this approach? The best junior developers treat AI output as a learning tool, not a solution — they analyze what the AI created to understand both what works and what doesn't. Seek out organizations and mentors who will teach you systems thinking alongside AI tools, because the developers who thrive won't be those who prompt best — they'll be those who learn to think in systems fastest.

The Bottom Line: AI Magnifies the Gap Between Experience and Inexperience

AI coding tools represent a genuine transformation in software development. But the real story isn't that AI replaces developers — it's that AI magnifies the gap between experience and inexperience.

These tools can **$10\times$ productivity** — but only if you bring the **$1\times$ of conceptual understanding** they lack. They're extraordinary assistants. They're not architects with complex system thinking.

Admittedly, new coding agents are beginning to embody more system thinking — but as long as they rely on current large language model approaches, they will still lack true judgment.

That's the **AI Vibe Coding Paradox**:

The better the machine gets at producing code, the more valuable human experience becomes.

So could I still code? I found that the same **systems thinking muscles** I'd developed earlier in my career came back quickly. Understanding architecture, debugging logically, seeing connections — the mindset was still there.

AI may write the code, but experience still writes the system.

And experience isn't exclusive — it's earned. If you're early in your career, the path forward is clearer than ever: learn the fundamentals, work alongside people who think in systems, and use AI as a mirror to accelerate your growth. The gap between junior and senior developers isn't closing — but the speed at which you can cross it has never been faster.

I would be fascinated to hear about your experience and best practices with vibe coding. Reach out to me and don't forget to clap.

• • •

Simon Greenman is Co-Founder and Partner at [Best Practice AI](#), and Co-Founder of MapQuest. He advises companies and boards on AI strategy, governance, and generative AI transformation.

[Disclaimer: ChatGPT and Claude helped me polish this article, but the core messages were all initially developed through my work with the vibe coding tools]

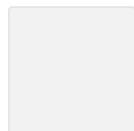
Vibe Coding

Artificial Intelligence

Software Engineering

Systems Thinking

Generative Ai Tools



Follow

Published in Data Science Collective

876K followers · Last published 18 hours ago

Advice, insights, and ideas from the Medium data science community



Follow

Written by Simon Greenman

3K followers · 414 following

All in on AI. Partner [bestpractice.ai](#). ex Member @World Economic Forum Global AI Council. MapQuest co-founder. Executive | NXD | CTO

Responses (16)





Bgerby

What are your thoughts?



Homotechnologicus

2 days ago

...

Very smart, thoughtful, and well written piece.



21

[Reply](#)



Roman Shushin

2 days ago

...

I started vibe coding recently and was so excited because AI took out the hardest part of coding for me - the need to remember the syntax. But very soon I discovered that I need to think twice as hard - for myself and for the LLM. Very often LLM is going in loops, losing direction, misleading and overcomplicating.



12

[Reply](#)



Christopher Peterson

1 day ago

...

It used to take many months or years to produce bad code at scale; now it takes seconds.

😂 love it



8

[Reply](#)

[See all responses](#)

More from Simon Greenman and Data Science Collective

 Simon Greenman

Best practices for LLM optimization for call and message compliance: prompt engineering, RAG, and...

From 80% to 98%: how we enhanced LLM model accuracy for compliance in medical marketing and sales calls

Jun 25, 2024  365  4



...

 In Data Science Collective by Amanda Iglesias Moreno

NotebookLM Just Got a Serious Upgrade—Exploring NotebookLM's Newest Features and Updates

What's New and Why It Matters

★ Sep 29 ⌘ 906 🗣 23



...

In Data Science Collective by Kalle Georgiev

Markov Chains for Humans: The Simplest Way to Predict What's Next

A plain English guide to modeling step-by-step behavior using only the present moment

★ Oct 7 ⌘ 861 🗣 11



...

Simon Greenman

Who Will Make Money from the Generative AI Gold Rush? Part I

Buckle up for the Generative AI gold rush! Will BigTech rule with its picks and shovels? Which startups will strike it rich? Will “copilot...

Mar 12, 2023  1.5K  24



...

[See all from Simon Greenman](#)

[See all from Data Science Collective](#)

Recommended from Medium

 In Towards AI by Teja Kusireddy

We Spent \$47,000 Running AI Agents in Production. Here's What Nobody Tells You About A2A and MCP.

Multi-agent systems are the future. Agent-to-Agent (A2A) communication and Anthropic’s Model Context Protocol (MCP) are revolutionary. But...

Oct 16 1.3K 24



In AI Guys by Vishal Rajput

Stanford Just Broke Prompt Engineering—VERBALIZED SAMPLING

I'm sure you have seen hundreds of posts on prompt engineering to date. But in few rare cases, you actually get something relevant...

4d ago 72 1



In Level Up Coding by Gao Dalie (高達烈)

PaddleOCR VL + RAG: Revolutionize Complex Data Extraction (Open-Source)

Not even a month ago, I made a video about MistralOCR that many of you liked.

◆ 5d ago 👏 52



Ignacio de Gregorio

LLMs are smarter than we thought.

Changing how they talk changes how they think.

◆ 2d ago 👏 729 💬 22



 In Data Science Collective by Ida Silfverskiöld

Agentic AI: Single vs Multi-Agent Systems

Building with a structured data source in LangGraph

 3d ago  285  7



...

 In Artificial Intelligence in Plain English by Sandra Bats

OpenAI is coming for your org*sms... and we should be worried

I thought it was a joke. I really did. But it's official. ChatGPT is going to get a 'pervert mode'. CEO Sam Altman announced yesterday that...

Oct 23  239  10



...

[See more recommendations](#)