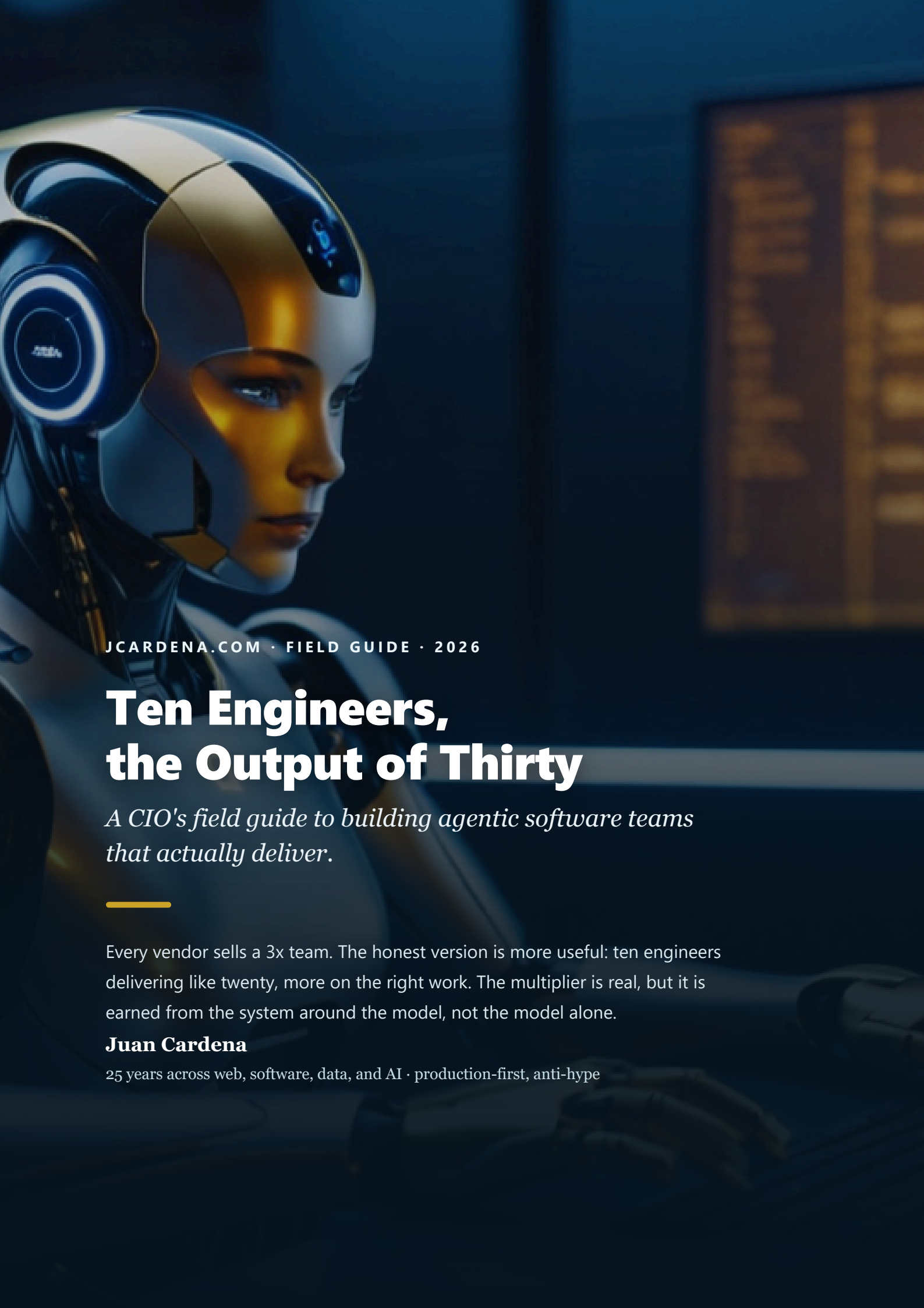


JCARDENA.COM · FIELD GUIDE · 2026

Ten Engineers, the Output of Thirty

A CIO's field guide to building agentic software teams that actually deliver.

Every vendor sells a 3x team. The honest version is more useful: ten engineers delivering like twenty, more on the right work. The multiplier is real, but it is earned from the system around the model, not the model alone.

Juan Cardena

25 years across web, software, data, and AI · production-first, anti-hype

Ten Engineers, the Output of Thirty

A CIO's field guide to building agentic software teams that actually deliver.

Edition 1.0 · June 2026
Written by Juan Cardena · jcardena.com
Published by jcardena.com

The cost models, multipliers, and benchmarks in this guide are illustrative early-2026 ranges drawn from real engagements, not investment, legal, or financial advice. Vendor prices and model scores move every quarter; re-quote your own numbers before you commit a budget.

How to read this guide. If you have ten minutes, read the Executive Summary and the one-page Cheat-Sheet at the back. If you own the budget, read Chapter Five before your next planning meeting. If you own the build, Chapters Two, Three, and Six are the architecture.

A note on the numbers. The multiplier and per-task figures come from real engagements, not a benchmark deck; treat them as a shape to expect, then measure your own. Model quality figures track public repo-level evaluations such as SWE-bench Verified. Pricing reflects published frontier API rates, the GPU rental market, and self-hosted throughput at the stated utilization, all as of early 2026. Every one of these is checkable, and every one moves each quarter, which is the whole reason this guide tells you to build the evaluation harness and re-quote your own numbers rather than trust anyone's slide, including this one.

CONTENTS

Table of Contents

All figures here are early-2026 ranges. The shape of the argument is durable; the exact numbers move every quarter, so re-quote yours before a board meeting.

EXECUTIVE SUMMARY

The argument, in a page

Your developers are about to get a great deal more productive, and your competitors' developers are too. The question is no longer whether to adopt agentic AI for software engineering. It is how to do it so the gains are real, the code stays yours, and the bill does not surprise the board.

The frame that gets a CEO to act is not headcount. It is unit cost. Done right, this work drops the cost of producing a feature on most of your portfolio by 35 to 45 percent, and it builds a routing-and-evaluation platform that becomes a real competitive asset. The headcount multiplier (ten engineers delivering like twenty to thirty on the right work) is a consequence of that, not the goal.

There is no single place to run the model. A serious organization runs four tiers at once and routes each task to the cheapest tier that can do it well: a small model on the developer's own machine for the routine, a self-hosted open model for the bulk of the volume, a frontier model for the genuinely hard work, and managed cloud only where compliance forces it. The layer that decides, which this guide calls **the Agency Router**, sits in front of every request: it assigns each task to the cheapest tier of model autonomy that can complete it reliably, enforces the budget, and keeps sensitive code from leaving the building. That router is the control plane of the whole system. The eval harness, the sandbox, and the retrieval layer sit beside it, but the router is where the economics are won: it spends the attention each task deserves, and no more, so most work never touches the expensive model.

1.7–2.1×
honest blended throughput in 6 months

8–15×
lower cost per token, self-hosted vs frontier, all-in at volume

75–85%
of tokens belong on a cheap or local model

A number your CFO can check.

Take 20 developers. All-frontier, used heavily and undisciplined, runs **$25,000 to $50,000 a month** in tokens and seats, and it still needs people to run it. The tiered mix in this guide (most volume on a self-hosted model, frontier only for the hard 15 to 25 percent) lands the same work near **$6,000 to $12,000 a month** in tokens and GPU, plus **one or two senior platform engineers, roughly $40,000 to $50,000 a month combined** and fully loaded (senior infrastructure people carry a higher rate than line developers) to own the gateway. Count that staff honestly and the tiered platform is about **$50,000 to $60,000 a month all-in**. The 10 extra engineers this output replaces would cost **$150,000 or more a month** fully loaded. That is the decision, fully loaded on both sides: a $50–60k platform against a $150k-plus hiring line, for the delivery of roughly 30 to 40.

How the 20-developer number is built

Assumptions, so you can plug in your own: ~20 active developers; agentic coding at the upper end of daily use; routing split 75 to 85 percent to a self-hosted 32B-class model on one or two high-memory cards at 60 percent-plus utilization, the rest to a frontier API; self-hosted at $0.20 to $0.40 per million tokens all-in, frontier blended at $4 to $8 per million; one to two platform engineers fully loaded. Idle GPUs, a lower routing split, or a thinner pipeline move the number. The point is the method, not the cents.

The one thing most organizations get wrong: *they buy the model and skip the platform. The model is the easy part. The router that decides how much to spend, the evaluations that govern quality, and the guardrails that contain the risk are the actual product, and they are the part nobody wants to fund. Fund them first. (The six questions to test any plan against are in Chapter Five.)*

CHAPTER ONE

The Death of the Copilot

For two years the assistant in the editor suggested the next line and waited for a human to accept it. That era is closing. The tools your developers will use this year do not suggest; they execute. They read an issue, plan a change across a dozen files, write the code, run the tests, read the failures, and fix them, in a loop, until the work is done or they ask for help.

How the work actually happens: an agent reads the task, plans the change, writes across files, runs the tests, reads the failures, and feeds the result back into the loop until the job is done or it asks for help.

The shift from copilot to agent moves the bottleneck. A copilot made a developer slightly faster at typing. An agent takes a unit of work off the developer's plate and hands it back as a reviewed pull request, which moves the constraint from writing code to reviewing logic. That changes the math of how much a team can ship, and it changes what a team has to watch.

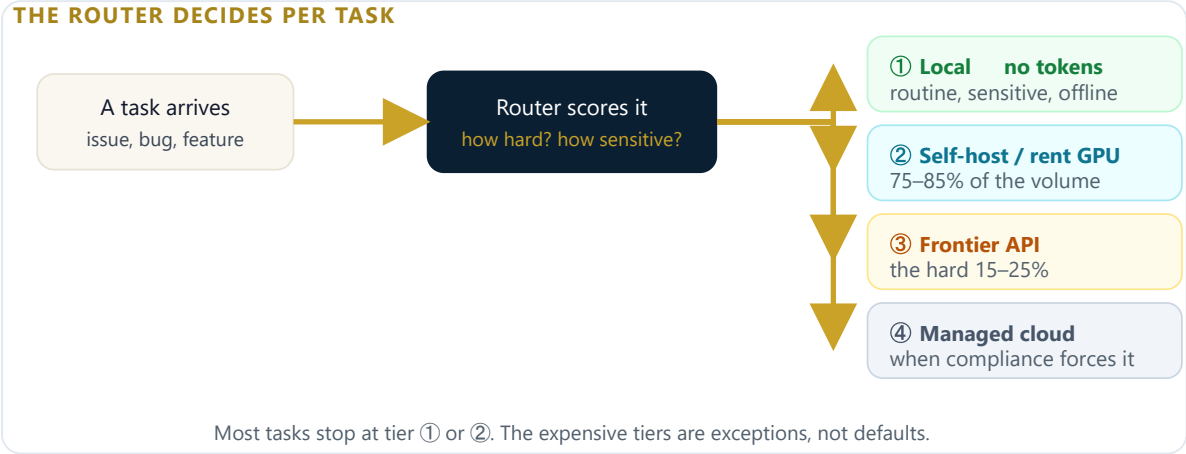
It also changes what can go wrong. A copilot's mistake is a line you do not accept. An agent's mistake is code that compiles, passes a weak test, and quietly ships a defect. The power and the danger arrive together, which is why this guide spends as much time on guardrails as on gains.

The rest of this book is about turning that capability into a system you can trust with your codebase, your secrets, and your budget. We start where every decision starts: where the model actually runs.

CHAPTER TWO

The Four Tiers

There is no single answer to "where do we run the AI." The right answer is "in four places, and a router decides which one per request." Each tier trades cost, privacy, latency, and quality differently, and the art is matching each task to the cheapest tier that handles it well.



The four tiers, and the router that chooses between them on every request.

① Fully local, no tokens

An open model on the developer's own machine. Zero cost per use, total privacy, works on a plane. It handles autocomplete, documentation, small edits, and private-repo questions. A capable desk needs 32 GB of memory as a floor and really wants 64 GB or more of unified memory (or a 24 GB graphics card) to run a 30B-class model well; the largest local models want 96 GB or more. What local cannot do well today is reason across forty files at once or debug ugly legacy in long chains. For that, you climb a tier.

② Self-hosted, on rented or owned GPU

This is the workhorse, and the sweet spot for most organizations. You rent a graphics card by the hour, serve a strong open model behind an internal interface, and your marginal cost per token collapses. A 32B or 72B model such as Qwen fits comfortably on a single high-memory card; a 600B-class model such as DeepSeek needs a multi-GPU node, so size the model to the hardware before you promise a number. On a single well-utilized card, a self-hosted model lands near twenty to forty cents per million tokens against roughly fifteen dollars for a frontier model. On token price alone that is forty to eighty times cheaper; once you count the operations team, power, and the frontier vendors' prompt caching on repeated context, the honest all-in advantage is closer to eight to fifteen times. It is still a large gap, and the code never leaves your network. The catch is utilization: a half-idle graphics card costs more than the API it was meant to beat. Commit to the volume, or do not buy the card.

③ Frontier APIs

For the hard fifteen to twenty-five percent: cross-repository migrations, debugging with tools, the spec that has to become a clean pull request on an important service. The rule of thumb is simple. If forty-five minutes of a frontier model saves more than two hours of a senior engineer, route it. Left undisciplined, twenty developers on frontier APIs can spend tens of thousands of dollars a month, which is precisely why the router exists.

④ Managed cloud

Bedrock, Azure, and Vertex earn their premium only when compliance, residency, and audit matter more than the token price. You pay more and the newest models arrive a little later, in exchange for the controls a regulated industry requires. Use it for the regulated subset, not as the default.

> *The pattern that wins in practice: a self-hosted open model as the default backend, a router that promotes only the genuinely hard tasks to a frontier model, and sensitive repositories pinned to local or self-hosted.* **Most of the quality for a fraction of the cost, with your code under your control.**

CHAPTER THREE

Buying Intelligence

A CIO has four ways to put a model in front of developers, and they are not interchangeable. This is the briefing a CTO gives a CEO: what each costs, who controls the code, and how good the output really is.

Four ways to buy intelligence, stacked from the cheapest, most private tier to the most capable. The skill is routing each task to the lowest tier that handles it well.

HOW YOU BUY IT	REAL COST (EARLY 2026)	CONTROL OF YOUR CODE	BEST FOR
Subscription seats	$20 to $200 per developer per month	Low	the fastest start, the smallest team, the least control
Frontier raw API	~$2–5 per million in, ~$15–30 out	Medium	the hard work, where quality pays for itself
Self-hosted open	$0.20 to $0.40 per million at scale	High	the bulk of the volume; the workhorse
Local only	$0 per use, after a workstation	Total	privacy-first, offline, air-gapped work

The trap hides in plain sight. Subscription seats feel cheapest to start and become the most expensive at scale, with the least control over where your code goes. They are a pilot tool, not an end state. The self-hosted row is the one most finance teams have never seen modeled, and it is what makes the economics work: a single well-utilized card serving a 32B-class open model carries eight to twelve heavy developers and beats per-token frontier pricing several times over. The only way to lose is to rent the card and leave it idle. Utilization is the whole game.

How good is open source, honestly?

On SWE-bench Verified and similar repo-level harnesses, frontier models now resolve real multi-file tasks somewhere in the 70s to low 80s of percent; the best open models trail them by roughly ten points. Both numbers climb every quarter, so check the current leaderboard before you quote them; what matters for budgeting is the gap, not the absolute. That gap is real and it is closing, and it lives almost entirely in novel system design, deeply entangled legacy, and subtle correctness. For the bread and butter of an engineering organization, building features, refactoring, generating tests, writing documentation, open source is already good enough. Buy frontier for the exceptions, not the routine.

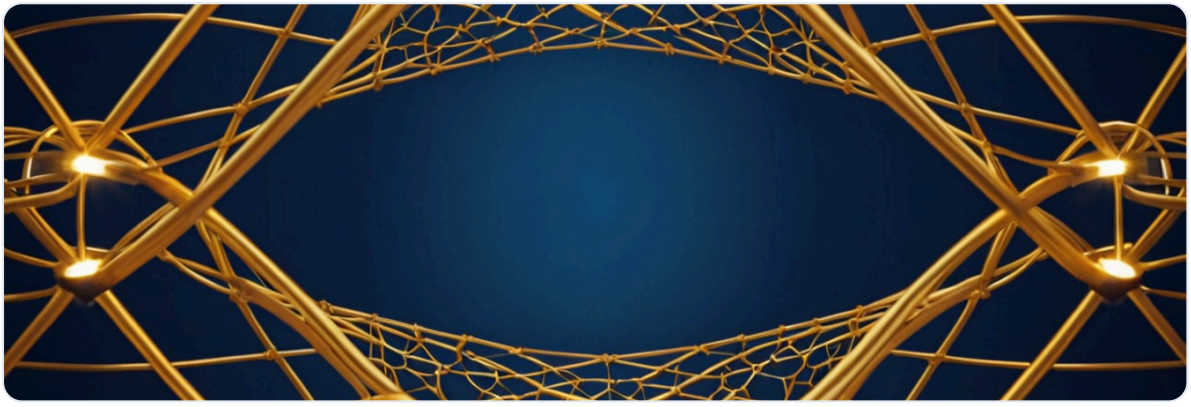
The privacy order, most protective first

Local only · Self-hosted internal · Private cloud endpoint · Standard API with a no-train agreement · and never, ever, proprietary code in a consumer chatbot. That last one is how intellectual property leaks, and it is a policy you enforce in the architecture, not a memo you hope people read.

CHAPTER FOUR

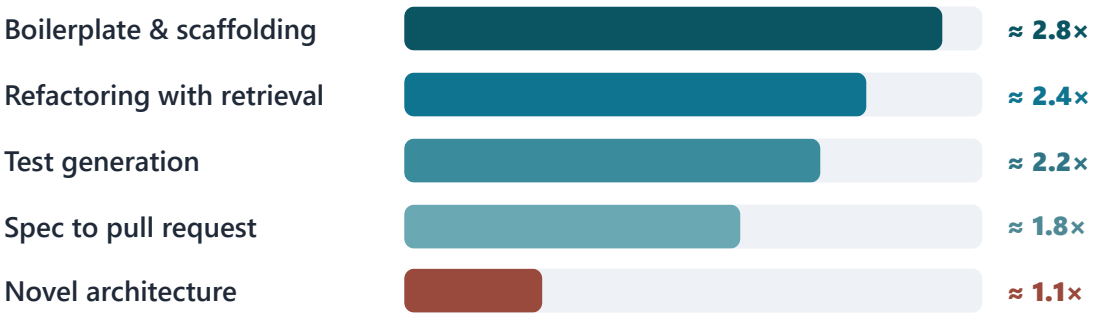
Earn Your Multiplier

The promise that sells the budget is ten developers with the output of thirty. Here is the version that survives contact with a real codebase. A well-run team of ten delivers like fifteen to twenty within six months, and like twenty-five to thirty on narrow classes of work. The honest blended figure is between 1.7 and 2.1 times, and you only reach it by changing how the organization works, not just which model it buys.



The multiplier is real, but it is leverage, not magic. One node of human judgment, multiplied across the mechanical work the agents take on.

The gains are not uniform. Some work compresses dramatically and some barely moves. The slice figures below are what we have measured on real pilots; treat them as a shape to expect, then measure your own. Plan around the slices, not the average.



The deeper the human judgment a task needs, the flatter the curve. So your multiplier depends on your portfolio. A team building greenfield features approaches three times. A team maintaining tangled legacy gets closer to 1.6, and on familiar code worked by senior engineers a careless rollout can land below 1.0, slower than no agent at all, because the review-and-correct loop costs more than the help is worth. None of these numbers is a failure or a guarantee; all are real, and you should budget for yours, not for the brochure's.

The prerequisites nobody puts on the invoice

The multiplier is gated by the codebase and the process, not the model. Agents fail in spaghetti. Before you count the gains, you pay these:

- **Codebase hygiene.** Clear boundaries and naming are what let an agent navigate without getting lost. Spaghetti caps your multiplier hard.
- **Fast, trustworthy tests.** The agent's loop is change, run tests, fix. No fast suite, no loop. Tests are the agent's eyes.
- **Disciplined review.** More code produced is more code to review. Without it, more output is just faster debt.
- **Guardrails.** Budget caps, sensitive-repo pins, and a human gate on production, so the multiplier does not multiply mistakes.
- **The hardware.** A throttled developer is a throttled multiplier. Give them the desk from Chapter Two.

> *The one-line truth for a CEO: this is not a 3x headcount. It* **buys back the hours your good engineers waste on mechanical work** *and redirects them to the judgment only they can provide. Run it with the prerequisites above and ten people deliver like fifteen to twenty; run it without them and you get a bigger bill and a messier codebase.*

CHAPTER FIVE

The CEO's Six Questions

A CEO signs the check; a CIO has to defend it. Every real decision here sits on four axes: cost, value, quality, and security. Optimize one and you pay on another. The job is the right balance for your risk, not the cheapest or the most private corner. Six questions cut to whether the plan in front of you is serious.

1. What is our **blended cost per developer per month** at the target mix, and what is the ceiling if usage triples?
2. Which **repositories can never leave the building**, and is that enforced in the architecture, not just in policy?
3. How do we **measure output quality**, and who sees the number before it reaches a developer?
4. What is the **honest multiplier** on our actual portfolio, and how will we know in ninety days if it is real?
5. If an agent ships a bug to production, **whose name is on the decision**, the routing one, not just the code?
6. Are we building a **platform we own**, or renting seats we will be locked into and cannot audit?

Quality is the question most plans cannot answer, so press on it. "Good output" cannot be a feeling. Concretely, an evaluation set is a golden set of fifty to a hundred real tickets pulled from your own history, each with a known-good resolution, that the agent has to solve correctly before you ship a new model or change the router. You run it on every such change, the same way you run tests on code. It is the cheapest high-leverage thing the platform team will build, and it is the difference between governing on evidence and governing on vibes. Vibes do not survive an incident.

CHAPTER SIX

Six Months to Production

A defensible plan is phased, cheap to start, and proves itself before it scales. Pilot in weeks, not quarters. Reach a production-grade internal platform by month six. Each phase ships value and earns the right to the next.

PHASE	WHAT YOU DO	WHAT YOU PROVE
Weeks 1–4 Pilot	One clean repo, three to five willing developers, subscription seats, no infrastructure yet. Build the evaluation set. Baseline the metrics.	Is the multiplier real here, on our work?
Months 2–4 Platform	Stand up the router with budget, logging, and policy. Self-host an open model on rented GPU. Pin sensitive repos, add redaction. Roll out the agentic editor and code retrieval. Wire in the frontier API for the hard tasks.	We own the stack and the code stays home.
Months 5–6 Production	Agents in the pipeline: test generation, review, and bug-fix pull requests before a human sees them. Expand with the proven guardrails. Evaluations gate every change. Cost and routing dashboards live.	It scales, it is governed, and the numbers hold.

What to invest, in what order

Fund people before silicon. One or two platform engineers to own the gateway come first, because the platform is the product and it needs an owner. Then the evaluation harness, the cheapest and highest-leverage thing you will build. Then the router. Then rented GPU and a self-hosted model. Then frontier contracts and the workstation refresh. Rent before you buy, and let real utilization, not a vendor pitch, decide whether you ever own a graphics card.

The ninety-day honesty check. If cycle time has not moved and every task is still going to the frontier model, you bought tools, not a multiplier. Stop, fix the prerequisites from Chapter Four, and re-baseline before you scale. The metrics that matter are cycle time, senior hours redirected, the share of tokens served cheaply, blended cost per developer, the evaluation score, and the escaped-defect rate. Watch the last one closely: quality must not fall as output rises.

CHAPTER SEVEN

Avoiding Agent Slop

Anti-hype means naming the failure modes before the vendor does. Every one of these has sunk a rollout, and every one is survivable with the discipline already in this guide. The pattern is consistent: the failure is rarely the model. It is the missing guardrail.

WHAT GOES WRONG	THE FIX
IP leak. Code or secrets sent to a consumer tool or a vendor without a no-train agreement.	Pin sensitive repos to local or self-hosted; redact at the gateway; verify no-train terms in procurement; ban personal accounts on company code.
Agent slop. Plausible-but-wrong code at volume. Three agents can agree and be wrong together.	Human review scales with output; the evaluation set catches regressions; tests gate the loop; a critic pass before anything merges.
Cost runaway. A "cheap" pilot becomes a five-figure monthly surprise.	The router enforces budget caps; cheap is the default; the runaway ceiling is alarmed; the routing split is a dashboard.
The false multiplier. Leadership budgeted for a flat 3x; reality on legacy is half that.	Set the honest expectation up front; measure on your portfolio; the ninety-day check; promise the narrow 3x only where it delivers.
Skill atrophy. Juniors who never learn to debug because the agent always did.	Agents draft, humans understand and own; review is a teaching moment; rotate through agent-off work.
Vendor lock-in. Built entirely on one vendor's seats and format.	Own the router and the evaluations; keep a standard interface so models are swappable; keep a self-hosted fallback that proves you can leave.

> *Notice that every failure above is a platform failure, not a model failure. The organizations that struggle hand out seats, expect three times, and never build the router, the evaluations, or the guardrails, and never fix the codebase underneath. So they pay frontier prices for autocomplete, leak code to vendors, cannot measure quality, and end with a disappointing result and a mess to clean up. Buy the model last. Build the platform first.*

APPENDIX A

The Decision-Maker's Cheat-Sheet

One page to take into the room. The whole guide, compressed to what you decide and what you check.

The four tiers (route per task)

- ① **Local**: routine, sensitive, offline. No tokens.
- ② **Self-hosted**: 75–85% of volume. The workhorse.
- ③ **Frontier API**: the hard 15–25% only.
- ④ **Managed cloud**: when compliance forces it.

The numbers that matter

- **1.7–2.1×** honest blended throughput in 6 months.
- **8–15×** lower cost per token, all-in at volume.
- **35–45%** lower cost per feature on most work.
- **~$50–60k/mo** platform for 20 devs vs **$150k+** hiring.

The CEO's six questions

- Blended cost per dev, and the ceiling at 3× usage?
- Which repos never leave the building, enforced in code?
- How is quality measured, and who owns the number?
- Honest multiplier on *our* portfolio, proven in 90 days?
- If an agent ships a bug, whose name is on the routing?
- Are we building a platform we own, or renting lock-in?

Six months to production

- **Wks 1–4**: pilot one repo, build the eval set.
- **Mo 2–4**: router, self-host, pin sensitive repos.
- **Mo 5–6**: agents in the pipeline, evals gate every change.

Avoid the slop (fund first)

- IP leak → pin + redact at the gateway.
- Agent slop → human review + the eval set.
- Cost runaway → router budget caps, cheap by default.
- False 3× → measure on your code, 90-day check.

The one rule

Buy the model last. Build the platform first. The router, the evals, and the guardrails are the product, and they are the part nobody budgets for. Fund them first.

THE AUTHOR

About Juan Cardena

Juan Cardena is an enterprise AI architect specializing in agentic systems for software teams. Across twenty-five years in web, software, data, and AI he has designed and operated production agentic pipelines on a multi-machine fleet, including a self-hosted LLM tier, a model-routing layer, and automated evaluation and guardrail platforms, shipping more than a hundred applications to a Kubernetes cluster. This is not theory: the blog and the guide you are reading were written, illustrated, and published end-to-end by the agentic system this book describes. I write the way I build, production-first and anti-hype, because the multiplier only shows up when the system around the model is real.

Rolling this out, and want it done right?

I help engineering leaders design the router, the evaluation harness, and the guardrails that turn agentic AI from a slide into a shipped, governed platform.

Read more and reach me at **jcardena.com** · the companion film and essay live at **blog.jcardena.com**.

Ten Engineers, the Output of Thirty · Edition 1.0 · jcardena.com · 2026