Talent in the Age of AI

Change Management for the Much-Needed AI Adoption in Organizations

By jpelayo · February 23, 2026

	Nombre	Qué cambia
0	Autocompletado Extremo	La IA sugiere la siguiente línea. Aceptas o rechazas. Solo una tecla tab más rápida. El humano sigue siendo quien escribe el software.
1	Becario de Código	Le das a la IA una tarea acotada: una función, un componente, un refactor. La IA ejecuta la tarea. El humano sigue siendo dueño de la arquitectura, el juicio y la integración.
2	Desarrollador Junior	La IA navega el código, maneja cambios en múltiples archivos, construye funcionalidades entre módulos. El humano sigue leyendo todo el código.
3	Desarrollador como Jefe	El humano deja de escribir y empieza a dirigir. Revisas a nivel de feature y de PR. La mayoría topa aquí porque soltar el código es psicológicamente difícil.
4	Desarrollador como Product Manager	Escribes una especificación, te vas, vuelves horas después y compruebas si los tests pasan. Dejas de leer código. Solo evalúas resultados.
5	La Fábrica Oscura	Entran especificaciones, sale software funcional. Ningún humano escribe o revisa código. La fábrica funciona con las luces apagadas. Aquí es hacia donde va la industria.

The core of a company is its talent and processes. A large part of process management weight is carried by systems deployed and maintained by the IT department, which often internalize software development to maintain, adapt, or improve the business system. Their departmental processes have evolved over 50 years with bigger or smaller leaps, but the one coming now is absurdly huge. Most companies are not realizing the magnitude of the change that comes with AI-driven code development. And it's not a change for the worse, because it will give more prominence to the company's core (its processes), but it will alter the definition of talent. The company that makes the right bet today will improve its long-term prospects. Whoever makes the losing bet or buries their head in the sand will sink, dragged down by a part of their organization that will not be competitive.

Where are we today? Let's see.

The Loop Has Closed and It's Moving Fast

The self-referential loop has been installed in both Anthropic and OpenAI. Codex 5.3 is the first cutting-edge model that was decisive in creating itself, and it's not a metaphor. It was launched as a direct product of its predecessor's coding work. OpenAI reported a 25% speed improvement and 93% fewer wasted tokens, gains that the model found by analyzing its own inefficiencies during the training process.

Claude Code does the same. 90% of its own code was written by Claude Code, a number converging toward 100%. Boris Cherny, who leads it, says he stopped writing code months ago. His role shifted to specification, direction, and judgment. At Anthropic, everyone is defining architecture and machines are writing.

The feedback loop has closed. The question is no longer whether AI will improve AI, but the acceleration of that loop; and what it means for the fifty million people who currently write software to earn their living.

The J Curve

However, here's where we are: most organizations are not seeing gains but are becoming slower. Developers trying to add AI to existing workflows spend their time evaluating suggestions, correcting almost-correct code, switching context between their mental model and the model's, and debugging subtle errors that seem correct-but-not. 46% of developers in broader surveys say they don't fully trust AI-generated code.

When you add a new tool without redesigning the workflow around it, you're putting a new engine in an old transmission. Productivity drops before it improves, sometimes for months. Most organizations are at that point right now, interpreting the slowdown as evidence that AI doesn't work.

Copilot is the clearest illustration. Twenty million users, 42% market share, and lab studies showing 55% faster code completion in isolated tasks. In production... the story is more complicated: larger pull requests, higher review costs, more security vulnerabilities. Organizations seeing 25-30% gains are those that redesigned their entire development process, how they write specs, review code, structure CI/CD.

The Real Bottleneck

Every ceremony in a software organization exists because of a human limitation. Standups synchronize people sharing code. Sprint planning manages human working memory. Code review catches human errors. QA exists because builders cannot objectively evaluate their own work. When humans no longer write the code, none of this disappears gracefully. It becomes friction.

Structural change is harder to see than technological change. The manager's value shifts from coordinating the team to defining the specification with enough clarity for agents to build the functionality. Writing a specification precise enough for an AI agent is a genuinely different skill. Machines build what-you-described. If what you described is ambiguous, filler appears in the gaps. The bottleneck has shifted from implementation speed to specification quality.

We'll leave for another occasion the discussion of the average technical person's ability to articulate complex messages verbally, and their written expression, grammatically and orthographically correct. For now just this: read more.

The Scenario Technique

Traditional tests live inside the code. The AI agent can read them, which means it can optimize to pass them rather than building source code appropriate to the objective function: it's the classic problem of studying for the exam. Perfect grades, superficial understanding.

You need to use scenarios. Scenarios live outside the code, stored separately so the agent never sees them. They work like a holdout set, the same concept used in machine learning to prevent overfitting. The agent builds the software; scenarios evaluate whether it actually works. The agent never sees the evaluation criteria. It cannot game the system.

At an appropriate volume of work, AI agents operate at a scale where compute cost becomes significant, and yet it's still usually cheaper than the humans they replace: if you haven't spent $1,000 in compute per engineer per day, your organization has room for improvement.

The Processes

Most enterprise software is existing systems vegetating over the years, running in production, sustaining revenue. Monoliths grown over fifteen years of adding features. Configuration management that lives in the heads of three people who remember why that environment variable has that specific number.

You can't apply the "dark factory" to a legacy system. The specification doesn't exist. Tests, if any, cover 30% of the code; the other 70% runs on oral tradition. The system is the specification and the only complete description of what the software does.

The path starts with documentation. Reverse engineering the implicit knowledge in a running system requires domain experience, unfiltered honesty, systems thinking: exactly the human capabilities that matter more in the age of the "dark factory," not less.

The migration looks like this: first, use AI at level 2 or 3 to accelerate your developers' existing work. Second, use AI to document what your system actually does. Third, redesign the validation and deployment pipeline. Fourth, begin moving new development to autonomous agents while keeping legacy in parallel. That path takes time. Whoever tells you otherwise is selling you something.

The Talent

Software engineering has always been a learning model dressed in business clothing. Juniors learn by doing: simple features, small bugs, immersion in code. Seniors mentor, review, catch errors. In five to seven years a junior becomes a senior through accumulated experience. AI breaks that model from the bottom. If AI handles simple features and bugfixes, where do juniors learn? If AI reviews code faster than a senior, where does mentorship happen?

And yet we need more elite engineers, not fewer. The minimum required level is rising toward exactly the skills that have always been hardest to earn. The 2026 junior needs the systems thinking expected of a mid-level engineer in 2020, not because the work has become harder, but because entry-level work has been automated.

What matters now is: systems thinking, customer intuition, ability to hold an entire product/project/program in your head, and ability to write a specification precise enough for an autonomous agent to implement correctly. Those skills have always separated great engineers from mediocre ones. The difference is that "adequate" is no longer a viable position at any level, because "adequate" is what AI models do.

The Demand

Every time computing costs have fallen, from mainframes to PCs, from PCs to cloud, from cloud to serverless, total software produced hasn't stayed flat: it has exploded. New categories have emerged that were previously economically impossible: SaaS, mobile, streaming, real-time analytics.

We're lowering the cost of software production by an order of magnitude. A regional hospital, a mid-sized manufacturer, a family logistics business cannot afford custom software at current labor costs. A custom inventory system can cost $500,000 and take over a year. That unmet need is now affordable.

Although it's no consolation to those suffering job loss, know that demand for software has never saturated. The constraint always shifts to the next bottleneck. Here, that bottleneck is judgment: knowing what to build and for whom. The "dark factory" doesn't replace the hardest people to replace. It amplifies them.

What's Coming

The "dark factory" is real. A small number of teams produce software without any human writing or reviewing code, delivering production-ready results that improve with each new generation of models. The feedback loop is closed and those teams move faster with each iteration.

Most companies are stuck at level 2, becoming measurably slower with tools they think are making them faster. Both things are true simultaneously. The vanguard is further ahead than almost anyone wants to admit. The center is further behind than vanguard teams recognize. The distance between them is not a technology gap. It's a people gap, a culture gap, an organization gap, a willingness to change that no tool or vendor can close.

The companies that survive are not those that buy the best coding tool. They are those that do the hard, slow, unglamorous work of documenting what their systems do, rebuilding their organizations around judgment rather than coordination, and are humble and honest enough with themselves to know this transition won't happen as fast as they want, because people change slowly.

The "dark factory" doesn't need more engineers. It needs better ones. Better means people capable of thinking clearly about what should exist, describing it with enough precision for machines to build it, and evaluating whether what was decided to be built actually serves real humans. That has always been the hard part of software engineering. We've just, for years, let implementation complexity hide how many people were really good at it. The machines have removed that cover. We're about to discover real meritocracy, the value of authentic talent.

Credit and acknowledgments: Nathan Jones