raia

From Users to Builders to Managers

The Evolution of the Agentic Workforce

A raia white paper

Executive Summary

Who this is for: leaders deciding how their company will build and operate AI agents — and how much of that decision to make irreversible.

Most enterprise AI programs are not failing because the technology is inadequate. They are failing because the technology was deployed without changing anyone's role, without preparing the knowledge agents depend on, and without a structure to govern what results.

This paper argues that an agentic workforce is a human transformation supported by a platform, in that order — and that the choices made in the next twelve months about ownership, independence, and where building happens will determine whether AI compounds for your business or quietly consumes it.

The argument in brief

1. The shift is from users to builders to managers. For fifty years software was built for people to operate. Agents are closer to co-workers than to tools, and your people must move from using software to creating, training, and supervising a workforce of agents. This depends on knowledge transfer, process conversion, and re-underwriting how the company makes decisions — which is why it takes years rather than quarters.

2. We do not claim this is easy. Zero-code does not remove difficulty. It relocates it from technical difficulty, which only engineers can absorb, to domain difficulty, which only the people doing the work can. That relocation is the entire point, because the second kind was never solvable by your IT team.

3. Most AI spend is being wasted. Our estimate is roughly 80%, driven by shadow AI, wrong model selection, over-engineered workflows, and absent governance. These are management problems, not model problems, and they require a platform-level answer.

4. Three hidden costs compound: unmanaged waste, the maintenance burden of infrastructure nobody planned to own, and a shelf life for custom work now measured in months. For many enterprises today, the damage from these exceeds the benefit AI delivers.

5. Three of the four available roads fail. Outsourcing the transformation to consultants, relying on a few internal champions, and waiting for your existing SaaS vendors all break in predictable ways. The fourth is building on a foundation you own but did not have to build.

6. Preparation determines outcomes, not deployment. Agents fail because knowledge was never made agent-ready and because no tooling exists to train, test, and correct them — not because models are inadequate. The industry has optimized for what demos well.

7. Your people are too busy doing the work to build the agents that would free them from it. This paradox stops more programs than any technical obstacle. It has to be funded with protected time, and the first agents should be chosen to buy time back.

8. Centralize the substrate, distribute the creation. Knowledge, security, policy, orchestration, and cost governance belong in one place. Building belongs permanently with the people who understand the work — and should become more distributed as the program matures, not less.

9. The market narrative is backwards. "Every company will become a software company" ignores that code generation compresses one stage of a lifecycle most companies have never run. Software companies will lead this transition rather than be leveled by it — and for everyone else, the opportunity was never building software. It was encoding what only you know.

10. The window for building from scratch closed in 2023. Both software and non-software companies now have to invest in a platform. The only real question is which one, and the criteria are control of the harness, flexibility to keep building on top, and genuine independence from any single model provider.

What we conclude

The organizations that benefit from AI will not be the ones that replaced their teams, and not the ones that bought the most tools. They will be the ones that moved their people from users to builders to managers, on a foundation they control but did not have to build.

Buy the parts that are the same at every company. Own the parts that are only true of yours.

Contents

Where This Comes From	*The basis for these claims*
I — The Credibility Problem	*Everyone says AI; how to tell who means it*
II — The Real Transformation	*Users to builders to managers, and what it depends on*
III — We Do Not Say This Is Easy	*Where the difficulty actually goes*
IV — The Hill	*Licenses are not a strategy, and the paradox that stalls programs*
V — The Unknown Unknowns	*Waste, infrastructure burden, and half-life*
VI — The Four Roads	*Three that fail, and why*
VII — From the Field	*What three years of deployment taught us*
VIII — Three Things We See Everywhere	*The conditions we find on arrival*
IX — What Companies Must Get Right	*Six requirements*
X — From the Edge to the Center	*The migration plan*
XI — What We Are Actually Building	*A category distinction*
XII — Preparation Is the Product	*Where we place the bet, and the build-versus-buy case*
XIII — The Platform	*CX, Command, and Copilot*
XIV — Staying Synchronous	*What to watch, and why independence matters*
The Future Is Human Plus AI	*The conclusion*

Where This Comes From

This is not a thesis written in a vacuum.

It is the accumulation of what we have learned building and deploying thousands of agents alongside customers who are getting real value from them. Every claim here was paid for in deployment — in the problems that surface only at scale, in the failure modes that never appear in a pilot, in the difference between an agent that demos well and an agent a business will put on its front line.

We build in partnership, not in isolation. That partnership taught us where the time goes, what can be compressed, and what must never be rushed.

What follows is what we believe as a result.

PART I

The Credibility Problem

Every company you engage with today has inserted the letters "AI" into its value proposition. Vendors. Employees. Customers. Competitors.

This has made something essential nearly impossible: **telling the difference between who is delivering value and who is trying to stay relevant.**

The distinction isn't rhetorical. On one side are organizations that have changed their core DNA — rebuilt how they work, how they staff, how they measure output. On the other are organizations that added a chatbot and a slide. From the outside, at the moment of the pitch, they sound identical.

The only reliable signal is evidence: what was actually deployed, what it cost, what broke, what it produced. Judge this document against the alternative claims you are hearing, and ask which ones come with scar tissue.

PART II

The Real Transformation

Fifty years of software built for users

Every interface, every workflow, every product decision has assumed a human being sitting in front of a screen, operating a tool. Forms to fill. Fields to complete. Buttons to click. The software held the logic; the person supplied the labor of driving it.

This assumption runs so deep that most organizations have never articulated it. It is simply what work is. Every job description, every training program, every productivity metric, and every org chart in your company quietly encodes it.

From interface to co-worker

AI breaks that assumption completely, and the break is not incremental.

Software has always been something you *operate*. An agent is something that *does work*. It receives an objective rather than a click, it performs a sequence you did not specify step by step, and it produces output you review rather than output you generated. It is far closer to a colleague than to an application — **AI operates very much like a co-worker**, and once that is true, the entire relationship between a person and their tools inverts.

You do not manage a spreadsheet. You do manage a co-worker.

The arc: user to builder to manager

This is the transformation every team member will move through over the coming decade, and each stage is a genuinely different job than the one before it.

Creator. You are no longer operating logic someone else designed. You are deciding what work should be done autonomously, what it means to do that work well, and where the boundaries are. This is a design act, and it requires judgment about the business that no engineer outside the function possesses.

Builder. You gather the information, documents, rules, and sources of truth that give an agent context. You curate them. You document a workflow that may have lived only in your head for a decade. You train an agent on a specific task, and you test whether it actually works.

Manager. Once the agent is live, you supervise it. You review its output, correct it, feed back what it got wrong, and keep it aligned as the business changes. You are managing a worker, and the discipline required is far closer to management than to software operation.

There is an enormous distance between sitting at a computer filling out forms and doing any of this. One is execution. The other is critical thinking, knowledge curation, instruction design, evaluation, and ongoing supervision.

What the transition actually depends on

The reason this cannot happen quickly is that its hardest components are not technical. Three in particular:

Knowledge transfer. What your people know has to become something an agent can use. Most of it has never been written down — process knowledge, exception handling, the reasons behind the rules, the judgment accumulated over years of doing the work. Extracting it is slow, it can only be done by the people who hold it, and it is the single largest determinant of whether an agent performs.

Process conversion. Day-to-day work has to move from 100% human to human-plus-AI. Not automated wholesale, and not left alone — genuinely redesigned around a division of labor that did not exist before. Which steps run autonomously, which require review, where a human enters, what happens at the exceptions. Every meaningful process in the business needs this treatment, one at a time.

Re-underwriting how the company operates. This is the deepest one. Underwriting is the act of examining the assumptions on which you accept risk and setting terms accordingly. A business that introduces an agentic workforce has to re-underwrite itself: how decisions get made and who is accountable for them, what a customer relationship means when the first interaction is an agent, and which systems, controls, and approval structures exist only because humans err in particular ways that agents do not.

Much of your operating model is an artifact of human constraints. Re-underwriting means finding out which parts were load-bearing and which were scaffolding.

Why it will not happen overnight

This is a human transformation before it is a technology transformation.

The technology is available today. Knowledge transfer, process conversion, and re-underwriting are the work of years, because they depend on people changing what they do, what they are accountable for, and how they understand their own value to the organization.

Any leader planning an AI program without understanding that is planning the wrong program — and will conclude the technology failed when what actually happened is that the transformation was never resourced.

What it requires: a plan and a platform

Both, and neither alone.

A **plan**, because this is a decade-long organizational change with sequencing, resourcing, and role implications — not a procurement decision.

A **platform**, because a plan without ground to execute on produces exactly the pattern described later in this paper: good intentions, scattered effort, and ideas that never take root.

PART III

We Do Not Say This Is Easy

Zero-code does not make building AI easy. We will not claim that it does.

What zero-code does is change **who is able to participate.** The difficulty does not disappear — it relocates. It moves from *technical* difficulty, which only engineers can absorb, to *cognitive and domain* difficulty, which only the people who actually do the work can absorb.

That relocation is the whole point, because the second kind of difficulty was never solvable by the engineering team in the first place. An engineer cannot supply the judgment about what "good" looks like in a claims review, a renewal conversation, or an escalation. Only the person who has done it can.

This is also why the alternative path fails so quietly. When a company centralizes AI on its IT team and commissions bespoke applications, it is **repeating the past**: building more software, shipping more interfaces, and leaving the rest of the organization exactly where it was — as users. The tooling is new. The relationship between people and their work is unchanged.

That is the single clearest explanation we have for why most businesses have not unlocked the productivity gains AI promised. They deployed AI without changing anyone's role.

This transformation is not led by technology innovation, and it is not about accelerating how fast code gets written. It is a human-first undertaking. The technology is what makes participation possible; it is not what makes the transformation happen.

PART IV

The Hill

Everything above is philosophy. Here is the practical shape of the climb.

Licenses are not a strategy

Most companies today are at exactly one place: they bought seats to Claude or ChatGPT, distributed them broadly, announced an AI initiative, and washed their hands.

Nothing that follows the purchase order has been done. And what follows the purchase order is the entire job.

Handing every employee a chat interface does not transform a business. It gives capable people a capable tool and leaves them to figure out, individually and without coordination, whether it helps. Some will find real value. Most will use it as a better search box. None of it compounds, none of it is governed, and none of it automates a single workflow. The organization has spent money and changed nothing structural.

The work that actually has to happen

Between "we have licenses" and "AI is changing our business" sits one enormous task: **transforming knowledge into an agentic framework.**

That knowledge lives in two places, and both are hard to extract:

- **In your people.** Process knowledge, exception handling, judgment developed over years, the reasons behind the rules. Most of it has never been written down because it never needed to be.
- **In your past software.** Systems of record, historical data, embedded business logic, and workflows encoded in products built for human users rather than for agents.

Until that knowledge is extracted, structured, and made available to agents, no work gets automated — regardless of how many licenses have been issued or how good the underlying model is.

The point is buying back time

Automation is not primarily a cost story. It is a **capacity** story.

The reason to automate a workflow is to free your workforce to do what only humans can do: apply judgment, think critically, handle the exceptions, design the next set of agents. That freed

capacity is the input to further scale. It is how an AI program compounds instead of plateauing after its first deployment.

The paradox that stops most programs

Here is the trap, and it is the most common reason AI initiatives stall:

Your people are too busy doing the work to build the agents that would free them from the work.

Unless time is freed, they remain fully constrained by day-to-day execution. But freeing time requires building agents, and building agents requires time they don't have. The initiative gets squeezed into the margins of people's calendars, progresses in fits and starts, and quietly dies — not because the technology failed, but because nobody ever funded the transition.

Two consequences follow. First, **the transition must be resourced deliberately** — with protected time, not enthusiasm. Second, **the first agents should be chosen to buy back time**, not to impress anyone. The earliest deployments should target the highest-volume, lowest-judgment work of the people you most need freed up, because those agents fund everything that comes after.

> *(The sequencing point above is our operating recommendation, drawn from deployment experience rather than from any single customer's mandate.)*

The hill itself

What all of this amounts to is a move from a **human-oriented operational model** to a **human-plus-AI operational model.**

That is not a tooling upgrade. It is a change to how work is organized, who does what, what roles mean, and how output is measured. It requires real investment of time, energy, and money.

A leader unwilling to fund that transition has not chosen a smaller version of transformation. They have chosen tools.

Why startups have the advantage — and what incumbents have instead

Startups have a structural edge here for one reason: **they have nothing to transition.** No established operating model to unwind, no legacy processes to reconcile, no workforce whose roles have to change. They can build human-plus-AI from the first day, and every process they create is designed for that model from inception.

Incumbents cannot. They have to run the existing business while changing it — which is the harder problem, and it is why the hill exists at all.

But the advantage is not one-directional. **A startup has no knowledge to encode.** No decades of process expertise, no accumulated exception handling, no proprietary data, no institutional judgment about what good looks like in a specific domain. They have speed and no substance.

An established company has exactly the thing that makes agents valuable — deep, hard-won, domain-specific knowledge — locked in people and systems. The entire strategic question is whether it gets extracted and encoded before someone with less knowledge but fewer constraints builds enough of it to compete.

That is the race. And it is winnable, because what incumbents have takes years to accumulate, while what startups have can be bought.

PART V

The Unknown Unknowns

Every early market has costs that become visible only after commitment. Enterprise AI has three, and they compound.

1. The waste problem

Cost is justifiable when there's a clear line from spend to measurable outcome. Most enterprise AI spend has no such line.

Our estimate: roughly 80% of what enterprises spend on AI today is wasted. Token consumption is the visible symptom. The causes are structural:

- **Shadow AI** — usage outside any budget, policy, or visibility
- **Wrong model selection** — frontier prices for work a smaller model handles
- **Over-engineered workflows** — reasoning-heavy approaches applied to basic tasks
- **Unmanaged adoption** — no view of who uses what, how much, to what end
- **No training** — users burning tokens through inefficient interaction
- **No oversight or governance** — nothing structurally preventing any of the above

None of these are model problems. They are management problems, and they require a platform-level answer.

2. The infrastructure problem

Building an agent is the small part. Maintaining and managing a fleet of them is the real cost, and most organizations skip that step entirely. Two failure paths:

Tool sprawl. Point solutions bought without centralizing the agentic workforce. Each extends one team's capability slightly. None of it compounds — no shared knowledge, no common policy, no orchestration between them.

Build from scratch. Applications constructed directly on top of models, often not genuinely agentic, producing sprawl, slop, and a decentralized headache that worsens over time. Once in production, they never stop consuming resources. Maintenance permanently diverts engineers from high-value work.

Both paths end in unmanageable budget pressure from a cost structure nobody modeled going in.

3. The half-life problem

AI is changing faster than organizations can absorb it. Without a dedicated team, most companies cannot leverage what's being released — and what they do build has a shelf life measured in **days and months, not years**.

Worth naming plainly: **every company building its own AI tooling is quietly becoming a software company.** A fundamental distraction from the core business, undertaken by organizations with no structural advantage in software, against a curve that obsoletes their work before they amortize it.

The compounding conclusion

Waste, resource misallocation, and distraction are not independent risks. They compound.

For many enterprises today, the damage from these three exceeds the benefit AI delivers. That is an uncomfortable claim from a company selling AI infrastructure. It is also the honest read of the market, and it is the situation this platform exists to prevent.

PART VI

The Four Roads

Every leader confronting this arrives at the same fork. Three of the roads are well-travelled. All three are reasonable-sounding, and all three fail in ways that only become obvious after significant spend.

Road One — Outsource the transformation

Hire the consultants. Engage the advisors and thought leaders. Buy a high-dollar solution aimed at one function or one use case. Wait for the roadmap to complete, and hope ROI appears at the end of it.

This is fundamentally the wrong road, for a reason that has nothing to do with the quality of the firm you hire: **you cannot outsource your own transformation.**

The hard work described in Part II — extracting knowledge, encoding judgment, changing what your people do all day — is work only your organization can perform. Paying someone else to do it produces a deliverable, not a transformation. You end up owning another tool. It may even generate real gains in its narrow lane. What it will not do is change how your business operates, because nobody inside your business changed.

Road Two — Let your AI champions figure it out

Identify the few people who are naturally AI-forward, give them room, and let internal leadership emerge.

There is genuinely something right about this. Internal conviction matters enormously, and these people are usually your best asset in the transition. But the road has two failure modes, and the second is fatal.

They become a bottleneck the moment they succeed. AI-forward people become the most valuable commodity in the company almost instantly. Every function pulls them in a different direction. Focus fragments, execution stalls, and the constraint on your entire AI program becomes the calendar of three people.

And if they build from scratch, they stop building. This is the harder lesson. AI presents itself as a technology where you can do anything — and so capable people start constructing everything themselves. But there is a natural ceiling on how much any group of humans can build and operate effectively, and more importantly, **as you deploy more AI, the complexity of supporting, scaling, and managing it compounds.** The acceleration you felt at the start disappears.

Any founder knows this shape already. You build a product, people use it at scale, and your time silently converts from building into supporting, fixing, and maintaining. You are no longer creating. You are running a system.

Your AI champions will hit exactly that wall, and faster than they expect. The way to protect them is not more autonomy — it is a foundation underneath them that reduces how much the organization depends on them personally, so that adoption can accelerate without their calendar being the ceiling.

Road Three — Extend your existing ecosystem

Wait for the vendors you already run. Your cloud provider, your development platform, your CRM, your HR system, your support desk — all of them will ship AI capabilities as add-ons, and adoption is easy because the products are already installed.

For some short-term needs, this is a perfectly good answer. But it carries two risks that compound over time.

You rent your intelligence. Your agentic workforce may become the single greatest asset your business owns — an encoding of your process knowledge, judgment, and domain expertise. Distributed across a dozen vendor ecosystems, it is not an asset you control. It cannot be moved, cannot be unified, cannot be orchestrated as one workforce, and cannot be taken with you when a vendor relationship or pricing model changes.

You lose control of cost. Most vendors have not yet landed on pricing structures that serve customers over the long run. Consumption-based add-ons priced by seat, by action, or by token — set by someone whose incentive is your consumption — mean your costs are determined outside your organization.

Combine those and you arrive at the most dangerous position in this entire paper: **you lose the ability to scale faster than your costs scale.** At that point growth in AI usage stops being leverage and becomes a liability, and the rational response is to slow adoption — which forfeits the entire opportunity.

Road Four — Build on a foundation you own

The fourth road accepts what the first three avoid: the transformation is yours to do, and it has to be done by your people.

What it changes is the amount of undifferentiated work standing between them and that transformation. Your team should be spending their effort on the things only they can do — extracting knowledge, encoding judgment, designing and managing agents. They should not be building harnesses, integrations, governance layers, orchestration, or cost controls, because that

work is identical at every company, compounds into a maintenance burden, and has a shelf life measured in months.

The requirements this road implies are specific:

- The **foundation is bought**, so you inherit acceleration rather than a maintenance obligation.
- The **harness is owned**, so intelligence, control, and portability stay with you.
- The **model is your choice**, per agent, and changeable.
- The **cost structure is yours to govern**, so scale and spend stay decoupled.
- The **building is done by your people**, so the transformation actually occurs.

This is not a compromise between the other three roads. It is a different premise: buy the parts that are the same everywhere, and own the parts that are only true of you.

PART VII

From the Field

Everything to this point is argument. This section is observation — what three years of deploying agents with real customers actually taught us.

Nobody has deep experience yet, including your best people

The science, the processes, and the proof of what works are all too new. **There is no one, at any company, with a decade of agentic AI deployment behind them.** That is simply not a thing that exists.

Which means: whatever industry you are in — including technology — your AI program is an early-stage venture. It has to be run like one. Some things will fail. The objective is not to avoid failure; it is to **fail cheaply and quickly enough to separate real opportunity from false promise** before you have committed serious money to the wrong thing.

This reframes the platform decision entirely. The question is not "which vendor is best." It is **"which choice makes information cheap?"**

Locking into a restrictive third-party solution inverts that. It typically requires full implementation, tens of thousands of dollars, and a long-term contract signed *before you have learned anything about whether the use case produces ROI*. You are asked to make an irreversible commitment at the point of maximum ignorance. Flexibility is not a nice-to-have in an early market — it is the thing that determines what your mistakes cost.

The use cases companies pick are usually the wrong ones

This is the most consistent pattern we see. The team assembled to identify AI use cases selects cases where AI cannot actually work well.

It is not incompetence. There is a genuine gap between what AI appears able to do and what it can do reliably, autonomously, and at scale — and that gap is not visible from the outside. Companies commit to a direction based on where the pain is, rather than where AI is actually effective, and the mismatch only surfaces after the investment.

Almost no one evaluates this before choosing a path.

This is why we built evaluation into the front of the process rather than the end. By curating the data that will ultimately train the agent and mapping the business process first, you can **run evaluations and audits on how effective an agent would be at the proposed work —**

before launching it. You learn whether the use case is viable while the cost of being wrong is still low.

It is also why breadth matters. A platform that supports a wide range of use cases means **a change of direction does not destroy the investment.** You keep the knowledge, the foundation, and the work already done, and you redirect. A functionally-focused, single-purpose AI solution offers no such recovery — if the use case turns out to be the wrong one, the entire investment goes with it. Single-purpose solutions are substantially higher risk than they appear, precisely because nobody can pick correctly on the first try.

Autonomy succeeds where adoption is required to fail

Adoption is the most underestimated variable in the entire category.

The clear pattern: **use cases where the agent does the work and humans supervise outperform use cases that depend on humans changing their behavior.** When success requires employees to adopt a new tool, remember to use it, and change established habits under deadline pressure, the technology can be excellent and the program still fails.

This runs against most people's instinct, which is that assistive AI is the safer starting bet because it keeps humans in control. In practice, assistive deployments carry hidden behavioral risk that autonomous deployments with human oversight do not.

The internal champion is a single point of failure

We covered the structural version of this in Part VI. In the field it looks like this: one person in the organization builds their own AI solution. Whether or not it works, their time is permanently drawn into supporting it. And frequently the solution they built — optimized for their function, their view, their priorities — is not aligned with productivity gains across the wider organization.

Single points of failure are among the most common drivers of failed launches we encounter. Not bad technology. Not bad intent. Concentration.

Your processes are not ready, and that is normal

In nearly every deployment, internal systems and processes turn out to be **fragmented, undocumented, inefficient, and heavily dependent on specific humans.** This is not a criticism of any particular company. It is the ordinary condition of organizations that grew by solving problems as they arose.

The consequence matters: for an agent to actually save time and money, the workflow itself usually has to change. That change requires humans and AI meeting in the middle — humans supplying the knowledge of what the process is really for and where exceptions live, the platform supplying what agents can reliably do.

A very large share of failures to launch come from companies trying to force AI to operate inside processes designed for humans. It does not work that way.

Human processes are built around human constraints — batching to reduce context switching, approval gates to catch fatigue errors, handoffs because no one person can hold all the information, documentation written as a memory aid. Agents have entirely different constraints. They do not tire, and they hold far more context at once, but they have no implicit knowledge, no hallway to walk down for clarification, and no instinct for when something is off. A process optimized for one set of constraints is rarely optimal for the other.

> *(The elaboration on differing constraints above is our framing of the pattern, offered to make the point concrete.)*

The thing every company lacks is intuition

Companies begin these programs with real intention and real investment. What they do not have — what they cannot have, because the technology is too new — is **intuition about what works, what doesn't, where the pitfalls are, and where the ROI actually lives.**

That intuition is only built by deploying. It is the one thing that cannot be bought as a license, hired as a consultant, or read in a report.

It is also the reason this document exists. Everything in it was learned the expensive way, and our position is that our customers should not have to learn it the same way.

PART VIII

Three Things We See Everywhere

Across our customer base, the same three conditions appear again and again — nearly independent of industry, size, or sophistication.

1. Nobody knows which half of the spend is working

There is an old adage in advertising: half of it is wasted, and nobody knows which half.

The AI version is now true in nearly every enterprise we encounter: **"I know 50% of our token spend is producing value. I just don't know which 50%."**

No system in place to manage token usage. No mechanism aligning that consumption to any outcome. Spend flows out, some of it produces something, and there is no instrumentation connecting the two. As a business model that is untenable — not because the spend is large, but because it cannot be defended, optimized, or scaled with confidence. Any budget that cannot be tied to outcomes eventually gets cut, and usually at exactly the moment it was beginning to work.

2. Ideas with no home

Most people in most organizations are still users. They have not crossed the chasm to builder, and nothing in their environment is helping them cross it.

There are exceptions, and they are worth paying attention to: someone in IT, or a technically-inclined person, or one of the naturally AI-forward employees starts building things inside whatever tools they have. This is exactly the instinct you want.

But there is no home for what they build. No platform to launch the agent on. No path to scale it. No way to establish confidence that it will behave correctly in production — whether it's a customer-facing support agent, an internal agent reviewing documents and producing analysis, or an agent enhancing the CRM and assisting with prospecting.

So the seeds of ideas that sprout at the edges never get enough sunshine or water to become trees the whole organization can stand under.

This is the quiet tragedy of the current moment. The initiative exists. The capability exists. The people exist. What is missing is the ground for any of it to root in.

3. The appearance of progress

There is a certain laziness in how many companies are embracing AI — and we use that word deliberately, though what sits underneath it deserves more sympathy than the word implies.

The pattern: buy an AI feature or add-on wherever an existing SaaS vendor offers one. Adopt the small application a team member vibe-coded over a weekend. Count both as wins.

They are not nothing. But **they do not move a company closer to thinking and behaving as an AI-native business.** They produce the appearance of progress while the actual lift — the knowledge extraction, the process redesign, the role change, the governance — remains untouched. And the appearance of progress is more dangerous than no progress, because it removes the urgency that would otherwise force the real work.

What the avoidance is actually made of

Calling it laziness is accurate about the behavior and unfair about the cause. Underneath it, consistently, are four things:

- **Fear of replacement.** People believe that building the thing that automates their work is building the thing that ends their job. Until that is addressed directly and credibly, expect quiet resistance rather than open objection.
- **Fear of distraction.** Capable, conscientious people worry that time spent on AI is time taken from the work they are actually measured on — and that they will do both badly. This fear is entirely rational, and it is the human face of the paradox in Part IV.
- **Disbelief.** Some people simply do not believe AI can do what they do. They are not always wrong, and they are rarely persuaded by argument. They are persuaded by seeing it work on something they know is hard.
- **Genuine lack of understanding.** Most people have no working model of how any of this functions.

That last one is worth being blunt about, because it is the widest gap and the least acknowledged.

There is an enormous distance between using ChatGPT to find a good spaghetti recipe and building a multi-agent workflow connected to several disparate systems, trained on volumes of company data, executing thousands of transactions a month.

Both are "using AI." They share almost nothing else. Most organizational confidence about AI readiness is built on experience with the first, and most organizational planning assumes it transfers to the second. It does not.

Closing that gap is not a communications exercise. It is the transformation described throughout this paper, and there is no version of it that skips the work.

PART IX

What Companies Must Get Right

1. Own the harness

Everything between the user and the LLM — routing, context, guardrails, memory, permissions, evaluation, audit — is where differentiation lives. Hand that layer to a vendor and you inherit their pricing, roadmap, and constraints.

Our answer is unusual: **customers build on our platform and still own their harness.** Open API, fully integratable, model-independent per agent. They extend it and build their own solutions on top of it.

This collapses the buy-versus-build tradeoff. Build gives control but costs years and creates the half-life problem. Buy gives speed but creates lock-in. The right answer delivers the acceleration of buying with the extensibility and ownership of building.

2. Organizing knowledge is the underestimated problem

Agents perform tasks. Performing them *well* — especially anything requiring reasoning — demands depth of context about the work itself. That comes from knowledge bases, training documents, sets of rules, systems of record, wherever the source of truth lives.

Two requirements, both consistently underestimated: knowledge must reach the agent **structured and AI-ready**, and agents need **live connectivity** to data sources, not a static snapshot.

3. Knowledge drift is the failure mode nobody plans for

Agents launch aligned with the business, then quietly fall out of sync as teams move to new projects, new documentation appears, and policies change.

The analogy is exact: an employee onboarded in January and told nothing since would be useless by June. Agents need continuous re-onboarding, not one-time training.

4. Participation cannot be delegated to IT

Dependence on a technical team creates inertia — every agent queues behind engineering priorities, and the backlog becomes the ceiling on scale. It misallocates your scarcest resources onto work functional experts should own.

But the deeper cost is the one described in Part III: it keeps your organization in the user role. The people who understand a process best remain the people not permitted to automate it, and the transformation never happens.

5. Cost has to be governed structurally

Token spend cannot be controlled through policy memos. It requires central visibility into usage, per-agent model selection so workloads run at appropriate cost, controlled distribution that eliminates shadow AI, and a feedback loop that improves how users interact with agents.

Cost governance is not a feature to add later. It is the difference between an AI program that survives its second budget cycle and one that doesn't.

6. The platform has to be the training

This is our central proposal, and it is the one we would most want a leader to take away.

Your workforce learns to build agents by building agents. Not through a training program. Not through a library of videos. Not through a certification track completed before anyone touches the real thing. Through doing the actual work — applying their expertise, curating their own knowledge, documenting their own functions, and seeing what the agent does with it.

A platform built for the average business user is therefore not a convenience. It is the mechanism of transformation itself. The interface is where the learning happens, or the learning does not happen.

And because it is centralized, that learning is shared. Agents built by different teams coordinate and orchestrate rather than fragmenting. People see each other's work, borrow from it, and improve on it. The entire organization moves through the transition **together, reading from the same sheet of music** — rather than a handful of technical staff moving while everyone else waits.

PART X

From the Edge to the Center

Almost every company reading this is in the same starting position: **AI is happening at the edges of the business, and nowhere in the middle.**

Individual subscriptions. A department that bought a point solution. Someone's weekend build. Shadow usage nobody has counted. Real energy, real ingenuity, and no center of gravity.

The destination is centralized command and control over the agentic workforce. The path between them is where most programs go wrong, so it is worth being precise about it.

The distinction that makes this work

Centralization is easy to get wrong in a specific way: a company decides to bring AI under control, routes everything through a central team, and recreates exactly the bottleneck described in Part IX.

That is not the goal, and it does not work.

Centralize the substrate. Distribute the creation.

What belongs at the center: knowledge, security, policy, compliance, orchestration, model selection, cost governance, and the platform itself. These must be common, or they do not exist at all.

What belongs at the edge, permanently: the building. The people who understand a process are the people who should be creating the agents that run it, and that should become *more* true as the program matures, not less.

Command and control is about the ground everyone builds on — not about who is permitted to build.

The sequence

1. See what is actually running.

You cannot centralize what you cannot see. Inventory the real state: individual subscriptions, departmental tools, shadow usage, internal builds, and anything already touching customer data. Most organizations are genuinely surprised by this exercise.

Approach it as discovery, not enforcement. The moment it becomes an audit, it stops being accurate.

2. Build the center before you consolidate onto it.

The most common sequencing error is shutting down edge activity before there is anywhere for it to go. This kills the initiative and, more importantly, kills the enthusiasm of the people you most need.

Stand up the platform, the knowledge layer, and the governance model first. Consolidation should feel like an upgrade, not a confiscation.

3. Migrate knowledge before you migrate tools.

The durable asset is not the agent someone built. It is the knowledge that went into it, and the process understanding that made it work.

Tools will be replaced within a year. Knowledge compounds. Prioritize getting company intelligence unified, structured, and agent-ready — regardless of which tool it currently lives beside.

4. Recruit the edge builders rather than overruling them.

The people already building at the edges are your first cohort of builders. They have proven both aptitude and motivation, which is the scarce combination.

Bring them in early, give them the platform, and let them shape how it gets adopted. A centralization effort that reads as a rebuke to the most AI-forward people in the company will succeed on paper and fail in practice.

5. Choose first agents that buy back time.

As described in Part IV, the earliest deployments should target high-volume, low-judgment work belonging to the people you most need freed up. These agents fund everything after them, and they prove the center works with evidence rather than argument.

6. Govern cost from the first day, not the first overrun.

Instrumentation, model routing, and usage visibility are far easier to establish before scale than to retrofit after it. A center that cannot answer what agents cost and what they produce will lose its budget regardless of how well the agents work.

7. Measure the workforce, not the tooling.

As the estate grows, the meaningful questions become managerial rather than technical: what capacity do we have, what does it cost, what is it producing, and where is it degrading. Build toward that view early, because it is what converts an AI program from an expense into a managed function.

What this produces

A company that completes this transition has something structurally different from a company with a lot of AI tools. It has **one place where knowledge lives, one place where policy is enforced, one place where cost is visible, and many people building on top of all three.**

Agents built by different teams can coordinate rather than duplicate. Knowledge gathered by one function becomes available to another. A policy decision applies everywhere at once. And the organization learns collectively, because everyone is working on the same ground.

That is the difference between having AI and operating an agentic workforce.

PART XI

What We Are Actually Building

Most of what is sold as agent infrastructure is a technology layer between a model and a set of users. That layer is necessary. It is not the point.

We are building a reimagination of how a company constructs its workforce — independent of any existing ecosystem.

This matters because of what it is *not*:

- Not an extension of your CRM
- Not an add-on bolted to your support desk
- Not a DIY project
- Not a set of vibe-coded applications waiting to become someone's maintenance burden
- Not a copilot for one department

Every business is, at bottom, a combination of **data and workflows**. Sales, support, operations, finance — information moving through a process, executed by people who cost time and money. Today a company spends thousands of dollars executing work that an agent, properly built and properly informed, can perform.

The ambition is that **every team member can build any number of agents that automate real work.** Not that IT can build agents faster. That the people doing the work build the agents that do the work.

That is a different category of thing than a feature added to software you already own.

PART XII

Preparation Is the Product

If there is one place where our view diverges most sharply from the rest of the market, it is here.

Almost everyone is focused on deployment. We are focused on preparation.

Deployment is the visible part. It demos well, it makes a good screenshot, and it is what a buyer thinks they are purchasing. So the entire industry has organized around it: launch an agent in minutes, connect it to your systems, watch it respond.

What gets skipped is the part that determines whether any of it works.

Garbage in, garbage out

The oldest principle in machine learning is also the most ignored in the agent era: **the quality of what goes in directly determines the quality of what comes out.**

Feed an agent substandard information and it will do substandard work. Not occasionally — reliably, and in ways that are difficult to detect until they have already cost you something. There is no model good enough to compensate for not knowing what your business actually does.

What agents are being deployed without

In the race to launch, agents are routinely put into production with three foundational gaps:

- **No real training.** Configured, prompted, connected — but never actually taught the work.
- **No access to core information.** Nothing that provides judgment, context, or an understanding of what the task is genuinely for. The agent knows the shape of the request and nothing about the business behind it.
- **No tooling to improve.** No mechanism to continuously train, test, evaluate, and correct. The agent is as good on day one as it will ever be, which in a changing business means it gets worse from there.

An agent deployed this way performs exactly as you would expect a new employee to perform if you handed them a login on their first morning, told them nothing about the company, and put them on the phone with customers.

That is the actual source of most agent failures. Not the model. Not the integration. The preparation that never happened.

Where we place the bet

We do deployment. We do post-launch management, governance, orchestration, and cost control — all of it is in the platform, and none of it is optional.

But the bet is on preparation, and the portfolio reflects it structurally:

- **Command exists to transform knowledge into something agents can actually use** — the distributed, undocumented, fragmented intelligence of a business, made agent-ready and kept in sync.
- **Copilot exists to train, test, and improve agents through human feedback** — the loop that turns a functioning agent into a trustworthy one.
- **CX is where prepared agents get deployed, governed, and scaled.**

Two of our three products exist for the work that happens before an agent goes live and continues after it does.

Why the rest of the market skips it

Preparation is unglamorous. It cannot be demoed in five minutes. It requires work from the customer that no vendor can perform on their behalf. It is, in every commercial sense, harder to sell than "deploy an agent instantly."

That is precisely why it is the differentiator. The market has optimized for what closes deals rather than for what makes agents work, and the gap between those two things is where most enterprise AI programs are currently dying.

> *(The commercial explanation above is our reading of why the pattern persists.)*

The same constraint applies to building software

The most confident claim in the market right now is that AI has transformed software development. It is worth examining, because it is true for some organizations and badly misleading for others — and the difference determines whether a build decision is sound.

The variable is not how complex the software is. It is how mature the builder is.

Writing code is one stage of the development lifecycle. Around it sit requirements and specification, architectural judgment, code review, QA, test infrastructure, integration, release process, monitoring, and long-term maintenance. Implementation was rarely the binding constraint. It was simply the most visible one.

AI compresses implementation dramatically. It does not compress the rest.

Software companies capture the gain, because they already have the rest. They know how to write a specification. They have QA discipline, test coverage, review culture, release engineering, and people whose judgment about architecture is earned. Drop a powerful code generator into a complete pipeline and it multiplies everything downstream of it. This is exactly why the frontier labs compound so quickly — they are software companies with excellent development practice, applying code generation to a machine that was already built.

Non-software companies do not get the same boost, because the missing stages are the actual work. They have a business need, someone who can code, and now a tool that writes code very fast. What they lack is everything that surrounds it. The result is predictable: **they produce code faster than they can specify, test, review, or maintain it.**

And the deficit compounds. Generated code still requires very specific instruction — requirements documentation, architectural context, an explicit account of what the system does, what it must preserve, why the data is modeled as it is, what each integration expects. Someone has to know those things and write them down. That knowledge is undocumented, held in people, and slow to extract, **which is the preparation problem again, appearing in the one domain where everyone assumed it had been solved.**

AI did not remove the bottleneck in software development. It moved it — from implementation to specification, from typing to knowing.

This carries a consequence worth stating directly, because it is shaping decisions right now. The belief that AI made building cheap is exactly what makes building your own platform look newly attractive. But if your organization has never had to run a full development lifecycle, the discount available to you is far smaller than the headlines suggest — and an agentic platform, being stateful, deeply integrated, governed, security-sensitive and long-lived, sits at the far end of the range where the missing stages matter most.

The tool that makes a prototype trivial does not make a platform trivial. Assuming otherwise is how a company ends up funding infrastructure it never intended to own, and discovering the cost only once it is in production.

The narrative that gets this backwards

There is a thesis moving through the market with a great deal of confidence behind it: **every company will become a software company, building its own software with AI.**

It is superficially compelling. Code generation is genuinely powerful, and the cost of producing working code has fallen sharply.

But the thesis rests on an assumption nobody is examining: that a company which can now generate code can therefore build software. **It omits the question of whether non-**

software organizations can do the work that comes before the building, and whether they understand the lifecycle that surrounds it.

Specification. Architectural judgment. QA and test discipline. Review culture. Release process. Monitoring. Maintenance over years. None of these are compressed by code generation, and none of them arrive with a tool purchase. They are institutional capabilities built over a long time by organizations that had to build them to survive.

Which leads to a conclusion the market has almost exactly inverted:

Software companies are positioned to lead the AI race, not to be disrupted by it.

They understand the full cycle. For them, code generation is a multiplier applied to a machine that already works — the frontier labs are the clearest example, but every competent software organization gets a version of the same compounding. The narrative predicts a leveling. What is actually happening looks more like a widening.

This is the same error identified in Part V, in a more confident form. There we described companies that drift into becoming software companies by accident, distracted from their core business. The market thesis recasts that accident as strategy and encourages it deliberately.

For a company that is not a software company, the opportunity was never building software. The advantage you hold is the thing a software company cannot easily acquire: domain knowledge, proprietary data, process expertise, customer relationships, and judgment accumulated over years of doing work that matters.

Encoding those into an agentic workforce is a fundamentally different act than becoming a software company. It plays to what you already have rather than to what you would have to build from nothing. And it is the version of this transformation that is actually available to you.

If you are a software company

Everything above argues that software companies hold the advantage in this transition. Which raises the obvious question for a software company reading it: *why would we not build this ourselves?*

You could. That is not in dispute, and we will not pretend otherwise. You have specification discipline, QA, release engineering, architectural judgment, and people who have shipped and maintained real systems. The constraint that stops a manufacturer or a distributor does not apply to you.

So the honest case is not about capability. It is about cost, time, and focus.

Cost and time. By our own accounting, reaching the point this platform is at today represents somewhere in the range of one to two million dollars and roughly two years of sustained, funded product work. Not a side project, not a quarter of one team's capacity — a real product effort with people dedicated to it.

And the target moves while you build. That two years is measured against where the platform stands today, not where it will stand when you arrive. Given the half-life problem described in Part V, a company that starts now is building toward a specification that will have moved substantially before it ships. This is the single most underestimated element of the build case.

Focus. Every engineer working on agent infrastructure is an engineer not working on your vertical product. That product is your moat — the domain depth, the customer relationships, the workflow knowledge that took years to accumulate and that a competitor cannot copy quickly.

Agent infrastructure is not a moat. It is nearly identical at every company that needs it: harness, orchestration, governance, knowledge pipeline, evaluation tooling, cost controls. Spending your scarcest and most differentiated engineering capacity on the part of the stack that is the same everywhere is the clearest opportunity cost in this entire paper.

And the dependence concern is answerable. The legitimate reason a software company wants to build is control — not wanting its AI future determined by someone else's roadmap, pricing, or priorities. That instinct is correct, and it is exactly why the first imperative in Part IX exists.

Building *on* a platform is only the same as depending on it when the platform is closed. With an open API, full integratability, model independence, and ownership of the harness, you get the two years back without surrendering the control you were trying to protect. You extend it, build your own solutions on top of it, and take your knowledge with you.

That is the actual choice: not build versus buy, but **whether to spend two years and two million dollars rebuilding the part that is the same for everyone, or to spend that time on the part that is only true of you.**

The window for starting from scratch has closed

Set aside cost and focus for a moment, because there is a timing argument that applies even to a company willing to absorb both.

The moment to build an agent platform from scratch was 2023. The surface area was small enough to cover, the pace was survivable, and a well-resourced team starting then could lay foundations while still tracking where the field was going.

That window has closed. AI is now moving fast enough that a from-scratch effort spends its first year building infrastructure the field has already moved past — and the gap widens rather than narrows, because a team heads-down on foundations is by definition not tracking the frontier. You do not catch up to a moving target by starting further behind it.

Building it properly also demands your best people. Not spare capacity or a motivated volunteer — the engineers with the strongest architectural judgment, for two years, on infrastructure rather than on product.

Which means the real conclusion of this section is not that software companies should buy and others should not. **It is that both now have to invest in a platform,** arriving at the same answer from different directions: non-software companies because they lack the development lifecycle around code generation, software companies because the window for building from scratch has passed and the opportunity cost has become indefensible.

The question is no longer build or buy. It is **which platform**, and the criteria follow from everything in this paper:

- **Maximum control** — you own the harness between your users and the model, not the vendor.
- **Maximum flexibility** — an open API and full integratability, so you can keep building your own solutions on top rather than working inside someone's box.
- **Genuine independence from the frontier model companies** — because a platform that resolves one dependency by creating another has not solved the problem, only relocated it.

That third criterion deserves particular weight. It is entirely possible to escape the burden of building infrastructure and land directly in dependence on a single model provider, with your economics, your capability ceiling, and your continuity all determined somewhere else. Independence is not a feature you can add later. It is either in the architecture or it is not.

And preparation is where the transformation happens

There is one more reason this matters, and it connects back to the argument at the start of this paper.

Preparation *is* the builder work. Curating knowledge, documenting a process that lived only in someone's head, deciding what good output looks like, testing an agent and correcting it — these are exactly the activities that move a person from user to builder to manager.

A platform focused only on deployment asks your people to remain users of a slightly different tool. A platform focused on preparation puts them through the transformation itself, in the course of doing real work.

That is not a feature difference. It is a difference in what the product is for.

PART XIII

The Platform

raia CX — the platform *(flagship)*

Zero-code environment where agents are built, configured, and governed. Model choice per agent. Orchestration rules, security, and compliance defined centrally and applied across the entire estate. Open API and full integratability make CX a foundation to build on, not a box to work within.

Addresses: own the harness (1), participation (4), cost governance (5), platform-as-training (6).

raia Command — knowledge into intelligence

Unifies intelligence distributed across documents, applications, and databases through connectors and direct ingestion, converting it into AI-ready knowledge that flows into CX to train agents. Because connectivity is live rather than a one-time import, Command is also the answer to drift.

Addresses: organizing knowledge (2), knowledge drift (3).

raia Copilot — where agents get used and improved

The interface through which internal users engage the agents built on CX, without seat-license budgets and with centralized token control. Its deeper purpose is the human feedback loop — the manager stage of the transformation. Agents nobody trains, tests, or corrects never reach the efficacy bar real deployment requires.

Addresses: knowledge drift (3), participation (4), cost governance (5), platform-as-training (6).

The arc

Make knowledge usable → make agents buildable by anyone → make agents trustworthy → make the workforce measurable: agents orchestrating across each other, managed the way headcount is managed, with capacity, cost, performance, and audit in one view.

PART XIV

Staying Synchronous

There is one more consideration, and it may be the one with the shortest fuse.

Every investment you make in AI right now is at risk of being obsolete within three to six months.

Not because it was built badly. Because the ground is moving faster than anything technology has moved before, and a capability that required six months of custom engineering in the spring can arrive as a free feature in the autumn.

The redundancy trap

This creates a failure mode that catches serious, well-run companies: **you spend real money and real team time building something your vendor was already going to ship.**

The work isn't wrong. It's redundant — and redundant work is worse than no work, because it consumed the scarcest resource you have while producing an asset with no remaining value. Worse still is building something that turns out to be a non-factor entirely, overtaken by a change in how the technology works.

Companies fall into a cycle of misalignment: investing energy in things that become free, or irrelevant, in a matter of months.

Nobody runs at frontier pace

The frontier model companies and the hyperscalers set the tempo, and it is not a tempo an operating business can match. Neither can most vendors. Almost no organization has the capacity to absorb what is being released, evaluate it, and adjust course at the rate the field is actually changing.

That is not a failure of ambition. It is arithmetic.

The keyword is synchronous

Which leads to the single most important word in this section: **stay in sync.**

If you cannot run at the pace of the frontier, your next best position is a close relationship with a partner who is oriented toward it — one that keeps you synchronized rather than leaving you to discover changes after they have already cost you.

Concretely, that means a vendor you can **influence** and whose **roadmap you can coordinate with.** Open communication about what is coming, so you can make the decision that actually matters: **what to build custom, and what to wait for.** Get that decision right consistently and you avoid the redundancy trap entirely. Get it wrong and you fund the same capability twice.

This has never been a significant selection criterion in enterprise software. In this market it may be the deciding one.

What to watch

Staying synchronous requires knowing what to track. Four dynamics are worth watching closely, because each one changes the build-versus-wait calculation.

1. Code generation is the frontier labs' flywheel.

The model companies have concentrated enormous effort on code generation, and not only as a product. It compounds: better code generation lets them build their next products faster, which improves the models, which improves the code generation.

The practical consequence is that **the cost of building software is falling fastest for organizations that were already good at building software.** Code generation multiplies a development pipeline; it does not substitute for one. The labs compound quickly because they are applying it to mature specification, QA, and release practice.

Read that carefully in light of Part XII before drawing conclusions about your own build decisions. If your company has never run a full development lifecycle, the productivity gain available to you is a fraction of what a software company captures — because the stages AI does not compress are the ones you would have to build from nothing.

2. Models are shifting toward reasoning and asynchronous work.

The trajectory is away from fast conversational response and toward considered, higher-quality output that takes longer to produce. Models increasingly think before they answer, and the emphasis has moved from latency to quality.

This is more consequential operationally than it appears. **Asynchronous work requires an entirely different design than chat.** Submitted tasks, queues, results returned later, output that needs reviewing rather than reading. It also means agents behave less like a search box and more like a colleague you delegate to — which is precisely the co-worker relationship described in Part II, arriving as a technical reality rather than a metaphor.

Most organizations have designed no process for receiving, reviewing, and acting on asynchronous agent output. That gap will become visible quickly.

3. There is no visible ceiling yet, and that matters for planning.

Open-source models are applying real pressure to frontier pricing. But the more important open question is capability: things have moved fast enough that nobody knows how far frontier models can actually go, and there has not been enough time for businesses to find out.

This has a strategic implication worth thinking through. **If model progress slows and training investment moderates, the frontier labs will move up the stack** — building applications above their own models to sustain market capture. That movement is already visible in products like Claude Code and ChatGPT.

For a company building on those models, this is a genuine consideration: **your model provider may become your application competitor.** It is not a reason to avoid them. It is a reason not to be structurally dependent on any single one.

4. SaaS vendors face a problem they may not be able to solve.

Incumbent software companies are struggling with AI adoption for two compounding reasons.

The first is the revenue model, discussed above: seat licensing and maintenance revenue do not map onto usage-based economics.

The second is deeper and gets less attention. **How do you make software built for humans into something AI-native, when it was never built for AI?** These products encode their capability in interfaces — screens, forms, wizards, workflows designed for a person to navigate. Agents do not need screens. They need programmatic access to the underlying operations and data, which in many legacy products either does not exist or exposes only a fraction of what the interface can do.

Retrofitting an agentic layer onto a human interface produces exactly what we see across the market: something that looks like AI, sits on top of the existing product, and does not change how the work is done.

This is the primary challenge facing incumbent software, and it is not a matter of effort or intent. It is architecture. Which is why an agentic workforce built independently of any existing ecosystem is a fundamentally different proposition than one bolted onto software designed for a different era.

Which argues for niche over generic

A generic provider builds for the median of a very large market. A partner who understands your business — and your customers — can build toward what you specifically need, and can tell you honestly when something you are considering is already on their path.

That relationship is also the strongest available defense against the DIY trap described in Part VI. Every hour your team spends building something that gets featured out from underneath them is an hour taken from work only they can do.

What the incumbents are actually doing

It is worth being clear-eyed about the posture of the large software vendors, because their behavior is easy to misread as progress.

Most are running a **land grab, and defensively.** The priority is retention — keeping customers from leaving — so they ship an agentic layer that does not fundamentally change how the software is used. It looks like AI. It does not produce the multiplier effect that a genuinely agentic approach does, because it was never designed to.

The strategic purpose is to **buy time and freeze the market**, preserving their position long enough to eventually build what customers actually want.

And they face a structural problem that no amount of engineering solves quickly: **their business model cannot easily make the transition.** Classic seat licensing and maintenance revenue do not map onto usage-based economics, which agentic AI essentially requires. That is not a roadmap gap. It is a revenue model built for a different technology, and unwinding it means damaging the numbers they are measured on.

Model dependency is a strategic risk

The most legible argument against building on a single model provider is cost. It is worth stating precisely — and then being clear that it is not the strongest reason.

The unit of account has broken. Several things happened at once over the past two years. Standard frontier pricing came down substantially. A premium reasoning tier appeared above it at an order of magnitude higher cost. Reasoning models began billing internal chain-of-thought as output tokens, so a fixed unit of completed work now consumes far more billable tokens than it used to — on models whose posted price never changed. And tokenizer revisions have shifted how much text a token represents, moving effective cost again without any change to the rate card.

The consequence for a buyer is not simply that things got more expensive. It is that **price per token stopped carrying useful information.** A finance team tracking that number could watch it fall while the invoice climbed, with nothing on the bill explaining the gap. The only measure that still means anything is cost per completed unit of work — and that can only be managed by an organization able to route workloads across tiers and providers as the economics move.

But cost is one exposure among several. Committing your agentic workforce to a single provider concentrates all of them:

- **Capability leapfrogging.** The best model for a given task changes every few months. A locked architecture cannot capture improvements that arrive somewhere else.
- **Availability and continuity.** Models retire on the vendor's schedule, not yours. Access can also be interrupted by forces neither party controls — in mid-2026, a leading provider suspended access to two of its models to comply with export controls, restoring it weeks later. A production agent depending on a single model has no answer to that.
- **Capacity and rate limits.** You inherit your provider's scaling constraints at precisely the moment your own volume is growing.
- **Policy and jurisdiction.** Usage policies, regional availability, and data residency requirements change, for reasons unrelated to your business.
- **Negotiating position.** You have none if you cannot credibly leave.
- **Task-fit routing.** Locking to one provider usually means locking to one price point, so work a smaller model would handle well runs on a frontier model by default. This is among the largest sources of the waste described in Part V.

None of these are hypothetical, and none are solved by choosing a better vendor. They are solved by not making the choice irreversible.

This is the same argument as owning the harness, arriving from the direction of risk rather than architecture. **Model independence is not a philosophical preference. It is the mechanism by which a company retains the ability to change its mind** — about cost, about capability, about jurisdiction — without rebuilding what it has already paid for.

The Future Is Human Plus AI

Especially in the enterprise.

The organizations that benefit from AI will not be the ones that replaced their teams. They will be the ones that moved their teams from users to builders to managers — and gave every person a way to make that transition alongside their colleagues rather than alone.

This is difficult work. We are not going to pretend otherwise. What we can do is make the difficulty the *right* difficulty: the judgment, the domain knowledge, the critical thinking that only your people possess — instead of infrastructure, integration, governance, and plumbing that we have already built and that no company should be rebuilding.

Your business does not need to become a software company. It needs to become a company where every person can build.

That is what we are building toward, and it is what our customers are already proving is possible.