AI PLAYBOOK · 2026

Bessemer Venture Partners

The Agentic Awakening

Why 10× faster coding doesn't translate into proportional organizational productivity, and how AI-pilled leaders do it.

Liran Eshel and Adam Fisher

IN THREE PARTS

I. Build the Churches · II. Convert the People · III. Assemble the Community

Copyright © 2026

FROM THE AUTHOR

How I came to this work.

About three years ago, after more than twenty years in CEO seats running software companies, I finished my last role and finally had time on my hands. I found myself completely absorbed by the generative-AI revolution that was just starting, reading and analyzing almost every piece of writing that came out, and not only on the technical side. The social, historical, political, and economic implications fascinated me at least as much as the models themselves. The whole shape of what AI was about to do to the world had me in a state I hadn't been in for years.

But the further I went, the more one specific itch kept returning. I had always believed that software development was the area where AI's future would be most visible first, because it was already the most advanced area of AI usage. If I really wanted to understand where this was going, I needed to *build*. There was just one problem: I had not written production code in a very long time. Different languages, different runtimes, a different operating model entirely. I felt completely out of shape. But the thought kept returning: *why can't I do it?*

So one day I just sat down and dove in. The first move was small: experience what it felt like to work inside an agentic IDE. Within hours I was addicted. That experience became the first platform I built, AI-assisted from end to end, letting me move on things I hadn't touched in years.

The second platform, just one year later, was already purely agentic: I directed intent and reviewed output; the code wrote itself. By that point I had forced myself through every serious tool in the space, and the agent harnesses layered on top of them, comparing them on real work rather than on benchmarks. I was running multiple agent sessions in parallel and routinely generating hundreds of thousands of lines of production code per month, operating in a mode that had no analog in how software was built even two years earlier, let alone two decades. In the gap between those two builds I watched the shift from 2024 to 2025 happen in real time. The whole way software gets built was changing under me. That was when I felt *AI-pilled*.

I use the word *addicted* deliberately. The pull is the same one that hooked me on programming in the first place, decades ago: you type something, the machine obeys, and a small jolt of *it worked*, a hit of dopamine, lands before you've thought about what comes next. Agentic coding turns that hit into a slot machine: you write a sentence, the agent disappears for a minute, and comes back with a working feature, or a near-miss that's one more prompt from working. The reward is variable and the friction is gone, the same loop that makes a video game hard to put down. Every win suggests three more things to try, and the cost of trying is a sentence. I lost more than one night to *just one more prompt*, and I was nowhere near alone.

When Adam Fisher from Bessemer Venture Partners approached me about this project, I saw an opportunity to study the same shift from the other side of the table (across their portfolio and beyond, at scale) and to put two perspectives in the same frame: the hands-on experience I had been accumulating as an AI-native builder, and twenty-plus years of running software companies. The combination let me ask sharper questions than either viewpoint alone would have allowed. This playbook is the result.

Across our conversations with portfolio companies, four patterns kept appearing. Some teams believed they were making progress but had no benchmark against which to measure it. Others were moving quickly but kept their methods close, making it difficult to compare approaches. Some were spending heavily on AI without seeing the investment translate into output. And some knew what needed to change but could not get the organization to move. We realized the field needed more than another report: it needed a practical guide for founders, CEOs, engineering leaders, and boards navigating the transition to agentic engineering.

The work behind this document

During the first half of 2026, we spoke in depth with CTOs and engineering leaders from more than twenty companies, including Ramp, Lemonade, Wonderful, and DriveNets. They ranged from small AI-native startups and mid-stage software companies to late-stage organizations with codebases over a decade old. The study spans consumer software, B2B SaaS, infrastructure, security, e-commerce, marketing, and several other verticals, broadly enough that the patterns are not specific to any one industry.

The interviews ran in successive rounds, with findings synthesized between each. The framework barely needed to change as the sample grew round over round, itself a finding. The underlying patterns are more consistent across companies than the surface differences suggest.

This playbook's production followed the same agentic model. The thesis, interpretation, editorial judgment, and final prose were human; agents helped analyze interview evidence, cross-check external research, and challenge drafts through several models. The source lived in HTML and Git rather than Word, allowing agents to work in parallel, propose changes as diffs, preserve a complete version history, test layouts, and generate web, PDF, and slide artifacts through code. This made iteration across research, writing, design, formats, and versions far faster than a traditional document workflow, while human judgment determined what the evidence meant and what merged.

Why a playbook, not a report

This is a playbook, not a report. A report tells you what other people are doing; a playbook tells you what to do next. The engineering leaders we spoke to did not want a trends deck; they wanted to know: *where am I on this curve, what am I missing, what is the next move?* The four-level maturity scales in each step are the answer. Read them honestly, locate yourself, identify the gap, start building. The field examples scattered through the asides are not the prescription; they are calibration points.

We deliberately included real stories from the field. Every leader we spoke to was curious how their organization was doing relative to the others, the most common question we heard, in some form, was *"am I behind, ahead, or about typical?"* The anonymized vignettes in the asides exist to answer that question without naming companies. They also do something a framework alone cannot: they generate ideas. More than once, a leader asked about a play another company had run, and went off to try a version of it the next quarter. Stories travel where frameworks don't.

What surprised us

What surprised us most through the interviews was the pace. We had expected to find a few aggressive movers and many cautious laggards. Instead we found something stranger: every leader we spoke to had made moves that twelve months ago would have been considered reckless: week-long org-wide tool cutovers, 60% management cuts, PM-to-engineer ratios inverting, $1M AI bills exceeded in two months. And every one of them was making those moves with conviction. The pace alone tells you something.

We came away convinced of one thing above all: the technical changes are real and moving fast, but already well ahead of most organizations' ability to absorb them. Many mistake that motion for absorption. Track lines of code as proof you're transformed and you've taken a placebo: the number climbs while nothing real changes. The decisive work over the next two to three years won't be just about the tools and infrastructure. It will be about **the people** who have to change how they work, moving from writing code to directing agents, and about **the organization** that has to change shape around them: the team structures, planning cadences, and ownership models that decide whether the new speed compounds or gets absorbed. **The three problems interlock: once the infrastructure is real and enough people convert, organizational redesign stops being optional. This is the agentic awakening: a wave moving through the software industry, overturning methods and practices that worked for decades. Some already see its shape; most are still trying to understand what it means for their work, their careers, and their companies.** That is the arc of this playbook: build the infrastructure, convert the people, restructure the organization.

Liran Eshel · June 2026

@liranesh

theagenticawakening.com

A three-part field report on becoming an AI-native engineering org.

I Build the Churches

The Foundation. Coding infrastructure, AI Ops ownership, measurement, and security, the environment in which AI-native work can happen at all.

II Convert the People

The human pillar. Moving engineers and non-engineers from traditional work into AI-native execution as the default professional mode.

III Assemble the Community

The organizational pillar. Redesigning teams, roles, planning loops, and ownership models around the new speed of execution.

Key Takeaways.

1 **"90% of our code is AI-written" ≠ agentic engineering.**
Without autonomous-agent infrastructure (multiple parallel agents per engineer), you're capped by human attention.

2 **From driver-assist to a fleet of robotaxis.**
The AI IDE is driver-assist; coding agents take your hands off the wheel; dark factories let you manage a fleet of independent agents.

3 **No tokens, no agents.**
Real agentic engineering runs at least $1K/month per engineer on average; if your spend is a small fraction of that, you've got an IDE plugin, not a fleet.

4 **AI Ops is the new DevOps, and the gate on autonomy.**
The stack turns over monthly, faster than anyone with a day job can track, so a dedicated team must own the internal AI infrastructure, and the wiring into your own runtime is the part you can't buy off the shelf.

5 **The top 1% of engineers ship 46× the median's AI-written lines.**
Agentic gains don't spread, they concentrate: manage the distribution, not the average.

6 **Bottom-up gets you demos; top-down gets you change.**
The organization's leaders must be AI-pilled by first-hand experience and drive a sharp turn from the front. This is a moment for bold moves, not incremental change.

7 **A Ferrari at every stoplight.**
Engineers work 10×+ faster, yet organization-level gains stall below +50%: rebuild the organization and its processes to capture the rest.

8 **Faster code makes requirements the bottleneck.**
The PM-to-engineer ratio breaks to one extreme or the other, widening well past 1:10, toward engineers absorbing the PM role, when they can proxy the customer and feed their own requirements, or tightening to 1:1 when each engineer needs a dedicated PM.

9 **The beta is the spec; the PR is the handoff.**
Product and design can hand engineering working software rather than representations of it. Engineering still owns merge and production approval, but the translation loop disappears.

10 **The org tree gets shorter and wider.**
AI reduces the coordination work that once required narrow spans. The most aggressive adopters now run 15–25 reports per manager; larger organizations keep management, but with fewer layers and managers per engineer.

11 **Four experienced chiefs at the wheel: Architect, Product, Designer, Security.**
Critical for consistency, product strategy, brand identity, and security, the big-picture supervision agents can't hold on their own.

12 **AI in the product is defense and dividend.**
Reinvent your product as an agentic skill before your new AI competitors do, and use it to build a new AI center of excellence, and a talent magnet.

PART I

Build the Churches

A transformation does not scale through belief alone. It needs places, rituals, tools, operating rules, shared language, and visible proof. The first pillar is the foundation: the technical infrastructure and operating system through which humans and agents can work together at scale.

It's tempting to call this done when 90% of your code is AI-written, but that isn't necessarily agentic engineering.

Using an analogy to self-driving, the AI IDE is driver-assist, coding agents take your hands off the wheel, and a lights-out "dark factory" lets one engineer run a fleet of independent agents.

"If you build it, he will come."

– FIELD OF DREAMS (1989)

The Foundation.

Before an organization can convert its people or restructure its teams around autonomous agents, it must first build the environment in which AI-native work can reliably happen.

The metaphor is intentional. The *churches* are the infrastructure, the operating processes, and the dedicated experts that turn new behavior from individual experiment into organizational practice. A company that has merely purchased AI coding licenses has not completed the foundation; it has only opened the door.

The foundation has four dimensions, each scored on its own maturity scale.

Coding Infrastructure	AI Ops	Measurement	Security & Compliance
CORE QUESTION	CORE QUESTION	CORE QUESTION	CORE QUESTION
Are AI coding tools deployed, mandated, and in daily use across engineering?	*Is there a dedicated team owning internal AI infra?*	*Can we quantify the AI acceleration we are getting?*	*Do we understand the new risk surface agents create, and have we bounded it?*

Coding Infrastructure puts the capabilities into your team's hands. **AI Ops** keeps you current with a pace of change no one doing their day job can track; it takes dedicated focus. **Measurement** tells you whether the investment is paying off: that the team is genuinely using the tools, and that it isn't overspending to do it. **Security & Compliance** keeps you from exposing yourself to new AI-era risks, and keeps you compliant. Underbuild any one and it becomes the ceiling on the rest.

Coding Infrastructure

This dimension covers the full arc of AI inside the coding loop, from a coding assistant the engineer steers keystroke by keystroke, to a team that triggers and ships work itself. The maturity scale below maps it as four levels; the paragraphs underneath walk each level in detail.

The Maturity Scale

LEVEL I	LEVEL II	LEVEL III	LEVEL IV
DRIVER ASSIST	SAFETY SYSTEMS	DESTINATION-LED	FLEET AUTONOMY
AI as the Assistant	**AI as the Reviewer**	**AI as the Engineer**	**AI as the Team**
Interactive human–AI coding inside the IDE. Engineer steers; multiple sessions; tight back-and-forth.	AI added to the CI / CD pipeline: PR review, test triage, runtime observability.	Long-running sandboxed agents. Multiple in parallel per dev, where 10×+ leverage kicks in.	The "dark factory" of code. Self-triggering and self-healing, with humans governing consequential changes.

The two jumps that matter: **Assistant to Engineer is taking your hands off the wheel**, the moment you stop reviewing every line. **Engineer to Team is no longer needing your own car**, the moment the loop closes around you. And just as no city skipped straight to robotaxis, no organization in the study reached the final stage without first building every layer underneath.

The four levels, in detail.

AI as the Assistant. Interactive human–AI coding inside the IDE: the engineer watches the AI write, runs several sessions at once, and guides every step with tight back-and-forth. The field grew up here: from tab completion (GitHub Copilot, 2022) to chat-in-the-IDE (Cursor, 2023), and **Cursor and GitHub Copilot** remain the dominant tools at this level. The acceleration is real but bounded: the human is still the throughput constraint.

AI as the Reviewer. AI added to the CI / CD pipeline as an automatic review layer: PR review, test-failure triage, security findings, runtime observability (Sentry, Datadog, Honeycomb piped to AI). First-party tools now cover it: **Cursor's Bugbot** (agentic review with auto-fix), **Anthropic's Claude Code review** and **/security-review** , alongside independents like CodeRabbit, Qodo, and Graphite's Graphite Agent. Easy to wire in because it doesn't change how engineers code, and routinely skipped for flashier autonomous work, even though it pays back on its own.

AI as the Engineer. Long-running, sandboxed agents that write code, open PRs, and run multi-hour tasks on their own. Multiple concurrent agents per developer is where 10x+ leverage kicks in and the job shifts from writing to directing. The edge existed earlier but was narrow: one study company had used Cognition's Devin since an early beta, well before autonomous agents became common.

It is the first level that demands real infrastructure (the progression runs *worktrees → containers → managed VPSs*) and a striking number of orgs **build that layer themselves** (parallel-session orchestration, make-agent-style commands, a sandbox fleet) rather than wait for a vendor, because off-the-shelf tooling doesn't yet match how they run agents at scale. **Claude Code** is the dominant tool here, with **OpenAI Codex** the major-vendor alternative. **Cursor** approaches the same level from the IDE side, with an agent-first workspace, automatic model routing, and its Composer model family. This is where the study's mid-stage and advanced organizations now operate.

AI as the Team. The closed loop, and not just for fixing. Triggers come from everywhere: caught bugs, CI failures, production incidents, telemetry anomalies, customer requests, and increasingly **agents listening on Slack threads, PRD drafts, or planning calls** that start building before anyone formally asks. The trigger feeds observability or the planning layer; agents act on tests deep enough to validate the work; deploy closes the loop. The industry calls this a *dark factory*, manufacturing's lights-out model (FANUC, Xiaomi), now applied to code. The line between the Engineer and the Team is that the system begins the work and closes the loop itself. Human approval becomes risk-based rather than the trigger for every step; people stay in the loop for judgment, not keystrokes.

The VP Engineering dream: software that maintains itself.

The dream is a self-healing feedback loop. Early signals across QA, staging, production, and live user behavior are continuously collected, scanned, and correlated. When something begins to break, agents reproduce it, identify the likely cause, and turn it into a tested fix, often before a user complains.

The human operator receives a PR with the evidence, approves the merge and rollout, and the system verifies the repair on the live system. Narrow, reversible fixes can eventually run automatically.

For legacy companies, the prize is not only faster development. It is reducing the maintenance burden: fewer hours spent investigating regressions, recurring incidents, dependency drift, and support escalations.

> **"The system identifies anomalies and broken parts, not only in code, but in usability, language, and conversion, and carries the repair through gradual deployment, testing, and validation, end to end, while we sleep."**
>
> – Technology leader, AI-native public company

AI Ops

AI Ops is the new DevOps. Just as organizations learned that infrastructure, deployment, monitoring, and developer productivity could not be left to every engineer to solve independently, the same is now true for AI. The name is deliberate and worth separating from an older one: AIOps, in Gartner's sense, means applying AI to IT operations. This is the reverse: the team that builds and owns the AI infrastructure everyone else engineers on top of. The field changes too quickly: new models, tools, workflows, pricing, security patterns, and agent architectures land constantly, and no engineer doing their normal job can credibly keep up.

The misconception is that this is a procurement task: buy the tools and switch them on. It is the opposite: a strategic function, the hardest role on this list to hire for, and the gate on autonomy. It can't simply be bought, because the infrastructure isn't generic; it has to be wired into your own runtime, and that wiring is always in-house. Licenses get you assistants and reviewers; independent, long-running agents run on infrastructure only this team builds.

The Maturity Scale

LEVEL I	LEVEL II	LEVEL III	LEVEL IV
No dedicated owner	**Official, part-time owner**	**Dedicated AI Ops team**	**Whole-org enablement**
Handled informally by the CTO, head of R&D, or scattered champions.	A named person with mandate, but still a side responsibility.	2–5 FTEs with a roadmap, budget, and ownership of internal infra.	Supports non-engineering builders; catalyst and guardrail for shadow AI.

The reason the team has to exist is the **pace**. New models, new harnesses, new pricing structures, new tools: any can land in a given week, and no engineer doing their day job can keep up. Dedicated AI Ops experts read every release as it ships, follow the practitioners building the tools directly: Boris Cherny at Anthropic (Claude Code), Peter Steinberger at OpenAI (creator of OpenClaw, now working across its agent products including Codex), and the wider set of public thought leaders, test what they find against the org's actual workloads, and bring what they learn back inside. Without that team, an org gets stuck on one environment or harness, paying in lost capability and cost efficiency. The role is the AI-era version of what DevOps became, except the cycle is measured in days, not quarters.

> **"There used to be a committee that vetted and approved every new AI tool. We replaced it with a dedicated AI-Enablement team (IT, security, infrastructure, and AI experts) chartered to set the architecture and move as fast as possible to remove blockers."**
>
> – Head of AI transformation, large infrastructure company

The shape recurs across the study: small (2–5 people), cross-functional, full-time. One legacy SaaS chartered an *Office of the CTO* with the same remit. The team also reshapes **procurement**: once the agent stack is standardized, every new SaaS purchase is filtered through whether it fits: does it expose an MCP (Model Context Protocol) server, can agents drive it.

At full maturity the function grows *outward*, beyond engineering, into the org's **whole-org enablement layer** for AI: PMs building prototypes, finance teams wiring reporting agents, and designers shipping production code all need a place to bring questions and stay inside the org's security and cost guardrails. Without it, the failure mode is *shadow AI* and *app sprawl*, teams running their own credentials and one-off apps no one knows about until something breaks or leaks.

Measurement

Measurement is how you monitor progress through the agentic SDLC. Done well, it answers the questions that actually matter: which teams and engineers are genuinely converted and which only say they are; which teams have hit real productivity acceleration and which are just burning tokens; and how you compare to industry benchmarks.

Measurement is where organizations in this study lag the most, and at both ends of the spectrum. The slower movers simply haven't reached it yet: still mid-rollout on tools and people, they leave it for later. At the other extreme, AI-native and sharp-turn orgs skip it on purpose, because the game, they feel, is speed, and instrumentation reads as drag. But even they are coming around, because without it token-maxxing becomes a runaway spend spiral and you can't make grounded decisions about where to push and where to pull back.

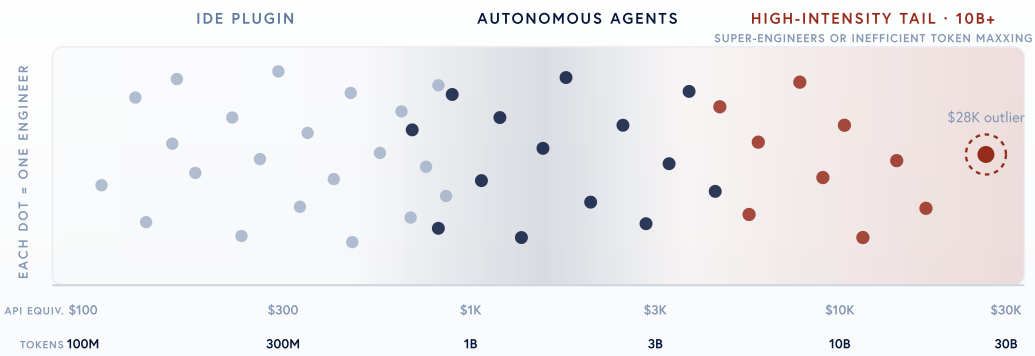
The scale below runs through four levels. It begins with **usage**: who's using which tools and how much, the easy data where most organizations stop. Then **quality**, whether the AI-generated code holds up. Then **productivity**, whether you're genuinely shipping faster, which only means something once work is complexity-weighted. And finally **benchmarking**, how your velocity, cost, and output compare to peers. Each level is harder to measure than the last, and more useful.

The Maturity Scale

LEVEL I	LEVEL II	LEVEL III	LEVEL IV
Usage	**Quality**	**Productivity**	**Benchmarking**
Stats collected from every AI dev tool: tokens, LOC, sessions, who is using what.	Code quality, regressions, security findings, test cycles: does AI-generated code hold up?	Is the org delivering faster? Requires complexity-weighted PR scoring to be meaningful.	Compare velocity, cost, and output against peer companies and industry cohorts.

Usage is the floor: measure tokens and spend

Monthly agentic-compute intensity per engineer



Locate each engineer with **two numbers: total tokens and API-equivalent spend**. Below roughly **1B tokens per month**, about **$1K** at Opus list prices after cache adjustment, you usually have an IDE plugin, not a fleet. An engineer operating at **Coding Infrastructure Level IV can exceed 10B tokens per month**, which at the ratio observed in this study can imply roughly **200K accepted LOC**, although project type can move the ratio materially.

High usage can mean two very different things: a super-engineer producing extraordinary output, or inefficient token-maxxing. Separate the two with quality-adjusted throughput and cost per accepted result. Packages, caching, and model routing can make the same intensity more than **10× cheaper**.

Most of the organizations furthest along averaged **$1–$2K per engineer per month**, but the average hides a wide internal distribution: a few heavy parallel-agent users at **$8–10K/month**, many lighter users near zero. The bimodal shape inside the average is the real story. One organization in this study reached **$12,000/engineer/month** as its org-wide average: the outlier of the cohort, and an early signal of where heavy-agentic spend can land.

The same intensity that converts a skeptic is what makes the spiral real. At one AI-native company, one engineer used roughly **$28,000** worth of tokens in a single month, **$16,000 of it from one `/goal` run that spawned fifty parallel agents** and left them looping; the team found it *"would run four hundred hours if you didn't stop it."* None of it was instrumented until after the fact. Measure *before* the spike, not after.

The picture has now surfaced publicly at scale. Uber rolled out Claude Code to its 5,000 engineers in December 2025; by March 2026, 84% were classified as agentic users and more than **70%** of committed code was AI-generated, and the company had burned through its entire 2026 AI budget in four months. CTO Praveen Neppalli Naga, candidly: *"I'm back to the drawing board because the budget I thought I would need is blown away already."* Salesforce is the public benchmark at scale: Benioff says Salesforce expects to spend close to **$300M** on Anthropic tokens in 2026 for the company's ~15,000 engineers (roughly $1,700/engineer/month), paired with an explicit hiring freeze on engineering and a public claim of 30% productivity gains. The framing is sharper than the dollar number: Salesforce is publicly substituting tokens for headcount, and there is plausibly more spend to come.

Pricing mechanics · accurate as of July 2026 · this layer changes monthly

Subscription pricing as arbitrage. Claude Max 20x and ChatGPT Pro 20x cost $200/month, versus $2–8K/month at API list price for a heavy agentic engineer. Many organizations cannot use individual plans because they lack centralized billing, SSO, and audit logs; where the control environment allows them, the gap is too large to ignore. The discount carries a control tradeoff: consumer accounts are not zero-data-retention environments, so disable model training and evaluate the provider boundary separately.

Routing becomes an economic requirement at this intensity. At the Claude Code mix measured across this study's own sessions in July 2026 (roughly 94% cache reads, 5% cache writes, and under 1% each of output and fresh input), 10B tokens would cost roughly **$10–12K/month on Opus** and **$20–23K on Fable** at list prices. The common pattern is a portfolio: subscription access to top models for planning and hard tasks; faster, lower-cost models for routine agent operations.

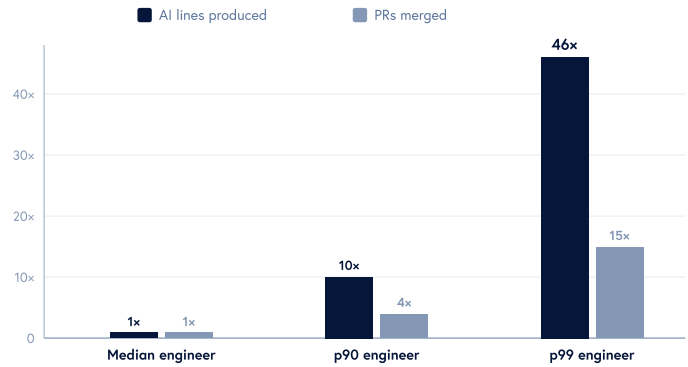
Lower-cost substitution is no longer confined to chores. GLM-5.2 brings long-horizon agentic engineering into low-cost coding plans; Qwen, DeepSeek, Kimi, and small fine-tuned models handle tests, documentation, triage, and other repeatable operations. Cursor's Composer 2.5 (built on Kimi K2.5) is another standout, matching frontier models on key benchmarks at **~10–60× lower cost per task** on Artificial Analysis's measurements.

It's the distribution, not the average.

The useful number is not the average; it is the **distribution**: by engineer, team, workflow, and accepted output. The skew that showed up in spend appears even more sharply in what gets produced.

The cleanest public proof comes from **Cursor's 2026 Developer Habits Report**: AI output is **brutally concentrated**: the p99 ships **46× the median's AI lines** (Gini 0.77). A widening median-to-p99 gap is the signal that AI compounds for the converted few, not the whole org. Other datasets rhyme: CircleCI found the **top 5% of teams nearly doubled throughput** while the median grew **+4%**, and one study company saw a third of engineers at 3–5× parallel throughput, a third single-threaded, and a bottom third barely using agents.

Concentration of AI output



*AI output by developer percentile – Cursor 2026 Developer Habits Report. Gini **0.77**: the median is dwarfed by the p99.*

And the organizations that moved *fastest* are often the ones furthest behind on measurement. Fewer brakes, but also less control: when the bill finally arrives, there is no usage-and-output dashboard to interpret it with, and the only responses left are to absorb the cost or cancel the program. Measurement, the subject of this step, is what would have given them a third option.

Quality is the second-hardest dimension to measure

AI-generated code adds two requirements on top of traditional quality measurement: an **attribution layer** (which lines came from an agent) and a **comparison layer** (is the agent's output better or worse than what the engineer would have shipped manually). Without those, every "AI quality" claim is anecdote.

One study CTO proposed a more operational quality bar: **merge-readiness**. Cognition's FrontierCode asks whether a repository maintainer would actually merge the PR, combining correctness with regression safety, test quality, scope discipline, style, and codebase fit. Internally, measure the same idea through first-review acceptance, rework, and revert rates, segmented by task complexity.

The emerging metric set:

Acceptance and revert rates. What share of agent-written PRs merge on first review, and what share get reverted within two weeks of merging. A high first-review merge rate paired with a low revert rate is the cleanest single signal that the team trusts (and can trust) what the agents produce.

Regression rate per AI-touched module. Tag modules agents have touched; track incidents against them vs. human-only. Orgs doing this report no meaningful delta, itself the justification for further delegation.

Accepted LOC vs. generated LOC. A 90%-generated, 30%-accepted ratio means expensive tokens are being thrown away, a direct AI-Ops tuning signal.

Security findings rate. *SAST* (static application security testing, scanning the source code) and *DAST* (dynamic application security testing, probing the running application) on every PR, with particular attention to agent-generated code. Adversarial-agent setups (writer + critic, cross-model review) exist to catch what one model misses with another. False-positive rate counts as a quality dimension too: noisy AI review erodes engineer trust faster than no review at all.

The gap: most organizations in this study aren't running these systematically. Cost is the only line almost everyone watches; quality remains aspirational for the majority. There's a reason it stays that way, beyond effort: engineering is extremely noisy. Output varies wildly across modules, developers, and the kind of feature being built, so statistical measures are hard to trust, and too often the result you see is mostly chance.

Productivity needs a complexity denominator

Counting lines of code or tokens is the easiest place to start and the most misleading place to stop. A principal engineer fixing the company's hardest bug might ship a single line in a 24-hour session; a junior writing routine CRUD might ship two thousand lines in the same time. Without a way to weight what was shipped by how hard it was, every "AI productivity" claim is suspect, and every "X% of our code is AI-generated" headline is an input metric, not an output one.

Two approaches to the complexity denominator have emerged, from opposite directions: a **learned model** and a **fixed taxonomy**.

The learned-model approach is visible in the open: the insurtech **Lemonade** publishes *complexity-analyzer* (`github.com/lemonade-hq/complexity-analyzer`), an open-source CLI that uses an LLM to score the intrinsic difficulty of a pull request, so each unit of work carries a difficulty weight before any productivity number is computed. The unit of productivity becomes *complexity points shipped per engineer per week*, not lines of code, and the "vanity metrics" framing for LOC and tokens used elsewhere in this playbook follows directly from treating complexity, not volume, as the denominator.

One study company arrives at the same place from the opposite direction: a fixed taxonomy rather than a learned model:

> **"Every unit of work we've classified into one of seven levels: T0 a basic config-file change, T7 a crazy advanced feature. We go tier class by tier class: how many tickets can we put through at each tier with zero human intervention? We call it the one-shot rate. By the end of this year we'll hit 100% one-shot through tier three, maybe tier four."**
>
> – CTO, late-stage software company

Both approaches converge on the same operational metric: **complexity-weighted throughput per engineer**.

One caveat grows as orgs approach the dark factory: "per engineer" inflates toward meaninglessness when a few people run large agent fleets: headcount shrinks while the token bill balloons, so the denominator stops reflecting real cost. The metric that survives the shift is complexity-weighted throughput **per dollar of total cost, human plus AI**. Per engineer is the practical metric today; per total dollar is the one to graduate to.

Without this denominator, even a well-instrumented organization cannot say what "productivity" actually means. Third-party analytics tools (DX, Hivel, Jellyfish, Worklytics, LinearB and others) are starting to fill this gap on top of Cursor and Copilot telemetry, but the study's strongest practitioners are building the complexity layer themselves, because the weighting reflects *their* domain, not a generic SaaS heuristic.

Benchmarking is peer-to-peer, not platform-to-planet

Benchmark against *peers* (same stage, sector, and architectural class) on *per-developer* metrics, not platform aggregates (planet-wide totals like GitHub's are the sum of millions of repos; no single org runs at that rate). Three numbers actually travel between companies: **AI-attributed share of code shipped** (the study runs <10% to 95%+), **effective FTE per human engineer** (human ICs × concurrent agents, 1× to 5–7×), and **token spend per engineer** (~$1–2K average, near-zero to $12K). Almost no one in the study measures these systematically yet; the benchmarking advantage is simply the discipline to define a peer set, instrument the same numbers, and update quarterly.

Security & Compliance

Agentic AI widens the security surface in four directions at once. There is risk in what the agents **write**: vulnerabilities shipped at a volume traditional review cannot fully catch. There is risk in what the agents **do**: prompt injection, tool misuse, supply-chain compromise, and PII leakage: familiar risks with new agent-specific paths, plus attack modes that emerge when models can act. And there is risk from the **attacker's** agents: AI that scans your code orders of magnitude faster than any human researcher could. The fourth vector is the **model and provider** themselves: bias in the weights, hidden serving routes, data retention, and trust in a company that increasingly competes across the software stack. On top of all four sits a **compliance gate** that maps agentic work to the organization's approval, evidence, and audit controls. Unlike the other dimensions, this one is not a progression you climb; it is a surface you have to cover.

The strategic implication is the one most underestimated in the study: *"we won't use AI in our codebase"* is no longer a defensible posture. The attacker is using AI against you regardless, so opting out doesn't lower your risk, it just means you find your vulnerabilities second, in a CVE filing, rather than first. The job is to cover the surface, not avoid it.

The Security Surface

RISK 1	RISK 2	RISK 3	RISK 4
What your agents write	**What your agents do**	**What other people's agents do**	**The model and provider**
Insecure code shipped faster than review can catch it.	Prompt injection, supply-chain attacks, PII leakage.	AI finds your gaps orders of magnitude faster than any human.	Bias, model-level attacks, hidden routing, retention, and trust.
COMPLIANCE GATE	Maps agentic work to existing approval and audit controls; AIUC-1, ISO 42001, and NIST AI RMF add AI-specific governance.		

Risk Surface 1: What Your Agents Write

The risk. The simplest vector is the easiest to underestimate: agents generate code faster than humans can review it, and not all of it is safe. AI-written code carries the same vulnerability classes as human code: hardcoded secrets, injection-prone queries, insecure defaults, over-broad permissions, but at a volume that overwhelms line-by-line review.

The mitigation. Move security *into the pipeline*: SAST and DAST on every PR, plus secret and dependency scans and an *adversarial-agent* stage: a writer model produces the code and a separate critic model or specialist service tries to break it before merge. Measure findings caught before merge, escaped issues, false positives, and added review time. Because this stage already reads every change, reuse it for complexity scoring and quality-adjusted throughput on the Measurement dashboard.

Risk Surface 2: What Your Agents Do

The risk. Two agent-amplified attack paths dominate. **Prompt injection** is the new SQL injection, and the CVEs have started landing. In *EchoLeak (CVE-2025-32711)*, indirect prompt injection could cause Microsoft 365 Copilot to disclose information over a network. The shape: an agent reads an issue, comment, document, website, tool result, or customer email and treats embedded instructions as authoritative.

Supply-chain attacks are the second. The 2025 npm *"Shai-Hulud"* worm self-replicated through hundreds of packages by stealing publish tokens. In February 2026, the Shai-Hulud-like *SANDWORM_MODE* campaign disclosed by Socket (named for the malware's own Dune-themed switches, not the Russian APT of similar name) added an agentic twist: it planted a rogue **MCP server** into AI coding assistants (Cursor, Claude Desktop, and the then-Windsurf, now Devin Desktop) whose prompt-injected tool descriptions turned the assistant into a *confused deputy*, quietly exfiltrating SSH keys, cloud credentials, and npm tokens. The same surface opens every time an agent installs a typosquatted package or connects to an unvetted MCP server.

The mitigation. The defense is layered, because no single layer holds. The *probabilistic* layer: input sanitization, *spotlighting* (tagging untrusted input so the model can tell data from instructions; in Microsoft's GPT-family experiments it cut injection success from over 50% to under 2%, although results vary by model and attack), and classifier guardrails (Llama Guard, NeMo Guardrails, LLM Guard), productized by the AI-gateway vendors: Noma and Lasso, plus the platforms now inside Check Point (Lakera) and SentinelOne (Prompt Security), with MCP-specific gateways like MintMCP underneath, on a *triple-gate* pattern: client→LLM, LLM→MCP-server, MCP-server→external-API. This layer shrinks the attack surface but never closes it; a clever enough input gets through.

The *architectural* layer is the durable one: the *dual-LLM / quarantine pattern* Simon Willison proposed in 2023 (a privileged model that holds the tools never reads untrusted content; a quarantined model that reads it cannot act), *capability scoping* (each agent runs under its own *non-human identity*, treated as an independent entity for permissions and auditing, sandboxed, with no write access that matters), and a human in the loop for privileged actions. Prompt injection is the new SQL injection with one cruel difference: there is no parameterized query that fully solves it yet, so you defend in depth, or not at all.

Persistent memory and agent-to-agent handoffs are security boundaries too. Validate provenance before anything is written to memory, isolate state by user and session, authenticate peer agents and delegated authority, and bind every action back to the initiating user. At runtime, monitor tool sequences and data egress, with the ability to isolate the agent, revoke access, and roll back poisoned state.

Risk Surface 3: What Other People's Agents Do

The risk. The surface widens even for organizations that don't use AI to build, because the attackers do. AI now scans thousands of code paths in minutes that once took researchers days. CrowdStrike's 2026 Global Threat Report logged an **89% year-over-year increase in operations by AI-enabled adversaries**. Separately, Anthropic described a 2025 cyber-espionage campaign in which AI performed **80–90%** of the tactical work, calling it the first documented large-scale attack executed without substantial human intervention.

The mitigation. *AI red teaming*: continuous adversarial testing, automated or researcher-led, hunting vulnerabilities across code, applications, cloud configuration, agent workflows, memory, and exposed interfaces, from vendors like **FireCompass**, **HackerOne**, **Zscaler**, and **Obsidian Security**. Track what it finds and how quickly those findings are fixed.

Risk Surface 4: The LLM and the Provider

This is the most complex surface because it raises a chain of trust: the model, the company behind it, and the infrastructure that runs it. The strategic question is whether the code, workflows, and feedback you send to an AI provider could help that same company compete with you later. Start with the workload: what data it handles, what actions it can take, and the consequences of failure. Only then choose the model, provider, region, retention mode, and execution boundary.

Frontier Providers: Training, Retention, and ZDR

The minimum control is simple: company data must not be used for model training. Verify that training is disabled for every account and plan, whether it is the business default or a setting you have to turn off. But training opt-out is not **zero data retention (ZDR)**. ZDR means eligible prompts and outputs are not retained in provider logs after processing. It generally requires an approved API or enterprise arrangement, is not available for every endpoint or feature, and does not necessarily eliminate application state. For API calls, set provider-specific persistence controls explicitly: on OpenAI's Responses and Chat Completions APIs, send `store: false` and verify that the SDK, agent framework, or gateway preserves it. This prevents response application-state storage; it does not create ZDR or remove standard abuse-monitoring retention. Without ZDR, API and temporary-session content is commonly retained for up to 30 days even when training is disabled; ordinary chat history may remain until it is deleted. Under the major providers' published terms, use of that retained content is limited to operating the service and providing the requested history, detecting abuse and security incidents, providing support, and meeting legal obligations. Safety-flagged content or data subject to a legal hold can be kept longer.

Frontier Providers: Trust

Those terms define what the provider is allowed to do with your data today. They do not answer how much you trust the company over time. Alex Karp framed the strategic risk as handing model companies your *"weights and alpha."* Providers' published policies restrict how customer data can be used, but his broader question remains: frontier labs are moving from model APIs into coding, agents, search, browsers, and more of the application layer their customers occupy. A toggle and a policy reduce the near-term risk; they do not guarantee that incentives will remain aligned. Decide which data should never cross that boundary.

Neo-Clouds and Model Routers

Identify the serving path before approving the model. A **hoster** runs the weights on infrastructure it controls. A **router** forwards the request to another provider. A **mixed provider** can do both behind the same catalog, account, and API. Mixed providers such as Together AI illustrate the issue: "available on Together" does not by itself tell you where the request is processed.

Together makes the boundary explicit in its organization privacy controls: pass-through models send prompts and responses directly to third-party providers under those providers' data policies, and prompt storage must be on to use them. Set both **prompt storage** and **Allow passthrough models** to **No**, and leave training consent off. Then allow-list and log the approved model, host, deployment, region, and retention mode. Otherwise a model selected through a familiar cloud account can still move company data to another provider or jurisdiction.

The neo-cloud itself is also part of the trust boundary. Even when it hosts the weights, it can retain prompts or outputs through caches, debugging logs, abuse monitoring, or support systems. Its contract (not the model's license) controls retention and staff or subprocessor access. Review its DPA, retention defaults, logging exceptions, subprocessors, and deletion controls with the same rigor as a frontier provider.

Open-Weight Models

Open weights were a powerful distribution wedge for Chinese model labs: Western inference clouds could host DeepSeek, Qwen, Kimi, GLM, and MiniMax models close to their customers, accelerating adoption and the surrounding deployment and evaluation ecosystem. That route remains important, but the market is splitting. Alibaba now pairs open Qwen releases with proprietary Plus and Max tiers; Qwen 3.8 Max Preview is, as of this writing, available only through Alibaba's own channels (its Token Plan and the Qoder and Qwen apps), with open weights promised but not yet released.

Running open weights on infrastructure you control can remove the external data path and reduce dependence on a frontier provider. It does not remove what the model learned. A 2026 EACL study across 36,000 prompts found that model origin and prompt language systematically changed political bias; a separate audit of DeepSeek across 646 sensitive prompts found suppression of references to transparency, government accountability, and civic mobilization. The answer is not to reject Chinese open models; it is to review independent evaluations and test the shortlisted models in the languages and domains where they will actually operate.

Private Hosting

Private hosting is becoming a credible architecture choice for a growing minority, although it remains operationally out of reach for most organizations today. At WWDC26, Apple showed a complete local agentic stack on MLX and a distributed path for models too large for one machine, explicitly citing a 1.6-trillion-parameter DeepSeek model whose weights require more than 800GB, sharded across multiple Macs. It was not one Apple CPU running a "1TB model"; the important point is that local silicon, unified memory, and clustered inference are making private execution more practical. Western tier-one companies are also expanding the open stack: NVIDIA's Nemotron family provides models, weights, datasets, and training recipes that can run on-premises, in a private cloud, or through a controlled enterprise endpoint. The likely end state is a portfolio: frontier cloud models where capability wins, privately run open models where control matters more.

The Compliance Gate

Start with the organization's documented SOC 2 change-management controls. CC8.1 states that the entity "authorizes, designs, develops or acquires, configures, documents, tests, approves, and implements changes to infrastructure, data, software, and procedures to meet its objectives." For agentic changes, document how each control activity is satisfied and confirm the treatment of automated and human approvals with the auditor. The most advanced study companies are working through that question now:

> **"We've been interested in doing this on bug-fix, but the biggest blocker right now is SOC 2 and compliance. Our production database is full of PII, and the agent needs prod-DB access to understand a bug the way a human developer would; that's a whole new level of sensitivity, and we haven't solved it. And somewhere in our SOC 2 there's an audited control about how we do human code review."**
>
> – CPTO, late-stage company

The infrastructure and test coverage are there; the remaining question is how each compliance-sensitive flow satisfies the company's audited control. Where human judgment or signature is required, it becomes a useful prompt-injection circuit-breaker only if the system presents the underlying evidence and enforces the decision outside the model. Alongside this, **a new certification stack is forming** for the agent autonomy SOC 2 was never written for: NIST AI RMF (risk guidance), ISO 42001 (AI governance), and above all *AIUC-1*, the first agent-specific standard, positioned as "SOC 2 for AI agents." AIUC-1 uses independent audits, at least quarterly technical testing, and annual operational-control reviews. It adds AI-specific evidence and controls; it does not itself rewrite the organization's existing SOC 2 control design.

Volume is the operating challenge. When agents write most of the code, a human-review control can become the bottleneck the pipeline backs up against, and much of the review is already theater: low-risk PRs get rubber-stamped in seconds because no one can meaningfully read that much. The workable middle keeps the control real without making it a rate limiter: a model performs the first pass and presents its judgment, underlying evidence, and risk signals rather than an unstructured raw diff; the change is surfaced and acknowledged in the channel where the work happens, which becomes the audited record; and genuine scrutiny is reserved for the tail of changes that actually carry risk, with automation applied where the documented control design and auditor permit it.

The Ten Commandments for Agentic Security

I **Choose your model path**
Match the provider, region, retention, and execution boundary to the data, actions, and failure impact. Move sensitive work toward regional, dedicated, or private execution.

II **Lock down access**
Before the first prompt, disable training, unnecessary storage, pass-through routing, and unapproved fallbacks. Require zero-data retention for sensitive work; otherwise, verify retention and permitted uses.

III **Guard external input**
Map every untrusted feed. Add injection guardrails or keep it outside automated flows; validate memory writes and isolate sessions.

IV **Separate exposure from authority**
Keep untrusted intake apart from high-impact execution. Require a human for sensitive or irreversible steps.

V **Give every agent an identity**
Record its owner, purpose, permissions, originating user, review date, and expiry. Trace actions and spend; revoke access when retired.

VI **Control tools. Shield credentials**
Allow-list tools and destinations; pin versions and restrict egress. Broker short-lived access so agents never see secrets.

VII **Monitor, contain, recover**
Log plans, actions, data access, and outcomes. Enforce sandboxes, budgets, kill switches, isolation, revocation, and rollback.

VIII **AI-review every PR**
Combine standard scans with independent AI review. Measure catches, escapes, false positives, and latency; reuse the pass for complexity scoring.

IX **Attack your systems first**
Continuously red-team code, cloud, agents, memory, and exposed interfaces. Track findings and remediation time.

X **Know where humans must decide**
Define what runs automatically, what needs review, and what requires human sign-off. Preserve evidence and map policy and customer commitments to those tiers.

How the Strongest Teams Build It.

At the leading edge, the agent itself is increasingly something companies build, not buy. Ramp runs an internal agent, Inspect, on the open-source OpenCode and on Modal sandboxes loaded with its full stack; it writes over half of all merged pull requests, and more than 80% of Inspect is now written by Inspect. Stripe's Minions merge over a thousand PRs a week on a fork of Block's open-sourced Goose. Shopify open-sourced Roast; Coinbase runs Forge. The pattern is consistent among teams with deep engineering benches: own the harness, because it only has to work on your code.

What ownership buys is control a buyer never gets: routing each task to the cheapest adequate model, running on your own sandboxes, wiring in your own context, multiplayer sessions the whole team can watch, and no lock-in to a single vendor. For a company whose edge *is* engineering velocity, that control can be worth a standing team.

One AI-native company sharpened the point: which harness you run, whether Claude Code, Codex or OpenCode, is the commodity layer, and increasingly interchangeable. They swap it freely (lately experimenting with Codex inside their own container); the part that compounds is everything wrapped around it: a **headless platform** the agent can drive end to end, the **curated skills** that encode how your best engineers build, the **tools**, and the **loops**. The durable advantage is *how you serve the agent its context, not which agent you serve.*

Two things keep this from being a blanket recommendation. First, the economics bite even at the top. That early playbook (unlimited use and competitive usage leaderboards) does not survive maturity. Uber's reversal from maximal use to per-engineer caps and gated tools shows why routing and budgeting belong inside the harness, not as a finance afterthought.

Building your own harness is not necessarily right for everyone. Credible off-the-shelf options already exist, and they keep improving. Every name here has a reason to lead on tooling and the bench to staff it; for almost everyone else the off-the-shelf agents are the right call, and a half-built internal harness is a tax. **Factory's Droids** show the buy-side alternative: a commercial agent runtime with model routing, integrations, permissions, and observability built in. These systems were also built into a gap that is closing; several of them predate good managed agents entirely. As managed options mature and open harnesses standardize, Block's open-source Goose now sits under the Linux Foundation's Agentic AI Foundation; the honest rule is narrower than "build your own": own the parts specific to you (your context, your sandboxes, your evals), buy the rest, and revisit the line every few months.

That line keeps moving because the ground beneath it does. Anthropic, OpenAI, Cursor, and others run roughly a phase ahead of everyone else: they turn each model on their own engineering before the rest of us feel it, so the way they describe their internal work is a preview of the infrastructure and workflows you'll need next year.

And the collaboration stack is being rebuilt beneath it.

GitHub and Slack were designed around human speed: developers make occasional pushes, teammates review a manageable number of changes, and people follow conversations in shared channels. Parallel agents clone, branch, push, and communicate continuously. The question is not whether to replace the stack, but where agent volume first breaks it.

This is why people are beginning to rethink GitHub. A large agent fleet can create rate limits, latency, and review overload, while Git records the resulting code without necessarily preserving the session and decisions that produced it. Cursor calls Origin a "Git forge for the agentic era," although it remains on a waitlist. Entire mirrors GitHub repositories onto infrastructure for heavy concurrent traffic and attaches agent sessions and decisions to commits. Both can sit alongside GitHub today, but are building agent-native alternatives intended to replace it when human-paced, centralized hosting becomes the bottleneck.

Slack has a related problem. It can host agent bots, but the workspace is still organized for humans reading messages. Block (creator of Goose) released Buzz, an open-source, self-hostable workspace where agents have their own identities and access, and messages, approvals, workflows, and Git events share one signed log. Its **"branch as room"** brings the discussion, patch, CI, review, and merge decision together, although parts of the Git integration are still being built. **Band** takes a similar approach as infrastructure, giving agents across frameworks shared rooms, memory, identity, and an audit trail.

These products are early, and more will emerge. But they show the direction of travel: development infrastructure built around agent-scale execution and coordination, rather than human-scale workflows with agents bolted on.

Build for the model you'll have.

Most planning treats today's model as a fixed constraint. Frontier teams treat its weaknesses as the fastest-expiring part of the stack. Architecture built around those weaknesses can age before the feature ships, leaving behind workarounds for problems the next model has already solved.

One AI-native company deliberately starts features today's model can only *barely* demonstrate, betting that by the time the work reaches users, the next checkpoint will make them reliable. That lets it pursue capabilities competitors avoid because they are evaluating them against the model available today.

This is not permission to ship unfinished software. The release must still pass the current quality, security, and evaluation gates. The bet is on **what you start building**, not on lowering the bar for what reaches production. The discipline is to build for the model you'll have, not the one you have.

SCORECARD

Score yourself, and see where the study sits.

Plot your own scores on the four dimensions, then compare against the study. Where your row sits darker than the study, you are ahead; where it sits paler, that is your next move.

	CODING	AI OPS	METRICS	SECURITY
Your organization	?	?	?	?
AI-native scale-up (advanced)	4	4	4	3
Late-stage converter (aggressive)	4	4	1	2
Mid-stage converter (steady)	3	3	3	2
AI-native startup (mid-stage)	4	3	2	2
AI-native startup (early-stage)	3	2	1	1
Late-stage SaaS (mid converter)	3	3	2	2
Legacy SaaS (slowest mover)	2	2	1	1
Study median	3	3	1	2

1 · starting 2 · org-wide basics 3 · mature 4 · advanced

Seven anonymized study examples plus the study median; Metrics and Security are the two weakest dimensions across the study.

*Score the first three columns from the four-level scales of Coding Infrastructure, AI Ops, and Measurement. Security has no single scale; read it from **1** (ad hoc SAST / DAST, no AI-specific policy) to **4** (compliance-as-code: policy enforced at the agent layer, audit trails generated as a byproduct).*

CLOSING · PART I

Build the Churches is a test of seriousness.

A company that has only bought AI coding licenses is still experimenting; one that has built the full Coding Infrastructure ladder, AI Ops ownership, measurement, and an agent-aware security posture has begun changing its operating system. But do not wait for the foundation to be complete before converting people. Engineers working agentically expose the missing tests, permissions, context, and observability, giving AI Ops its real backlog; every improvement then enables deeper conversion. Infrastructure and conversion advance together. The church is where the new behavior becomes repeatable.

PART II

Convert the People

Part I began building the environment for AI-native work. Part II asks the harder question: have the people actually changed how they work?

Do not wait for Part I to be finished. The engineers who begin working agentically are the ones who discover what infrastructure is still missing and can tell AI Ops what to build next. Conversion advances the infrastructure just as the infrastructure advances conversion.

But the direction must come from the top. Bottom-up adoption produces isolated demos; an AI-pilled leader sets the standard, funds the missing infrastructure, and holds the line. Then map the population, apply the right conversion play, and move engineers toward a new job built around orchestrating and verifying agents.

"Wake up, Neo."

– THE MATRIX (1999)

It Starts at the Top.

One precondition governs all of Part II: conversion only works **top-down**, and a mandate alone isn't enough. The decisive variable in this study was whether the top of the house was itself **AI-pilled**: a CEO, CTO, or VP Eng who had personally run a fleet of agents, not merely sponsored the idea. Such a leader calls the bluffs a sponsor can't: "I'm 90% AI" doesn't survive someone who knows what 90% looks like, and holds the line when the org pushes back. Bottom-up enthusiasm without that reverts within a quarter; every transformation that stuck was led from a keyboard, not a strategy deck.

One trait runs through both ends of it: **high agency**. The leaders who make it happen don't wait for consensus: they set the mandate and move first; and the people who convert fastest are the high-agency ones who treat that mandate as license to experiment, not an instruction to comply. And it is mostly pre-existing: the best-prepared organizations didn't build this culture *for* AI, they already prized speed, ownership, efficiency, and automation, and had invested in systems that could scale, so they met the moment rather than scrambling to manufacture a culture for it. Handing these tools to a team without those bones, as one engineering leader put it, is *"an AK-47 to a monkey"*: the foundation is far easier to amplify than to create.

You cannot hire your way through the conversion.

That shortage determines the talent strategy. AI-native companies grew engineers with genuine product instincts and agentic fluency internally; converters cannot wait for the market to produce them. The practical answer is hybrid: hire from the small available pool, then pair every external hire with an intensive internal program that retrains the strongest existing engineers into the new mode.

The companies that have moved fastest treat **hiring and training as the same workflow**: every new hire's first month is also a learning curriculum for two or three existing engineers paired with them. One profile travels especially well into the new mode: people with a machine-learning or data background, who already think in datasets and evaluation: the same muscles that benchmarks and golden datasets now demand of everyone.

> **"ML engineers make the best AI engineers; they understand the data, and they understand what having a golden dataset means."**
>
> – Head of engineering, fintech

Map: Locate the Population

With that in place, the work begins, and conversion is a **distribution, not a number**. Organizations convert in pockets, not evenly, and there is no "typical" shape. The four-cluster model below is the diagnostic for locating yourself: Believers and Tool Users separate cleanly on metrics, while Guardians and Resistant share the same low signals; what distinguishes them is whether the resistance has ground.

Conversion readiness · high to low

01	02	03	04
Believers	**Tool Users**	**Legacy Guardians**	**Resistant**
Defined by behavior. 50K+ accepted LOC/month, 10×+ velocity, and 3+ parallel agents.	Low tens of thousands of accepted LOC/month, 1–2× velocity, and almost no parallel agents.	Low activity with grounded resistance: real architectural constraints or identity tied to the legacy system.	Low activity with no convincing ground: fear, performance theater, or moral opposition.

Two forces sit beneath the clusters: **People readiness** (willing and able to work AI-natively) and **Environment readiness** (does the system allow it: Part I infrastructure plus code architecture). A converted engineer in an unconverted environment is paralyzed; the reverse just gets ignored. The Guardian sub-types that follow surface this directly. One caution carried over from Part I: the activity signals above help locate people on the distribution; they are not measures of productivity. Judging the work still needs the complexity denominator.

Convert: Move the Population

The map determines the intervention. Protect the Believers, deepen the Tool Users, and give the Resistant a fair path with a firm deadline. The Legacy Guardians require most of the conversion effort.

Believers don't need conversion: they need *visibility* (workflows captured as reusable playbooks), *distribution* (the strongest made champions inside legacy teams), and *pressure* (ambitious targets that pull the rest along).

Tool Users already accept AI; the gap is delegation depth. The plays: mandated agentic-workflow training on the team's own code, a delegation requirement (one ticket-to-PR via AI per sprint), side-by-side bakeoffs, pairing with Believers.

Resistant won't move regardless; the conversation is fairness of process and acceptance of consequence: equal access, a clear bar, a clear timeline (months, not years), performance-review consequences, then role change or exit.

Legacy Guardians: diagnose, then move

Guardians are the largest cluster, the slowest to move, and the most consequential. Their resistance is partly true; the leader's job is to diagnose which part.

DIAGNOSTIC QUESTION	ARCHITECTURE-BOUND	IDENTITY-BOUND
Whose concerns are these about?	The system	The code's character / history
What's missing for AI to work safely?	Tests, sandboxes, observability, decoupled boundaries	Nothing, the system is fine
Willing to use AI for adjacent tasks (analysis, tests, review)?	Yes, caution is targeted	Often no, the resistance generalizes
What softens the resistance?	A real safety net	A demonstration that their judgment is preserved

The distinction matters because the same resistance calls for opposite responses. If the environment cannot verify and contain agent work, fix the environment. If the safeguards already exist and the objection still generalizes, the work is human conversion.

Conversion Plays from the Field.

Decisive moves from the study: the bets that reset what a team could do and surfaced who would convert.

The bake-off. An early-stage SaaS built its entire engineering offsite around a single exercise: hands-on time with the agents, capped by a side-by-side, agent-vs-human bake-off on the same tasks. The wager: that watching the productivity delta happen in the room converts the skeptics faster than any presentation can, and the holdouts move themselves.

Write me a CRM. A decade-old software company (not AI-native by birth) shut down all of engineering for a week in October, built a custom internal course, and set every engineer one task: build a CRM end-to-end with Claude Code by the end of the week. They thought it absurd. They built it. Since then, **98% of shipped code is Claude-written.**

"I saw what one of them came out with and I was like, this is the future, so we shut down all of engineering for a week."

– CTO, late-stage software company

Mandatory by Sunday. A public consumer company announced its switch to an agentic IDE on a Thursday and made it mandatory by Sunday, expecting 100% agentic code within two months. Engineers are now discouraged from reading code at all: if you're reading it, the thinking goes, you don't yet trust the system. The stated culture is unsentimental: *"get with the program or get out."*

Rewrite the whole product in Go. An early-stage SaaS decided its legacy .NET stack was the real blocker: too heavy to run many agents in parallel, and not worth patching. Rather than keep fixing an architecture built for a pre-agent world, a small team rebuilt it from scratch in Go using agentic programming, with the old codebase serving as both the spec and the verification layer: the agents read the existing system to recover intent, and the old app ran alongside the new one, dual-read / dual-write, to a clean quarter-end cutover. A from-scratch rewrite that would once have been unthinkable for a team that size.

The whole-company hackathon. A ~1,800-person fintech ran a company-wide AI build-a-thon, and only about a sixth of the company were engineers. Roughly a thousand people entered **600 projects**, auto-scored by an ensemble of models because no human could watch them all. The standout came from sales: a real-time AI co-pilot that listens on a live call, scores the customer's sentiment, and tells the rep what to say next, *"like a senior SDR whispering in a junior SDR's ear."* Conversion isn't only an engineering problem.

Architecture-Bound: Fix the System First.

Architecture-bound Guardians are often right. Agentic AI does not create a new class of architectural problems; it makes the old ones much more expensive.

Software teams have spent decades wrestling with sparse test coverage, undocumented APIs, service boundaries, dependency sprawl, and code that only a few veterans understand. At human speed, review, coordination, and institutional memory kept many of those weaknesses under control. Connect an agentic engine to the same architecture and you do not remove the debt; you generate changes against it faster. A brittle system becomes more brittle at machine speed.

Dated languages, slow build and test loops, and legacy package dependencies deepen the gap. The organizations that already invested in modular boundaries, documented contracts, observability, modern dependencies, and fast verification arrived better prepared for agents. Others have to catch up before they can capture the same acceleration.

Fortunately, AI can accelerate that preparation too. The first agentic program does not have to be a new feature: point agents at the changes that expand their own safe operating envelope. Let them map, document, test, decouple, and modernize the system first. This is not a detour from adoption. It is the work that makes sustained adoption possible.

Six plays for an agent-ready stack.

The Map	Use agents to locate high-fan-in modules, cycles, shared state, and cascading dependencies; then generate dependency maps, context files, service contracts, architectural decisions, and runbooks. One public infrastructure company had to document a multi-million-line, eight-year codebase before agents could traverse it effectively.
The Verification Layer	Generate characterization tests against existing behavior, contract tests around APIs, and end-to-end traces that connect a code change to its runtime consequence. The goal is a verification function strong enough for agents to check their own work.
The Headless Split	Separate the data and domain layer from the UI behind stable APIs and contract tests. Agents can then rewrite either layer independently, and a legacy front end no longer constrains backend velocity.
The Boundary Redraw	Break the system into units agents can hold in context. Modularization can split a monolith, while consolidation can pull fragmented services back into a clean monorepo. Direction matters less than clear boundaries and predictable agent behavior.
The Strangler	Put a proxy, facade, or abstraction layer at a stable seam, then route one capability at a time to a new implementation while old and new coexist. Where no clean API exists, intercept events or use an anti-corruption layer. Remove the transitional seam after the migration.
The Rewrite	When incremental displacement is slower or riskier than rebuilding, move a bounded component onto modern languages, runtimes, and packages. Treat the existing product as both specification and oracle: run both systems, mirror traffic, compare outputs, and use dual-read or dual-write before a staged cutover.

Identity-Bound: Convert the Person.

The codebase is fine. The resistance is human. What's at stake: *mastery* (years of expertise feel devalued), *status* (visibly slower than AI-native peers), *identity* (writing code → supervising it), *control* (can't audit every line). The phrase to listen for is "I read every line." It sounds like rigor. Often, it is the last defended position.

> **"I don't like tools that show me code; it distracts me. Intent in, validated result out."**
>
> – Public-co leader, on the inversion an Identity-bound Guardian must make

Plays: convert expertise into validation criteria (they define "good," AI does the work) · hero rewrite with old system as oracle · pair with a Believer · name the fear out loud.

The canonical conversion moment in our dataset: a late-stage engineering leader took an "impossible task" himself (rewriting an entire test suite), completed it with agentic AI, then handed the result to a resistant veteran and asked him to code-review the AI's output (even building review tooling to make the inspection easy). *Seeing work he knew firsthand would have taken him weeks shifted the veteran's mindset permanently; he became one of the strongest internal pushers for AI.* Hero rewrites work because they bypass argument: people can debate theory forever, but they cannot easily ignore a working result they have personally reviewed.

The same conversion, a different way in. At one AI-native company, the most senior engineer (a generation older than the rest of the team) pushed back: *"this is nonsense, I'll be faster by hand."* The leader didn't argue; he set a deadline: *"trust me, one week, no manual code."* A few months later that same engineer was running **twelve agent terminals across two screens**, and had the agents build him a dashboard to hand out and coordinate his own tasks: *"I fed my kids while they ran."* He became the team's loudest advocate. A hero rewrite converts through a result the skeptic reviews; a hard one-week trial converts through self-discovery; he out-built his old self before the week was out.

The Engineer's New Job Is Orchestration.

Once engineers own a broader product surface, the job itself changes. It is not the old work sped up. Running a squad of agents is an orchestration role: you hold three to five parallel threads, each an agent mid-task, and the work is the stream of decisions they surface: an architecture call here, a rejected approach there, a "no, do it this way" before a bad path compounds.

The dominant cost is context switching, and it changes how you think. Every thread carries its own mental model, and recovery never fully completes before the next demands attention, so you hold several partly-loaded models and none of them whole, thinner still as the loops lengthen and the threads multiply. It rewards a specific temperament: sustained concentration, fast judgment, and the discipline to run one thread fewer than feels possible. Cognitive bandwidth, unlike the agents, does not parallelize, and the gap shows up as what Addy Osmani calls *comprehension debt*: supervising more than you can deeply understand. Boris Cherny, who built Claude Code, describes where it ends up: *"I don't prompt Claude anymore. I have loops running that prompt Claude and figuring out what to do."*

The closest analogy is a civilization-building strategy game like *Age of Empires*: you can hold your own for a while on frantic micro, clicking every villager by hand, but you win by building the economy and advancing through the ages. Creating loops is that economy, and the gains compound: get them right and the same person goes from three threads to thirty. It is, not by accident, the same high-stimulation loop from the foreword, which is what makes the work both addictive and draining.

The other half of the job is raising that ceiling: making each thread independent enough to need fewer interruptions. The lever for that is **verifiability**. A thread runs unattended only as far as you can confirm its output is right. Past a point the constraint is not how fast agents generate code but whether you can prove it correct, quietly, the single largest bottleneck on autonomy. The orchestrator's real craft is less writing prompts than building the checks (tests, types, evals, shadow runs, the old codebase as a spec) that let a thread prove its own work. Where that proof holds, the loop runs; where it doesn't, you are back to reading every line.

Were you vibing, or were you engineering?

This is the line between *vibe coding* and *agentic engineering*, Andrej Karpathy's terms. Vibe coding is letting the agent run and accepting what comes back; it's fine for a throwaway, but it hits a wall the moment the project scales or reaches production, because you can't operate or maintain what you never reasoned through.

Agentic engineering is the craft of managing agents, leaning on AI just as hard while keeping judgment in the driver's seat. You hold the design and architecture, and you can step in. Control comes not from inspecting code at human speed but from reviewing the plan with the agent, asking leading questions, and confirming the result is what you intended. Teams that master this scale with agents; teams that default to vibe stall the first time something breaks in production.

The Project Has to Learn You.

The beginning of building this way can be deeply discouraging. I have started projects where working through agents initially felt slower than writing the code myself. Every assumption had to be explained. The agent misunderstood the architecture, chose the wrong dependency, broke something, or produced work that looked finished but was not. You spend hours correcting it, tightening the environment, and wondering whether the promised acceleration is real.

But each correction leaves something behind: a test, a rule, a script, a documented decision, a reusable pattern, or a piece of project memory. The agent is not simply completing the current task; you are gradually constructing the environment in which the next task can complete correctly. This resembles training a new team. Doing the work yourself may be faster today, but it teaches nobody and builds no additional capacity.

The difficulty is that the target keeps moving. As the project grows, dependencies accumulate, the architecture hardens, and earlier decisions sometimes need to be rewritten. The agents are improving at the same time the system is becoming harder to change. Progress comes through repeated cycles of building, breaking, correcting, and encoding what was learned.

Then the experience begins to change. Gradually, the agents anticipate how the project should work. They plan within its architecture, reuse the right components, run the right validations, and repair failed loops. Routine features begin arriving complete enough to move directly into staging or a controlled production path. What once required constant supervision starts to flow.

Reaching that point can take time on a serious project. The early friction is not necessarily evidence that the model has failed; often it is the cost of building the agentic team around the codebase. The first stage feels slow because you are doing two jobs at once: building the product and building the system that will eventually build the product.

When you reach escape velocity.

This is the point at which the experience becomes difficult to explain to someone who has not felt it. You see a feature in another product and send the agent a screenshot. A customer asks for something during a call. An idea occurs to you and you describe it in a few sentences. The agents already understand the architecture, product language, design system, tests, and path to production. They can plan the change, build it, validate it, repair what fails, and even deploy it directly to production, often behind a feature flag or to a trial group. In most cases, it just works.

You are not trusting the model blindly. You are trusting everything you built around it: the accumulated context, rules, tests, scripts, architectural boundaries, and hundreds of previous corrections now encoded in the project. The difficult decisions have already been made and preserved. What once required repeated explanation has become part of how the system works.

That is what defines the escape-velocity moment: when you can one-shot a complex feature. The agentic team carries it through architecture, implementation, validation, and deployment with little intervention. The acceleration is no longer limited to writing code faster. The entire loop from intent to production has changed.

The Transition Cost.

Some people will not make the transition. This is real, and the organization should not pretend otherwise. A data scientist of eight years quit to become a Torah scribe, feeling AI had surpassed his capabilities. An engineer left to help his father's hardware store. The performance-theater pattern (claiming "I'm 90% AI" while the instrumentation shows near-zero token spend and no parallel agent sessions) is its own quiet form of refusal.

These are not failures of the conversion program. They are the conversion program working honestly, separating who can adapt from who cannot, with fair support but without indefinite tolerance. The companies that struggled most allowed legitimate caution to become permanent shielding, or allowed Resistant clusters to set the pace under cover of "we're being thoughtful." The companies that moved hardest named the standard, gave fair support, measured the conversion, and held the line. None of which is costless: a mandate lands hardest on people whose careers were built on the skills it discounts, and the leaders who came through it best said so out loud rather than calling it an opportunity for everyone. Training them is both fair and the only route that scales: the talent to replace an engineering org wholesale does exist, but not in the numbers or at the speed a conversion needs. Say early who has no seat in the new shape. Done badly, the cost is the quiet loss of the people you meant to keep.

AI in the Product: Defense and Dividend.

Converting the engineers you already have is half the work; the other half is building the *capability* (and the magnet for talent) that makes the conversion stick. The most powerful single move in the study does both at once: an initiative to put AI *into the product itself*. Almost every company has one, and not as a feature: the alternative is watching their own software become the legacy in real time.

The threat is not that competitors will ship faster. It is more fundamental: **the product itself is becoming the legacy**. A general-purpose agent with a single domain skill can now do what an entire specialized SaaS used to do: the HR analytics product, the BI tool, the niche CRM, the form builder are each one prompt away from being a 50-line skill on someone's agent platform, and customers are beginning to choose the agent. This is the **Innovator's Dilemma in its purest form**: the specialized software you built the company around is exactly the thing that gets disrupted. The study companies that have absorbed this are racing to cannibalize their own products, to ship the AI-native version of what they sell before a competitor (or a horizontal agent platform with a domain skill bolted on) does it for them.

The response across the study is near-universal.

Almost every company in the study had a named, funded initiative to embed AI into the product layer. The shapes vary:

Agent-as-primary-user. One late-stage SaaS company spun up an "agent experience" team whose job is to make the company's platform usable by AI agents: text-based interaction, indexing, search, MCP integration, semantic layers, rate limiting tuned for agent traffic. The implicit acknowledgment: agents are now a real share of the platform's users and need a deliberately designed surface.

AI as the product's analytical brain. One mid-stage marketplace is building a "trusted analyst" experience that gives its customers AI-driven insight on their own data, explicitly framed as a moat against AI-native competitors entering the same market.

Whole product is the agent system. The AI-native companies in the study skip the integration question because their products are agents already: agentic content generation, agentic operations, agentic customer support.

AI plus the security wrinkle. One small AI-native company in a security domain flagged that integrating AI into a security product creates a new attack surface: prompt injection becomes a real risk. The AI-in-product team and the security team converge.

The compounding dividend.

Investing in AI-in-product pays back across multiple dimensions at once, which is the under-appreciated case for treating it as a top-tier initiative rather than a side project.

The team becomes the org's AI center of excellence. The people shipping AI features in production are solving the hardest real-world AI problems: model evaluation, failure modes, eval suites, prompt injection, cost-per-inference, what to expose versus what to abstract. That expertise then leads the rest of the company's adoption: **the team that "sells AI" understands AI better than the team that only "uses AI."**

It is usually greenfield. A new product, a fresh codebase, no legacy to defend. The AI-in-product team can move at a speed the rest of the org can't match, and that speed becomes proof of what's possible, which is itself a conversion tool for the slower-moving parts of the organization.

It is a talent magnet. The most AI-curious engineers and PMs want to work on AI projects. Having an AI-in-product initiative gives the company a recruiting story for exactly the AI-native talent this transformation runs on, and a place to put the people you most want to hire once you have them.

In short, the same investment defends against the Innovator's Dilemma *and* builds the institutional AI literacy that makes converting everyone else far easier. The study companies that moved earliest on AI-in-product are visibly ahead on AI everywhere else too.

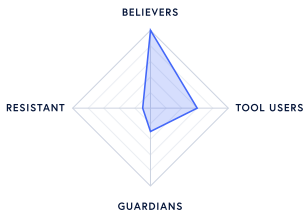
The agility dividend: faster delivery changes what you can sell.

There is one more dividend, and it is strategic rather than internal. The classic SaaS posture (*"we ship one product, you adapt to it,"* and *"that's not on the roadmap"*) was a financial position, not a product opinion: engineering capacity was scarce, every customer-specific feature pulled engineers off the roadmap, and the only profitable shape was one-to-many. Once a single engineer with an agent fleet can ship a customer-specific capability in a day, that cost wall is gone, and *"that's not on the roadmap"* becomes *"that's a two-day pull; we'll have it Friday."*

That agility opens business-model moves that were previously uneconomic, each now being tested across the study: a **more responsive product** (the threshold for accepting customer requests for enhancement drops); a **bespoke custom-agent layer** on top of the one-to-many core, priced on outcome rather than seats; and the deepest, **outcome-as-a-service**: selling the resolved ticket, the completed review, the closed conversation, with the customer never logging into a UI. That direction is already visible in the forward-deployed-engineering ventures both frontier labs stood up in 2026: Anthropic's **Ode**, a joint venture with Blackstone and others, and OpenAI's majority-owned **Deployment Company**: small teams that embed with a customer and build a bespoke agentic workflow against its data. The risk for the incumbent is blunt: if it doesn't make the move, a horizontal AI provider delivers the outcome directly, and owns the relationship.

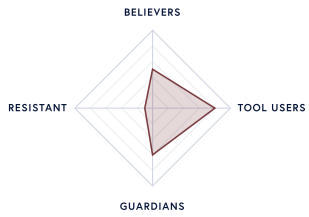
Plot your distribution, and see where the study sits.

With Part II behind you, the deliverable is your own four-axis polygon: the four clusters as axes, percentages summing to 100. The shape that emerges is your conversion picture, and it tells you which playbook to run next.



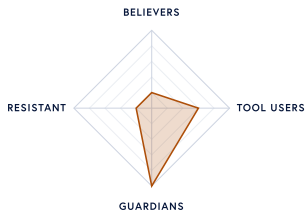
Public consumer software co

Aggressive cutover. Believer-heavy after a mandated weekend switch (100% agentic within two months); the Resistant cluster shrank after performance-review consequences and exits.



Late-stage SaaS

Broad rollout, uneven depth: half the heavy token consumers show no productivity gain (measurable performance theater). Guardians sat on a coupled monolith; modularization became a 2026 OKR.



Public infrastructure co

Guardian-heavy by structure: small new teams at 100% AI-native; the 200+ engineer team on an eight-year, multi-million-LOC codebase at ~20% (documented with context files first). 280 hires frozen, AI-fluent funnel only.

Plot your own shape honestly. Where your weight sits heaviest is the playbook to run first: Believer-heavy → visibility and pressure; Tool-User-heavy → a ticket-to-PR mandate; Guardian-heavy → diagnose architecture vs. identity. Expect the shape to move; re-plot every quarter.

CLOSING · PART II

Once people are converted, the organization itself becomes the bottleneck.

Once engineers can orchestrate agents from intent to implementation in hours, old planning loops, PM handoffs, review processes, and team structures become the constraint. Engineering gets faster, but the company does not, which is the subject of **Part III: Assemble the Community**.

PART III

Assemble the Community

Part I built the infrastructure. Part II converted the people. Yet a recurring pattern ran through the study: organizations that did both (with individuals accelerating 10×+ on their own work) still couldn't push organization-level productivity past roughly 50%.

Without the org changes that come next, it's like upgrading a sedan to a sports car and then stopping at every stoplight: the speed is real, the organization just won't let you use it.

Capturing the rest takes a new shape: requirements become the bottleneck, so the PM-to-engineer ratio breaks to an extreme; product-minded engineers organize into smaller autonomous squads; four experienced chiefs hold company-wide architecture, product, design, and security direction across them; and the org tree gets shorter and wider. Building capacity then spreads beyond engineering into the wider organization. If your structure hasn't changed, you haven't really absorbed the first two parts.

"Adapt or die."

– MONEYBALL (2011)

The 50% Ceiling

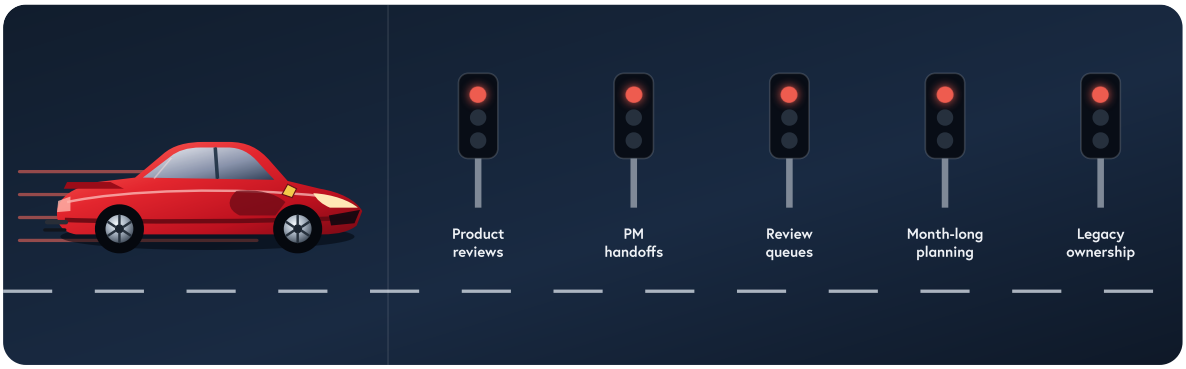
Parts I and II make individual engineers dramatically faster. They do not, by themselves, make the organization move at the same rate. That gap between local velocity and organizational throughput is the central finding of Part III.

The pattern first surfaced at one of the study's most advanced companies: agentic for more than a year, with 85–90% of production code generated by agents and a strong infrastructure layer. If any company should have translated local velocity directly into company output, it was this one. It could not.

> **"We accelerated code generation by close to 10×. But the actual end-to-end productivity gain (from intent to deployed value) never crossed 50%. Most of the time it sat closer to 25–30%."**
>
> – Head of engineering, late-stage company, on the moment the bottleneck shifted

The Ferrari at every stoplight

The analogy lands instantly with every CTO who has hit this wall. Replace an old, slow sedan with a sports car: the acceleration is sharper, the top speed is higher, and every measure of the vehicle improves. Then drive it along the same city route, with a red light every 200 meters. You arrive only slightly sooner. That bottleneck also shows up in outside data. Across more than 400 companies, DX found median pull-request throughput rising 7.76% as AI adoption increased 65%, meaning much of the local acceleration was absorbed before it reached wider organizational output.



The cars got faster. Now we need to redesign the roads.

This is what happens when an organization installs AI-native engineering inside a pre-AI org shape. Some controls still protect quality and safety; others are habits inherited from the old speed. The answer is not to run every red light. It is to redesign the route: remove unnecessary stops, automate controls that can operate at machine speed, and create high-throughput lanes for agentic work. Product reviews, PM handoffs, review queues, month-long planning, human-paced status meetings, and legacy ownership boundaries all have to be reconsidered.

The lights were gone. He still stopped.

I spoke with an engineer who had been given a new project and an explicit mandate to work in AI mode. The organization had cleared away the normal approval gates and told him to move fast. Yet at each meaningful decision (architecture, scope, product tradeoffs), he still went looking for permission. The stoplights had been switched off; he kept stopping at the junctions.

Decades of product and software management taught the same sequence: define, align, review, approve, and only then build. That discipline made sense when code was expensive and rework was slow. It also trained engineers to seek permission before deciding, protect work once written, and treat discarded code as waste.

Agentic work asks for different instincts: make more reversible decisions, test them in software, and throw work away without ceremony when the evidence changes. Removing the gates is only half the transformation; people must learn to move without waiting for them.

When two AI-pilled engineers outpace your fifty.

The danger isn't only your own ceiling. Remember the **46×** concentration from Part I. Picture an R&D org of fifty that came late to AI, up against just two converted engineers at a competitor. On throughput alone, the two can outpace the fifty. But the arithmetic understates their advantage. At two people, there is barely an organization available to slow them down: no management layer, no PM handoff, and no queue of teams waiting on one another. If they also have the rest of the stack: deep domain expertise and the right architectural decisions underneath them, they are operating in full entrepreneur mode, while the fifty keep stopping at every junction. Your existing codebase is still a moat, but a bounded one: it makes an excellent spec, and two engineers like that can rebuild an equivalent baseline from it and then race past it. It's a scary thought, which is exactly why you want those two working for you, not for someone else.

The organizations that broke through 50% rebuilt the workflow around the new speed. Part III is the map of that rebuild.

WHAT PART III REBUILDS

Product–engineering interface. Reset PM ratios, planning, and handoffs.

Teams and direction. Build small autonomous squads, held together by four company-wide chiefs.

Organizational structure. Shorten reporting paths, widen spans, and extend building beyond engineering.

The PM-to-Engineer Ratio Inverts.

Historically, the typical software organization averaged roughly one PM for every six engineers. The ratio held because engineering was the slow side; product could define what to build faster than engineering could deliver it. That balance has now flipped. When a feature ships in a day instead of a sprint, the PM becomes the bottleneck. The question every CTO in the study has had to face is not whether to restructure the ratio, but in which direction.

Two valid directions. Customer proximity decides which.

Direction A: The ratio widens toward 1:10 or beyond. Engineers absorb product work. When engineers can credibly proxy the customer (because they are the customer in devtools or infrastructure, or because they know the domain deeply), the PM function compresses. Product work does not disappear; it moves into engineering. Each engineer becomes a *mini-entrepreneur* for a product surface: close to the customer, choosing what to build, shipping it, and learning from use. A PM, where one remains, covers a broader portfolio instead of feeding requirements to a single squad.

Direction B: Ratio inverts toward 1:1. PMs become hybrid PM/engineers. When engineers cannot proxy the customer (wrong geography, demographic, or domain), the PM cannot be removed; it has to scale up. PMs work in coding tools, ship clickable prototypes, and engineering's job shifts from *"translate the spec"* to *"harden the prototype the PM already built."* One late-stage converter is moving from 1:6 toward 1:2, every PM in Claude Code, on this logic:

> **"I have a build surplus. You define it, I will build it. I want to build it faster than you can define it."**
>
> – CTO, late-stage software company

Pair with the expert. Uber paired AI-proficient engineers one-to-one with domain experts in two-week pods: shadow the work, choose a workflow, build with the people doing it, and ship. Sixteen functions were covered in two months; capital allocation fell from **15 hours to 30 minutes**, and financial pacing reports from two days to ten. The rule: *"build with them, not for them."*

The beta is the spec; the PR is the handoff.

Amazon made APIs the interface between teams. AI is now making working software the interface between functions. Designers can submit functioning interfaces; PMs can submit working features. People who did not know Git a year ago are now opening pull requests.

The handoff remains, but the translation loop disappears. Everyone can react to the beta instead of interpreting a specification. Engineering still owns architecture, quality, security, and production approval, but it begins with a working change.

The same handoff runs in reverse: production evidence, support patterns, and FDE learning can arrive as a tested PR rather than a ticket passed through the organization.

Cat Wu, who heads product for Claude Code, describes the convergence: *"Our roles are blending together: designers ship code, engineers make product decisions, product managers build prototypes and evals."* The study's working hypothesis on hiring tracks this:

> **"It's easier to teach a developer PM skills than to teach a PM to think like an architect."**
>
> – Engineering leader, late-stage SaaS company

Within twelve months, the company that has made one of these moves will not look like the company that made neither.

One mid-stage co-founder named the unsolved half of the work out loud: *"I'm systematically trying to help the engineering team change how they work, but no one's helping the product managers systematically change how they work."* That gap (engineers being enabled, PMs being left to figure it out) is the most common failure in this part of the study.

That logic collapses the planning cadence too.

The ratio is not the only casualty of fast shipping. The four-quarter OKR cycle, the six-month roadmap, the two-week sprint: all are artifacts of an era when a feature took a month. When it takes a day, planning six months out means planning around constraints that won't exist by the time you ship; the loop has to shrink to match, with more experiments run and discarded and fewer long-dated commitments. And the model itself keeps moving underneath the plan: Anthropic's internal benchmark puts it at *"a roughly 41× jump in 16 months"* in the length of task a frontier model can complete unaided, measured from Sonnet 3.5 to Opus 4.6. A planning cycle longer than the gap between capability jumps is planning around a world that no longer exists.

The Squad Is the Unit That Ships.

Product-minded ICs organize into small squads and command their own agents. Companies in the study converged on the unit, but not its size.

The range is narrower than the debate around it suggests: one to five people. **Two to three is where most land.** One AI-native company runs ~5 per vertical (2 backend + 2 AI engineers + an architect-shaped person). Two heads still beat one on most non-trivial problems, even when both are operating agents, and the social dynamics of pair-coding-with-agents stay healthier than full isolation.

The one-person squad is the extreme end of the same line, not a different structure. It remains contested: one AI-native startup in the study dismisses it as *"romance; it's not good enough yet."* But an advanced late-stage company runs five-to-six such squads in production. A single engineer owns products end-to-end, with a team of agents working nearly twenty-four hours a day. Each person operates as the *mini-entrepreneur* described earlier: they understand what the product needs to do, handle their own PM work, and command the agents. That configuration fits a domain where engineers can credibly proxy the customer; this company reports **7×** velocity versus its pre-restructure baseline.

Junior engineers are back, conditionally.

Counter to the "junior is dead" narrative, multiple study companies report the opposite, on two conditions. The hiring filter is now *AI proficiency* and *product sense*, not years of experience; and the architectural spine is the precondition.

> **"A junior with product sense, paired with the architect, is stronger than yesterday's veteran programmer."**
>
> – CEO, AI-native startup

At scale, the picture inverts. One mid-stage SaaS company in the study, instrumenting its own engineering org, found exactly the opposite pattern: its L2s, the most recent grads, are faring *worst* with AI, while staff and principal engineers are the most engaged. The CPTO described it as *"experience and taste: being able to make decisions faster with more context."* In a small AI-native organization, a strong architect nearby can carry more of the judgment load. In a larger company without that proximity, individual judgment matters more, and experience is what produces it. The filter still is not tenure, but at scale the taste that comes with experience compounds.

Four Chiefs at the Helm.

Small, autonomous squads ship with fewer handoffs. That freedom creates the speed. It also creates a new risk: every project can make sensible local decisions and still pull the company in a different direction. That is why the flatter organization needs a small set of company-wide functions to keep the system under control and help teams navigate at speed.

In the old structure, architects, senior managers, product reviews, design critiques, and security gates carried decisions across teams. Once those layers thin out, alignment no longer happens by default. A squad knows its product and an agent knows the repository and task in front of it. Neither naturally sees why another team chose a particular database, how two products should share identity, what customers have already been promised, or which details make the company's brand recognizable. Four human roles hold that company-scale context:

The Four Chiefs · Human Judgment at Company Scale

1	2	3	4
Chief Architect	**Chief Product**	**Chief Designer**	**Chief Security**
Keeps architecture consistent across projects and current with the best technology elsewhere.	Aligns autonomous teams to one strategy and the customer demands that matter most.	Gives the company one visual and product voice, distinct and consistent across every surface.	Maintains one security boundary across models, agents, squads, and Part I's risk surface.

The chiefs should be the company's most experienced people in their respective fields and its strongest AI operators. Squads use agents inside one project; chiefs use them across many projects at once, carrying architecture, product, design, and security judgment from one team to another. Agents do the scanning, retrieval, comparison, and routine communication, so each chief can remain an individual or a very small team rather than growing into a large CTO office. Concentrating the role keeps standards consistent and reserves the company's best human judgment for the decisions that matter most.

When every squad chooses its own route.

Fewer layers and routine gates mean more choices are made locally: databases and data-access layers, authentication, API patterns, UI components, state management, testing, observability, and external libraries. Model diversity widens the range of answers. Advanced teams should use several models: their strengths differ, independent opinions improve important decisions, and no company should depend entirely on one provider.

But models bring defaults of their own. A 2026 Findings of ACL paper, *A Study of LLMs' Preferences for Libraries and Programming Languages* compared eight models and found a consistent popularity bias: they reached for familiar languages and libraries even when they were a poor fit. The operational point is broader: models can favor popular technology over the best fit for the company. Several models are still the right practice, but their differing defaults add variation to the freedom squads already have, while shared training data can pull all of them toward the same solutions. Across independent squads, locally reasonable choices can accumulate into incompatible stacks and duplicated infrastructure.

The cost of a missing company-wide review. "The wrong DB choice or wrong architecture can sink a project," one chairman told us. In one AI-native startup, a team used agents to build a critical internal application without an explicit architecture review. The system failed, costing **roughly $500K in lost billing over two months**. The company responded by placing a System Design Review between requirements and code, with the Chief Architect confirming that the proposal fits the wider company. The problem was not that agents helped design it; a company-wide decision had been treated as a local implementation task.

When all products look the same.

Visual design makes that convergence easiest to see. Models trained on much of the same product corpus reach for the same safe defaults: stock component libraries, rounded cards, predictable layouts, and an increasingly familiar visual grammar. If the company has not set a visual voice of its own, the model fills the gap with the statistical average. The result can be polished and usable while still making one product difficult to distinguish from the next.

That voice must be set before it can be scaled: typography, color, composition, imagery, motion, and the conventions the company chooses to reject. The Chief Designer owns that graphical language and the human point of view behind it. Agents can then apply it consistently across hundreds of screens and generate variations without averaging the brand away. Three study companies independently made the same observation: *"AI design is derivative, trained on Tailwind and Bootstrap, everything looks the same."* Without that human hand, the company can ship a technically distinct product that looks like every other AI-generated one.

One company memory.

A squad's working context is intentionally local: requirements, repository, tests, and the customer problem in front of it. A chief's context spans the shared architecture and stack, product strategy, security policy, design system, and customer commitments. It also records which code, data, prompts, and design assets can be reused, along with the exceptions accumulated over time. Maintaining that context is a real operating job: keep decisions consistent across repositories, preserve the company's IP, and make what every team builds maintainable by the next one.

The chiefs use the strongest available models to search that history and compare outside approaches. On important questions they use more than one, so disagreements expose the defaults and assumptions described above. The models broaden the view; the final call rests on accumulated human experience.

Traffic control without stoplights.

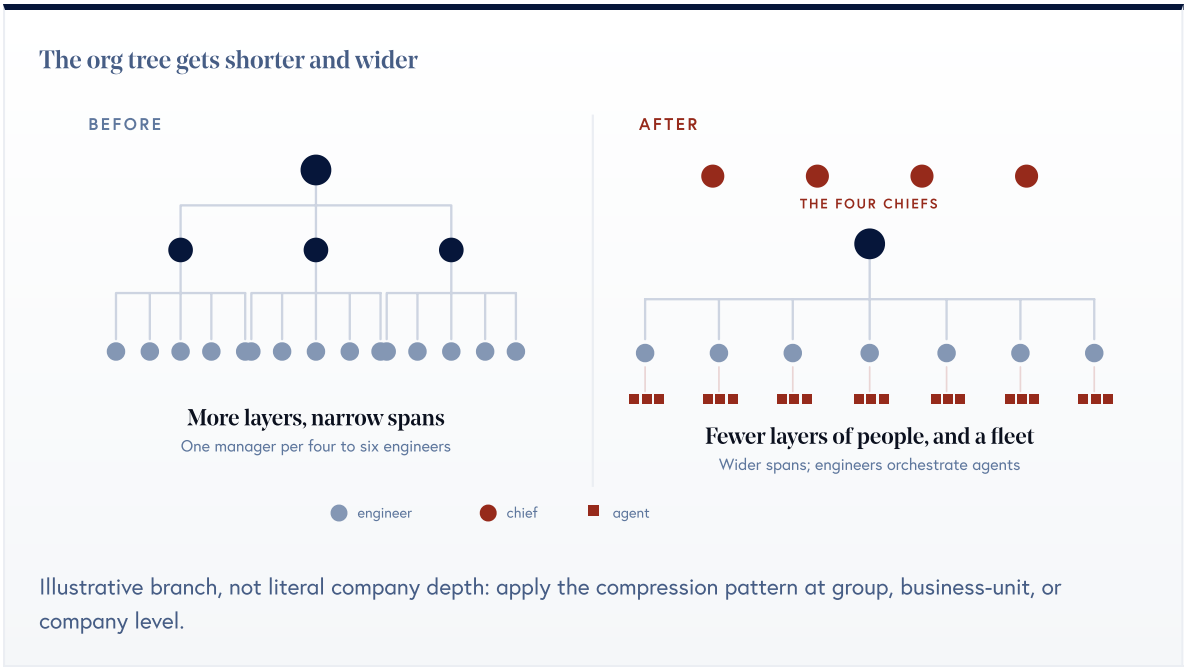
Removing the old traffic lights is what gives autonomous squads their speed. But removing control altogether would produce chaos: standards diverge, local decisions collide, and risks surface only after the fact. Squad leaders are still expected to consult the appropriate chief before major architecture, product, design, and security decisions. But the chiefs do not depend on teams recognizing and escalating every issue. Each chief operates agents that scan plans, code, dependencies, design changes, security exceptions, and customer commitments across all active projects. The agents compare that stream with company standards and earlier decisions, giving the chiefs visibility no human could maintain alone. They can consume it asynchronously and step in when a local choice is becoming a company-wide one, while routine decisions remain with the squads and the work keeps moving.

The chief without the queue.

That context also gives the organization a way to reach the chiefs. In one AI-native company, the CEO built a personal agent connected to Slack, production, meeting transcripts, and monitoring; people speak with it as his representative before coming to him. Another advanced organization gives every product a bot that knows its conversations, code, and customer calls; its Head of Product uses one as the first pass on launch approval. These agents answer routine questions and carry accumulated context back into the squads, while decisions requiring judgment still reach the person.

The Engineering Org Tree Gets Shorter and Wider.

Once product-minded ICs can orchestrate agents, small squads can ship independently, and the four chiefs can hold direction across them, the management structure around the work can compress. As agents absorb status collection, task coordination, and routine handoffs, managers can cover more engineers and squads. Engineering management does not disappear, but its density and purpose change.



Illustrative branch, not literal company depth: apply the compression pattern at group, business-unit, or company level.

In a 25-person startup, that can remove the team-level manager entirely. In a 500-person engineering organization, it more often removes a layer, widens spans, and shifts the remaining managers from coordinating work toward people, portfolio, and cross-team judgment. The unit of change is *management density*: fewer managers per engineer, fewer hops from builder to executive, and more scope per manager.

The traditional engineering-manager role bundled people leadership with flow coordination: status meetings, sprint planning, capacity allocation, blocker removal, and cross-team negotiation. Agents reduce much of that coordination load, but not the need for coaching, performance management, staffing, governance, or decisions that span products and platforms. What compresses first is the layer whose primary job is translating status and coordinating handoffs between other layers.

More capacity can mean fewer people, or more software.

AI expands effective software capacity; it does not determine the size of the organization. Some people make the transition and multiply their scope; others do not. Some companies hold demand constant and use the gain to reduce headcount. Others widen the roadmap, enter new markets, and keep or expand the workforce. Demand for software is close to unlimited. The limiting factor is whether the company can identify valuable work and reorganize around the people who can deliver it.

At one AI-native hypergrowth company in the study, the new capacity is funding expansion rather than a reduction in force. Its roughly 40-person R&D group sits inside a company approaching 600 people, with a substantial FDE organization, and the business is still hiring about **80 people a month** across functions, including engineering and FDE roles. Leadership described an almost unlimited backlog: new features, existing-product work, and a widening scope toward a horizontal platform. The point of the leverage is not fewer people. It is moving faster into a much larger opportunity.

At one late-stage converter in the study, engineering-manager headcount has been cut by roughly **60%**. Some former managers returned to coding and now run squads of agents; the remaining managers cover broader product surfaces. The CTO's framing: *"Cut the middle management. Everyone becomes a squad of agents. More flat. That's the shape."*

At one mid-stage SaaS company, the remaining engineering managers cover **15–25 reports each**, up from four to six. The organization is also experimenting with *effective agentic FTE count* per team to measure the human-equivalent throughput of a small squad plus its parallel agents.

The manager-free structure is the frontier endpoint, not the universal design. One 25-person AI-native startup in the study has no team-level managers or PM layer at all; a flat mesh of senior generalists works beneath four company-wide chiefs. That shape is possible partly because the organization is small, senior, and AI-native from inception.

One late-stage company has made organizational shrinkage, rather than team growth, an explicit management metric. Managers are rewarded for reducing headcount without losing throughput. The deeper signal is not that every manager disappears; it is that *team size no longer proxies for leadership importance.*

Large organizations still need managers to develop people, make staffing and performance decisions, hold regulatory and customer commitments, and coordinate shared platforms across hundreds of engineers. But they need fewer management hops and fewer managers whose scope ends at a single small team. The role becomes broader and more judgment-heavy as its coordination machinery becomes increasingly automated.

Part of what makes the wider span workable is that the residual management job is itself being automated. The parts that scaled poorly with headcount (sitting in every meeting, writing status updates, prepping one-on-ones, chasing blockers, stitching together what happened across a dozen workstreams) are increasingly handled by AI: **call and meeting transcriptions that summarize themselves**, **agents that draft status reports and surface risks** straight from the team's own activity, and assistants that turn a manager's intent into the follow-ups. A manager who spends less time manufacturing visibility can hold far more people in view, which is how some organizations now run 15–25 reports where the old ceiling was six.

The bigger shift is in communication itself. AI is collapsing the cost of the visibility that used to cap how many people one person could track. Ramp runs its agents in shared, multiplayer sessions, so a team watches each other work in the open rather than waiting for a status sync; call and meeting transcripts harden into searchable corporate memory that outlives the conversation; and at one AI-native startup the CEO built a personalized agent of himself (his identity, wired into Slack, production, and monitoring) that people query before they go to him, in effect his standing representative. When visibility is ambient and a leader's context is queryable on demand, the span a single person can hold stretches further still.

The management career path branches.

Engineering managers have built careers on the assumption that larger teams mean greater seniority. That equation is breaking. As spans widen and layers compress, the path separates into three credible directions. All three share a non-negotiable condition: the manager must become AI-pilled. Supervising people who use AI is not enough; leaders need to operate agents themselves, understand the new work firsthand, and recognize strong agentic performance.

(a) Broaden the management span. Manage several squads or a product area rather than one small team. AI supplies visibility and follow-through; the manager remains accountable for people, priorities, performance, and cross-team tradeoffs.

(b) Return to hands-on engineering and run agent squads. The former manager-of-four becomes a senior individual contributor orchestrating several agents, with comparable scope of impact but a different shape of work.

(c) Move into cross-squad leadership. Architecture, platform, product-surface, security, and other company-wide roles become more important as autonomous squads make more local decisions. These roles hold the context and judgment no single project can maintain.

The common thread is leverage, not team size. Large organizations will retain middle management, but fewer roles will exist primarily to relay status between layers. Managers who broaden their scope, deepen their technical contribution, or carry judgment across squads become more valuable; coordination-only roles become less durable.

When Building Leaves Engineering.

As the rest of the company awakens.

Engineering moved first, and its progression offers a template for what comes next. That does not mean turning people in revenue operations, finance, HR, or support into software engineers. It means giving each function agents and tailored internal applications that work across the systems where its work already lives. Instead of a person moving between the CRM, billing system, email, spreadsheets, and support queue, an agent assembles the context, executes the routine steps, and brings back the decisions that still need human judgment. As these systems improve, manual processes and thin SaaS point products begin to give way to agentic tools built around the company's own data and operating model.

The study already shows this happening. One RevOps team is building sales-cycle analysis and deal-scoring software with coding agents and automation tools. Another company has a dedicated five-person GTM AI team building call briefings and other systems for sales and marketing. Elsewhere, companies have built their own candidate-screening tool for HR, an internal pricing application, help-desk and meeting-intelligence systems, and dashboards that combine sales, engineering, finance, and people data. These are not employees moonlighting as product engineers. They are functions rebuilding how their own work gets done.

The IDE-to-agent shift will repeat in business software. In engineering, assistance inside the existing interface was only the first step; the larger change came when an agent could take an outcome and operate the toolchain. A better sidebar inside the CRM or an AI extension inside Excel is still the IDE stage: useful, but waiting for a person to drive the application. The agentic shift begins when an agent reads the customer history, updates the record, triggers the next action, and works across CRM, email, billing, and support without waiting for a person to click through each application.

The chiefs' context-management practice extends to strategy and marketing. The same operating model used to keep architecture, product, design, and security coherent can connect business operations with the market outside. Agents become active sensors across customer conversations, sales activity, support, finance, product usage, competitors, and public sources. They preserve not only what happened but why decisions were made, giving leadership and marketing one living body of context for roadmaps, positioning, and company strategy instead of another stack of fragmented reports.

The org tree gets shorter and wider across functions. The same tools that compress engineering management can strip routine coordination from every management role. Meetings become transcribed and searchable; agents track progress, surface blockers, prepare status, and follow up without another reporting cycle. Removing that work removes many of the human-paced stoplights between teams. Managers can carry wider spans and broader portfolios, reserving their time for coaching, judgment, and the decisions that actually need them.

AI Ops becomes a company-wide function. It no longer serves only engineering. It provides every department with tools, training, reusable patterns, and a common security and data foundation. It also gives the chiefs and functional leaders visibility into what is being built, which models and data it uses, and where local choices are beginning to diverge. That is how the company gains speed without ending up with shadow software, inconsistent controls, and ten different databases solving the same problem.

That moves the build-versus-buy boundary. As custom development gets cheaper, companies can build the agentic layer and the distinctive workflows around their data instead of buying a separate product for every process. The durable systems of record, regulated cores, and products with real networks still get bought; rigid point solutions and thin interfaces are easier to replace. The likely split is to buy the trusted core and build the workflow that makes the company different, which is why APIs, MCP support, and agent access are becoming procurement criteria.

Cat Wu observed the same pattern across Anthropic: *"Our data science, finance, marketing, legal, and design teams picked up these tools on their own. The whole organization moves at the same speed instead of waiting on handoffs."*

Where You Stand, Where We Go from Here.

The thesis is easy to state and hard to live: agentic engineering is an infrastructure problem, then a people problem, then an organizational one, and most companies stall at that last step, the one with the real leverage. So rather than restate it, the more useful question is: **where do you stand?**

Read the clock. Then read the slope. The window has been short. The AI-enabled IDE made assistance ambient in 2023–24, but a human still drove every line; the real inflection came over **Q4 2025** (Claude Code 2.0 in September, Opus 4.5 in November), when agents crossed from demo to dependable. How thin the ground was before that is measurable: METR's randomized trial on early-2025 tools found experienced developers **19% slower** with AI than without, while estimating they had been 20% faster. Re-run on the same developers late in 2025, the direction reversed, though they caution the newer figure is a weak signal, partly because too many of them now refused to work without AI at all. Almost everything in this playbook unfolded in the months around that moment, and where an organization sits today tracks closely to when it started, on two fronts at once: how far its **tooling** has gone, and how far its **people** have.

The AI-pilled organizations PAST THE TURN

The AI-natives that were agentic from day one, plus the established companies that took a sharp turn in Q4 2025, straight into autonomous agentic development, not merely AI-assisted IDEs. They are already past tooling and conversion, into the hard part: restructuring the org and tuning it on real measurement. Most of their staff sits in the Believers camp by now, moved there through training, recruiting, and replacement, and the obstacles their Legacy Guardians stood on have been cleared.

The cautious movers NEARING THE TURN

Where most organizations probably sit. They usually have some agentic projects underway (new products, AI in the product), but most of the org is still IDE-based, working on the infrastructure that would let agentic development scale. Not yet ready for a sharp turn: their teams still carry a large bloc of Legacy Guardians and Resistant who need convincing or replacement, and many feel constrained by their existing architecture and wary of losing control.

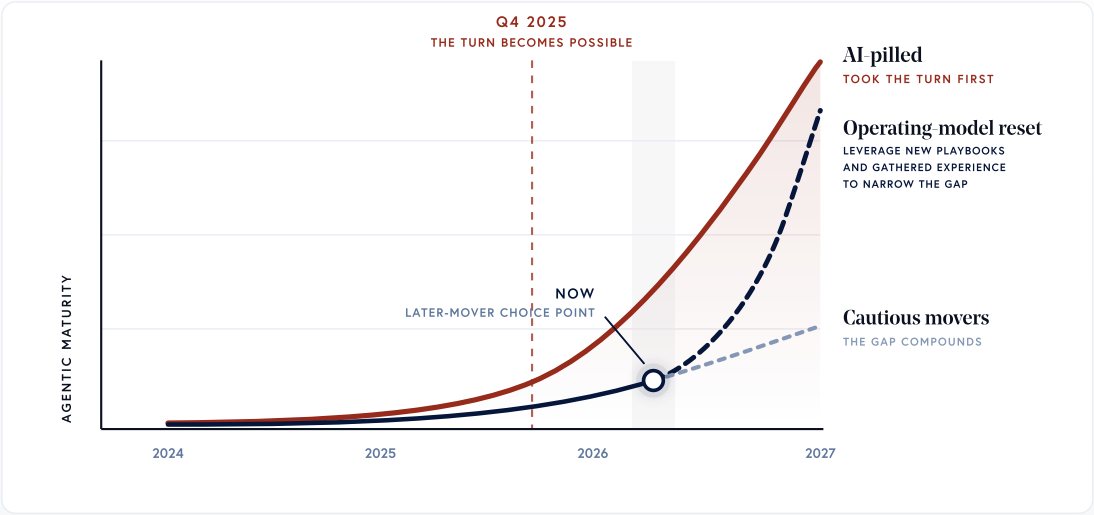
The holdouts YET TO TURN

Some IDE tooling added, but every change still routes through mandated human review, by choice or by compliance. Earliest on both fronts, with the most ground to cover and the least time to cover it.

The turn graph

Timing created the lead. Operating choices set the slope.

The early advantage is real and compounding. The trajectory available to later movers is still a choice.



For later movers, the choice point is now. Better tools and a proven playbook shorten the path, but only an operating-model reset changes its slope.

Timing created the early lead. High-agency companies took the turn first, and the systems and learning they accumulated are still compounding. But timing is not destiny. Later movers now inherit better tools, gathered experience, and a proven playbook, so they can traverse the early curve much faster than the pioneers did.

Operating choices determine the slope from here. Cautious adoption can produce meaningful local gains while leaving the operating model intact; on that path, the gap continues to compound. An operating-model reset (redrawing infrastructure, workflows, roles, and decision rights around agent-enabled work) can put a later mover onto the same accelerating curve and begin to narrow the gap. For later movers, the choice point is now.

The turns keep coming, and you need to be prepared.

The Q4-2025 turn was not the end of the story; it was the first in a series, and the ones after it land faster. Each one resets the structure: an agent that works unattended for an hour and one that works unattended for a day are not the same tool used longer; they call for different teams, different review, a different shape of organization around them. Every breakthrough sends a shockwave through the people and the org chart.

This is what makes the moment genuinely hard. We plan linearly (next quarter looks like this one, plus a little), and the capability does not. A structure built for a 5× world is behind in the 20× world that shows up while you're still staffing for the 5×.

What might a year from now look like? Imagine you no longer open an editor on Monday morning: you brief a fleet of agents instead. Not the five or six that feel aggressive today, but hundreds, spread across features, fixes, tests, migrations, research, and review, most of them working through the night and the weekend without a prompt from anyone. Your job is to decide what gets built and to confirm that what comes back is right. Meanwhile, the codebase you counted on as a moat protects you a little less each quarter, now that an equivalent can be rebuilt from scratch in a fraction of the time. None of this needs a breakthrough you can't already see coming; it's just today's curve, drawn out twelve months.

It won't only be more agents; it will be faster ones. Today's agentic loop will feel to you like dial-up speed: you hand off a task and wait minutes, sometimes longer, for it to come back, the modern equivalent of a 300-baud modem, watching a page of text crawl onto the screen a line at a time. That latency is collapsing the way bandwidth did.

The cycles that cost you minutes today (a build, a test run, an agent reworking a change) compress toward a snap and return not just quicker but more accurately, with fewer wrong turns to unwind. When the short loops that need your input close in seconds rather than minutes, interactive work stops being a chain of hand-offs and starts to feel like a live conversation. Longer loops move in the other direction: agents remain on a problem for hours or days without pulling you back in. The one thing that doesn't speed up is your judgment about what to build and whether the answer is right, which is precisely why that becomes the whole job.

And it spreads past engineering. Revenue operations, finance, HR, support, and legal are already building their own agents and internal workflows. R&D moved first; it had the longest history with the tools and the strongest incentive to automate itself, but the rest of the company does not have to discover the path from scratch. The playbook for tooling, conversion, cross-functional context, and organizational change can now be carried over.

The only real certainty is that it doesn't stop. A finished reorganization will not be enough; the discipline is to stay close to the edge and see each shift early. **The agentic awakening is already moving through the industry. Those who see it early will define the next era of software. Everyone else will wake up inside it.**

BACK MATTER

Sources.

Public claims and where they come from. Figures from the study itself (the vignettes, the distributions, the token measurements) come from the interviews described in the foreword. The model, pricing and tooling layer moves monthly; treat Measurement and What Comes Next as a snapshot of July 2026.

Part I · Coding Infrastructure

GitHub Copilot (2022), Cursor (2023). Copilot entered technical preview in June 2021 and reached general availability in June 2022. github.blog/news-insights/product-news/github-copilot-is-generally-available-to-all-developers

Cursor's agent-first workspace, routing, and Composer models. Cursor 3 introduced a distinct agent-first interface alongside the editor. Cursor's Router selects a model per request based on task type and complexity; Composer is Cursor's model family for coding agents. cursor.com/blog/cursor-3 · cursor.com/changelog/router · cursor.com/composer

Devin's early-access period. Cognition introduced Devin publicly in March 2024 and initially made it available through early access. The company-specific adoption claim in the text comes from the study interviews. cognition.com/blog/introducing-devin

AI review tooling. Cursor Bugbot (agentic review, Autofix added February 2026); Anthropic's Claude Code review and */security-review*; CodeRabbit, Qodo, and Graphite Agent (formerly Diamond, renamed for renewals after January 2026). github.com/anthropics/claude-code-security-review · graphite.com/blog/introducing-graphite-agent-and-pricing

Dark factories in manufacturing. FANUC's lights-out robot plants; Xiaomi's Changping facility, opened July 2024. imeche.org/news/news-article/inside-the-rise-of-unmanned-dark-factories

Part I · AI Ops

Practitioners named. Boris Cherny leads Claude Code at Anthropic (he left for Anysphere in July 2025 and returned within weeks). Peter Steinberger, creator of OpenClaw, joined OpenAI in February 2026 to work across its agent products including Codex; OpenClaw itself remains independent open source. fortune.com/2026/06/08 · techcrunch.com/2026/02/15/openclaw-creator-peter-steinberger-joins-openai

Part I · Measurement

Uber: rollout, adoption, and budget. Claude Code deployed to ~5,000 engineers in December 2025; 84% classified as agentic users by March 2026; the 2026 AI budget exhausted in four months; spend later capped at $1,500 per engineer per month per tool, with some agentic tools gated behind VP sponsorship. CTO Praveen Neppalli Naga quoted via The Information, April 2026. theinformation.com/newsletters/applied-ai · bloomberg.com/news/articles/2026-06-02 · techcrunch.com/2026/06/02

Salesforce: token spend and hiring. Marc Benioff, All-In podcast, May 2026: Salesforce expects to spend close to $300M on Anthropic tokens in 2026 across ~15,000 engineers, alongside a continued engineering hiring freeze and a claimed 30%+ productivity gain. thenextweb.com/news/salesforce-benioff-300-million-anthropic-tokens-slack-coding · fortune.com/2026/05/28

List pricing and the cost arithmetic. Anthropic published rates: Opus $5 / $25 per MTok, cache write $6.25 (5-minute) or $10 (1-hour), cache read $0.50; Fable $10 / $50, cache read $1.00. The $1K, $10–12K and $20–23K figures in the text are cache-adjusted, applying the July 2026 session mix quoted there: roughly 94% cache reads; naive tokens-times-output-rate math overstates them by more than 20×. platform.claude.com/docs/en/about-claude/pricing

Subscription tiers. Claude Max 20x and ChatGPT Pro 20x at $200/month. OpenAI added a $100/month 5x Pro tier in April 2026. claude.com/pricing · techcrunch.com/2026/04/09/chatgpt-pro-plan-100-month-codex

Lower-cost substitution. GLM-5.2 released June 2026 (open weights, long-horizon agentic coding, low-cost plans). Cursor's Composer 2.5 is built on Moonshot's Kimi K2.5; the ~10–60× cost-per-task advantage is Artificial Analysis's independent measurement. venturebeat.com/technology/z-ais-open-weights-glm-5-2 · artificialanalysis.ai/articles/cursor-composer-2-5-coding-agent-index

Concentration of AI output: 46×, Gini 0.77. Cursor, *2026 Developer Habits Report* (data through May 2026): p99 ships 46× the median's AI-written lines and 15× merged PRs; p90 ships 10× and 4×. cursor.com/insights

Top 5% of teams doubled throughput; median +4%. CircleCI, *2026 State of Software Delivery*, February 2026, from 28M workflows. Throughput measured as daily workflow runs. circleci.com/blog/five-takeaways-2026-software-delivery-report

Complexity scoring in the open. Lemonade's *complexity-analyzer*, an open-source CLI that uses an LLM to score pull-request difficulty. github.com/lemonade-hq/complexity-analyzer

Merge-readiness. Cognition's FrontierCode evaluates whether repository maintainers would merge model-generated pull requests, including correctness, regression safety, tests, scope, style, and repository fit. cognition.com/frontiercode

DX: AI adoption and pull-request throughput. A longitudinal study of more than 400 companies from November 2024 through February 2026 found AI adoption up 65%, median PR throughput up 7.76%, and mean throughput up 13.1%; most companies landed between 5% and 15%. getdx.com/news/new-data-ais-impact-on-engineering-velocity-is-more-modest-than-expected

Engineering-analytics vendors. DX, Jellyfish AI Impact, LinearB, Hivel and Worklytics all now report on AI-assistant telemetry. DX also publishes an AI measurement framework connecting adoption, system performance, developer experience, and business impact. getdx.com/blog/ai-roi-engineering · jellyfish.co/blog/measure-ai-impact-copilot-cursor-gemini-sourcegraph · linearb.io/use-case/measure-ai-impact

Part I · Security & Compliance

EchoLeak (CVE-2025-32711). Indirect prompt injection in Microsoft 365 Copilot allowing information disclosure over a network; Microsoft CVSS 9.3. nvd.nist.gov/vuln/detail/CVE-2025-32711

Shai-Hulud. Self-replicating npm worm, September 2025, spreading via stolen publish tokens; the November 2025 second wave reached ~796 packages. krebsonsecurity.com/2025/09/self-replicating-worm-hits-180-software-packages · securitylabs.datadoghq.com/articles/shai-hulud-2.0-npm-worm

SANDWORM_MODE. Disclosed by Socket's threat research team, February 2026; corroborated by CrowdStrike and Endor Labs. Named for the malware's own SANDWORM_* switches, not the Russian APT. socket.dev/blog/sandworm-mode-npm-worm-ai-toolchain-poisoning

Spotlighting. Hines et al. (Microsoft), "Defending Against Indirect Prompt Injection Attacks With Spotlighting": attack success from over 50% to below 2% across GPT-family models. arxiv.org/abs/2403.14720

The dual-LLM / quarantine pattern. Proposed by Simon Willison, April 2023; credited in OWASP's prompt-injection prevention cheat sheet. simonwillison.net/2025/Jun/13/prompt-injection-design-patterns

AI-gateway and guardrail vendors. Noma and Lasso remain independent; Lakera was acquired by Check Point (November 2025) and Prompt Security by SentinelOne (September 2025). MintMCP provides MCP-specific gateways. checkpoint.com/press-releases/check-point-acquires-lakera · sentinelone.com/blog/sentinelone-acquires-prompt-security

Adversary use of AI: 89% year over year. CrowdStrike, *2026 Global Threat Report*, February 2026. crowdstrike.com/en-us/press-releases/2026-crowdstrike-global-threat-report

AI performing 80–90% of a campaign. Anthropic, "Disrupting the first reported AI-orchestrated cyber espionage campaign," November 2025. anthropic.com/news/disrupting-AI-espionage

AI red teaming. FireCompass (continuous automated red teaming); Zscaler (via its SPLX acquisition, announced November 2025); HackerOne (researcher-led AI red teaming); Obsidian Security (AI security posture management). firecompass.com/continuous-automated-red-teaming · zscaler.com/press

Retention, training, and zero data retention. OpenAI retains API abuse-monitoring logs up to 30 days by default, with ZDR requiring prior approval; the Responses API stores application state by default unless *store: false* is set. Anthropic deletes inputs and outputs within 30 days on the backend, with longer retention for policy enforcement and legal hold. developers.openai.com/api/docs/guides/your-data · privacy.claude.com

"Weights and alpha." Alex Karp, CNBC, July 2026, alongside Palantir's accompanying statement on model dependence. cnbc.com/2026/07/01/palantir-karp-open-ai-anthropic-tokens.html

Pass-through routing. Together AI's privacy documentation: pass-through models forward prompts and responses to the upstream provider under that provider's policy, and prompt storage must be enabled to use them. docs.together.ai/docs/privacy-and-security

Qwen tiering. Alibaba previewed Qwen3.8-Max in July 2026 alongside its personal Token Plan; open weights announced but not released at the time of writing. marktechpost.com/2026/07/19

Political bias by model origin and prompt language. Lim & Röttger, "Bias in the East, Bias in the West," Findings of EACL 2026: 36,000 parallel prompts across 60 political issues. aclanthology.org/2026.findings-eacl.122

Suppression in DeepSeek. Carragher, Williams & Carley (Carnegie Mellon), "Information Suppression in Large Language Models": 646 sensitive prompts, comparing chain-of-thought with final output. arxiv.org/abs/2506.12349

Local and distributed inference. Apple, WWDC26 sessions 232 ("Run local agentic AI on the Mac using MLX") and 233 ("Explore distributed inference and training with MLX"), citing a 1.6-trillion-parameter DeepSeek model requiring more than 800GB sharded across several Macs. NVIDIA's Nemotron family publishes models, weights, datasets, and training recipes for on-premises or private-cloud use. developer.apple.com/videos/play/wwdc2026/232 · developer.nvidia.com/nemotron

The compliance stack. SOC 2 criterion CC8.1 quoted from the AICPA Trust Services Criteria. AIUC-1 is positioned as the first agent-specific standard, using independent audits with at least quarterly technical testing; ISO/IEC 42001 covers AI management systems and the NIST AI Risk Management Framework provides voluntary risk guidance. aiuc-1.com

Part I · How the Strongest Teams Build It

Internal agents. Ramp's *Inspect* runs on open-source OpenCode inside Modal sandboxes, writes over half of merged PRs, and is now >80% self-written. Stripe's *Minions* ship ~1,300 PRs a week on a forked Goose. Shopify open-sourced *Roast*; Coinbase runs *Forge* (formerly Claudebot, then Cloudbot). modal.com/blog/how-ramp-built-a-full-context-background-coding-agent-on-modal · blog.bytebytego.com/p/how-stripes-minions-ship-1300-prs · github.com/shopify/roast

The buy-side alternative. Factory's Droids, a commercial agent runtime with model routing, integrations, permissions, and observability. factory.ai/product/droids

Goose under the Linux Foundation. Block contributed goose to the Agentic AI Foundation, formed December 2025 alongside MCP and AGENTS.md; the move completed in April 2026. linuxfoundation.org/press/linux-foundation-announces-the-formation-of-the-agentic-ai-foundation

Part I · What Comes Next

The collaboration stack. Cursor's *Origin*, announced June 2026 as a "Git forge for the agentic era," waitlist-only at the time of writing. *Entire*, founded by former GitHub CEO Thomas Dohmke, previewed its distributed Git network on July 2026. Block's *Buzz*, an open-source agent workspace built on signed events, launched July 2026. *BAND* provides shared rooms, memory, and identity for agents across frameworks. cursor.com/origin · geekwire.com/2026 · siliconangle.com/2026/07/21 · venturebeat.com/orchestration

Part II · Convert the People

Legacy-modernization plays. The Strangler Fig, Branch by Abstraction, Parallel Change, event interception, and transitional architecture all replace a system through temporary seams while old and new implementations coexist. docs.aws.amazon.com/prescriptive-guidance/latest/cloud-design-patterns/strangler-fig.html · martinfowler.com/bliki/BranchByAbstraction.html · martinfowler.com/bliki/ParallelChange.html · martinfowler.com/articles/patterns-legacy-displacement

AI-assisted modernization. Microsoft's GitHub Copilot App Modernization follows an assess–plan–execute workflow; Meta's large-scale ingestion migration used shadow jobs, automated validation, and staged promotion before cutover. learn.microsoft.com/en-us/dotnet/core/porting/github-copilot-app-modernization/overview · engineering.fb.com/2026/05/12/data-infrastructure/migrating-data-ingestion-systems-at-meta-scale

Forward-deployed engineering ventures. Anthropic's *Ode*, a joint venture with Blackstone, Hellman & Friedman and Goldman Sachs, launched May 2026; OpenAI's majority-owned *Deployment Company*, announced May 2026. techcrunch.com/2026/07/15 · forbes.com/sites/janakirammsv/2026/05/28

Comprehension debt. Term coined by Addy Osmani, late 2025; republished by O'Reilly Radar. addyosmani.com/blog/comprehension-debt

"I don't prompt Claude anymore." Boris Cherny, in conversation, June 2026. lucumr.pocoo.org/2026/6/23/the-coming-loop

Vibe coding and agentic engineering. Both terms are Andrej Karpathy's: "vibe coding" from February 2025 (Collins' word of the year for 2025), and "agentic engineering" proposed in February 2026. x.com/karpathy/status/2019137879310836075

Part III · Assemble the Community

Model bias toward familiar technology. Twist, Harman, Syme, Noppen, Yannakoudakis, Nauck & Zhang, *A Study of LLMs' Preferences for Libraries and Programming Languages*, Findings of ACL 2026. Across eight models the pull toward the popular is universal: the same three Python libraries top every ranking, and Python stays the dominant choice in 58% of tasks where it is the wrong tool. arxiv.org/abs/2503.17181

Roles blending; the whole organisation moving at one speed; the 41× benchmark. All three from Cat Wu, Head of Product for Claude Code, "Product management on the AI exponential," March 2026. The 41× figure is Anthropic's internal task-length benchmark, measured from Claude Sonnet 3.5 to Opus 4.6. claude.com/blog/product-management-on-the-ai-exponential

The 1:6 to 1:10 PM ratio. Long-standing industry rule of thumb, traceable to Marty Cagan and consistent with subsequent practitioner surveys. bringthedonuts.com/newsletter/ideal-ratio-engineers-to-product-managers

APIs as the interface between teams. The 2002 Bezos mandate at Amazon, described publicly in Steve Yegge's 2011 platform post. nordicapis.com/the-bezos-api-mandate-amazons-manifesto-for-externalization

Pairing engineers with domain experts. Uber's Agentic Pods programme, described publicly by CTO Praveen Neppalli Naga, mid-2026: ~30 engineers, two-week pods, sixteen functions in two months. thestateofai.com/news/uber-unveils-agentic-pods-structure

The pre-turn productivity measurement. METR (Model Evaluation & Threat Research), randomized controlled trial, July 2025: 16 experienced open-source developers were 19% slower on their own repositories with early-2025 AI tools, while estimating they had been 20% faster. A late-2025 re-run of the same cohort reversed the direction, with the authors cautioning it is a weaker signal, in part because participants increasingly declined to work without AI. metr.org/blog/2025-07-10-early-2025-ai-experienced-os-dev-study