The Engineer of 2026 Doesn't Write Code

A 10-Minute Podcast Script on AI-Native Engineering

There's a question keeping engineering leaders up at night, and most of them are asking it wrong. They're asking: "How do we get our engineers to use AI?" That's the wrong question. The right question is: "What happens to the engineering role when AI does the coding?"

Because that's where we are. Not in five years. Now.

Let me give you a number. One engineer — a real person at a real company — shipped over three hundred pull requests in a single month. December. No overtime. No burnout. No team behind him. One person, three hundred PRs. That's not a productivity improvement. That's a different job.

Here's how he did it. He runs ten to fifteen AI agent sessions in parallel. Five terminals on his machine, another five to ten in the browser. Each one working on a separate task. His workflow has three steps. First, he plans — he spends time with the AI getting the plan right before any code gets written. Second, he lets the agents execute — they write the code, run the tests, commit, push, and open the pull request automatically. Third, he verifies — he reviews output, catches errors, and steers. Plan. Execute. Verify. That's the job now. The coding part? The agents handle that.

And this is the part people miss. This is a harder job, not an easier one. Writing code is a well-understood skill. Most engineers have spent years getting good at it. But decomposing a problem into fifteen parallel tasks, choosing the right delegation strategy for each one, spotting when an agent is going off track, and verifying output quality at speed — these are new muscles. They require more clarity of thought, not less. More architectural thinking. More judgment. The bottleneck is no longer typing code. The bottleneck is thinking clearly.

So there's a framework I find useful for understanding where teams actually sit on this curve. Four levels.

Level one: AI User. You use AI as a tool — autocomplete, asking questions, occasional code generation. This is where most of the industry is right now. AI is helpful, but your workflow hasn't changed. You work the same way you did before, just with a smarter assistant in the corner.

Level two: AI First. AI is your first option for the majority of tasks. Before you write code, before you Google the answer, before you ask a colleague — you ask the AI. A few people on any given team have made this shift. It's a frequency change, not a structural one.

Level three: AI Native Individual. This is where it gets interesting. You've restructured your entire workflow around AI. You're not using AI more — you're working in a fundamentally different way. You design plans, orchestrate agents, verify output. The three-hundred-PR engineer is a textbook Level three. He didn't get faster at coding. He stopped coding and started managing a fleet.

Level four: AI Native Team. The whole team operates this way. Shared tooling, shared norms, shared context. The team's output is structurally impossible without AI — not because individuals use it, but because the systems, culture, and infrastructure assume it. A new person joining the team reaches Level two within a week because the environment carries them there.

Here's the critical insight. Getting individuals to Level three is a training problem. You teach people new workflows, new tools, new habits. It's hard but straightforward. Getting a team to Level four is a systems problem. It requires shared infrastructure, shared norms, and a culture that makes AI-native the default. Those are fundamentally different interventions, and most organizations are only investing in the first one.

Now, you might be listening to this and thinking — okay, the framework makes sense, the proof point is compelling, but what actually stops teams from making this jump? It's not the technology. The tools exist today. It's not the skill gap either — that's trainable. The number one blocker is fear.

Fear of breaking something. Fear of expensing a tool without permission. Fear of running an experiment that might not work. Fear of looking like you're not "doing real work" because you're planning and verifying instead of typing. In most organizations, the immune system kicks in the moment someone tries to work differently. The process says no. The procurement cycle says wait. The culture says don't rock the boat.

The organizations that are going to win this transition are the ones that name this problem and kill it explicitly. Not with a vague "we encourage innovation." With a specific, loud, leadership-backed mandate: **experiment freely, expense the tools, break the rules if the rules are the blocker.** One organization I know calls this "Be Pirates" — and it's not a slogan on a wall. It's sanctioned at the director level and above. When someone experiments and it works, they celebrate it publicly. When it doesn't work, they celebrate the learning. Both outcomes are valued. Both are visible.

That's the unlock. Not a new tool. Not a training program. A cultural license that removes the friction between knowing what's possible and actually doing it.

But culture alone doesn't get you to Level four. You also need to rethink the shape of the team itself.

The traditional model: ten people on one initiative. A product manager, a designer, eight engineers. Heavy coordination. Lots of meetings. Slow but aligned. The emerging model: same ten people, four parallel tracks. One or two engineers per track, each backed by AI agents. Product and design float across tracks, providing direction and quality rather than managing tickets. The team covers four times the ground with the same headcount.

Inside each track, the work changes shape too. Instead of one person working on one long task sequentially, each engineer runs many short tasks in parallel through agents. The unit of work gets smaller. The throughput explodes. The engineer's job is to keep the fleet loaded — define the next task before the current one finishes.

This is the organizational expression of the same shift. At the individual level, the engineer becomes a manager of agents. At the team level, the team becomes a network of small, fast-moving crews. The coordination overhead that used to justify large teams collapses when the unit of execution is one person plus agents.

So where does this leave us? Let me bring it back to something practical.

If you're an engineering leader, here are three things to do this quarter. First, **measure where your people actually are** on the four-level framework. Not where they think they are — where their workflows prove they are. Most will be at Level one. A few at Level two. That's your baseline, and honesty about it is the starting point.

Second, **invest in the Level three jump for your strongest engineers.** Give them permission, tools, and time to restructure their workflows. The three-hundred-PR proof point didn't come from a training course. It came from one person who was given the freedom to rethink how they work from the ground up. You need those proof points on your own team — people who can show the rest what's possible.

Third, **start building Level four infrastructure now**, even if most of your team isn't there yet. Shared agent configurations. Shared prompt libraries. Automated pipelines that handle the last mile — CI, review, merge — so agents can ship without human bottlenecks. Internal documentation that captures mistakes so agents don't repeat them. These are the systems that turn individual AI-native practice into team-level capability.

The engineer of 2026 doesn't write code. They plan, orchestrate, and verify. They manage fleets of agents the way today's engineering managers manage teams of people. The job is harder, the thinking is deeper, and the output is an order of magnitude larger. The question isn't whether this shift is coming. The question is whether your organization is building the culture and the systems to make it happen — or waiting until someone else figures it out first.

Based on emerging practices in AI-native engineering, 2025.