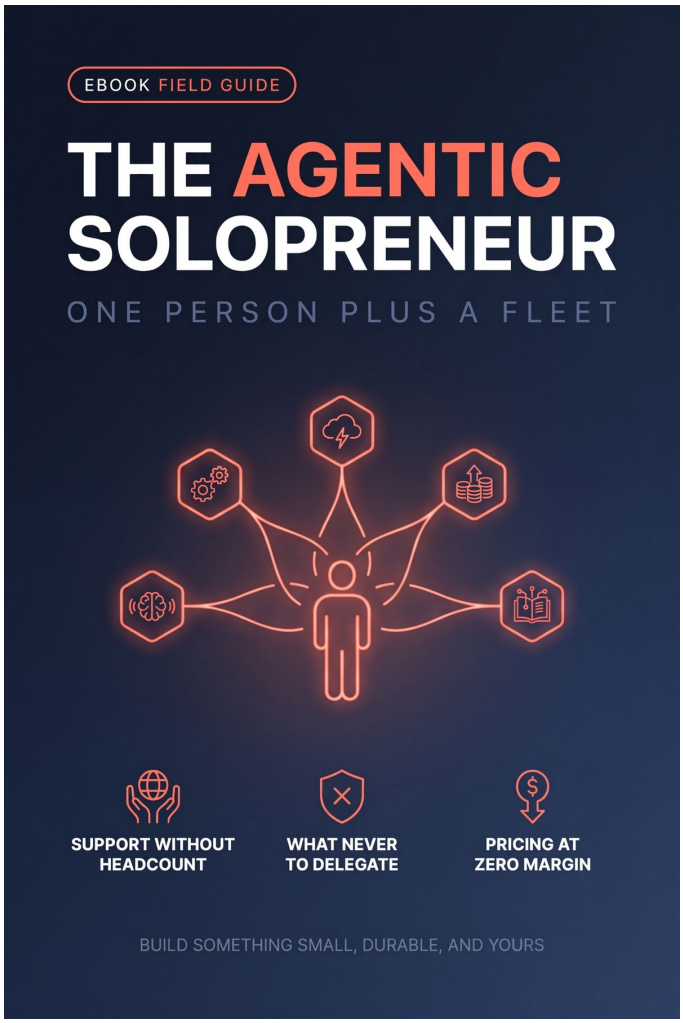

The Agentic Solopreneur

Every piece of good advice about running a business alone used to come back to one constraint: there is one of you, and you have a fixed number of hours. Don't build software that needs support, because questions scale with customers and you can't hire. Sell content instead, because content is finished once you publish it.

That advice was right, and it stopped being right.

This is a guide to building a small, durable business on your own in 2026, using leverage that didn't exist when most of the advice you've read was written. One person plus a fleet: a handful of narrow, well-guarded agents handling the parts of a company that used to need headcount, run by someone who stays close enough to the customer to notice when it's going wrong.

It covers choosing what to build now that the content-versus-software split has been rearranged, what a fleet actually costs in money and attention, the parts of the job you must never delegate, pricing when your marginal cost of delivery approaches zero, and the specific ways this fails.

It includes the year I grossed ten thousand dollars, because a version of this genre that skips to the good part is worse than useless.

This is a living document and will be updated as the tools and the economics keep moving.

Roger Stringer · rogerstringer.com

September 13, 2026

Contents

The Ceiling Moved	4
One Person Plus a Fleet	6
How I Got Here	8
Choosing What to Build	10
Your Operating Layer	12
Support Without Headcount	14
Content and Marketing at Throughput	16
Selling as One Person	18
What You Don't Delegate	20
Guardrails That Hold	22
Pricing When Delivery Is Nearly Free	24
Launching	26
How This Fails	28
What I Would Do Now	30
Toolkit: The Context Layer	32
Toolkit: The Delegation Line	34
Toolkit: The First 90 Days	36
About Roger	38

The Ceiling Moved

Don't build software that needs support, because questions scale with customers and you can't hire. Sell content instead, because content is finished once you publish it and then it works while you sleep. Don't take on anything with a pager. Don't sell to enterprises, because enterprises want a phone number and a person on the other end of it.

That's the standard advice for building a business alone, and for about twenty years it was right.

All of it came back to one constraint: there is one of you, and you have a fixed number of hours. Past a certain number of customers, a one-person business stops being a business and becomes a queue with your name on it.

That constraint moved.

Not disappeared. Moved. And almost everything that follows in this guide is about working out where it moved to, because the old rules were tuned to where it used to be.

The claim that stopped being true

Here's the specific sentence that used to end the argument, in one form or another: *the number of questions scales linearly with the number of customers.*

Take it seriously for a second, because it was the reason solo founders got pushed toward courses and ebooks and away from anything with a login. Ten customers is a handful of emails a week. A thousand customers is a full-time support job that you personally do, forever, including the week you're supposed to be on holiday. It didn't matter how good the product was. The math was the math.

Now: most of those questions have been asked before, by someone, and answered. They arrive as text. They need a correct, kind, specific reply that knows your product and your docs and what this particular customer already tried. That is a well-shaped job for a system, and has been for a while now.

I'm not claiming support disappears. I'm claiming it stopped scaling linearly, which is a different thing and a much bigger one. The queue with your name on it became a queue where you handle the interesting fifth and review the rest.

Once that's true, a pile of decisions you were told not to make are back on the table.

What actually changed, and what didn't

The honest version of this, without the breathlessness.

What moved: support and triage. First drafts of nearly everything. Research and synthesis. Routine ops. The long tail of small engineering work that used to sit in a backlog forever because it was never worth a whole afternoon. Content throughput. Anything where the work is real but the judgment required is low and the pattern is well established.

What did not move: deciding what to build. Knowing which customer complaint is a symptom and which is the disease. Taste. Whether a price is right. Whether you'd be proud of the thing. Relationships with actual people, which remain the only durable moat a one-person business has ever had. And responsibility, which cannot be delegated to something that can't be held accountable.

If you take one idea from this guide, make it that split. Nearly every failure in this space comes from moving an item off the second list onto the first.

One person plus a fleet

The shape I'm arguing for is one person plus a fleet.

Not "AI employees." Not a company of agents that runs itself while you sit on a beach, which is a thing people sell and not a thing that exists. A fleet is a set of narrow, well-scoped, well-guarded workers that handle the parts of your business that are legible and repeatable, run by one person who stays close enough to the customer to know when something has gone wrong.

The person is load-bearing, and that's the whole design. A fleet with nobody paying attention produces a great deal of confident output and no business.

What this buys you is not a bigger company. If you wanted a bigger company you'd hire, and hiring is a well-understood technology with two decades of writing behind it. What it buys you is a business whose ceiling is set by your judgment rather than by your typing speed, which for most people is a much higher ceiling.

What this guide is

This is what I'd tell you if you asked me over coffee how to build something on your own in 2026 without it eating your life.

We'll go through choosing what kind of business to build now that the content-versus-software split has been rearranged. What a fleet actually consists of, and what it costs in money and in attention. Which parts of a company used to need headcount and mostly don't anymore, and which still do and always will. Pricing when your marginal cost of delivery approaches zero, which breaks some assumptions you probably hold. And the ways this goes wrong, because it goes wrong in specific and repeatable ways.

It is not a guide to getting rich, and I'd be suspicious of anyone who says otherwise. It's a guide to building something small, durable, and yours, using leverage that genuinely didn't exist when most of the advice you've read was written.

Two things before we start.

I'm going to assume you can build, or are willing to learn. Not deeply, and not everything, but enough to know when something you've been handed is wrong. That's now the core skill.

And I'm going to be specific about numbers where I can, including my own bad years. The version of this genre where someone tells you it worked out without telling you what it cost is not useful to anybody.

Next, what a fleet actually is, in concrete terms, and what it isn't.

One Person Plus a Fleet

"Fleet" sounds grander than it is. Let me take the mystique out of it, because the mystique is where people lose money.

A fleet is a handful of narrow workers, each of which does one job you understand well, wrapped in enough code that you can see what they did and stop them when they're wrong.

That's it. If you were expecting something more impressive, the unimpressiveness is the point. A setup that works is four or five of these, not forty.

The unit is the loop, not the model

The most common mistake is thinking the model is the thing you're buying. It isn't. Models are a commodity you rent by the token, and they change every few months.

The thing you actually own is the loop around the model: what context it gets, which tools it can reach, what happens when it fails, how you know what it did, and what it's forbidden from touching. Two people using the same model get wildly different results, and the difference is almost never the prompt. It's everything wrapped around it.

I've written the long version of this in Loop Engineering (https://rogerstringer.com/guides/loop-engineering), so I won't repeat it here. For our purposes: when I say a business runs five agents, I mean it runs five loops, each with its own context, its own tools, and its own blast radius.

What a small fleet actually contains

Here's a realistic starting set for a one-person business. Yours will differ, but the shape holds.

The support agent. Reads incoming questions, has your docs and your product's actual behavior in context, drafts a reply. Sends the easy ones, escalates anything it's unsure about or anything touching money or account access. This is the one that broke the linear-scaling problem, so it's usually first.

The build agent. Does the small engineering work: the bug with a clear repro, the copy change, the migration, the endpoint that should have existed months ago. Not the architecture. The long tail.

The research agent. Reads what customers said, what competitors shipped, what's in the support queue this week, and tells you what's actually going on. This one replaces the thing you were always going to do on Friday and never did.

The content agent. First drafts, repurposing, the mechanical half of publishing. Never the final word in your voice, which is a distinction worth more than it sounds, and I'll come back to it.

The ops agent. The boring recurring jobs. Reconciliation, reports, the weekly checks you'd otherwise skip until something breaks.

Five loops. None of them clever. Each one covers a category of work that used to be either yours or nobody's.

Mine run on Claude Code for the build work, n8n for the scheduled and event-driven jobs, and Directus as the data layer they all read from. None of that is load-bearing to the argument. Use whatever you'll actually maintain.

What it costs

Two currencies, and people only budget for one.

Money is the easy one, and it's usually smaller than people expect. A coding agent subscription, an automation runner, a database you were probably already paying for, and token spend on top. For five narrow loops that lands under what a single day of contract help costs, and almost all of it is fixed rather than per customer, so it doesn't move when your customer count does. Whatever it comes to, it keeps falling, and it is not the constraint.

Attention is the real cost, and it's the one that catches people. Every agent you add is a thing that can be confidently wrong in a way you have to notice. Five well-understood loops that you check are worth more than twenty you've stopped reading. The failure mode is a fleet nobody is auditing, quietly producing plausible garbage for a month.

Budget your attention first and your money second. When you're deciding whether to add a sixth agent, ask whether you'll actually read what it does.

The layer underneath

There's one piece of infrastructure worth building before the fleet itself, and skipping it is the most common reason people conclude that agents don't work.

Your agents need shared context: what this business is, who it's for, how you write, what you'd never do, where things live, what "done" looks like. Written down once, in files, available to every agent. Without it you re-explain your business at the start of every session, and each agent produces something locally reasonable that doesn't match anything else you've shipped.

I've built this out in detail in Building Your Agentic OS (https://rogerstringer.com/guides/building-your-agentic-os), and the multi-agent coordination side in Running the Fleet (https://rogerstringer.com/guides/running-the-fleet). If you only read one before continuing, make it the first.

What it feels like day to day

Less magical and better than the demos suggest.

You start the day reading, not writing. What came in overnight, what the fleet did about it, what it flagged. Then you spend your actual working hours on the things that needed a person: the customer conversation that's really a product decision, the architecture call, the thing you're building that nobody has asked for yet because nobody knew to.

The change isn't that you work less. Often you don't, because the interesting work expands to fill the space. The change is what the work *is*: far less of it goes to things you were doing purely because there was nobody else to do them, which for a solo operator is most of the job.

Where this goes wrong

Briefly, because there's a whole chapter on it later and you should know the shape now.

Over-automation, where you hand over a decision that needed you and find out three weeks later. Drift, where the fleet keeps working and you stop reading and the quality decays somewhere you're not looking. And volume, where you can now produce ten times the content or ten times the features and mistake that for ten times the business.

Every one of those is a version of the same mistake: treating the fleet as a replacement for attention rather than a multiplier on it.

Next, how I got here, including the year it didn't work.

How I Got Here

I went out on my own in 2015.

I'd been building software for about a decade before that, on other people's payrolls, on everything from travel platforms to internal tools nobody outside the company will ever hear about. By 2015 I knew exactly what I wanted to spend the next decade on, and none of it was ever going to arrive as a ticket, so I stopped waiting to be assigned it.

What I want to tell you about is not the part where it worked.

The ten thousand dollar year

There was a year in there where I grossed ten thousand dollars.

Not profit. Gross. For the year.

I'm putting that number in print because the version of this genre where someone skips to the good part is worse than useless, it's actively misleading. You read enough of those and you conclude that the people who made it had some quality you're missing, when usually what they had was a longer runway or better timing or a partner with a salary.

Here's what a $10k year actually feels like, in case you're in one. You are working constantly. You are not lazy and you are not stupid and the work is genuinely happening. What isn't happening is any of it compounding. Every month starts at zero. You take whatever comes in because you can't afford to be selective, which means you never build a reputation for any specific thing, which means next month also starts at zero.

That's the trap, and it's a structural one rather than a character one. Selectivity is what compounds, and selectivity is exactly what you can't afford when you're broke.

What broke it was narrowing, and I did it years later than I should have. I stopped taking everything that came in and started turning down anything that didn't look like more of the work I wanted. The first few months of that were frightening and the income dipped. What it bought was a description of myself that fit in one sentence, and other people started repeating that sentence for me. The first client who sent me a second client is the point where the math actually changed. Compounding is small and unglamorous when it starts. It looks like one referral.

The cruel part is that you can't narrow while you're panicking, and the ten thousand dollar year is nothing but panic. If that's where you are: take the work, keep the lights on, and narrow by one degree at a time on the way through. Waiting for a clean moment to specialize is how people stay general for a decade.

What the business is now

A mix, deliberately.

Consulting and fractional CTO work is the backbone, most of it through Data McFly (https://datamcfly.com). That's the part that pays reliably and it's also the part that keeps me honest, because it puts me inside other people's real systems with real constraints instead of theorizing from the outside. Nearly everything I write comes out of something that broke on a real engagement.

Around that sits the writing: these Field Guides, the blog, the things I build in public. That started as marketing and turned into something closer to a product over time.

I'm not going to publish a full revenue breakdown, partly because a lot of it is covered by client

confidentiality and partly because a single year's numbers from one person's business is not the kind of data you should be making decisions from. What I will say is that the mix is the point. The consulting smooths the volatility of everything else, and the writing is what makes the consulting inbound rather than something I have to chase. Neither works nearly as well alone.

Where the agents came in

A couple of years ago the tooling crossed a line. Agents could hold a real task for more than a few minutes without going sideways, and I stopped treating them as a curiosity.

The first thing that changed wasn't code. It was everything around the code. Research I'd been putting off. Documentation that had been stale for a year. The long tail of small fixes on my own projects that never justified an afternoon. Work that was real and necessary and that I had simply been failing to do, for years, because there was one of me.

The second thing that changed was scope. There's a category of engagement I used to turn down because it needed more hands than I had. Some of those are now takeable, because a meaningful chunk of the grunt work in them is legible and delegable. That's the concrete commercial version of the ceiling moving.

What didn't change: I still read every line that goes out under my name. I still take the calls. The judgment about what's worth building and whether it's any good is still mine, and if I ever hand that over the whole thing stops being worth doing.

Why I'm telling you this

Because the advice in the rest of this guide is downstream of a specific vantage point, and you should know what it is.

I'm not writing this from a position of having built something enormous. I run a small, deliberately small, business that pays well and doesn't own me. I've had years that didn't work. I've taken client work I didn't want in order to make rent, and I'd do it again in the same situation rather than pretend that's beneath anyone.

What I have is a decade of doing this alone, and a stretch of doing it alone with a fleet, which is a genuinely different job. The second part is what's new and it's most of what follows.

Next, the decision that used to define a solo business and now needs re-making: what kind of thing you're actually going to build.

Choosing What to Build

The oldest decision in this genre is content or software.

Content meant courses, books, newsletters, videos. You make it once, you sell it many times, and crucially nobody can page you at 2am because a course doesn't go down. Software meant a product with a login, which meant uptime, which meant support, which meant you had built yourself a job with worse hours than the one you left.

That split was drawn where it was for one reason: the cost of *serving* a customer after you'd sold to them. Content's serving cost was near zero. Software's was not, and it grew with every customer.

Both sides of that comparison moved, in opposite directions. Which means the decision is genuinely open again, and most people are still making it on the old information.

Software: back on the table, with conditions

The serving cost of software fell. Not to zero, but far enough that the old blanket "don't" no longer holds.

The conditions matter though, and they're specific:

It has to be a product where being wrong is recoverable. Support that drafts a reply about a billing question is fine, because a wrong draft gets caught. A product where an automated response could cost someone money or data is a different risk profile, and you're one person with no one to catch you.

You have to be able to verify it works without watching it. Tests, monitoring, alerts that mean something. If the only way you know the product is healthy is by using it yourself every morning, you've rebuilt the job.

The domain has to be one you actually know. This is the one people skip. A fleet accelerates building the thing. It does nothing for knowing whether the thing should exist, and building the wrong product faster is not progress.

If you clear those three, software is a better business than it was, because the thing that used to eat solo founders alive is now mostly handled.

Content: still good, but the moat moved

Content got easier to produce, which sounds like good news and mostly isn't.

When anyone can generate a competent article on any topic in a minute, competent articles stop being worth anything. The floor rose and the value of being at the floor collapsed. If your plan was to publish a lot of solid, useful, unremarkable posts and rank for them, that plan is worse than it was in 2023.

What survives is the stuff a model can't produce from public information: what actually happened when you tried it, the number nobody publishes, the opinion you'd defend in an argument, the thing you got wrong and fixed. Specificity and stance. Everything else is now commodity.

So content is still a good business. It's just that the work moved from volume to judgment, and judgment doesn't parallelize.

Services: underrated, and the thing that funds the rest

The genre tends to treat consulting as the thing you escape from. I'd push back on that.

Services have a property nothing else has: they pay immediately, at a real rate, without needing an audience first. They also keep you inside real problems, which is where the material for everything else comes from. A product built by someone who hasn't touched a customer's actual system in two years tends to show it.

The trap is real. Services don't compound, and they stop paying when you stop working. But treating them as a funding mechanism and a source of raw material, rather than as a failure state, is a much better frame. Most durable one-person businesses I'd point at run some version of this mix.

How to actually decide

Three questions, in this order.

What do you know that most people in your field don't? Not "what are you interested in." What have you done enough times to have opinions about. This is your only real input, and everything downstream is a format decision.

What can you verify without babysitting? Whatever you build, you'll be the only person checking it. Pick the shape where "is this working" has an answer you can automate.

What still pays if you disappear for two weeks? Ask it now, not later. The whole point of building alone is the freedom, and it's very easy to build something that removes it.

The honest answer for most people is a mix, weighted differently over time. Services early because they pay. Content throughout because it's how people find you. Product when you've found something specific enough to be worth building and you understand it well enough to verify it.

Next, the layer that has to exist before any of the fleet is worth building.

Your Operating Layer

People try agents, get mediocre results, and conclude the technology is oversold. Usually what's missing is underneath.

An agent starts every session knowing nothing about your business. Not your customers, not your pricing, not how you write, not the thing you decided six months ago and would be annoyed to see relitigated. If you don't hand it that context, it invents something plausible. Plausible and wrong, consistently, in a way that takes you longer to fix than doing the work yourself would have.

So before the fleet, the files.

What actually goes in

I've written the full version of this for engineering work in Building Your Agentic OS (https://rogerstringer.com/guides/building-your-agentic-os). For a business rather than a codebase, the contents are different. Roughly:

What this business is. What you sell, to whom, what problem it solves, what it explicitly does not do. Two paragraphs, not a strategy deck. The "does not do" line earns its place more than the rest combined.

Who the customer is. Their actual situation, in their words where you have them. Not a persona document. The three things they always ask, the two objections that come up every time, the thing they're really worried about underneath the thing they say.

How you sound. Not adjectives. Examples. A paragraph you wrote that landed, a paragraph you'd never send, and a short list of the specific tics you don't want (for me that list starts with the long dash and gets more opinionated from there).

Your standing constraints. Prices and what's negotiable. What you'll never automate. What always comes to you before it goes out. Which claims you're not allowed to make because you can't back them up.

What "done" looks like. Per kind of work. A support reply is done when it answers the question and the customer doesn't have to write back. A draft is done when it has a point and a specific example. Write this down and your fleet stops handing you 80% work with confidence.

Why files and not prompts

You could put all of this in a prompt. People do, and then it lives in one tool, drifts out of date, and exists in four slightly different versions across four different agents.

Files fix that. One source, version controlled, referenced by everything. When your pricing changes you change it once. When you notice an agent making the same mistake twice, you don't correct it again, you go add the line that prevents it. That habit is the whole discipline: **the second time you give the same piece of feedback, it belongs in the files, not in the conversation.**

This is also what makes a fleet coherent rather than five tools with different ideas about your business. Shared context is the difference between a team and five strangers who happen to work for you.

It rots

Written once and left alone, this layer decays quietly. Prices change, the product changes, you change your mind about something and only tell one agent.

The failure is hard to spot because nothing breaks. Output keeps arriving, confident and slightly wrong,

matched to a version of your business that stopped existing in March.

Two habits keep it honest. Read the files end to end once a month, which takes ten minutes and is never wasted. And when something goes out that made you wince, fix the file before you fix the output. The output is one instance. The file is every future instance.

Start smaller than you think

The instinct is to write everything before starting. Don't.

Write the two paragraphs about what the business is and who it's for. Add your constraints. Start one agent. Every time it gets something wrong because it didn't know something, add that thing. Inside a month you'll have a context layer built entirely from real failures, which is far better than the one you'd have written from imagination.

The smallest example I have is also the one I point at most often. For months every draft came back to me stitched together with long dashes, and I stripped them out by hand every time. Two minutes an article, and it annoyed me for considerably longer than two minutes. The fix was one line in the voice file: no em dashes, use a comma or a period. I haven't removed one by hand since. That's the whole discipline in miniature. The correction I kept making in conversation bought me nothing. The same correction, written down once, bought me every draft after it.

Next, the piece that makes the whole argument work: support that doesn't grow with your customer count.

Support Without Headcount

This is the chapter the rest of the guide rests on. If support still scales linearly with customers, the old advice was right and you should go build a course.

It doesn't. But the way it stops scaling is more specific than "point an AI at your inbox," and the difference between doing this well and badly is the difference between a business and a slow-motion reputation fire.

Three tiers, not one switch

The mistake is treating support as a single thing to automate. Sort it into three piles first.

Deflected. The question that's answered in your docs, asked for the hundredth time. Nobody needs to write this reply, including a machine. Good search, a decent FAQ, and an answer surfaced at the moment of confusion handles most of it. The cheapest support ticket is the one that never opens.

Drafted. The real question with a knowable answer. Something specific about their account, their setup, their error. An agent with your docs, your product's actual behavior, and this customer's history writes a reply that's usually right. You read it and send it. This tier is where most of the volume lives and where most of the win is.

Escalated. Anything about money, anything about access, anything from someone who is upset, anything the agent isn't confident about. These come to you, untouched. No draft, no suggestion, just a flag saying this one is yours.

The proportions shift as you tune it, but the shape holds. The thing that broke linear scaling is that the middle tier, which used to be all of your time, is now mostly review.

What never goes out unread

Be strict about this list, because every item on it is a way to lose a customer permanently:

- Anything that touches billing, refunds, or what someone is charged
- Anything that grants, changes, or revokes access
- Anything making a promise about a future feature or a date
- Anything to someone who is already angry
- Anything where being wrong is expensive to undo
- Anything the agent flagged as uncertain, which you should make it easy for it to do

The rule underneath: **if getting it wrong costs more than the two minutes you'd spend reading it, read it.** That's nearly always the right trade for a one-person business, where a single badly handled refund can produce a review that outlives the customer.

Deflection is a product signal

Here's the part most people miss. Your support queue is the highest-quality product feedback you will ever get, and automating it badly means you stop reading it.

Wire the loop the other way. Every question that gets asked twice is a documentation gap. Every question that gets asked ten times is a product gap. Have something watch for the repeats and tell you weekly, in plain language: here is what people kept asking about, ranked.

That report is worth more than the time saved. It's a list of exactly where your product confuses people, generated from real behavior, and before agents almost nobody in a one-person business had the hours to

compile it.

What to measure

Two numbers people track, and one they should.

Deflection rate tells you how many questions never reached you. Useful, and easy to game by making it hard to contact you, which is why it can't be the only one.

Escalation precision tells you whether the things that came to you actually needed to. If you're rubber-stamping most escalations, tighten the rules. If you're frequently rescuing sent replies, loosen them.

The one that matters: are people getting their problem solved on the first reply? A fleet that halves your support time while doubling the number of back-and-forths has made your business worse and your dashboard better.

How it fails

One failure mode dominates, and it's worth naming precisely: **a wall of cheerful unhelpfulness.**

Every reply arrives fast, reads politely, and doesn't quite answer the question. The customer writes back. The next reply is also fast, polite, and slightly off. Nothing is technically broken and your metrics look good, and what you've actually built is the thing everybody hates about big-company support, at a scale of one.

It happens when the agent lacks context to answer properly and hasn't been given permission to say so. The fix is to make "I don't know, a human will pick this up shortly" a first-class, unpenalized outcome. An honest escalation beats a confident near-miss every time, and customers can tell the difference immediately.

The second failure is quieter. You stop reading. The fleet handles it, the queue is calm, and three months later you have no idea what your customers are struggling with. Read a sample every week even when nothing is wrong. Especially then.

Next, the same shift applied to the other thing that used to eat all your hours: making and distributing the work.

Content and Marketing at Throughput

The first thing most people do with a fleet is make more content. It's the obvious move and mostly the wrong one.

Publishing got cheap for everyone at the same moment. That means the marginal value of one more competent post fell to roughly nothing, because your reader can generate an equivalent one on demand without visiting your site. Ten times the output into a market where output is free is not a strategy. It's a way to dilute whatever attention you'd already earned.

The leverage is real. It's just somewhere other than volume.

Where it actually helps

Research and synthesis. Reading everything in a space and telling you what's actually being argued about, what nobody has written yet, which claim everyone repeats without checking. This is genuinely hours of work and it's the input to good writing rather than a substitute for it.

The mechanical half of publishing. Formatting, cross-linking, alt text, metadata, checking that the code in the post runs, finding the three older pieces this one should link to. Unglamorous, endless, and exactly the sort of thing that used to make publishing feel expensive.

Repurposing across formats. One real piece becoming the newsletter, the thread, the talk outline. The thinking happened once. The reshaping is mechanical.

Distribution follow-through. Knowing what got traction, what died, what someone asked in a comment that deserves its own post. The part everyone intends to do and nobody does.

Notice the pattern. All of that is the work *around* the idea. None of it is the idea.

What stays yours

The take. The specific example. The number you measured. The thing you got wrong.

That sounds like a values statement and it's actually a market observation: those are the only parts a reader can't get elsewhere for free. Everything else in your post is now commodity, which means everything else is packaging.

So the practical rule I'd apply: **an agent can touch any part of a piece except the part that makes it worth reading.** If you can't identify which part that is, you don't have a piece yet, and no amount of drafting help will produce one.

The one thing you should never automate

Publishing under your name without reading it.

This sounds too obvious to state until you're eight weeks into a content system that works, the drafts are decent, and it starts feeling reasonable to let a good one through unread. Then one goes out with a confident claim you'd never make, or a tone that isn't yours, or a factual error about something you're supposed to be the expert in.

The cost of that is not the one post. It's that every other post you've written becomes suspect to anyone who noticed.

What to measure

Output metrics are actively misleading here. Posts per week, words published, pieces shipped: all of these go up the moment you add a fleet, and none of them tell you whether the business got better.

Three that mean something:

- **Did anyone come back?** Returning readers over new ones. Volume buys new. Only quality buys returning.
- **Did it produce a conversation?** Replies, questions, someone quoting it at you six months later. This is the leading indicator of the thing that actually pays.
- **Did it bring work in?** For a solo business, content is usually the top of a funnel that ends in a conversation with a human. If that number doesn't move, the content isn't doing its job regardless of how much of it there is.

The search shift

Worth naming because it changes the calculus. A growing share of people asking questions get an answer without visiting anyone's site, assembled from sources they never see.

You can't fight that, and writing more mediocre pages aimed at it is the worst possible response. The thing that survives is being the primary source: the original measurement, the firsthand account, the opinion with a name on it. Those get cited, quoted, and sought out directly, because they can't be synthesized from anything else.

Which lands in the same place as the rest of this chapter. Whatever is genuinely yours is the whole asset now, and the machinery around it is the part worth handing off.

Next, what happens when the content works and someone actually wants to buy something.

Selling as One Person

There is no sales team. There is you, and whatever system you built, and a person deciding whether to trust you with money.

That last part is worth sitting with, because it's the constraint that shapes everything else in this chapter. People buying from a one-person business are not buying a product with a company behind it. They are buying you, with the specific risk that you might get bored, get hired, get ill, or vanish. Every part of your sales process is either addressing that or ignoring it.

Inbound beats outbound, by a lot

Cold outreach works. It also costs an enormous amount of the one thing you don't have, and it puts you in the worst possible negotiating position, which is wanting the deal more than they do.

Inbound inverts that. Someone who found you through something you wrote arrives already half-convinced, already knowing how you think, and already self-selected for the kind of work you want. The conversation starts at "can you help with this" rather than "who are you."

This is the actual return on the writing, and it takes far longer than anyone tells you. The piece that brings in work is often eighteen months old. That lag is why most people quit writing right before it starts working.

What the fleet does here

Research before the call. Who they are, what they've shipped, what they've said publicly, what their stack looks like, what's probably going wrong given all of that. Fifteen minutes of preparation that used to be forty, and it shows in the first two minutes of the conversation.

Drafting the boring documents. Proposals, statements of work, follow-up summaries. The structure is the same every time and the specifics come from your notes. First draft in a minute, edited by you in ten.

Follow-up that actually happens. The single biggest leak in solo sales is the conversation that went well and then nothing happened for three weeks because you got busy. Something that tracks open threads and tells you what's gone quiet is worth more than any clever tactic.

The record. What was agreed, what was promised, what the price was, what they said their real problem was. Written down without you doing it.

What it doesn't do

The call. The read on whether this is a client you want. The negotiation. The moment where you say the uncomfortable true thing about their situation.

That last one is the entire job. The reason someone hires an experienced person is to be told something they don't want to hear by someone with no incentive to soften it. Nothing you can automate does that, and a proposal generated without it is a document, not a sale.

Saying no is the strategy

For a solo business, the client you decline is as load-bearing as the one you take.

You have no bench. A bad engagement doesn't cost you margin, it costs you the entire quarter and usually

your appetite for the work. The three that reliably go wrong: the one who wants a full-time employee at contractor rates, the one who won't say what success looks like, and the one who was rude to you before any money changed hands.

The last is the most reliable signal in the whole business, and the easiest to talk yourself out of when the pipeline is thin.

Address the bus problem directly

Every serious buyer is quietly wondering what happens if you disappear. Most won't ask.

Answer it before they do. What's documented, what's in their repo rather than your head, what the handover looks like, who they'd call. For fractional and consulting work I'd put it in writing early, because the client who was worried and didn't ask is the client who quietly picks the agency instead.

This is also just good practice. A business where everything lives in your head is one you can never step away from, which defeats the purpose of building it alone in the first place.

Next, the boundary this whole guide has been circling: what you must never hand over.

What You Don't Delegate

Every chapter so far has been about what to hand over. This one is the boundary, and it matters more than any of them, because the failures in this model are almost never "the agent couldn't do it." They're "the agent did it and shouldn't have."

Five things stay yours. I'd treat this as close to non-negotiable.

Deciding what to build

The most expensive mistake available to you is building the wrong thing efficiently.

A fleet makes execution cheap, which quietly raises the cost of a bad decision, because you'll get further into it before the wrongness surfaces. The judgment about what deserves to exist comes from talking to people, noticing what they do rather than what they say, and having a view about where things are heading. None of that is available to something that has only read about your market.

The quality bar

Taste doesn't survive delegation, and it degrades in a specific way: everything becomes fine.

Fine is the natural output of a system optimizing for plausible. Fine is also indistinguishable from good on a first read, which is why it slips through, and why a business can spend six months getting worse without a single visible failure. You are the only thing standing between your work and a slow drift to the median, because the median is exactly what a model was trained to produce.

The practical version: read your own output cold, a week later, and ask whether you'd have been proud to publish it as a first attempt. If the honest answer is "it's fine," something's wrong.

Relationships

The conversation with the unhappy customer. The call where a client tells you the real problem forty minutes in. The peer who sends you work because you helped them once with nothing in it for you.

For a one-person business this is the only durable advantage you have. Products get copied, content gets summarized, prices get undercut. People who trust you specifically do not transfer.

Automating the appearance of a relationship is worse than not having one. Everyone can now tell the difference between a message written for them and a message generated at them, and getting that wrong costs you the relationship you were trying to maintain.

Responsibility

You cannot delegate accountability to something that cannot be held accountable.

When an agent sends the wrong invoice, deletes the wrong record, or promises a customer something you can't deliver, the sentence "the AI did it" does not exist as a defence. Not legally, not commercially, and not in the mind of the person you let down. It's your business, your name, and your problem.

This is an operational point. It means every irreversible action needs either your hand on it or a guardrail you designed deliberately, which is the next chapter.

Enough of the work to stay sharp

The one people skip.

If you delegate all of the doing, your judgment about the doing decays. Not immediately. Over a year or two, quietly, until you're reviewing work you can no longer evaluate properly and calling it oversight. At that point you've become a manager of systems you don't understand, which is a worse job than the one you were trying to escape, and a much more fragile one.

Keep building things yourself. Not everything, and not to prove a point. Enough that when something looks subtly wrong, you know it, and you can say why.

The test

When you're unsure which side of the line something falls on, one question sorts most cases:

If this goes wrong, will I be able to explain how it happened without embarrassment?

"I reviewed it and misjudged it" is a survivable answer. "I don't know, it just went out" is not, and if that's the honest answer for a given category of work, that category needed you.

Next, the machinery that makes the boundary real rather than aspirational.

Guardrails That Hold

The previous chapter drew a line. This one is about making that line hold when you're tired, busy, and not watching.

A guardrail you have to remember to apply is a preference. A guardrail the system enforces is a control. For a business with one person in it and nobody to catch your mistakes, the difference decides whether the whole model is safe to run.

Gate on blast radius, not on difficulty

The instinct is to supervise the hard things. Wrong axis. Supervise the *irreversible* things.

A complicated report that comes out wrong costs you an hour. A simple, easy, one-line action that emails four hundred customers cannot be recalled at any price. Sort your fleet's capabilities by what happens if this fires incorrectly, and put the gates there.

The list that should require your hand, for almost any business:

- Anything that sends to a customer list
- Anything that moves money, changes a price, or issues a refund
- Anything that deletes, and anything that changes access
- Anything that publishes under your name
- Anything that makes a commitment on your behalf

Everything else can usually run, provided you can see what it did.

Secrets never go in the context

The most common serious mistake I see is putting an API key, a password, or a token into a prompt or a context file because it was the fastest way to make something work.

Those files get read by every agent, get copied, get pasted into a session you'll later share, and persist in conversation histories you've forgotten about. Credentials belong in a secret store your code reads at runtime, never in the text you hand a model. If a secret has already gone into one, rotate it. Not later.

Scope every credential to the minimum it needs. An agent that only reads your support inbox should hold a token that can only read your support inbox, so that the worst case is bounded by design rather than by hope.

Customer data is not yours to be casual with

You are one person, which does not reduce your obligations to the people who gave you their data.

Know what you hold and why. Know which of your agents can see it and which third parties it reaches when they do. Be able to answer "what happens to my data" without improvising, because eventually someone will ask, and increasingly it'll be in a procurement questionnaire attached to the largest deal you've been offered.

I'm not going to give you legal advice, and anyone who does in a guide like this is selling something. What I will say is that "I was small and moving fast" has never once worked as a defence, and the regimes that apply to you don't scale their expectations to your headcount.

The stop button

Build the thing that halts everything before you build anything clever.

One switch that stops every agent, and per-agent switches for the individual ones. It sounds paranoid until the morning you find something has been quietly doing the wrong thing since 3am, and the difference between a bad hour and a bad week is whether you could stop it in ten seconds or had to work out how.

The same applies to rate limits. Cap what any agent can do per hour: how many messages, how many writes, how much spend. Not because you expect a runaway, but because the cost of the cap is zero and the cost of not having one is unbounded.

You need to be able to reconstruct what happened

When something goes wrong, the question is always the same: what did it see, what did it decide, what did it do?

If you can't answer that, you can't fix the cause, and you'll end up patching symptoms while the underlying problem stays live. Log the inputs, the actions, and the outputs, in something you can search. The longer version of this is in Loop Engineering (https://rogerstringer.com/guides/loop-engineering) and Running the Fleet (https://rogerstringer.com/guides/running-the-fleet).

The weekly ten minutes

Guardrails decay because businesses change. New agent, new integration, a permission widened during a busy week and never narrowed.

Once a week, briefly: what can each agent currently reach, what did it do that surprised you, is anything running that you'd forgotten about. Ten minutes, and it's the difference between a fleet you operate and a fleet that operates you.

Next, the economics, which change in ways most people get exactly backwards.

Pricing When Delivery Is Nearly Free

Here is the mistake, and it's close to universal: your costs fell, so you lower your prices.

It feels honest. It is also the fastest way to convert a real advantage into nothing at all, and once you've done it you can't undo it with existing customers.

Nobody is buying your hours

Cost-plus pricing was always a bad idea and it's now incoherent. If delivery costs you almost nothing, cost-plus prices your work at almost nothing, which is obviously wrong for work that solves an expensive problem.

The buyer never cared about your costs. They care about the gap between what the problem costs them and what you charge to remove it. That gap is where your price lives, and it didn't move when your costs did.

Two clients, same deliverable. One where it saves a weekend of annoyance, one where it unblocks a launch that's burning money every day it slips. Same work, and the same price for both is leaving an enormous amount on the table in one case and being unaffordable in the other.

"Cheaper because AI" is a trap

Say it out loud and listen to what the buyer hears: *this is a commodity, produced by a machine, and its main virtue is being cheap.*

You've told them the work is undifferentiated, invited them to compare you on price alone, and set an anchor that everyone else in your market can undercut next quarter when their costs fall too. That's a race with no finish line and no winner who's a one-person business.

The people doing well are quiet about their costs and specific about their outcomes.

Where the gains should go

The efficiency is real. You have three places to put it, and the choice is strategic.

Lower prices. Buys volume, which is the one thing a solo business is worst at absorbing, and permanently resets what customers expect to pay. Rarely right.

Wider scope at the same price. Deliver more than you used to for the same number. Feels generous, and quietly becomes the new baseline within two engagements. Use deliberately, if at all.

Keep it. Same price, less of your time, more margin and more room. This is usually the correct answer and the one people feel guilty about, so let me be blunt: the customer is buying an outcome at a price they agreed was fair. How efficiently you produce it is your business, exactly as it was when you were slower.

The gains are what buy you the thing you started this for. Spending them on being cheaper buys you more work.

What actually justifies a higher price now

Since output is cheap and getting cheaper, price on the things that aren't:

Judgment. Knowing which of five plausible options is right here, and being willing to say so. Nobody can generate that about your customer's specific situation.

Accountability. There is a name attached and someone to call. Increasingly rare and increasingly valuable as more of what people buy is machine-produced by parties who won't stand behind it.

Speed to a decision. Not speed of typing. The compression from "we have a vague problem" to "here's what we're doing and why," which used to take weeks of meetings.

Being right the first time. Worth more than it ever was, because everyone now has infinite access to plausible-looking wrong answers, and the cost of following one has gone up.

On subscriptions

The old argument for recurring revenue was partly that ongoing support cost you ongoing money, so it needed ongoing payment. That justification weakened.

Recurring revenue is still excellent for a solo business, for a different and better reason: it makes income predictable, and predictable income is what lets you turn down bad work. That's the actual product of pricing well. Not the number, the ability to say no.

Just don't charge subscription for something that isn't continuously delivering. People notice, and the churn arrives quietly, six months later, in a batch.

One concrete suggestion

Raise your prices on the next new customer, not the current ones. Not a rounding adjustment, a real step.

You'll find out quickly whether the market agrees with you, and you'll find out at zero risk, because your existing revenue is untouched. Nearly everyone I'd point to who runs a good one-person business is charging more than they were two years ago, and none of them regret it.

Next, actually putting something in front of people.

Launching

Launching used to be an event because it had to be. You got one shot at attention, so you saved everything up, made a lot of noise on one day, and hoped.

That model was built for scarcity you no longer have. You can ship continuously, respond within hours, and rewrite the whole thing in an afternoon if the first version misses. The launch stopped being a moment and became a process, which is better in every way except that it's less exciting.

Launch to people who already know you

The single biggest predictor of whether a launch works is whether anyone was waiting for it.

Almost every failed solo launch I'd point at has the same shape: someone built for months in private, appeared on launch day, and discovered that shipping into silence produces silence. The product was often fine. Nobody knew it existed and nobody had reason to care.

The fix is unglamorous and slow. Talk about the problem publicly while you build. Show the thing half-finished. Collect the people who react. By the time you launch you want a list of people who have already told you they have this problem, and the launch is just you telling them it's ready.

If you're reading this with nothing built and no audience, that ordering matters more than anything else in this chapter.

Ship it embarrassingly early to a small number of people

Not publicly. To five or ten people who have the problem and will tell you the truth.

What you're testing isn't whether they like it. People are polite. You're testing whether they use it twice without being asked, and whether they get stuck in the place you expected or somewhere else entirely. Watching one person use the thing you built is worth more than a hundred survey responses, and it's uncomfortable enough that most people skip it.

Fix what that surfaces. Then do it again with the next ten.

What the fleet is good for here

Launch weeks are mostly logistics, and logistics is exactly the shape of work worth handing over.

The assets in every format. The FAQ assembled from questions the early group actually asked rather than the ones you imagined. Monitoring for whether anything is broken, which matters more on launch day than any other day. Draft replies to the wave of similar questions that arrives in the first forty-eight hours.

What it can't do is have the conversation with the first person who says this doesn't work for me. That one is worth your full attention, because early on, one detailed complaint is a larger sample than all your analytics.

The day is not the thing

Expect a spike, then near-silence, and don't read the silence as failure. Almost nothing worth building gets its customers on day one. The pattern is a small launch bump followed by a long flat stretch where the only thing that changes anything is continuing to show up.

The businesses that work are the ones still publishing, still improving, still talking to customers in month nine. That's the actual mechanism, and it's why the guide spent so long on the machinery that makes month

nine sustainable for one person.

The relaunch is allowed

You can launch the same thing again. Nobody is keeping score, and the overwhelming majority of your potential customers missed the first one entirely.

New angle, new audience, a version that fixed the complaint, a case study from someone using it. Treating the launch as a repeatable act rather than a single irreversible event removes most of the pressure that makes people delay shipping for months.

Which is the real point. The reason to stop treating launches as events is that the event framing is what keeps unfinished things in private repositories forever.

Next, the ways this whole model goes wrong.

How This Fails

I've been making the case for this model for twelve chapters. Here is the other side, honestly, because the failure modes are specific and mostly invisible while they're happening.

Drift

The commonest one by a distance.

The fleet works. The queue is calm. Output arrives, competent and on time, and you gradually stop reading it closely because there's never anything wrong. Then one day you look properly and discover that quality has been sliding for two months, that replies have been subtly missing the point, that the tone stopped being yours somewhere around March.

Nothing broke. That's what makes it dangerous. A crash gets fixed within the hour. A slow decline gets fixed when a customer finally mentions it, which is long after several others quietly left.

The only defence is a scheduled read of raw output when nothing is wrong. Put it in the calendar, because you will never feel like doing it.

Distance from the customer

Related, worse, and harder to reverse.

Support gets handled. Feedback arrives summarized. Research comes as a digest. Every individual step is an improvement, and the aggregate effect is that you haven't heard a customer's actual voice in a quarter.

You lose the thing summaries can't carry: the hesitation, the frustration, what they tried before giving up, the workaround they invented that tells you exactly what your product is missing. Product instinct is built from that raw material and it starves without it.

Read some unfiltered feedback every week. Take a call you could have avoided. This is a deliberate inefficiency and it's worth protecting.

Volume mistaken for progress

You can now produce ten times the output. It's very easy to believe that means ten times the business.

More posts, more features, more everything, and the metrics all point up. Meanwhile nobody is returning, nothing is compounding, and you've spent six months busy. Volume is the easiest thing to increase and the least correlated with anything that matters.

If your output tripled and your revenue and returning-customer numbers didn't move, that's not a reason to publish more.

Over-automation

Handing over something that needed you, and finding out weeks later.

It's almost never a dramatic failure. It's a pricing question answered slightly wrong, a commitment made you wouldn't have made, a customer handled correctly by policy and badly by judgment. Each one small. Collectively they're the difference between a business people recommend and one they merely use.

The test from chapter nine holds: if it goes wrong, could you explain how without embarrassment?

Skill decay

The slowest and the most expensive.

Delegate all the doing and your ability to evaluate the doing erodes. It takes a year or two and you won't notice, because reviewing feels like competence. Then something subtly wrong goes past you, and then something else, and eventually you're the person nodding at work you can no longer really assess.

For a one-person business this is close to fatal, because your judgment *is* the product. Keep your hands in the work.

The isolation, which nobody automates away

The thing this genre skips. Working alone is genuinely hard, and a fleet makes it more so, because the collaborators who used to be a reason to talk to people are now processes.

You can go a full week without a professional conversation and feel productive throughout. That works for a while and then it doesn't. Peers, a group, regular calls with people doing similar work: this is infrastructure, not a nice-to-have, and it's the first thing to disappear when you get busy.

The honest summary

Every failure here is a version of the same thing. The fleet multiplies whatever attention you bring. Bring less and it multiplies less, quietly, while everything looks fine.

The model works if you stay in it. It fails, slowly and without alarms, if you use it to leave.

Next, what I'd actually do starting from nothing today.

What I Would Do Now

Starting from nothing today, with a year or two of runway and everything I currently know: agents all the way.

Not as a product to sell. As the way the business is built, from the first week, in every part of it. That's a different answer from the one I'd have given three years ago and it's worth being precise about why.

Why not the safe answer

The conventional move is content. It's still defensible: nothing to run, nothing to break, no support burden, publish once and it works while you sleep.

I wouldn't, and the reason is the one from chapter four. Competent content became free at the exact moment everyone got a machine that produces it. Building a business on being reliably useful in writing means competing with infinite free reliably-useful writing. What survives is the part that comes from doing the work, which means the writing has to be downstream of something real. It works as an output. It's a thin thing to make the whole business.

The old argument against software was support scaling with customers. That was the correct read for twenty years and it isn't now, which reopens the option that used to be closed.

What I'd actually build

Something narrow, in a domain I already know cold, for people I can name.

Narrow because a solo business cannot out-feature anyone and shouldn't try. The advantage available to one person is being unreasonably good at a specific thing for a specific kind of customer, in a way that's uneconomic for a larger company to bother with.

In a domain I know because a fleet accelerates building and does nothing for knowing what to build. Starting in an unfamiliar market with fast execution is the most efficient way to build the wrong thing I can think of.

For people I can name because the alternative is guessing, and I'd rather have ten real conversations before writing anything than three months of building for an imagined user.

The order I'd do it in

Fund it first. Consulting, fractional work, whatever pays. Not as a failure state, as the thing that buys the runway and keeps me inside real problems. The mistake is treating this as temporary and then never building anything; the schedule has to be deliberate.

Write from week one. Publicly, about the actual work. Not for an audience I don't have yet, but because it's the only reliable way to be findable in eighteen months, and the lag means starting late is expensive.

Build the context layer before the fleet. The files from chapter five. An afternoon, and it's the difference between agents that help and agents that produce confident noise.

One agent at a time. Support first, because it's the one that changes the economics. Then the long tail of small work. Nothing clever until the simple things are running reliably and I'm reading their output.

Charge properly from the first customer. Prices are much harder to raise than to set. And the point of the margin is the ability to refuse work, which is the actual freedom being purchased here.

What I'd refuse to do

Take on anything where an automated mistake costs a customer money or data before I have the guardrails from chapter ten in place.

Hire. Not out of principle, but because the entire premise is that the ceiling moved far enough that one person plus a fleet covers the ground that used to need a small team, and adding people re-imports every coordination cost I'm trying to avoid.

Chase volume. In content, in features, in customers. The failure mode in chapter thirteen is real and it's seductive precisely because it feels like progress.

The part I'm least certain about

Being honest: I don't know how long the current advantage lasts.

Everything in this guide rests on a gap between what's possible and what most people have adopted. Gaps close. What's differentiating now is table stakes in three years, and the businesses that will still be standing are the ones whose advantage was never really the tooling.

Which points back at the same place. Build something where the durable part is your judgment, your relationships, and your specific knowledge, and use the fleet to remove everything standing between you and doing more of that. That advantage doesn't close.

That's the bet I'm running now, and it isn't hypothetical. The consulting funds it and keeps me inside real systems. The writing is what makes the consulting inbound. The fleet has taken over the middle: the first drafts, the small fixes, the reading I would otherwise never get to. What comes back is the only thing I wanted out of any of this, which is more hours pointed at the work that has to be mine.

If you take one thing from this guide, take the order. Fund it. Narrow it. Write the files down. Add one agent at a time and read what it does. Guard the parts that are yours. The tools will keep changing under you, and every version of them so far has rewarded that order.

Toolkit: The Context Layer

Chapter five said build the context layer before the fleet, and then didn't hand you the files. Here they are.

Drop these in a folder in your business repo. Every agent reads all of them. Keep them short enough that you'll actually maintain them, which in practice means one page each at most.

```
# business.md

## What this is
[Two sentences. What you sell, to whom, what problem it removes.]

## What this explicitly is not
[The single most useful section here. What you don't sell, who you
don't serve, the adjacent thing people keep asking for that you have
decided not to do. Prevents an enormous amount of wrong output.]

## How we make money
[Offers and prices. What's negotiable and what isn't.]

## Where things live
[Repos, docs, dashboards, inbox, CRM. Just enough to orient.]
```

```
# customer.md

## Who they actually are
[Their situation, not a persona. What's on fire for them.]

## What they always ask
- [The three questions that come up every single time]

## What they're really worried about
[Usually different from what they say. This is the one that makes
replies land.]

## Words they use
[Their vocabulary, not yours. If they say "reporting" and you say
"analytics", write that down.]
```

```
# voice.md

## A paragraph that landed
[Paste one. Real example beats any adjective.]

## A paragraph I'd never send
[Paste one. Say why in a line.]

## Never
- The long dash. Use a comma, colon, period, or parentheses.
- [Your other tics. Be specific and be petty about it.]

## Always
- Contractions. Short sentences mixed with long ones.
- Say the uncomfortable true thing rather than hedging it.
```

```
# boundaries.md

Never send without me reading it:
- Anything touching billing, refunds, or pricing
- Anything granting or changing access
- Anything promising a feature or a date
- Anything to someone who is already unhappy
- Anything where being wrong is expensive to undo

Never do at all:
- [Your list. Be concrete.]

Always escalate rather than guess. "A human will pick this up shortly"
is a correct answer and is never penalised.
```

```
# done.md

A support reply is done when it answers the question and they don't
have to write back.

A draft is done when it has a point, one specific example, and nothing
that could be cut without loss.

A code change is done when tests pass, the diff is small enough to
review in one sitting, and nothing was refactored that wasn't asked for.

[Add one per kind of work you delegate.]
```

The maintenance rule

One habit keeps this alive: **the second time you give the same piece of feedback, stop giving feedback and add a line to a file.**

That single rule is what turns the context layer from a document you wrote once into something that compounds. Everything an agent got wrong becomes a permanent correction rather than a recurring annoyance.

Read all five end to end once a month. It takes ten minutes and it's the cheapest quality control in the business.

Toolkit: The Delegation Line

Chapter nine argued about where the line sits. This is the version you can actually run a task through in ten seconds.

Use it when you're about to hand something over and aren't sure, and when you're auditing a fleet that grew without anyone deciding it should.

```
# The delegation line

Run the task through in order. First "no" stops it.

## 1. Reversibility
- [ ] If this fires wrongly, can I undo it within the hour?
      NO  -> gate it. Your hand on the button, every time.

## 2. Blast radius
- [ ] Does it touch money, access, customer data, or my public name?
      YES -> gate it, regardless of how simple the task looks.

## 3. Verifiability
- [ ] Can I tell it went wrong without manually inspecting it?
      NO  -> either build the check first, or keep the task.

## 4. Judgment content
- [ ] Does doing this well require knowing something not written down?
      YES -> keep it, or write the thing down and try again.

## 5. The embarrassment test
- [ ] If this goes wrong, can I explain how without embarrassment?
      NO  -> keep it.

## 6. Skill
- [ ] Is this one of the few things my judgment is actually built on?
      YES -> keep some of it, permanently, even though delegating
             would work fine. Especially because it would work fine.

Cleared all six -> delegate it, log what it does, read the log weekly.
```

The three-pile sort

For a whole business rather than one task, sort everything you do into three piles once and revisit quarterly.

Yours forever. Deciding what to build. The quality bar. Relationships. Anything irreversible. Enough hands-on work to keep your judgment honest.

Yours to review. Drafted, not sent. Most support, most first drafts, most small engineering work. This pile should be the biggest and it's where nearly all the leverage lives.

Not yours at all. Genuinely mechanical, verifiable, reversible. Formatting, repurposing, reconciliation, monitoring, research gathering.

The mistake that hurts is items migrating from the second pile to the third without a decision being made. It happens by drift: the reviews are always fine, so you stop reviewing, and nobody ever decided anything. Quarterly, ask which items moved and whether you meant them to.

The signals you've got it wrong

Too much delegated: you can't explain why a decision was made. Output is uniformly fine and never surprising. You haven't heard a customer's actual words in a month. You're reviewing work you couldn't produce yourself anymore.

Too little delegated: you're doing things a checklist could do. Your week is full and nothing compounded. You keep not doing the important thing because of the urgent ones.

Most people reading this are wrong in the second direction first and the first direction eighteen months later.

Toolkit: The First 90 Days

Ninety days from nothing to a business with a fleet under it. Not to revenue, necessarily. To the point where the machine exists and you know whether the idea has legs.

The order matters more than the dates. Funding before building, context before agents, customers before features.

```
# The first 90 days

## Week 0: decide
- [ ] Name the domain you already know cold
- [ ] Name ten real people who have the problem. Actual names.
- [ ] Book conversations with three of them. Ask what they do now,
      not whether they'd buy.
- [ ] Write the "what this is not" paragraph before anything else

## Days 1-30: fund it and be findable
- [ ] Line up the consulting or contract work that pays the runway
- [ ] Decide the split now: days on client work, days on your thing.
      Put it in the calendar or it won't happen.
- [ ] Publish something real about the actual work. Week one.
- [ ] Keep publishing weekly. The lag is 12-18 months, so the cost
      of starting late compounds.
- [ ] Build the context layer (business, customer, voice, boundaries,
      done). One afternoon.

## Days 31-60: first agent, first version
- [ ] Stand up ONE agent. Support if you have customers, research if
      you don't.
- [ ] Read everything it produces. All of it, for two weeks.
- [ ] Every mistake becomes a line in a context file, not a correction
      in a chat
- [ ] Ship the smallest useful version of the thing to 5-10 people
- [ ] Watch one of them use it. In person or on a call. Uncomfortable
      and worth more than any survey.
- [ ] Set your price. Higher than feels comfortable.

## Days 61-90: second agent, real customers
- [ ] Add the second agent only if you're still reading the first one
- [ ] Guardrails before anything touches money, access, or your name:
      gates on irreversible actions, secrets out of context, a stop
      button, rate caps, a searchable log
- [ ] Charge the first real customer
- [ ] Weekly: read raw customer words, unfiltered by any summary
- [ ] Day 90: decide. Evidence of pull, or a clean pivot. Both are
      fine. Drifting is not.
```

What "evidence of pull" means

Be strict, because this is where people lie to themselves.

Not: people said it was interesting. Not: signups from a launch. Not: your content did well.

Pull is someone using it twice without being prompted. Someone asking when the next thing ships. Someone paying. Someone telling a colleague. If you have none of those at day 90, you have a project, and the honest move is to change something substantial rather than add features.

The three ways this schedule breaks

The client work eats everything. It pays now and your own thing pays later, so it wins every scheduling conflict until there's no schedule left. The fix is calendar blocks you treat as immovable, which sounds weak and is the only thing that works.

You build for sixty days without talking to anyone. The most common and most expensive. Everything

in weeks 0 and 31-60 exists to prevent it.

You add agents faster than you read them. Five running by day 45, none of them audited. You've built a machine that produces confident output nobody is checking, which is worse than having no machine at all.

After 90 days

If there's pull, the next ninety are about narrowing rather than expanding: fewer customer types, sharper positioning, higher prices, and the third agent only if the first two still get read.

If there isn't, the domain knowledge and the context layer and the audience all transfer. Almost nothing is wasted except the assumption about what people wanted, which was the cheapest thing in the pile.

About Roger

I'm Roger Stringer. I build things, break them, and write up what I learned so you don't have to learn it the hard way. These Field Guides come straight out of that work.

Working on something bigger? I take on a handful of fractional CTO (https://rogerstringer.com/guides/the-fractional-cto-field-guide) engagements, helping founders and teams set technical direction, build AI-powered workflows, and actually ship the hard parts. If you're wrestling with the kind of problem this guide covers and want someone in your corner who's done it before, that's exactly what I help with. Drop me a line. (mailto:roger.stringer@hey.com)

And if a guide helped, got something wrong, or you just want to compare notes, I'd love to hear from you:

- **Email**: roger.stringer@hey.com (mailto:roger.stringer@hey.com)
- **X**: @freekrai (https://x.com/freekrai)
- **GitHub**: github.com/freekrai (https://github.com/freekrai)
- **LinkedIn**: linkedin.com/in/rogerstringer (https://www.linkedin.com/in/rogerstringer)

New guides go up as I hit problems worth documenting. Follow along wherever suits you.