Vamshi Jandhyala

Vibe Coding and the Boundary of the Firm

✠

May 2026

When building beats buying, and when it becomes a trap.

Vibe coding has moved the line between what a company builds and what it buys, but the more important change is quieter: it has turned building into a cheap option. The easy saving is at the glue layer, the last-mile workflows no vendor will ever prioritise. The harder and more interesting question is what cheap code does to the core, the systems of record the cloud era taught everyone to rent. The answer is not that you should now build them, but that you can finally afford to find out, and that the costs which ought to settle the matter are no longer the ones the make-or-buy story usually tells.

The boundary, and why it just moved

Start with the question Ronald Coase asked in 1937: if markets are so efficient, why do firms exist at all? In a frictionless market a company could buy every task from a specialist and dissolve into a web of contracts. It does not, because using the market is never free. There is a supplier to find, a price to negotiate, an integration to manage, and a contract to police. A firm exists wherever doing the work in-house costs less than buying it, and its boundary falls where those two costs meet. Transaction-cost economics is the right tool for locating that line, though, as we will see, it is the wrong tool for the build decision itself.

For decades the buy side has carried costs the invoice never shows. The average company now runs more than a hundred SaaS applications (BetterCloud, 2025), and each arrives with a procurement cycle, an integration to maintain, a workflow bent to fit the tool, data that cannot quite be exported, and the standing tax of a product that almost does what you need. What vibe coding changes is the other side of the ledger. The term is Andrej Karpathy's, coined in February 2025 for letting a model write the code while you describe the intent, and the practice is already close to universal: 84% of developers use or plan to use AI coding tools (Stack Overflow, 2025). A four-hundred-dollar-a-month subscription starts to look like a weekend project, and the question quietly shifts from "can we build this?" to "should we?"

Moving the boundary does not abolish cost, however. It relocates it, and four costs travel with it.

The four costs of building

Maintenance. A SaaS vendor was never only selling you software; it was selling an insurance policy. Someone else patches the zero-day at three in the morning, absorbs the breaking API change, and tracks the shifting compliance regime, and because that work is spread across thousands of customers, your share comes to pennies. Build the tool yourself and you hold the entire policy, at a bus factor of one, on an asset that today tends to arrive weaker and age faster: AI-generated code carried a security flaw in 45% of Veracode's 2025 tests, and across 211 million lines GitClear found duplicated blocks up roughly eightfold and refactoring down from a quarter of changed lines to under a tenth.

Governance. When any team can build its own tool, data silos form quickly, "customer" acquires six definitions, and the security surface expands where no one is watching. This is shadow IT, already substantial before AI: Gartner found in 2022 that 41% of employees were building or buying technology outside IT. Cheap code removes the last check, and SaaS sprawl, expensive but visible, gives way to a thicket of bespoke tools that talk to nothing and that no one has mapped.

Coordination. Coase's second insight is the one people skip: a firm stops growing at the point where coordinating one more activity in-house costs as much as buying it. The 2025 DORA report, from organisations where AI use is near-universal, found AI adoption now tracks higher throughput yet still lower delivery stability. It behaves as a multiplier of existing conditions, lifting disciplined teams and exposing disorganised ones, because it produces code in ever-larger batches and larger batches carry more integration risk. Conway's law (1968) did not stop being true when the code got cheap.

Talent. When software was expensive to produce, building it well was itself the advantage. When software is cheap, that advantage migrates up the stack, to the quality of the data, the design of the process, and the discipline of governance, the things that turn working code into a working system.

The counterargument: won't AI eat these costs too?

The strongest objection is that the very curve which collapsed the cost of writing code is now coming for these costs as well. On SWE-bench Verified, the best model has gone from resolving about a third of real GitHub issues at the benchmark's August 2024 launch to roughly 88% by 2026, and GitHub's Copilot Autofix already closes security alerts around three times faster than a developer. If agents can fix issues, patch vulnerabilities, and write their own integrations, those costs are the next dominoes to fall.

The honest concession is that some of them are. The security flaws and maintenance drag of the current generation are real, but they are also temporary; they fall as the models improve, and an argument that rests on them has a short shelf life. So the maintenance paradox is the one to stop leaning on. Three others do not depend on how good the model gets.

The saving lands on both sides. Coase's line is set by relative cost, not absolute cost, and AI cuts both. If an agent can maintain your bespoke tool, it can equally maintain the vendor's product, and the vendor still spreads that work across thousands of customers while you bear it alone. Cheaper maintenance therefore strengthens the case for building the glue, where there is nothing to spread the cost against, without

rescuing the case for rebuilding the core.

Accountability is operational, not just contractual. It is tempting to say a SaaS contract gives you someone to sue, but the legal version is weak: most enterprise contracts cap liability at a year's fees and pay out little when an SLA slips. The real difference is operational. The vendor keeps a security team, a roadmap, and a business that depends on fixing the problem, whereas an agent quietly maintaining your internal tool keeps none of these, and the consequence of an autonomous change that corrupts the payroll run at three in the morning lands entirely on you.

The benchmark is not the job. A SWE-bench score measures a model resolving isolated, well-specified issues in clean open-source repositories. Your internal tool lives somewhere far messier, among half-defined requirements, private context no model has seen, and a definition of "working" that extends well beyond passing the test that was set. An MIT study of roughly 300 enterprise efforts found 95% delivered no measurable profit, tracing the failure not to the models but to the "learning gap" of fitting them into real workflows (MIT, 2025); a randomised METR trial found experienced developers 19% slower with AI in their own codebases, even as they were convinced it had sped them up (METR, 2025).

The build is now an option, not a purchase

Here is where the conventional answer, build the glue and buy the core, starts to wobble, because it was the settled wisdom of the cloud era and cheap code is exactly the thing that unsettles it. Transaction-cost economics explains recurring exchange; it was never built for a one-off capital decision taken under uncertainty. A custom tool is not a repeated purchase but an investment whose payoff you cannot know in advance, and the right lens for that is real options. What vibe coding lowered is not only the cost of building but the cost of the *option* to build, and that is the larger shift. You can now prototype a replacement for a core system in a week and abandon it for almost nothing if it disappoints. When the experiment is that cheap, the expected value of trying rises even for systems the make-or-buy rule would leave firmly on SaaS.

Cheap to try, though, is not cheap to own, and three things the ledger usually omits decide ownership, all of them on the build side of the core. The first is data: every system of record you rent leaves your most strategically valuable information in someone else's warehouse, increasingly as training material for models that may be sold back to your competitors, and the case for owning a core system is strongest exactly where its data is most valuable. The second is optionality: a tool you control can be reshaped on your own schedule, while a SaaS roadmap is the vendor's to set. The third is capability, which is not the fixed constraint the question "can we staff a software company?" implies. Teams that only integrate SaaS slowly lose the judgement that building develops, and the muscle you need for the glue is the one you let atrophy by buying everything else.

The decision

It helps to define the two categories, because they are less fixed than the labels suggest. A core system is a shared record of truth, evolving on the outside world's schedule, where failure is expensive and the data is strategic. Glue is the single-company workflow with a small blast radius that no vendor will build. The complication is

that the boundary drifts: most core systems began as someone's glue, Salesforce was a contact list before it was a system of record, and a useful internal tool accretes dependents until it quietly becomes core. Build anything as though it may graduate, and revisit the decision when it does.

Build it yourself when

- it is glue, not infrastructure
- your data or optionality is strategic
- failure cost is low, or the experiment is cheap
- building keeps a capability you need sharp

Keep it on SaaS when

- amortisation across the vendor's base dominates
- operational accountability matters more than control
- the domain evolves faster than you could track
- you would be founding a software company you cannot staff

The certain prize is still the glue tissue: the workflows no vendor will build because they belong to one company alone. The lead-routing rules your sales team actually uses, the report that stitches three systems into the single view finance reviews each month, the reconciliation of vendor invoices against their contracts, the joiner-mover-leaver sequence across a dozen tools. Each used to be a quarter's project; most are now a few days' work. The newer move is to treat the core as something you can cheaply prototype rather than a fortress bought once and never questioned, while remembering that the failures are well documented too: a popular Ask HN thread on build-versus-buy regrets reads, almost without exception, as a list of teams that rebuilt infrastructure they should simply have bought.

The verdict

None of this resolves into a law, only into a sharper way of pricing the decision. Build the glue without hesitation. Treat the core as something you can now afford to prototype, cheaply and disposably, and then decide on the costs that did not fall with the price of code: keep it on SaaS where amortisation and operational accountability dominate, and pull it in-house where your data and your optionality do. "Build the glue, buy the core" was itself the consensus of the cloud era, and the genuinely interesting effect of cheap code is to put even that back in play. What vibe coding lowered is the cost of producing code and, more than that, the cost of trying. What it left untouched is the cost of coordinating the result, owning the data, and standing behind the system when it breaks, and that is now where the decision actually turns.

Sources

- Ronald H. Coase, "The Nature of the Firm", *Economica*, 1937; Nobel Memorial Prize, 1991.
- Andrej Karpathy, origin of "vibe coding", February 2025: `https://x.com/karpathy/status/1886192184808149383`

- Stack Overflow Developer Survey 2025: https://survey.stackoverflow.co/2025/ai/
- BetterCloud, 2025 State of SaaS: https://www.bettercloud.com/resources/state-of-saas/
- Veracode, 2025 GenAI Code Security Report: https://www.veracode.com/resources/analyst-reports/2025-genai-code-security-report/
- GitClear, AI Copilot Code Quality 2025: https://www.gitclear.com/ai_assistant_code_quality_2025_research
- Gartner, "Rise in Business Technologists", 2022: https://www.gartner.com/en/newsroom/press-releases/2022-03-13-gartner-survey-finds-rise-in-business-technologists-is-driving-funding-for-tech-purchases-outside-of-it
- DORA, State of AI-assisted Software Development 2025: https://dora.dev/dora-report-2025/
- SWE-bench Verified leaderboard (~33% at the August 2024 launch to ~88% by 2026): https://www.swebench.com/
- GitHub, "Secure code more than three times faster with Copilot Autofix", 2024: https://github.blog/news-insights/product-news/secure-code-more-than-three-times-faster-with-copilot-autofix/
- MIT Project NANDA, "The GenAI Divide: State of AI in Business", 2025: https://fortune.com/2025/08/18/mit-report-95-percent-generative-ai-pilots-at-companies-failing-cfo/
- METR, "Impact of Early-2025 AI on Experienced Open-Source Developer Productivity", 2025: https://metr.org/blog/2025-07-10-early-2025-ai-experienced-os-dev-study/
- Melvin E. Conway, "How Do Committees Invent?", 1968: https://www.melconway.com/Home/Committees_Paper.html