The 5 Levels of AI Coding: Where Are You, and Where Is This Going?

February 18, 2026 · by Mark Bartlett

Download this guide as PDF

> **Quick Answer:** Dan Shapiro's framework maps six levels of AI-assisted coding, from spicy autocomplete (Level 0) to dark factories where no human writes or reviews code (Level 5). A 2025 METR study found experienced devs using AI were 19% slower — while believing they were 20% faster. Meanwhile, StrongDM runs a 3-person team shipping production Rust entirely by agents. The gap isn't about tools. It's about workflow redesign and spec quality. Most of us are at Level 2-3. The bottleneck has moved from implementation to specification.

📚 **More on this topic:** Best Local Coding Models · Building a Local AI Assistant · Tiered AI Model Strategy

Here are two facts that seem contradictory:

A 3-person team at StrongDM has been shipping production software — 16,000 lines of Rust, 9,500 lines of Go — entirely written by AI agents since July 2025. No human writes code. No human reviews code. The factory runs with the lights off.

A randomized controlled trial by METR found that experienced open-source developers using AI tools were **19% slower** than working without them. Those same developers believed they were 20% faster. Wrong about both direction and magnitude.

Both facts are real. The gap between them is where software development lives right now.

The Framework

Dan Shapiro — CEO of Glowforge, Wharton Research Fellow — published a framework in January 2026 mapping six levels of AI-assisted coding, modeled after the NHTSA's five levels of driving automation. Nate B. Jones surfaced the implications of where most developers actually sit.

Level 0: Manual. Human writes all code. AI not involved.

Level 1: Task Offloading. You hand AI a discrete task: write this function, generate this test. You review everything. AI reduces keystrokes. GitHub Copilot's original autocomplete lives here.

Level 2: Active Pairing. Flow state with AI as a junior buddy. AI handles multi-file changes, navigates codebases, understands dependencies. You still read all the code. Shapiro estimates 90% of developers who call themselves “AI native” are here.

Level 3: Human-in-the-Loop Manager. You direct the AI and review what it produces. You approve or reject at the PR level. The model does implementation. You’re managing, not coding. Almost everybody tops out here because of the psychological difficulty of letting go of the code.

Level 4: Autonomous Development. You write a spec. You leave. You come back hours later and check whether the tests pass. You’re not reading the code. You’re evaluating outcomes. This requires deep trust in your specification quality.

Level 5: Dark Factory. Spec goes in, working software comes out. No human writes code. No human reviews code. The factory runs with the lights off.

Most of us are at Level 2-3 and think we’re at Level 4. The METR study proves we’re bad judges of our own AI productivity.

Why the METR Study Matters

The METR study (July 2025) was a proper randomized controlled trial. Sixteen experienced open-source developers completed 246 tasks on their own repositories — code they already knew well. Each task was randomly assigned as AI-allowed or AI-disallowed. The primary tool was Cursor Pro with Claude Sonnet.

The result: **19% slower with AI.**

Before the study, developers predicted AI would make them 24% faster. After, they still believed they’d been 20% faster. The perception gap — believing you’re faster when you’re measurably slower — is the headline finding.

The researchers emphasize caveats: these were experienced developers working on mature codebases they knew well. AI may help more on unfamiliar codebases, for less experienced developers, or on different task types. But the finding challenges the default assumption that AI tools automatically improve productivity.

This matches lived experience. Think about the first time you used Copilot or Claude Code. You spent time evaluating suggestions, correcting almost-right code, context-switching between your mental model and the AI’s output. The generation was fast. Everything else was slower.

The J-Curve

When you first adopt AI coding tools, you get slower before you get faster. This is the J-curve.

The dip happens because AI disrupts your existing workflow before you've built a new one. You're debugging AI suggestions instead of writing code. You're figuring out how to prompt effectively. You're losing context every time you switch between thinking and reviewing. The generation speed is impressive, but the integration overhead eats the gains.

The developers in the METR study were in this dip — experienced with their codebases, relatively new to AI-assisted workflows. They hadn't had time to redesign their process around AI capabilities.

The teams operating at Level 4 and 5 have climbed out of the dip by fundamentally changing how they work. They don't bolt AI onto their existing process. They build a new process where AI is the default execution layer.

The Spec Quality Bottleneck

This is the single most important shift: **the bottleneck has moved from implementation to specification.**

When AI writes the code, the quality of your spec determines the quality of your software. A precise spec with clear acceptance criteria produces working code. A vague spec produces code that looks right but does the wrong thing — and takes longer to fix than writing it manually.

We experience this directly at InsiderLLM. This site has 130 articles written in three weeks, almost entirely by Claude Code. When the prompt is specific — exact structure, data points, tone, word count, cross-links — the article comes out right the first time. When the prompt is vague, the output requires more editing than writing from scratch would have taken.

The same pattern holds for code. StrongDM's entire workflow is built around spec quality. Their agents consume markdown specifications and behavioral scenarios, not code prompts. The humans don't write code or review code — they write and refine specs.

If you're using AI coding tools and not getting faster, the problem probably isn't the tool. It's the spec. Level 2 developers give the AI code-level instructions. Level 4 developers give it outcome-level specifications.

Scenarios vs Tests: A Pattern Worth Stealing

StrongDM does something clever: they keep **evaluation criteria separate from the codebase.** The AI agents don't run unit tests. They run behavioral scenarios defined in markdown — descriptions of what the system should do in specific situations, written in natural language.

Why? Same reason teaching to the test produces students who pass tests but don't understand the material. If the AI can see the test suite, it optimizes for passing tests. If it can only see behavioral descriptions of correct operation, it has to actually build correct software.

Simon Willison called this approach "a glimpse of one potential future of software development" while noting the team behind it has 20+ years of experience in high-reliability systems. They're not naively handing code to AI. They're applying deep systems engineering knowledge to a new execution model.

This is a design pattern local AI builders should internalize. When you're using AI to build software, separate your evaluation from your implementation. Describe what correct behavior looks like before you start generating code.

The Numbers Around Us

Some context for where AI coding stands in early 2026:

- **4% of public GitHub commits** are now authored by Claude Code, per SemiAnalysis analysis — projected to reach 20% by end of 2026
- **70-90% of Anthropic's code** is AI-generated company-wide, with individual engineers like Boris Cherny (creator of Claude Code) claiming 100% for months
- **Microsoft** reported ~30% AI-generated code as of April 2025
- **Entry-level tech job postings** dropped 67% in the US between 2023 and 2024 (Stanford Digital Economy Lab / ADP data)
- **UK graduate tech hiring** fell 46% year-over-year (Institute of Student Employers, October 2025)

The junior developer numbers look grim until you consider the demand argument.

Why More AI Doesn’t Mean Fewer Opportunities

Every time computing cost dropped — mainframes to PCs, PCs to cloud, cloud to serverless — total software produced exploded. The addressable market expanded because projects that were too expensive became feasible. Companies that couldn’t afford custom software at $500K found they could build it at $50K.

The same dynamic is starting now. When a 3-person team can produce what used to require 20, the cost per project drops by an order of magnitude. That doesn’t eliminate demand — it creates demand that didn’t exist before. Small businesses, niche applications, internal tools, custom workflows — software that was never worth building at old prices becomes worth building at new prices.

The roles change. The demand for someone who writes functions decreases. The demand for someone who understands a problem domain well enough to specify correct behavior increases. This is the generalist advantage: when AI handles implementation, human value is in understanding the problem broadly — systems, users, business context, failure modes — not in knowing one framework deeply.

This is good news for the local AI builder audience. People who build things on their own hardware, who understand systems end to end, who are inherently generalists because they do everything themselves — that’s the skill profile that Level 4 and 5 workflows reward.

Where We Actually Are

Let me be honest about our position. At InsiderLLM, we run what you might call a dark-factory-lite for content. The workflow is: write a spec (the CC prompt you see at the top of each article) → AI writes the article → human evaluates the outcome → deploy. We’re operating at Level 4 for content production. 130 articles in 3 weeks with one person.

But I read every article. I check the facts. I catch errors. That’s not Level 5 — that’s Level 4 with strong quality control. And for code? Most of us — myself included — are at Level 2-3. We read the diffs. We verify the logic. We haven’t built the specification and testing infrastructure that would let us stop reading code.

The path from Level 3 to Level 4 isn’t about better AI models. It’s about better specs, better evaluation criteria, and the willingness to let go of reviewing every line. The METR study shows we can’t even accurately perceive whether AI is helping us. Trusting our intuition about when to let go is probably the wrong approach.

What might work: measuring. Track your actual completion times with and without AI. Build evaluation criteria before you start coding. Separate your scenarios from your tests. These are the patterns the Level 4+ teams are using, and they're available to anyone willing to redesign their workflow rather than just plugging AI into their existing one.

Bottom Line

The five levels framework is useful because it makes the gap visible. Most developers are at Level 2-3, getting the AI to write code and then carefully reviewing everything it produces. A tiny number of teams are at Level 4-5, writing specs and evaluating outcomes without reading code at all.

The gap isn't closed by better models. It's closed by better specifications, separated evaluation criteria, and a fundamental shift from "AI helps me code" to "I direct AI and verify results." The bottleneck was implementation. Now it's specification. The developers who internalize that shift — who invest in writing precise specs instead of reviewing generated code — are the ones who'll make the jump.

The dark factory exists. Most of us aren't ready for it. The honest thing to do is figure out exactly where we are, measure whether our tools are actually helping, and build the spec-writing and evaluation skills that make higher levels possible. The tools are already good enough. The workflows haven't caught up.

Framework credit: Dan Shapiro, "The Five Levels: from Spicy Autocomplete to the Dark Factory." Analysis sourced from Nate B. Jones, Simon Willison, and the METR study (July 2025).

Source: https://insiderllm.com/guides/five-levels-of-ai-coding-dark-factories/

Free guides for running AI locally