

The Engineering Org of the Future

Peter Inge · November 2026

The Engineering Org of the Future will be made up of AI-native developers working alongside fully autonomous agents. Agents will do the majority of the engineering work. Humans will architect the system, guide agents on the hardest problems, and set the guardrails.

Getting there will mean building two workforces at once: AI-native engineers and autonomous agents.

- You have to upskill your engineering org to become AI-native and reconfigure org structures to match.
- You also have to start building software factory capabilities, where agents do development work on their own.

The first is well underway; the second is just kicking off. We believe these two workforces will progress in parallel over the next 12 to 18 months, and then they will merge into one system: AI-native people and autonomous agents, each doing the work they do best. **Those who merge the two will win.**

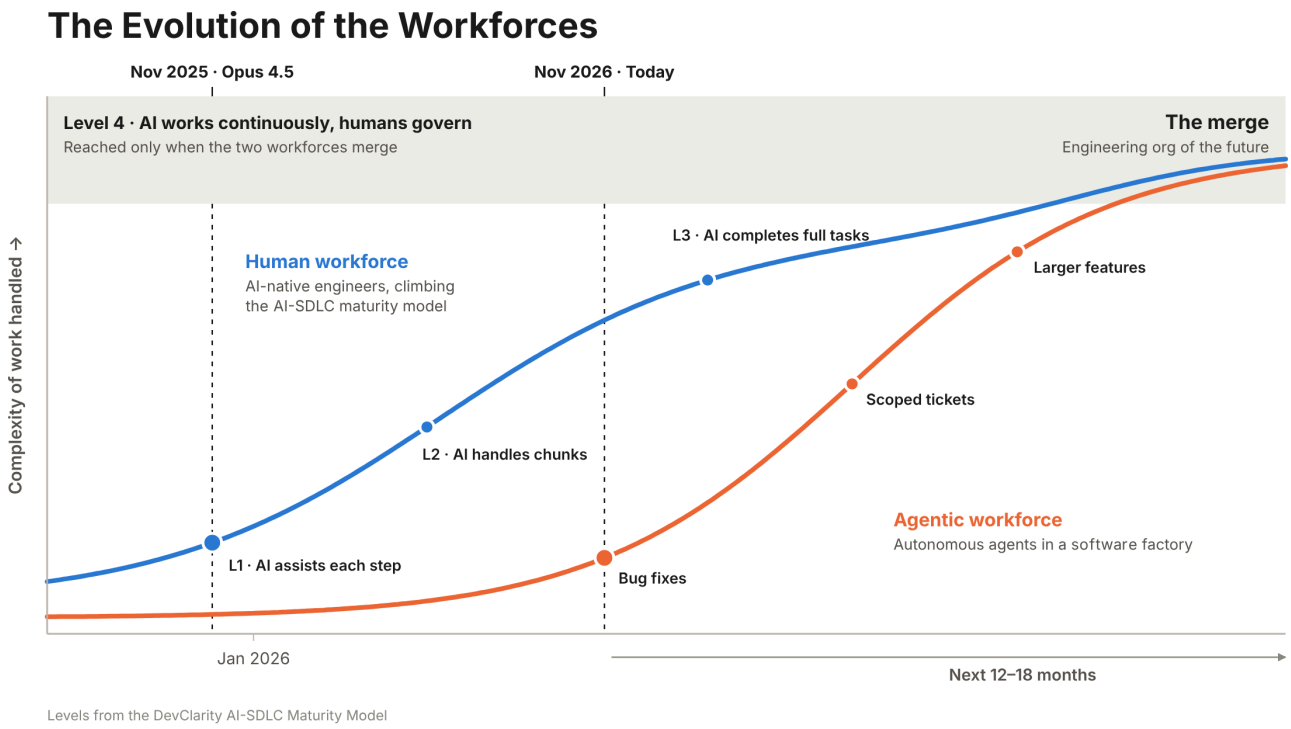
We've worked with more than 100 software companies through the first wave of AI coding adoption over the past year. Below is what we're seeing in each workforce, why the merge is where this is heading, and what we'd do about it in 2027.

Two workforces

For the first time in history, companies are going to have workers who aren't human. While software has automated work for decades, it has been fixed, repeatable processes that someone scripted in advance. AI agents are different. They take on work that changes shape and requires judgment. In the context of development, autonomous agents will pick up tickets, write the code, test it, and ship it. That is a new kind of workforce, and it needs to be onboarded, trained, and managed as one — with the right context, access restrictions, and feedback loops, as well as increasing responsibilities as trust is earned.

The human workforce is changing, too. The engineers who thrive won't be the ones who write the most code. They'll be the ones who know how to manage a fleet of agents: designing the system, deciding what work the agents get, setting the guardrails, and stepping in on the problems agents can't handle yet.

Both workforces will be required, and both will have to be built. **Most companies have started transforming the human side, but the agentic evolution has just begun.**



Levels from the DevClarity AI-SDLC Maturity Model

Two inflection points

Each workforce has its own "oh, this is real" moment.

For humans, it was November 2025. When Anthropic released Claude Opus 4.5, engineering leaders who treated AI coding tools as a nice-to-have started paying attention. The tools could finally do real work, and the push up our AI-SDLC Maturity Model got serious. In twelve months, the best teams we work with went from AI-assisted (Level 1) to AI-native (Level 3).

For the agentic workforce, it's now: November 2026. The questions CTOs bring us have shifted toward impact and what it takes for agents to close tickets on their own. Adoption still matters; most companies have some engineers at Level 3 and a long tail behind them that needs to keep moving. But now there is an equal amount of attention on the second workforce.

The human workforce: AI-native engineers

"AI-native" gets thrown around a lot, so it's worth getting more specific. An AI-native engineer isn't someone who uses AI to write code faster. They aren't writing code at all. They're building the skills, workflows, systems and architecture that let agents do the work, and then orchestrating those agents.

It may be easier to define by process of elimination:

- If you're not using AI tools every single day, you're not AI-native.
- If you're still writing code by hand, you're not AI-native.
- If you haven't multiplied your throughput over the past year, you're not AI-native.

By that test, most engineering orgs still have a long way to go, even after all the progress made in 2026. But we believe that **within the next 12 months, every software company that wants to stay competitive will need its whole team to be AI-native. Not a pilot pod. Everyone.**

The agentic workforce: the software factory

The agentic workforce lives in what's being called a software factory: an autonomous pipeline that can take an engineering ticket from idea or spec all the way to merged, deployed code, with no engineer driving each step.

This workforce is less mature, and as a result there's a lot of noise about software factories right now. Very few companies have one doing real work, and most are just starting to wrestle with the hard questions. *What context does an agent need to do this job well? What should it be allowed to touch? How do we know the output is good enough to merge? Who reviews what, and when?*

Nobody has all of the answers yet, but the companies that start working through these questions now will figure it out first. The answers depend on your codebase, your review process, and your people — and the only way to find them is to run real work through a factory.

The merge

In our maturity model, the highest level is Level 4, "Autonomous": agents doing work nobody explicitly triggered, with humans governing the process. It's tempting to read that as the next rung for your engineers to climb, but it isn't. **Level 4 is what you get when the two workforces merge.** Your people can't reach it alone, and neither can a factory.

A software factory doesn't work in isolation. Someone has to set the guardrails, define the workflows, write the trust criteria, and decide the exit gates a change has to pass before it ships. **Many of those skills grow out of what your engineers learn as they climb to Level 3.** Building skills and codifying workflows is the same work that makes a factory run well.

Those skills, workflows, access rules and validation checks add up to a team's harness: the system it works within. A well-built harness, shared through plugins and governed like any other code, works the same whether a person or an agent is doing the work. **That's where engineering is heading: less writing code, and more designing the harnesses that people and agents both work within.**

And the merge doesn't stop at engineering. As the factory matures, it pulls in the work on either side of it. Product has to write specs that agents can actually execute. QA, code review and DevOps get absorbed into the pipeline. Team sizes and structures change with it. Over time, the same pattern will reach past engineering into go-to-market, operations and finance.

> **So the engineering org of the future isn't a factory with a few humans watching it. It's AI-native people and autonomous agents, each doing the work they do best, built to work as one system. That's the merge, and it's where we think the next few years will be won or lost.**

What to do in 2027

If we were planning 2027 for a software company today, these are the five themes we'd build around.

1. Get every engineer to AI-native (Level 3)

Daily AI usage, custom context & skills in every repo, and throughput that is a multiple of last year's. Engineers review outcomes, not individual lines of code. Not a pilot pod. Everyone.

2. Start your factory, even if you don't feel ready

Most companies we talk to don't feel ready for this, and don't know exactly what to do first. That's ok: **the factory is a forcing function.** Every ticket it gets wrong tells you exactly where your context, skills, specs or tests are weak. An agent stalls on a service because there's no context file for it. Its PR ignores a convention that only lives in people's heads. A ticket fails because the spec was two lines long. It turns a vague goal ("mature our AI practices") into a specific backlog your engineers can work through. The pain is what pulls your people up the curve.

Start small: bug fixes, dependency updates, well-scoped tickets. Expand from there as trust is earned. Run it in parallel with theme 1; each one makes the other move faster.

3. Own your factory

There are vendors who will happily run a software factory for you. For a quick proof of concept, that can be a fine way to see what's possible. As a long-term strategy, we think it's a mistake.

Our reasoning is simple. If you're a software company, building software is the job. Your factory will hold the things that make your engineering org yours: the context about your codebase and customers, the workflows your best people have codified, the judgment about what's safe to ship. Handing all of that to someone else is no different from outsourcing your engineering org.

This is where the job of engineering is heading. The act of building software is getting commoditized. Your **engineering system** is not: the company-specific context, preferences, workflows and guardrails that tell a factory how *your* company builds, and that connect it to the rest of your systems. It's also what turns your understanding of your customers into the product they pay for. **More and more, engineering becomes the work of building and maintaining that system.**

That doesn't mean you have to build every piece yourself. Get help with the engine: keeping it current as new models ship, running the evals that catch when its behavior changes, and maintaining the integrations with your tools. Keep ownership of the context, the workflows and the guardrails. This is

the split we help companies make at DevClarity. We work alongside your team on the engine and help you build the in-house muscle to own the rest.

4. Measure adoption *and* impact

Adoption tells you how the people journey is going. Impact (PR throughput, cycle time, escaped defects, engineering allocation) tells you whether it's paying off. Tickets closed autonomously tell you how the factory is doing. Our metrics framework maps which numbers matter at each level.

5. Redesign the org around both workforces

Decide who architects, who governs the agents, and which work goes to which workforce. Push toward smaller teams with product and engineering together. Make product responsible for defining quality as a requirement, and have engineering design the system to enforce it.

The work ahead is significant, but so is the opportunity. If you want a trusted partner to help build your Engineering Org of the Future, **reach out to DevClarity**.