THE AI ADVANTAGE FOR HOME SERVICE BUSINESSES

How Smart Business Owners Are Using AI and Intelligent Automation to Increase Profits, Improve Operations, and Outperform the Competition.

INCREASE PROFITS | SAVE TIME & REDUCE COSTS | DELIGHT CUSTOMERS | IMPROVE OPERATIONS | OUTPERFORM THE COMPETITION

PRACTICAL STRATEGIES. REAL RESULTS. A SMARTER FUTURE.

JOHN HODSDON

The AI Advantage for Home Service Businesses

How to Win with AI and Agentic Systems
A Step-by-Step Guide for Business Owners

John H

2026

© 2026 John H. All rights reserved.

No part of this publication may be reproduced, distributed, or transmitted
in any form or by any means without prior written permission.

This guide is educational material and does not constitute
legal, insurance, or professional advice.

First Edition · 2026

Table of Contents

Part III — ends with you holding:

Part IV — — Put It In Motion

Part IV — ends with you holding:

Part V — — Keep Getting Better

Part V — ends with you holding:

Introduction

This guide is for you — the owner of a home service business. Maybe you run a remodeling shop. Maybe HVAC, plumbing, electrical, roofing, landscaping, painting, or restoration. Regardless of the trade, the problems are the same: things fall through the cracks, admin time eats into the field time that actually makes money, and every tool you've been offered was built for *any business* instead of *your* business.

We're going to walk through **how to make your business measurably better** — less waste, tighter margins, faster response, happier customers. And we're going to do it using **AI agents** as the engine that actually runs the improvements you design. Not because agents are cool tech, but because they're cheap, fast, and they get better every time they run.

Quick note on the example. Throughout this guide, I'm going to use a kitchen-and-bath remodeling shop as the running example, because it's the industry I know best and it illustrates most of the patterns. If you're in HVAC, your "proposal approval cycle" is different from a remodeler's in the details but identical in the structure. If you're a plumber, your "emergency call" is different from a remodeler's "change order request" — but both are places where jobs stall and money leaks. The patterns transfer. Where an example is specific to remodeling, I'll say so.

By the end of this guide you will:

1. Know exactly where your business is losing money and time
2. Have a clear picture of what "fixed" looks like for each problem
3. Have built and deployed 1–3 real improvements that are already helping
4. Have a system that keeps learning and getting sharper

You'll end up with an AI agent team — but that's the **result** of good thinking, not the starting point.

The Core Thesis

"Your edge isn't technology. Your edge is that you know how the business actually works."

For decades, if a home service owner wanted to improve their shop, they had three bad options:

- Buy expensive software built by people who'd never run a job site
- Hire a consultant at $200/hr who'd leave when the budget ran out
- Give up and keep doing things the way they'd always done them

None of those options worked well.

Today, there's a fourth option: **you describe what "good" looks like, and an AI agent builds it for you.**

That's not a hype statement — it's genuinely how it works now. Code is cheap. What's expensive, and what nobody can give you, is your 20 years of knowing the real workflows, the actual bottlenecks, the domain rules that no textbook captures. You know why the Elm Street change order needs a specific approval step, or why an HVAC lead that comes in at 3pm on a Friday gets a different reply template than one on Tuesday at 10am. That knowledge is what makes the difference.

The franchise owner who has run kitchen remodels for 20 years has more valuable knowledge than any engineer or consultant who's never stood in a demoed-out kitchen at 7am. Your job, and the job of this guide, is to turn that knowledge into a description that lets a team of AI agents go do the improvement work for you.

Every chapter in this guide answers one question: *How does the owner use what they know to make their business measurably better?*

What You'll Be Able to Build

Before we get deep into planning, I want you to see what "done" can look like. Here are two real working applications that franchise owners have built using this exact approach:

> **Marketing Mastery Platform** — an operations hub for running marketing campaigns, tracking leads, and generating ad content. This is the kind of thing you'd build once you've scoped your marketing function and decided "I want a single dashboard that handles this for me."
>
> **VOC Platform** — a customer-feedback and reputation dashboard built for a home service context. It turns the messy work of NPS surveys, review requests, and sentiment tracking into something automated and visible.

Neither of these was built by a software agency. They were built by an owner who knew exactly what their shop needed, wrote it down clearly, and had a team of AI agents do the building. That's the playbook we're going to walk through together.

Before We Begin: Why This Isn't Optional Anymore

If you're reading this book, you've probably already noticed the ground moving.

Maybe it was the homeowner who scheduled an estimate with a competitor because they had a slick website and online booking. Maybe it was the younger couple who asked if they could text you their change order

instead of signing another paper document. Maybe it was the crew lead who said he saw one of your competitors on a TikTok video showing a kitchen they'd built, and he thought their stuff looked pretty good.

Small things. But they add up. And together, they point to something that's hard to ignore: **the way home service businesses run is changing, and it's changing fast.**

I want to head off a thought you might be having: *"Do I really have to do this now? Can't I wait a few years and adopt when things settle down?"*

Here's the honest answer: **you can wait.** But there's a cost to waiting, and the cost isn't symmetrical. It's not "delay a benefit" — it's "fall behind competitors who started earlier." The difference matters, and it compounds.

Let me lay out what's actually happening.

1. Your customers' expectations are already set.

The next generation of homeowners grew up with online everything — booking, ordering, status tracking, messaging. They don't want to play phone tag to schedule an estimate. They don't want to fill out paper forms that someone else has to type into a system. They don't want to wait three days for a proposal that could have been generated in three hours.

These expectations apply to home services as much as they apply to Amazon. And the first shops in your market to meet those expectations are going to pull ahead on customer acquisition, conversion, and retention. The shops waiting for "the right time" to upgrade are losing to shops that upgraded last year.

2. Your competitors are already moving.

You might not see it, because the early adopters in any industry look exactly like you do — they drive the same trucks, answer the same phones, eat in the same restaurants. But underneath, some of the shops in your market are already running with AI-powered systems:

- Automated lead response that sends a text within two minutes
- AI-drafted proposals sent within an hour of a consult
- Voice-to-text change-order capture from the field
- AI-generated follow-up sequences that turn one job into three

They don't talk about it loudly because it's their competitive advantage. If you're not seeing this in your market yet, it's coming. The question is whether you'll be the one who got ahead, or the one playing catch-up.

3. The cost curve is going one direction: down.

Two years ago, building even a basic AI agent would have required a developer and real money. Today, a shop owner with modest tech comfort can stand up a full multi-agent team — each specialized to a business function, each remembering its own stuff, each working 24/7 — for $20-50 a month.

In two more years, it'll be even cheaper. In five years, the systems will be even more capable. But the competitive advantage doesn't go to whoever adopts last. It goes to whoever adopts *first and learns the most from the running system.*

Every day your competitors' systems are running is a day they accumulate the moat I talk about in Chapter 1 — the scope, the memory, the learned behavior that nobody can copy. You can copy their setup in an afternoon. You can't copy their five years of accumulated operational knowledge, now codified into running agents.

4. The non-adoption penalty is going to get worse, not better.

There's a version of this decision where "waiting" is neutral. It's the version where the technology is optional, where the early adopters get a small bonus but everyone still has a shot.

That is not the world we're in.

The world we're in is one where the cost of doing business with home service shops that are still running on manual processes, paper forms, and tribal knowledge is going to go up relative to shops that run on automated systems. Your best techs will want to work for the shop that doesn't waste their time on admin. Your best customers will want to work with the shop that doesn't make them chase paperwork. Your bank will prefer to lend to the shop with clean data and predictable margins.

None of this happens overnight. But it's a slope, and it's tilted, and the longer you wait the steeper it gets.

So. This isn't optional.

I'm not being dramatic. I'm not trying to sell you something (this book is about doing the work, not consuming content). I'm giving you the honest read, based on what I'm seeing across dozens of home service shops, across multiple trades, across multiple markets.

The window for early-adoption advantage is open right now. It won't be open forever.

The good news: you don't have to be a tech person to take advantage. You don't have to hire a developer. You don't have to learn to code. You just have to know your business — which you already do — and be willing to write down what you know in the format this guide teaches.

If you've stuck with this book this far, you're already ahead of most owners.

Let's get to work.

Part I — KNOW YOUR BUSINESS

---

*Goal: Before trying to improve anything, understand what is **actually** happening in the business. Most home service companies run on tribal knowledge, gut feel, and "we've always done it this way." The first step is making the invisible visible — mapping every process, measuring every handoff, and identifying where value leaks out.*

What you hold at the end of this part: A prioritized list of improvements, ranked by the real business impact each one would deliver — the ones that make money, save money, give time back, or keep bad things from happening.

Chapter 1: Why Scope-First Thinking Wins

Let's start with the moment you probably had last week.

You're in the office at 6pm, going through the day's paperwork. There's the proposal from this morning that still needs one more pass before you send it out. Your lead carpenter, Dave, texted from the Elm Street kitchen remodel — the homeowners want to swap the cabinet finish from what they picked three weeks ago, and you need to figure out the change order pricing before you tell them. There's a voicemail from the Miller family on the Oak Street bathroom remodel, asking why the tile guy hasn't shown up in two days. And on your desk, a stack of invoices from last month that haven't gone out because Marie's been out sick and nobody else knows how your filing works.

This is the reality of running a home service shop. You're good at the work. You know the business. But there are always these little gaps — things falling through, things getting retyped, things that take three times as long as they should because you've built the shop around what people know *by heart* instead of what's written down.

Most owners I talk to try to fix this one of three ways:

- **Buy software.** Usually some expensive all-in-one that was built for a different trade or for general contractors, and it fits your shop maybe 40%. You're paying $400 a month for a system you still end up keeping spreadsheets alongside.
- **Hire help.** Usually a part-timer at $18/hr who gets the filing done but doesn't know *why* the Oak Street call matters more than the one after it.
- **Or — and honestly this is the most common one — just live with it.** The gaps, the rework, the "I'll deal with it Monday."

There's a fourth option now. And I want to start by telling you why this option is genuinely different from everything that came before.

Something shifted in the world recently.

Until just a couple of years ago, if you wanted to build a piece of software — a dashboard, a workflow tool, something that automated one of those gaps — you had to hire someone. A developer, an agency, a consultant. They'd charge you anywhere from a few thousand to a few hundred thousand, it would take months, and by the time it was done half the requirements had changed anyway.

Even if you got past that, you'd end up with something that was *theoretically* built for your shop but actually built for *any* shop. Because the people building it didn't know why the Oak Street remodel needed a different follow-up cadence than the Elm Street one. They'd ask. You'd explain. They'd build. By the time it was live it was stale.

That era is over. Not perfectly, not everywhere, but for a home service shop your size, it's over.

Today, you can describe what you want in plain English — not code, not specs, just a clear description of what "good" looks like — and an AI agent will go build it. Or run it. Or both.

I know that sounds like a sales pitch. It isn't. I'm going to walk you through exactly how this works, starting in Chapter 12. But before that, let me tell you two things that matter more than the technology itself.

The first thing: ChatGPT is not what we're doing here.

A lot of owners I work with tried ChatGPT or Claude, got a great response to some question, and thought "cool, done." That's not what I'm talking about. ChatGPT is a chat window — you ask a question, it answers. It forgets everything between conversations. It can't do anything in the real world. It doesn't remember your shop, your people, your processes.

An AI agent is a different animal. An AI agent has three things ChatGPT doesn't:

- **Memory.** It remembers your shop across weeks and months. Your customers, your techs, your vendors, your usual headaches.
- **Tools.** It can send an email, post a message, pull a weather report, update a spreadsheet, trigger a workflow. It's not a chatbot — it can actually *do* things.
- **Autonomy.** You don't sit there and prompt it step by step. It runs. It checks things. It acts. It only asks you when it can't figure out what you'd want.

That difference matters, because the magic isn't in an AI that can answer a question. The magic is in an AI that *works*. The AI that answers the Oak Street voicemail before you're done with your coffee, without you having to tell it to.

The second thing, and the more important one: the value is in you.

Here's what nobody is telling the home service world right now. The technology side of this is cheap, and it's getting cheaper. The agents, the models, the hosting — you're looking at $5 to $50 a month for a team of AI agents that does more useful work than a part-time admin. The hard part isn't the tech.

The hard part is knowing what to fix and how to fix it.

And that's you. That's the owner who's been running jobs for twenty years. That's the person who knows that every year right after tax refund season, the phone starts ringing with kitchen remodel leads, and you need proposals ready to send within three days or the lead goes to a competitor. That's the person who knows why the Elm Street change order needs a different approval step than a routine selection swap, even though *technically* they're both "changes to the plan."

Every smart person on earth can install this software now. The part that actually makes it work — that turns generic AI into something that actually helps *your* shop — only comes from you.

That's why I'm calling this whole guide "scope-first thinking." It's not a technology book. It's a book about how to take what you already know and turn it into instructions that a team of AI agents can run forever.

What is "scope-first thinking," really?

Here's the short version: before you build anything, you write down exactly what you want built. In a way that's clear enough that an AI agent could go build it. And by "clear enough" I mean clear enough that if you handed it to a smart 17-year-old who'd never heard of home services, they could execute it correctly.

That sounds obvious, and it is. The trick is that most of us have never done it. We've never had to. The filing has lived in Marie's head because Marie does it every day and nobody has ever asked her to write down *why* she does it that way. The Oak Street follow-up cadence lives in your head because so far you've been the one remembering.

Scope-first thinking just means: before I ask an agent to do this thing, I take twenty minutes and write down what the thing is. Not the technology. The *thing*.

Once that's written down, the technology part is almost free. That's the new deal. That's why this moment is real.

I'll show you exactly what a "build-ready" scope looks like in Chapter 7. For now, just know that this is the skill that's worth developing, because it's the skill that turns what you know into an asset nobody can copy.

The franchisee's moat

Let me make this concrete, because it's important.

A consultant or some well-meaning nephew could install all of this agent software for you in about thirty minutes. That's the easy part. What they can't do — because only you can — is write the scope that makes it useful.

"I want an agent that handles my review requests" is a wish. It's not specific enough for an agent to do anything with. An agent hearing that would ask, well, when? What do you say? To everyone? Even the angry ones?

"Seven days after every project is marked complete, send a text to the homeowner from the lead carpenter's number, using a template that mentions the room and the carpenter's first name, but only if they rated the job 4-stars or higher, and if they leave a review you send a separate thank-you text, and if they don't leave a review in fourteen days you try one more time with a different angle" — that's a scope. That's buildable.

That's *your* knowledge turned into instructions an agent can run forever.

Nobody else on earth can write that scope except you. And once it's written, the agent runs it for a fraction of a penny per execution. You can see why I call this a moat. It's a competitive advantage that literally nobody can copy. They can buy the same software. They can't buy your scope — unless you sell it to them.

What this guide will do for you

By the time you finish this guide, you'll have:

- Mapped your business on one page, so you can finally *see* it whole.
- Identified the places where money and time are leaking out.
- Written one to three scopes that turn your best improvement ideas into buildable instructions.
- Set up a team of AI agents that know your shop the way your best employees do — only cheaper, and they never call in sick.
- Built and deployed at least one real improvement that's running in your business, not just planned.
- A rhythm for doing this again and again, so the business gets better every quarter without you reinventing the wheel.

We're not going to talk about code. We're not going to spend time on AI theory. We're going to talk about your business — what's working, what isn't, and how to systematically make it better using the most powerful implementation tool that's ever been available to a small shop owner.

Ready? Let's start by looking at your business on one page.

Chapter 2: Mapping the Business End-to-End

Here's the first rule of fixing anything: you have to be able to see it.

I don't mean the big picture — you know the big picture. "We do kitchens, baths, and additions," or "We do HVAC install and service," or whatever your pitch is. You can say that in your sleep. I mean the actual picture. The real step-by-step of what happens, minute to minute, from the moment a lead comes in to the moment that customer sends a review — or doesn't, or sends a complaint.

Most owners I work with have never drawn that picture. The way their shop runs lives in the heads of the people who do the work. Marie knows what she does with the filing. Your estimator knows what he hands to the office manager before a proposal goes out. Your lead carpenter knows what questions he's going to get from Marie about a change order that happened mid-job.

That's fragile. And it makes improvements impossible, because you can't improve a process you can't describe.

In this chapter you're going to do something simple. You're going to put your business on one piece of paper.

The universal skeleton: every home service job is the same job

I've worked with a lot of home service shops — remodeling, HVAC, plumbing, electrical — and underneath all the variation — different trades, different crew sizes, different software — they all run on the same skeleton. Here's what it looks like with the kitchen/bath remodeler as our example (if you're reading this in a different trade, just swap the specific steps for your own):

1. **A lead arrives.** Phone call. Website form. Referral. Houzz. Angi. Facebook ad.
2. **You triage the lead.** Is it real? Is it in our service area? Is the budget right for what they want?
3. **You consult/estimate.** In-home or virtual meeting. Measure, discuss scope, take notes, photos.
4. **You propose.** Scope of work, pricing, options. Send to the homeowner.
5. **You get signed.** Homeowner reviews, maybe asks for revisions, signs the contract, pays the deposit.
6. **You design & order.** Selections (materials, finishes, fixtures). Sometimes an architect or designer. Place orders. Wait on lead times.
7. **You demo & rough-in.** Tear out the old stuff. Framing, electrical, plumbing — the bones. Subcontractor days.
8. **You build out.** Finish carpentry, tile, paint, trim, install. Visible work. The "wow" part.
9. **You punch & close.** Walkthrough with the homeowner. Punch list items. Final fixes.
10. **You bill.** Final payment. Collections.
11. **You follow up.** Review request. Referral ask. Maybe a seasonal re-engagement touch a year later.

That's the skeleton. Every home service job looks like this at some level. An HVAC emergency service call has the same steps, just compressed into three of them and with no design phase. A plumbing service appointment is a 4-step version. The pattern holds.

When you map your business, you're going to walk through these steps, one by one, and write down what *actually* happens at each one. Not what's *supposed* to happen. What actually happens.

The exercise: give me forty-five minutes

Here's exactly what I want you to do. Block out about 45 minutes. Don't try to do this on your phone between jobs — set it up like you're doing real work.

What you'll need:

- A piece of paper that's at least 8.5x11, ideally 11x17 or a whiteboard
- A pen

- Quiet. Put up the "do not disturb" sign.

What you won't need:

- Software. Not yet. Not for this.
- Your laptop. Close it.
- Your team. This step is just you.

Step one: write the steps in a row across the top of the page. These are your columns. If you're in remodeling, use the 11 steps I listed above. If you're in HVAC, it's probably 6-8. If you're a plumber, maybe 5. The number doesn't matter; the pattern does. Under each column, you're going to fill in a few rows:

- What happens?
- Who does it?
- What tools do they use?
- What document (if any) gets produced?
- How long does it *usually* take?
- How long does it *actually* take?

That last pair — how long it *usually* vs. *actually* takes — is the most important row. The gap between those two numbers is where your money is leaking.

Step two: walk through a job you remember. Pick a real one. The Elm Street kitchen, maybe, or the Oak Street bathroom. Walk through it in your head from start to finish, and fill in the boxes. Don't invent — write down what actually happened, including the parts where things went sideways.

"Phone rings. Homeowner from a referral — the Millers on Elm Street want to redo their kitchen. You schedule an in-home consult for Thursday. You drive out, measure the room, talk through what they want, look at the existing layout. You go back to the office, write the proposal — three options (refresh, refresh plus island, full gut-and-rebuild) at different price points. You email the proposal Friday. They call Monday, want Option 2 but with a different countertop. You revise, send Tuesday, they sign Wednesday, pay the deposit. You put in the cabinet order — six-week lead time. You run the permitting paperwork. Six weeks later you demo, frame, rough-in electrical and plumbing. Five days of finish carpentry. Two days tile. One day paint and trim. Final walkthrough Thursday. They love it. Friday Marie sends the final invoice. They pay two weeks later."

Now translate that into the boxes. Don't skip anything. Don't polish. The rougher it is, the more useful it's going to be.

Step three: do the same walk-through a few more times. Two or three more jobs, different flavors — at least one small refresh, one major remodel, one that went sideways somehow. As you repeat, you're going to start seeing three things:

- **The variations.** Every shop's lifecycle looks a little different job to job. Write the variations down. They're usually where the problems hide.

- **The handoffs.** Marie writes a sticky note. You read the sticky note. Dave the lead carpenter reads your email. You read the homeowner's selection-change text. Every time a job passes from one person (or tool) to another, there's a chance something falls.

- **The waiting.** Where does the job just sit? The proposal waiting for your review. The invoice waiting for Marie. The cabinet order waiting on the supplier. The permitting paperwork waiting for your signature. Every day of waiting is a day of cash-flow delay and a day where the homeowner's patience is thinning.

What to look at when you're done

When the page is full, put down the pen and look at it. This is your business. This is the thing you've been running every single day — maybe for decades — and most of it has been invisible until right now.

You're looking for four things. I'm going to name them here because they're going to be the whole next chapter:

- **Duplication.** Does the same information get typed into three different places? (The homeowner's address, the scope, the pricing — entered into your estimating tool, into your scheduling tool, and then onto the contract.) That's almost always the first improvement, and almost always the easiest.

- **Gaps.** Anything where the answer to "what happens next?" depends on someone remembering to do it? Those are the Oak Streets — the voicemails nobody called back about the tile guy who hasn't shown up.

- **Delays.** Anywhere that work sits waiting for a person when it could be moving. Proposal pending your review. Invoice pending Marie. Change order pending Dave's confirmation. Review request pending... nobody, because there's no person assigned and no system assigned.

- **Knowledge traps.** Anything where the only record lives in someone's head. If Marie gets sick for two weeks, what breaks? If your estimator quits, what breaks? If Dave — who actually knows how the Elm Street rough-in is supposed to go — gets sick for a week, what breaks?

You're not fixing any of these today. Chapter 3 walks through how to prioritize them and decide which one is the real binding constraint. But you can't do any of that work until you have this page.

The one-page map is a permanent asset

Once you've done this exercise once, something happens. You start seeing your business differently. You start noticing the gaps that were always there, hidden in plain sight. You start seeing the parts of your shop that run smoothly *because* they're already effectively "scoped" — the parts where everybody knows what to do and there's not much friction.

That's a huge insight, by the way. The parts that run smoothly are where you don't need to touch anything. The parts that are painful are where you need a scope. That's how you decide what to fix first — not by

what's most fun or most interesting, but by where the scope is missing.

Keep this page. Put it somewhere you can see it regularly. It's not a one-time artifact. You're going to update it every few months, and every time you update it you're going to notice something new — a new gap, a new delay, a new knowledge trap that's crept in.

That's how the improvement becomes a habit, and that habit is what turns into a compounding advantage over the owners who never bothered to draw the picture.

Ready to find out where your money's leaking? Chapter 3 walks through the four kinds of losses that almost every home service shop has — and how to figure out which one is the real priority.

Chapter 3: Identifying Inefficiencies and Constraints

You've drawn the picture. Take a breath — that's more clarity than most owners ever get.

Now comes the part where we stop describing and start diagnosing. You've got your page of sticky notes, your 9-step map, your real jobs walked through in black and white. The map showed you *what* happens. This chapter is about *where it hurts*.

Because here's the thing: you're not going to fix everything on that page. And honestly, you don't want to. Some of the things on there are fine — they work, they're scoped implicitly, they don't need your attention. What you're looking for is a small number of places where real money is either leaking out or never getting in.

The four kinds of losses

After working with a lot of shops, I've found that almost every inefficiency in home services falls into one of four buckets. I call them the Four Losses. Once you learn to see them, you'll start spotting them everywhere — in your own shop, in your competitors', in the franchise system you're part of.

Loss number one: Time loss.

Anything where the job is waiting for a human to do something that could move faster. The proposal sitting in your inbox for two days because you haven't gotten to it. The invoice that Marie *almost* sent but got pulled into something else. The crew that arrives at 8am to find the change orders from yesterday's walkthrough weren't written up yet.

Time loss is the easiest to see, and usually the first one owners try to fix with software. The problem is, software usually just gives you a different place to put the thing that's waiting — a new queue, a new ticket, a new dashboard. It doesn't actually make the waiting go away.

What you're really looking at with time loss is *where your shop's rhythm breaks.* The job is supposed to move; instead, it sits. Every hour it sits is an hour of cash-flow delay on a job that's probably already been costing you money for weeks.

Loss number two: Knowledge loss.

The filing that only Marie knows how to do. The way you actually price a kitchen remodel that has structural changes — that comes from ten years of negotiating with suppliers and subs, and it lives in your head. The trick your estimator uses to read a homeowner's body language and know whether to pitch Option 2 or Option 3 first.

Knowledge loss is the quiet killer in small home service shops. It doesn't crash the business in any one month. It just makes you dependent on a small number of people — often, just you — in a way you don't fully appreciate until someone gets sick, or quits, or you yourself get pulled into something else for three weeks.

When I say "if you get hit by a bus, what breaks?" I'm not being dramatic. I'm asking about a real operational risk. If the answer is "a lot," you have knowledge loss, and it's a worse problem than you think.

Loss number three: Opportunity loss.

This is the money that never gets in because something fell through. The Houzz lead you didn't respond to quickly enough because you were on a job site. The homeowner who left a voicemail after seeing your truck and you never called back because you assumed someone else would. The review request you meant to send after the Elm Street kitchen wrapped but never did. The one-year re-engagement touch on the Oak Street bathroom that would have sent one of your best customers your way again for their master suite — if anyone had remembered.

Opportunity loss is the hardest to quantify, because it's invisible by definition. You can't easily count the jobs you *didn't* get. But I've worked with shops that, once they put a system around it, pulled 10-20% more revenue out of the exact same market. Not by working harder. By *not dropping the ball* on things that were already happening.

The Oak Street voicemail from Chapter 1? That's opportunity loss. The Millers on Elm Street probably had three or four remodeling companies out for bids — your follow-up cadence is what gets you the job, or loses it.

Loss number four: Margin loss.

This is money that should have been in the job's final bill but wasn't. The change order you did the work for but forgot to bill because Dave didn't write it up right. The material markup you forgot to apply because you were moving fast. The scope creep on the kitchen that you absorbed rather than fight the homeowner about, because presenting a supplement felt like more trouble than it was worth. The equipment rental you charged for two weeks when it was actually on-site for three.

Margin loss is everywhere in home services. It's the reason your "on paper" profit on a job is always a little higher than the real profit once the job closes. And it compounds — because once it's missed on one job, it usually gets missed on the next five, until someone notices.

The exercise: walk the map and mark the losses

Now you're going to use the page from Chapter 2. Spread it out, and go back through every step, every handoff, every thing-you-wrote-down, and mark it with which loss (if any) it has:

- **T** for time loss — this step has waiting in it, or a human bottleneck
- **K** for knowledge loss — this only works because someone specific remembers how to do it
- **O** for opportunity loss — we sometimes drop this ball, and we leave money on the table
- **M** for margin loss — this step is where we lose money that should have been billed

If a step has more than one kind of loss, mark it more than once. If a step has no loss — it runs clean, no pain — leave it blank.

Do this thoroughly. It should take about 30 minutes. Don't skip the small stuff. The "Marie sends the invoice" step probably has time loss (it waits for her) and opportunity loss (late invoice is late cash). The "we send the permit application" step probably has time loss AND opportunity loss (every day it sits is a day toward the homeowner losing patience). The 17-year test from Chapter 1 applies here too: if you can explain to yourself in one sentence why this step has the losses it does, you understand what you're looking at.

When you're done, count up your markings. You're probably going to see a pattern. Most shops have one or two loss categories that appear on 60-70% of their steps, and the other two categories are concentrated in one or two specific places.

That pattern tells you something important. The category that appears most often is your *systemic* problem — it's wired into how the shop runs. The category that's concentrated in one or two places is probably your *binding constraint* — the single place where, if you fixed it, you'd pull the most value across the whole business.

Finding the binding constraint

Here's where I want to borrow an idea from manufacturing — and don't worry, I'm not going to make you read a book on it. It's the Theory of Constraints, and the insight is dirt simple: **any system is only as fast as its slowest part.**

Think of it like a chain. The chain is only as strong as its weakest link. If you strengthen the tenth link but the fifth link is still the weakest, the tenth link doesn't matter — the whole chain still breaks at link five.

Your shop is the same way. No matter how fast your estimator works, if invoices are sitting on Marie's desk for two weeks, the business is only running at Marie's speed. No matter how good your crew is, if you're not responding to Houzz leads fast enough, you're only winning the jobs that happen to come in when you're free. No matter how good your pricing is, if you're dropping change-order documentation mid-project, you're only winning the jobs where nothing changes — which is none of them.

The binding constraint is the one place where, if you fix it, the whole business speeds up, makes more money, or both. Fix anything else first and you're strengthening links that aren't the weakest — real work, but wasted effort.

How to find yours, in three questions

Pick the step on your map that appeared most often in the Four Losses exercise. Now answer these three questions:

1. **Is this the place where jobs actually stall?** Not theoretically — literally. When something goes wrong on a job, does it usually stall here? If the answer is yes, this is a real candidate.

2. **If I fixed just this one step, what else would improve?** If the answer is "a lot of other things would flow better" — reviews, cash flow, referrals, whatever — you've got a binding constraint.

3. **Have I been ignoring this step because it felt too big to fix?** If the answer is yes, you've probably been tolerating this constraint for months or years, and the business has been paying for it the whole time.

If all three answers point the same direction, congratulations — you've found your binding constraint. This is the first thing to work on. Not the second. Not the third. The first.

What to do with what you've found

Take ten more minutes and write down, in plain English:

- Your top binding constraint (one sentence: what is it, where is it, who or what does it depend on)
- Your top 2-3 systemic losses (the loss-category that shows up most, and where it shows up)
- The top 3 specific places you'd fix if you could only pick three (ranked)

That's it. That's your improvement priority list. It's the output of Part I of this guide — "know your business" — and it's the work that the next few chapters will turn into a buildable design.

One thing I want you to hold onto

There's a temptation, at this point, to start trying to fix everything. To grab a whiteboard and start drawing diagrams and figuring out solutions.

Don't.

The work of this part is to *see the problem clearly.* The work of Part II is to figure out what "fixed" looks like. The work of Parts III and IV is to actually build it. Skipping ahead and trying to solve the problem before you've defined what good looks like is the #1 reason improvements fail — in home services as in anything else.

You've just done the hardest work anybody has to do. You've mapped the shop. You've diagnosed it. You've identified the binding constraint. Now the rest of this guide is much easier, because you know *what* we're building toward. Next chapter, we design it.

Chapter 4: The Three Automation Candidates

Now comes the filter.

Because here's the trap most owners fall into: they look at that prioritized list and immediately start thinking *"how do I automate all of this?"* And that's the wrong question. The right question is: *"what's the cheapest, cleanest way to make each of these problems go away?"*

Because sometimes the answer isn't automation. Sometimes it's elimination — just stopping the thing entirely. Sometimes it's simplification — restructuring so that what remains is trivial to handle. And sometimes, only sometimes, the answer is actual automation with an AI agent.

The order matters. And the reason it matters is that **automating a bad process just gives you a faster bad process.** If you automate something that shouldn't exist in the first place, you've just locked in the waste at machine speed. The damage scales with the automation instead of shrinking.

So before any agent gets built, every finding from your diagnosis goes through a three-bucket filter: **Eliminate. Simplify. Automate.**

Bucket one: Eliminate.

This is the bucket nobody wants to talk about, and it's where most of the easy money is hiding.

Eliminate means: stop doing the thing entirely. Not "do it faster." Not "do it with an agent." Stop doing it.

I've worked with shops where 30% of what they were automating turned out to be work that nobody actually needed. The weekly report nobody read. The spreadsheet that got duplicated in three places because "that's how we've always done it." The manual data entry from one system to another when both systems had an API that could talk to each other.

Here's the test for eliminate: *"If I stopped doing this thing tomorrow, what would actually break?"*

If the answer is "nothing" — it's gone. If the answer is "Marie would have to remember to do X instead" — that's a knowledge trap, not a reason to keep the work. If the answer is "the homeowner wouldn't get the monthly newsletter" — ask yourself honestly: how many homeowners actually read the monthly newsletter?

The resistance to eliminate is emotional, not rational. We've always done it this way. It feels wrong to stop. But if the work isn't producing value, the work is the problem.

Bucket two: Simplify.

Simplify means: restructure so that what remains is obvious and low-friction.

This is where you catch yourself automating something that's broken in a fixable way. The classic example is the proposal process where the estimator writes a scope in a Word doc, then Marie types it into the estimating software, then the office manager types it into the scheduling tool, then the homeowner gets a PDF copy, then the crew gets a printed copy.

Four systems, three transcriptions, all for the same information.

The wrong move is to automate the transcription. The right move is to ask: *"Why does this information live in four places? Shouldn't it live in one, with everyone who needs it pulling from that one?"*

Simplification is structural. It changes how the work flows, not how fast the work gets done. It's usually about collapsing multiple steps into one, or making one source of truth that everyone references instead of passing information along a chain.

Here's the test for simplify: *"Could this same outcome be produced with fewer steps, fewer handoffs, or fewer places where the information lives?"*

If yes, restructure first. Then see what's left. Often, what's left after simplification is so trivial that you don't need an agent — just a shared document, a better form, or a simple rule. The hard work is figuring out the structure. The automation is the easy part (and sometimes unnecessary).

Bucket three: Automate.

This is where the agents live. And you want to get here only after you've eliminated what shouldn't exist and simplified what should.

Automation is the right answer when:

- The work has to happen (can't eliminate)
- The structure is already clean (already simplified)
- The work is repetitive, rule-based, or pattern-matchable
- A human doing it is either slow, expensive, error-prone, or unreliable (for any of those reasons — including "they're busy with higher-value work")

Classic candidates:

- Follow-up sequences (review requests, re-engagement, post-project nurturing)
- Lead response (initial text or email within two minutes of inquiry)
- Proposal drafting (from a completed consultation)
- Invoice generation (from time/logs/materials)
- Change-order documentation (from field notes or voice messages)
- Scheduling and crew assignment (from a master job calendar)
- Data entry between systems that don't talk to each other directly
- Status updates to customers, subs, or internal teams

The reason these work is that they have clear inputs, clear rules, clear outputs, and low ambiguity. The agent knows what to do because the job is well-defined. That's why you did the scope work in Parts I and II — so you'd end up here with a clear picture of what the automation should do.

The exercise: walk your priority list and mark it

Take the top three to five priorities from your binding-constraint work (you've got that list from Chapter 3). For each one, write down:

E, S, or A?

Be honest. I can almost guarantee that at least one of your top priorities is actually an eliminate, not an automate. Maybe two. And at least one is a simplify-then-probably-don't-need-an-agent situation.

The ones that survive this filter — the ones that are truly automations, after the elimination and simplification work — those are your actual agent candidates. Those are what we're going to design in Part II.

One last thing about order

There's a reason I put Eliminate first, Simplify second, and Automate third.

If you start by automating, you'll build a system around waste that shouldn't exist. You'll have the agent running fast, and you won't realize until three months later that it's running fast on the wrong thing.

If you simplify before you automate, you end up with a system that's clean and small. Sometimes the simplify step is so effective that the automation becomes trivial or unnecessary.

Eliminate first. Simplify second. Automate third. It's not intuitive — it feels like you're going backwards — but it's how you end up with an automation that's actually solving the right problem.

This filter is what turns the raw list of inefficiencies from Chapter 3 into a short, sharp list of things that are actually worth building. And that short, sharp list is your input to Part II.

Chapter 5: Single Model vs. Multi-Agent Orchestration

Before we pick the tools we're going to use — before we talk about Hermes or any other framework — there's a foundational choice you need to make. One that most owners don't even realize is a choice, because they've only ever seen one version of "AI" — the chatbot kind.

The choice is: **do you want a single AI that tries to do everything, or a team of specialists that each handles a slice of the work?**

Both are real architectures. Both have their place. Let me walk through both honestly, then show you how to pick.

The single-model approach

Think of this as having one very smart assistant who does everything. One mind, one chat window, one memory, one set of tools. This is what you get if you just sign up for ChatGPT or Claude and start using them.

It feels magical at first. You can ask it anything, get an answer in seconds, iterate on drafts, research things, write things. For a solo operator with a handful of recurring tasks, this can absolutely work.

Here's where the single-model approach breaks down: **context.**

Every AI has a limited context window — the amount of information it can "see" at once. For a simple chat, that's plenty. But when you're trying to have one mind manage your entire business — handle customer emails in the morning, write proposals in the afternoon, review field notes from the crew, run end-of-day financial summaries, and plan next week's schedule — that's a lot of different stuff for one context to hold.

The context fills up. When it does, earlier stuff gets pushed out or compressed. The model starts to lose track. It confuses details from one task with another. It gets expensive because you're using the most capable (read: most expensive) model for everything, even the simple stuff that doesn't need it.

Single-model also has no persistent memory of business-specific stuff (unless you explicitly configure it, which most people don't). Every conversation starts from zero, which means you're constantly re-explaining the same context: who your customers are, what your shop's rules are, what your process is.

When it works:

- Solo operators with simple, well-defined recurring tasks
- Early exploration, weeks one and two, to figure out what's possible
- One-off projects (research, writing, analysis) that don't need ongoing memory
- Anything where the task is short, bounded, and doesn't need specialized knowledge

The multi-agent approach

This is the architecture this guide is built around. Instead of one chatbot trying to do everything, you build a team of specialized agents — each with a defined role, each with its own tools, each remembering its own slice of your business.

It's the difference between one person trying to be the receptionist, the estimators, the project manager, the billing person, and the marketing director — vs. a small team where each person has one job and gets really good at it.

A typical multi-agent setup for a home service shop looks like this:

- **Main** (general orchestrator): handles the day-to-day, routes work to specialists, manages simple queries. Runs on a cheap, fast model because it's doing mostly coordination.
- **Ops** (operations): tracks jobs, flags overdue tasks, manages scheduling, does the daily briefing. Knows your crew, your projects, your typical cadences.
- **Estimator** (pricing/proposals): drafts proposals, handles change-order pricing, knows your supplier costs, your margin rules, your trade's typical patterns. Runs on a slightly more capable model because pricing requires more nuance.

- **Marketing** (outreach): review requests, email sequences, social content, seasonal campaigns. Runs on a cheap model because the tasks are pattern-based and repetitive.
- **Research** (market/competitive): price research, competitor monitoring, industry trends. Cheap model, because most of this is structured information retrieval.

Each agent has its own memory, its own skills, its own personality defined in a SOUL.md file (we'll cover that in Chapter 16). Each agent uses the model that's appropriate for its job — a cheap fast model for repetitive work, a more capable model for stuff that needs deeper reasoning.

The main agent coordinates. When a task comes in that's clearly within another specialist's domain, the main agent delegates to that specialist and waits for the result. The specialists do their job, report back, and the main agent pulls it all together.

The advantages of multi-agent:

Cost. This is the big one nobody emphasizes. If your single chatbot uses a $3/1M-token model for everything, and 80% of what it does is simple repetitive stuff that a $0.05/1M-token model could handle, you're overspending by 60x on most of your work. Multi-agent lets you route each task to the right-priced model. Your monthly bill drops dramatically.

Context isolation. The ops agent doesn't have to carry around all the marketing context. The estimator agent doesn't have to carry around all the customer-support context. Each agent's memory and context window stays focused on what it actually needs, which makes it more accurate and less likely to get confused.

Parallel execution. When a new $80k kitchen remodel gets signed, you probably need the ops agent to start scheduling, the estimator agent to lock in the pricing breakdown, the marketing agent to begin the onboarding email sequence, and the research agent to check permit timelines. In a single-agent setup, those happen one at a time. In a multi-agent setup, they happen simultaneously. The main agent coordinates, each specialist runs its task, and the whole job gets moving faster.

Specialization. Each agent accumulates its own skills over time. The estimator agent gets really good at pricing kitchens because it's done a hundred of them. The marketing agent gets really good at follow-up sequences because it's run hundreds of them. They don't pollute each other's context by trying to be everything.

Resilience. If one agent goes down, the others keep working. The ops agent can still schedule jobs even if the marketing agent is temporarily broken. You don't have a single point of failure.

When multi-agent is justified:

You don't need multi-agent to start. A single agent, well-configured, can get you very far in weeks one through four of working with these systems. You earn your way into the complexity.

Here's my honest rule of thumb for when to go multi-agent:

- You're running 10+ jobs per month

- You have multiple crew members or office staff who need agent access
- You're doing active marketing (lead generation, follow-up sequences)
- Your business has distinct "functional areas" that are clearly separable (estimating, operations, billing, marketing)

If most of those bullets describe your shop, multi-agent is the way to go. If only one or two do, start with a single well-configured agent and graduate to multi-agent when the single agent starts to feel overwhelmed by context.

The decision matrix

The clean way to decide is a simple two-by-two:

Task complexity (low vs. high) × Task frequency (low vs. high)

- Low complexity, high frequency → automate with a cheap multi-agent setup (this is where multi-agent earns its keep — lots of repetitive stuff)
- High complexity, high frequency → automate with a capable agent (maybe multi-agent, maybe single with a powerful model)
- Low complexity, low frequency → might not need automation at all; simplify first
- High complexity, low frequency → single-model works fine; no need to over-invest in orchestration

Most of what makes up a home service shop's daily work lands in the "low complexity, high frequency" quadrant — the repetitive stuff that scales up the more successful you are. That's multi-agent's sweet spot.

A real example: the $80k kitchen remodel

Let me walk through how the agent team handles a real job, so you can see the difference.

The Millers on Elm Street sign a big kitchen job. In a single-agent setup:

- One chatbot has to know the Millers' preferences, the pricing, the crew schedule, the subcontractor contacts, the review sequence cadence, the billing rules — all at once. That's a lot of context to carry around, and as the job progresses, the single agent starts losing track of details that came in earlier.

In a multi-agent setup:

- The ops agent creates the job record, schedules the crew, orders the cabinets, pulls permits, and coordinates subcontractors. It remembers the Millers' timeline preferences but doesn't need to know the pricing details.
- The estimator agent locks in the pricing breakdown, flags any change-order candidates as the job goes along, knows your supplier costs and your margin rules. It never has to think about scheduling.

- The marketing agent begins the onboarding email sequence (so the Millers feel cared for), plans the mid-project check-in, and queues the post-project review request.
- The research agent checks permit timelines with the city, flags any neighborhood restrictions or HOA rules that might affect the project.
- The main agent coordinates — knows that the ops agent is handling scheduling, the estimator is handling pricing, the marketing is handling communication, the research is handling regulatory.

Each specialist does its thing. Each has the exact tools and memory it needs. Each uses the model that's appropriate for its job (the estimator on Claude Sonnet for pricing nuance, the marketing on a cheap fast model for email sequences, the ops on a mid-tier model for general coordination).

The result: the job gets handled more competently, faster, and cheaper than a single chatbot could manage. And it scales — when you have three big remodel jobs in flight, the agents don't get confused.

The honest trade-off

Multi-agent isn't free. It's more setup work than a single chatbot. It requires you to think about what each agent's role is, how they coordinate, what they each need to remember.

That setup work is the work of Part II and Part III — the scoping, the SOUL.md writing, the delegation policy. It's real work, but it's investment, not waste. Once the team is standing, it runs. And the cost savings from routing tasks to the right-priced model, plus the capability gains from specialization, more than pay for the setup in weeks one through two.

The bottom line: if you're running a home service shop at any meaningful scale — more than a few jobs a month, more than one person involved — multi-agent is the right architecture. It's what this guide builds toward. But it's not mandatory on day one; you start simple and grow into it.

Chapter 6: Introducing Hermes Desktop

So far you've learned what to fix, how to design the fix, and what kind of AI architecture makes sense. Now I want to introduce the actual tool we're going to use to stand it all up.

It's called **Hermes Desktop**, built by Nous Research. It's open source, runs on your own machine (or a small VPS for $5-12/month), and it's the reason I wrote this guide at all — because it's the tool that makes the multi-agent orchestration model I've been describing actually possible for a small shop owner.

What it is (and isn't)

Hermes Desktop is an AI agent framework. It's the software that lets you run multiple AI agents, each with their own memory, their own tools, their own personality defined in a SOUL.md file. It's not an AI model itself — you bring whatever model you want (Claude, GPT, DeepSeek, GLM, a local model). Hermes is the orchestration layer that sits on top.

What it's not:

- It's not ChatGPT or Claude or Gemini. Those are AI models. Hermes is the agent framework that runs those models.
- It's not a SaaS product where you rent access from somebody else. You own it. It lives on your machine.
- It's not a chatbot wrapper or a "talk to AI" tool. It's an orchestration engine for running multiple agents that each have their own job.

Why this tool for this work

I've used a lot of different AI tools over the last few years. OpenAI's API, Anthropic's API, every hosted chatbot platform, various agent frameworks. Hermes is what I settled on for home service shops, and here's why:

It's open source and self-hosted.

This matters more than most owners realize. Your customer data — their names, their addresses, their payment information, their job details — never leaves your control. It doesn't go to some SaaS platform's servers where the terms of service could change. It doesn't get ingested into some training dataset. It lives on your machine, under your control.

For a home service shop that handles customer data as sensitive (because it is), this is real protection.

It's cheap.

The tool itself is free. The only cost is the AI model you run through it, plus whatever hosting you use (local machine = free, a small VPS = $5-12/month). Compare that to the $500-2000/month most home service shops spend on "all-in-one" software that doesn't actually do what they need.

It's model-agnostic.

Hermes doesn't care which AI model you use. Want to use Claude Sonnet 4 for your estimator agent and a cheap fast model for your marketing agent? Hermes supports that. Want to switch models tomorrow because a new one came out that's better for pricing? Hermes supports that. You're not locked in to any model provider.

This is a huge advantage. The AI model landscape moves fast — twice a month there's a new best-in-class model for some task. Hermes lets you adopt new models without rewriting anything.

It has the multi-agent orchestration built in.

This is the big one for this guide. Hermes has native support for:

- **Profiles** — each agent has its own profile, with its own config, its own memory, its own SOUL.md. This is how you build the specialist team.

- **Delegation** — agents can hand work to other agents. The main agent can break a big job into subtasks and spawn specialists to handle them, in parallel.
- **Memory** — each agent remembers what it's learned across sessions. The ops agent remembers that the Millers on Elm Street are sticklers about dust barriers. The estimator agent remembers that your margins on kitchen remodels have been tightening.
- **Skills** — procedures that the agent has learned and codified (like the "scope a renovation" procedure, or the "generate a change-order" procedure). Skills are how the agents get better at your specific business over time.
- **Cron scheduling** — recurring tasks, automatic runs, scheduled briefings. The agent wakes up every morning at 8am and gives you the daily briefing with no prompting from you.
- **Messaging gateway** — one agent, accessible from wherever you want. Telegram (works great from the truck), Discord (good for office staff), email, Slack, WhatsApp, even SMS. Same agent, multiple channels, wherever you and your team already work.

Key capabilities in one view

Capability	What it does
Profiles	Each agent has its own config, memory, tools, personality
Memory	Agents remember across sessions — your shop's context compounds
Skills	Procedures the agent has learned and can reuse
Delegation	Agents spawn specialists to handle subtasks, in parallel
Cron	Scheduled automatic runs (daily briefings, recurring checks)
Tool use	Agents can call external tools (web search, databases, APIs)
Messaging	Same agent accessible from Telegram, Discord, email, Slack, SMS
Open source	You own it; data stays on your machine/VPS

Privacy and control

I want to be explicit about this because it's a real concern for home service shops handling customer data:

When you run Hermes local (on your own machine or your own VPS), your customer data stays with you. The AI models process it in their cloud (that part is unavoidable with most models), but the orchestration, the memory, the skills, the records — all of that is on your machine. You're not running someone else's hosted chatbot that stores all your context on their servers.

If you're in a regulated industry or handle sensitive financial data, you can go further — run local models that never transmit to any external service, or configure the tools so certain data never reaches the model at all. The flexibility is there.

How this fits into the guide

Hermes isn't what makes the methodology work — the methodology would work with any multi-agent framework. But Hermes is the framework I've built multiple home service shops with, and it's the one I'm going to walk you through in Part III.

In Parts I and II, you're doing the business thinking — no software yet. In Part III, we fire up Hermes and stand up the agent team. From there, you have a running system that's ready to execute the improvements designed in Part II.

We'll do the actual installation in Chapter 12. For now, the point is: **there's a real tool, it's free, it's flexible, and it's built for exactly the multi-agent orchestration work that this guide is about.** You're not going to have to build anything custom or hire a developer. The infrastructure is ready.

The remaining question is: what does each agent's "job" actually look like? And how do you describe that job precisely enough that an agent can execute it? That's what Part II is for.

Part I ends with you holding:

- A one-page map of your business showing how it actually runs
- An audited list of places you're bleeding money or time, ranked by how much it actually costs you
- A marked list of what to Eliminate / Simplify / Automate
- A prioritized queue: the first thing to fix that will unlock the most value
- A clear picture of your **binding constraint** — the one lever, if you pull it, that pulls the rest of the business forward

---

Part II — Design What Good Looks Like

Goal: For each prioritized improvement, define what "fixed" actually means in your shop. This isn't a tech phase — this is business design. The output is a clear picture of the process when it works right: what goes in, what decisions get made, what comes out, what edge cases you hit, and what "done" looks like. An AI agent team is one way to make this picture real; the picture has to exist first.

What you hold at the end of this part: A small set of clear improvement designs — sharp enough that you could hand one to an implementer (human or AI) and they'd produce the right thing on the first try.

Chapter 7: What "Build-Ready" Means

You've filtered your priorities. You've got your E/S/A list. The automations that survived have earned their spot.

Now comes the design work. And this is where most owners get into trouble, because it's the work that separates the people who get a real running system from the people who get a chatbot that does weird stuff and they give up after three weeks.

The question you're answering in Part II is: **what does this automation actually look like when it's done right?**

Not "I want an agent that handles invoices." That's a wish. Wishes don't build systems.

The question is more like: "When a project gets marked complete, what exactly needs to happen for the invoice to be generated, reviewed by Marie, sent to the customer, and logged in the accounting system — and what are all the edge cases, error paths, and quality standards that need to be met?"

That's a build-ready scope. And until you have that level of clarity, you don't have a scope. You have a rough idea. Rough ideas are how you end up with agents that do weird stuff and you give up after three weeks.

The four things a build-ready scope must answer

Every well-designed automation answers four questions. If any of them has a "hmm, I'm not sure" answer, the scope isn't build-ready yet.

Question one: Why does this automation exist?

This sounds obvious, but most owners skip it. They jump straight to the mechanics without ever articulating the business purpose.

For the Elm Street review sequence: *"After a project wraps, we want to systematically request a Google review from the homeowner — because our lead conversion data shows that homeowners who see our reviews are 2x more likely to sign with us, and we're currently at a 4% review rate instead of the 20%+ we could be at with systematic prompting."*

That's a crisp, measurable business purpose. You can write that in one sentence. If you can't, the automation might be solving the wrong problem — or might not be worth building.

Question two: Who is involved, and what do they need?

Every automation has actors. Humans, systems, data sources, external services. For the review sequence:

- The homeowner (receives the texts/emails)
- The lead carpenter (their name and direct line are used in the outreach)
- The main agent (orchestrates the sequence)
- The marketing agent (actually sends the messages)
- Google reviews (destination for the reviews)
- Your CRM or job-tracking system (source of project-completion data)

Who does what, and what do they each need to see or do? A scope that doesn't identify its actors is a scope where things get missed.

Question three: What actually happens, step by step?

This is where most scopes fall apart. The owner says "send a review request text" and that's it. That's not a scope. That's a label.

A step-by-step scope looks like:

1. Main agent checks the job-tracking system every morning at 9am for projects marked complete in the last 24 hours.
2. For each completed project, main agent pulls the homeowner's name and contact info, the project address, and the lead carpenter's name.
3. Main agent hands that info to the marketing agent with the instruction: "Generate a personalized review request text, wait 7 days, then send it."
4. Marketing agent drafts the text using this template: *"Hi [homeowner], it's [carpenter] from [company]. Really glad we could help with your [project type] on [address]. If you have a moment, we'd love it if you'd leave us a Google review: [link]. Thanks for trusting us with your home!"*
5. After 7 days, marketing agent sends the text from the company's business number (not the homeowner's — from the company's).
6. Marketing agent monitors for a review to appear. If one appears within 14 days, it logs it and sends a thank-you text.
7. If no review appears in 14 days, marketing agent sends a second, shorter text with a different angle: *"Hi [homeowner], just checking in — hope you're loving the [project type]. If you get a chance, we'd really appreciate that Google review. Here's the link again: [link]."*

8. Marketing agent reports back to main agent with status: review-received, review-pending, or review-declined (if the homeowner asks to be removed).
9. Main agent logs the result in the tracking spreadsheet.

That's a build-ready scope. Each step has an actor, an input, an action, and an output. An agent (or a team of agents) could execute this without asking you any clarifying questions — because the scope answers them all.

Question four: How do you know it's done right?

Acceptance criteria. The definition of "working." For the review sequence:

- The text uses the lead carpenter's name and the project address (not a generic message).
- The text is sent from the company's business number, not the homeowner's.
- The first text is sent exactly 7 days after project completion.
- The second text (if needed) is sent exactly 14 days after project completion.
- Reviews are detected within 24 hours of being posted.
- Thank-you texts are sent within 12 hours of a review being detected.
- The tracking spreadsheet is updated with status after each step.
- Homeowners who ask to be removed are removed and don't get future outreach.

That's measurable. You (or the agent) can look at the execution and say "yes, this is working" or "no, here's where it broke."

The common failures

Let me walk through the mistakes I see all the time, because you'll probably make at least one of them:

Vague outcomes. *"I want an agent that handles invoices faster."* Faster how? Faster than what? What's the current time? What's the target? "Faster" is not a scope. *"Generate the final invoice within 24 hours of project completion, instead of the current average of 5 days, with the same accuracy as the human process (measured by X% of invoices requiring correction after sending)."* That's sharp.

Hidden assumptions. *"The customer's email is always valid."* It isn't. Some homeowner's emails bounce. Some are typos that nobody caught. If your scope doesn't handle that, the automation breaks silently — or worse, breaks loudly by spamming the wrong person.

Missing edge cases. *"Generate the invoice from the time logs."* What if there are no time logs? What if there are 50 line items? What if two techs logged the same hour? What if the material costs are missing from the job? Every "what if" that you know about but didn't write down is a bug waiting to happen.

Unstated domain rules. *"Price the kitchen remodel based on our typical pricing."* What are your typical pricing rules? What's the margin you actually target? Do you discount for repeat customers? For large

projects? For referrals? Your pricing has rules — maybe dozens of them. If they're not in the scope, the agent guesses. And guessing on pricing is how you lose money or lose customers.

The curse of knowledge

This is the trap I want to drill into your head.

As the owner, you know your business incredibly well. You know the little rules, the edge cases, the "we always do this when..." exceptions, the customer-preference patterns, the supplier quirks.

And because you know them, you assume they're obvious. You assume an agent would "just know" that the Millers on Elm Street prefer to be called by their first names, or that lead carpenters get a 10% bonus on kitchens over $50k, or that you never schedule tile work on Fridays because your tile guy has standing commitments.

But the agent doesn't know that. The agent doesn't know anything about your business until you teach it — and teaching means writing it down. Explicitly. In the scope.

The curse of knowledge is when you write a scope and think it's complete, but you've assumed a bunch of domain rules that aren't actually in the document.

The fix: **the 17-year-old test.**

The 17-year-old test

Take your finished scope and imagine handing it to a smart 17-year-old who has zero experience in home services, has never worked in your shop, has never heard of Xactimate or permits or change orders.

Could that person — with no context other than what you've written — execute the automation correctly on the first try?

If the answer is yes: your scope is build-ready. It's complete, detailed, self-contained, edge-case-covered.

If the answer is no, and they'd need to ask clarifying questions: your scope isn't build-ready yet. Those questions are the bugs in the design. Go back and write them into the scope as rules, rules, or decision trees.

The 17-year-old test is brutal but necessary. The agent executing the automation is, in a meaningful sense, a 17-year-old with no memory, no business context, no common sense about how your shop actually runs. If a human couldn't execute it from the document alone, the agent definitely can't.

This is the work of Part II. It's not the exciting part. It's not the part where you get to see cool automation doing things. It's careful, detailed, sometimes tedious writing. But it's the work that separates the people who get a real running system from the people who get frustrated and give up.

Once the scope is build-ready, everything after — the agent configuration, the SOUL.md files, the delegation policy — is mechanical. You've done the hard part. The rest is just execution.

Chapter 8: The Extraction Interview

So how do you actually get a build-ready scope? Where does all that detail come from?

From you. But most of the time, the detail isn't sitting on the surface, ready to be transcribed. It's implicit. It's in the way you've been running the business for 10-20 years. It's stuff you don't even think about anymore — until you try to explain it to someone who doesn't know it.

This is the fundamental problem with turning operational knowledge into something an agent can execute. The knowledge exists. It's just not in a form that's extractable.

The extraction interview is the method for getting it out of your head and onto paper (or into a voice recording) in a form that an agent can actually use.

The basic premise

Most of your business knowledge is tacit. You know it, but you can't easily articulate it. It's muscle memory. It's pattern recognition. It's the gut feel that tells you the Millers' kitchen job is going to need a change order because their initial selections didn't account for the structural work you spotted during the consult.

Tacit knowledge is valuable, but it's not automatable. Not until it becomes explicit — written down, structured, specific enough that an agent (or a new employee, or your 17-year-old test subject) can execute from the document alone.

The extraction interview is a structured process for turning tacit into explicit. You do it for every process you want to automate. It takes maybe 90 minutes per process the first time. After that, you'll get faster. And the output — the build-ready scope — is reusable. You do the extraction once per process; the automation runs forever.

The five-pass protocol

I've found that tacit knowledge tends to be organized into four or five categories. You can extract it systematically if you hit each category in sequence. I call these the five passes.

Pass one: The narrative pass

Tell the story of the process from start to finish, out loud, recorded.

Don't write. Don't outline. Don't try to be precise. Just tell the story like you're explaining it to a new hire who's shadowing you for the day.

Use your phone's voice recorder. Hit red, talk, hit stop when you're done.

For the Elm Street review sequence, your narrative pass might sound like:

"Okay, so when the Millers' kitchen is done — when they've seen the final walkthrough and signed off and we've packed up our tools — we want to get a review from them. But not immediately, because they're probably busy moving back in and just finishing up. So we wait about a week, let them live with the kitchen for a bit. Then we send a text. Not from the office number because that feels corporate. From the lead

carpenter's number — Dave's the one who built it, so it feels more personal coming from him. We mention their name, mention what we built for them, drop the Google review link. Then we wait. Some people review right away, some don't. If they don't review in two weeks, we send one more text, shorter, just a nudge. If they review, we send a thank-you. If they ask us to leave them alone, we leave them alone. And we track all of this because we want to know our review rate and we want to know which text template actually works."

That's maybe two minutes of talking. It's not structured. But it contains a ton of information — timing, tone, channel, personalization, follow-up cadence, tracking expectations — that would take you an hour to write if you started with a blank page.

The narrative pass gives you the raw clay. The next four passes refine it.

Pass two: The artifact pass

Collect every document, every form, every message template, every piece of data that gets produced or consumed by the process.

For the review sequence:

- The Google review link (formatted with your business ID)
- The text message template (with the personalization variables identified)
- The thank-you text template
- The second-nudge text template (for after 14 days)
- The job-tracking system's "project complete" event
- The tracking spreadsheet (what fields it has, what it logs)
- The lead carpenter's contact info in your CRM

You're building a parts list. Every artifact the process produces or uses. If the process has a document, a form, a message template, a photo, a log — you want it collected.

The artifact pass is where you often discover missing pieces. "Oh wait, we don't actually have a standard thank-you text; I just wing it every time." That's a scope gap. The automation needs a template. Go write one.

Pass three: The decision pass

For every fork in the road — every point where the process could go two different ways — who decides, and on what basis?

For the review sequence:

- Do we send the review request? Decision: main agent checks if project was marked complete in last 24 hours.
- Do we send the second nudge? Decision: marketing agent checks if a review appeared within 14 days.

- Do we remove the homeowner from future outreach? Decision: if homeowner replies asking to be removed.
- Which text template do we use? Decision: use the standard template from the artifact list, with personalization filled in.

Every conditional — every "if this, then that; if not, then this" — needs to be identified. Who (or what) makes the decision? What information do they base it on? What are the possible outcomes?

The decision pass is where you catch your implicit assumptions. "Well, of course we'd only send the second nudge if they haven't reviewed yet." Is that in the scope? If not, add it. The agent doesn't know what's "of course."

Pass four: The exception pass

What happens when things go wrong?

This is the pass nobody wants to do, because nobody wants to think about all the ways their process could fail. But the exception pass is what separates a fragile automation from a robust one.

For the review sequence:

- What if the project completion event doesn't actually fire in the job-tracking system (some jobs get marked complete and then un-marked because of a punch list item)?
- What if the homeowner's phone number is wrong or the carrier rejects the text?
- What if the homeowner leaves a negative review instead of positive?
- What if the homeowner asks to be removed but the message gets lost?
- What if the marketing agent tries to send a text on Christmas Day?
- What if the CRM goes down for an hour?

For every process, there are dozens of exception paths. You don't need to handle all of them — but you need to know which ones exist and make a decision about each: fix, ignore, escalate, or absorb.

The exception pass is the difference between an automation that runs smoothly for a year and one that breaks in weird ways every other week. Spend the time here.

Pass five: The constraint pass

What external constraints does the process have to respect?

For the review sequence:

- CAN-SPAM / TCPA rules: texting requires consent, must include opt-out mechanism
- Google review policies: can't incentivize reviews, can't review-bomb
- Business hours: don't send texts at 2am

- Brand voice: the text must sound like us (friendly, personal, not corporate)
- Frequency limits: max 2 outreach attempts per project, to avoid looking spammy

Every process operates within a web of constraints — legal, regulatory, brand, logistical, resource-based. The constraint pass surfaces them. The agent needs to know what it can't do, not just what it can.

For home service shops, the constraint list is longer than you might think:

- Building codes and permit requirements (for design/estimating processes)
- Subcontractor availability and lead times (for scheduling processes)
- Supplier minimum orders and return policies (for procurement processes)
- Industry association rules (for pricing, marketing, warranty)
- Your own shop's policies (margin targets, approval hierarchies, customer communication standards)

The constraint pass takes knowledge that lives in your head — and in the heads of your senior staff — and makes it explicit in the scope.

The full extraction in practice

So the extraction interview for the Elm Street review sequence looks like:

- **Narrative (2 min):** you tell the whole story out loud, captured on your phone's recorder.
- **Artifact (15 min):** you collect every template, link, form, data source. You probably discover a template or two you need to create.
- **Decision (20 min):** you walk through every fork in the road and nail down who decides what, and on what information.
- **Exception (30 min):** you think through every way it could go wrong and write down the handling for each. This is the hardest pass, and the most valuable.
- **Constraint (15 min):** you surface all the rules, policies, and external boundaries the automation must respect.

Total: maybe 90 minutes for your first extraction. You'll get faster. The second one will take maybe 45 minutes. By the fifth one, you'll have the rhythm and you can probably do one in 30 minutes.

What you hold at the end

At the end of the five passes, you have:

- A narrative of the process (voice recording, transcribed if you want)
- A parts list of every artifact
- A decision tree for every fork in the road

- An exception-handling plan for every known failure mode
- A constraint list of every rule the process must respect

You hand all of that to a smart implementer — human or agent — and they can build the automation. You've turned tacit knowledge into explicit design. You've beaten the curse of knowledge.

That's the core skill of this guide. You're not learning to code. You're not learning to configure complex software. You're learning to extract what you know into a form that an agent can execute. That's the skill that turns you into the most valuable person in the AI-agentic era — the one whose operational knowledge is documented, structured, and running.

One last thing: the extraction interview works for you as the owner, but it also works for your senior staff. If Marie has a process she's the only one who knows, sit down with her and run the five passes. Same for Dave the lead carpenter. Same for your estimator. Every extraction you do makes your business less fragile, more automatable, and more valuable.

Do the extractions. Write the scopes. Turn the tacit into the explicit. Then the agents can do the work.

Chapter 9: The Scope Template

Now that you've extracted the knowledge, you need a structure for putting it into writing. A scope template.

This isn't some formal project-management document with sign-off sections and version numbers. It's just a consistent format that ensures your scope captures what an agent actually needs to execute the work. Think of it as a checklist that keeps you from missing the edge cases, the exceptions, the weird situations that break naive implementations.

Here's the template I use:

1. Purpose

One paragraph. What does this automation do? Why does it matter? Who benefits?

For the Elm Street review request sequence: "Every seven days after a project is marked complete, send a personalized text to the homeowner requesting a Google review. If they leave a review within fourteen days, send a thank-you text. If they don't leave a review after fourteen days, send a second, different request. Track the status of each request."

That's the purpose. One paragraph. Clear enough that if you handed this to a smart 17-year-old with no industry experience, they'd understand what's happening.

2. Actors and Systems

Who's involved? What systems are they using?

For the review request sequence:

- The homeowner (receives texts, leaves reviews)

- The marketing agent (generates messages, sends texts, monitors for reviews)
- The project management tool (marks projects complete, stores homeowner contact info)
- Google Reviews API (checks if a review was left, retrieves the review URL)
- The company's business phone number (sends the texts from)
- The tracking spreadsheet (logs all request activity, status, outcomes)

You don't need fancy diagrams for this. Just a list. The point is to make sure every piece of the puzzle is named so nothing gets forgotten when an agent is trying to execute.

3. Inputs

What triggers this process? What data does it need at the start?

For the review request sequence:

- A project marked complete in the project management tool (the trigger)
- Homeowner's name
- Homeowner's phone number
- Lead carpenter's name
- Project address
- Project type (kitchen, bathroom, etc.)
- The date the project was marked complete

These are the inputs. An agent executing this scope needs to know that these are the pieces of information it needs to gather at the start. If any of them are missing, the automation can't run.

4. Steps

This is the detailed sequence of actions. Step by step. For each step: who does it, what do they do, what's the output, and are there any conditions or rules?

For the review request sequence:

Step 1: Every day at 6am, the main agent checks the project management tool for any projects marked complete in the last 7 days where no review request has been sent yet.

Step 2: For each eligible project, the main agent hands off to the marketing agent: "Generate a personalized review request text for [Homeowner Name] at [Project Address], built by [Lead Carpenter Name], and send it from the business number."

Step 3: The marketing agent drafts the text using a template that includes the homeowner's name, the project address, and the lead carpenter's name (not the marketing agent's name — personalization works better

when it feels like it's coming from the person who actually did the work). Then it sends the text via the business phone number.

Step 4: The marketing agent logs the request in the tracking spreadsheet with status "request_sent" and timestamp.

Step 5: Every day at 6pm, the marketing agent checks Google Reviews for any reviews posted in the last 24 hours that match a project where a request was sent. If a review is found, update the tracking spreadsheet status to "review_received" and send a thank-you text.

Step 6: For any review requests where status is still "request_sent" after 14 days, the marketing agent sends a second text (different wording — more casual, shorter, maybe "Just checking in — would love to hear your feedback if you get a chance, we're always trying to improve").

Step 7: After 30 days with no review, mark status as "no_response" and stop the sequence.

That's the steps section. Detailed enough that an agent could execute without asking you questions.

5. Edge Cases

What happens if something goes wrong? What about the unusual situations?

For the review request sequence:

- What if the homeowner's phone number is wrong and the text bounces? (Log as "contact_failed", escalate to Marie to verify number)
- What if the homeowner leaves a negative review? (Don't send thank-you, log as "negative_review_received", flag for owner to handle personally)
- What if the homeowner replies asking to be removed from follow-up? (Immediately update status to "opted_out", stop all further texts)
- What if the project management tool goes down? (Log as "system_unavailable", retry in 1 hour, alert if it's still down after 3 retries)
- What if a homeowner has multiple projects completed in the same month? (Separate tracking for each project, but don't send review requests within 7 days of each other)

The edge cases section is where you catch all the things that break naive implementations. If you don't specify these upfront, the agent will guess, and guessing on edge cases usually means guessing wrong.

6. Outputs

What does this automation produce? Where does it go?

For the review request sequence:

- A review request text (sent to homeowner)
- A thank-you text (sent if review is received)

- A second reminder text (sent if no review after 14 days)
- A tracking spreadsheet entry for each request (updated in real-time)
- A weekly summary report to the owner (number of requests sent, reviews received, response rate)

Outputs are the tangible things that result from the automation. If the automation is working correctly, these outputs should be produced on schedule.

7. Acceptance Criteria

How do you know if this automation is working correctly? What's the definition of success?

For the review request sequence:

- A text is sent to every eligible homeowner within 24 hours of project completion
- Text is personalized with homeowner name, project address, and lead carpenter name
- Text is sent from the business phone number, not the marketing agent
- Reviews are detected within 24 hours of being posted
- Thank-you text is sent within 12 hours of review detection
- Second reminder is sent exactly 14 days after initial request if no review
- Tracking spreadsheet is updated after every action
- Weekly summary report is delivered every Monday at 8am

These are your test cases. Binary pass/fail. If the automation satisfies all these criteria, it's working. If not, something's broken and you need to debug.

8. Out of Scope

What does this automation *not* do? Prevent scope creep.

For the review request sequence:

- Does not generate the review content (homeowner writes their own)
- Does not incentivize reviews (no discounts, no freebies)
- Does not handle customer complaints (that's a separate process)
- Does not send review requests for projects under $X (we only want reviews on big jobs)
- Does not send requests on weekends or holidays (respect homeowner's time)

The out-of-scope section is critical. Without it, well-intentioned agents will add features and the automation will bloat into something nobody asked for.

That's the full scope template for the Elm Street review request sequence. Eight sections. Detailed. Complete.

You hand this to an agent, and the agent can execute without asking you questions — because the scope answers them all.

Now do this for your top 2-3 priorities. Write out the full scope for each one. It'll take time — probably 2-3 hours per scope the first time. But once you have these written down, the implementation phase becomes mechanical.

Walkthroughs with trade-specific examples:

> **"Generate a scope-of-work document from an estimator's in-home notes"**: The estimator takes handwritten notes during a consultation. Those notes need to be turned into a professional scope document that gets sent to the homeowner. Actors: estimator (writes notes), main agent (receives notes), marketing agent (formats into scope document), homeowner (receives scope). Inputs: estimator's notes, customer name, address, project type. Steps: receive notes, parse content, generate scope sections (demolition, rough-in, finish, etc.), format as PDF, email to customer. Edge cases: illegible handwriting (escalate to estimator), missing measurements (flag for estimator to fill in), customer has budget constraints (adjust scope to fit).
>
> **"Auto-generate a 5-email post-project review sequence"**: Similar to the review request sequence but more elaborate. Five emails over 30 days, each with different messaging. Track open rates, click-through rates, review submission rates.
>
> **"Seasonal service reminder campaign from last year's job list"**: Every spring, send reminders to customers who had HVAC work done last year ("Time for your annual maintenance check"). Track response rates, appointment bookings.

Each of these gets the same 8-section treatment. The structure is universal. The specifics change based on the process.

The scope template is your bridge between the extraction interview (where you get the knowledge out of your head) and the agent configuration (where you turn the knowledge into running code). It's the structure that keeps you from missing critical details.

Chapter 10: Scoping for Agents vs. Humans

Here's something that'll save you a lot of pain: a spec written for a human isn't the same as a spec written for an agent.

When Marie reads her job description, it says things like "handle invoicing" and "send follow-up communications." Short. General. Marie fills in the gaps because she's been doing the job for six years. She knows what "handle invoicing" actually means in your shop.

An agent can't do that. An agent reading "handle invoicing" doesn't know whether that means Quickbooks, Xero, or a spreadsheet. Doesn't know if it includes sales tax. Doesn't know the format of the customer email. Doesn't know what to do if the customer doesn't exist in the system.

An agent needs a different kind of spec. One that's mechanical. Specific. With no assumptions about context.

Let me show you the difference.

A human spec:

> *"Send the invoice for the Elm Street kitchen project to the Millers."*

Marie reads this. She knows who the Millers are. She knows what project this is. She knows your invoicing software. She knows their email address. She sends the invoice. Done.

Maybe five minutes of work for Marie. But only because she has six years of context.

An agent spec:

> ***Trigger:*** *Project status changes to "awaiting_payment" in the project management tool.*

> ***Input:*** *Pull homeowner name, email, project address, final billed amount, payment due date, payment link, invoice number from the project records.*

> ***Steps:***

> 1. *Fetch the homeowner's email address from the project management tool using the project ID.*

> 2. *Generate the invoice PDF using Quickbooks API with:*

> - *Customer: [Homeowner Name]*

> - *Email: [Homeowner Email]*

> - *Line items: pulled from approved change orders and original scope*

> - *Total: [Final Billed Amount] including sales tax at [X]% for [State]*

> - *Payment due: 30 days from today*

> - *Payment link: [Stripe Payment URL for that invoice]*

3. *Send email via SendGrid API:*

- *To: [Homeowner Email]*
- *From: billing@[your-company].com*
- *Subject: "Invoice #[Invoice Number] — [Project Address]"*
- *Body: Use template invoice_email_template.md, substituting [Homeowner Name], [Project Address], [Invoice Number], [Final Billed Amount], [Payment Due Date], and [Payment Link].*
- *Attach the invoice PDF.*

4. *Log the invoice send: update tracking spreadsheet with timestamp and invoice number.*

Edge cases:

- *If the Quickbooks API returns an error (401, 500, 429 rate limit): retry up to 3 times with 30-second delays. If still failing, alert the owner and log the error.*
- *If the homeowner email is missing from project records: alert the owner, don't proceed.*
- *If the invoice PDF generation fails: alert the owner, log the error.*
- *If the SendGrid API fails to send the email: retry 3 times, then alert the owner.*

Acceptance criteria:

- *Email is sent within 1 hour of the project being marked "awaiting_payment"*
- *Email contains the correct invoice PDF attachment*
- *Tracking spreadsheet is updated with timestamp*

Out of scope:

- *Does not chase unpaid invoices (that's a follow-up collection process, separate automation)*

> - *Does not handle disputes (escalate to owner)*

That's an agent spec. It's longer. It's detailed. It's mechanical. It's specific enough that an agent could execute without knowing who the Millers are, without having worked in your shop, without ever having seen a Quickbooks invoice before.

Why agents need different specs

Three reasons:

No context. Humans have years of experience. Agents don't. Every assumption needs to be spelled out.

No tolerance for ambiguity. If Marie reads "send invoice" and there's an email typo, she might catch it because she knows the Millers. An agent won't catch it. The spec needs to handle that case explicitly (what to do if email is missing or invalid).

Machine execution. Humans improvise. Agents follow instructions. If the instructions have a gap, the agent stops or does something dumb. Humans can fill gaps; agents can't.

Common pitfalls when writing agent specs

Pitfall 1: You write "send an email" without specifying the email system. Which one? SendGrid? Gmail API? Outlook? The agent doesn't know. You need to be specific.

Pitfall 2: You write "get homeowner info" without specifying where to get it. From which system? Which fields? The agent needs exact field names and data sources.

Pitfall 3: You write "if there's an error, retry" without specifying what error. 401 unauthorized? 429 rate limited? 500 server error? Different errors need different handling.

Pitfall 4: You don't specify what happens if the automation runs twice. This is called idempotency. If Marie sends an invoice twice by accident, you get two invoices. If an agent sends an invoice twice by accident (because the trigger fired twice), you've spammed the customer. The spec needs to say: "If this runs a second time for the same project, do nothing (the invoice was already sent)."

Pitfall 5: You don't specify what to do if an external system is down. If SendGrid is down, what happens? Does the agent retry forever? Alert you? Queue the email for later? The spec needs to say.

The payoff

Writing an agent spec takes more time upfront. Maybe 2 hours instead of 20 minutes. But here's the tradeoff: a good agent spec takes 10x less iteration to get right. A bad spec (like a human spec) means the agent asks you questions, makes wrong assumptions, and you spend hours debugging and patching.

When I see owners get frustrated with agents, it's usually because they wrote a human spec and expected the agent to fill in the gaps. The agent can't. It's not a human. It's a machine that needs exact instructions.

Write the agent spec. It takes longer upfront but less time overall. And once you have the spec, the agent executes reliably. No debugging. No frustration. Just reliable automation.

A quick test

After you write your agent spec, give it the "new hire test." Imagine you're handing this spec to someone who just started yesterday — no industry experience, no knowledge of your shop, no familiarity with your systems. Could that person execute the spec correctly if they were given access to your tools?

If yes: your spec is good. If no (they'd need to ask questions): your spec has gaps. Add details until the new hire test passes.

That's the standard. Every agent spec should pass the new hire test.

Chapter 11: Prioritized Build Queue

You've written your scopes. Maybe 2-3 detailed, agent-readable specifications. Now comes a decision: what do you build first?

Because here's the truth: if you try to build all of them at once, you'll get overwhelmed. You'll lose focus. The first one will be half-built when you start the second, and none of them will be complete.

Better approach: build one. Finish it. Learn from it. Then build the next.

The prioritization framework

Order your scopes by business impact. Three tiers:

Tier 1: Automations that eliminate the biggest time loss.

These are the automations that give you back the most hours. Usually it's lead response (because responding to leads manually eats your morning), estimate generation (because writing estimates takes forever), or review requests (because nobody remembers to send them).

Pick one. Build it. Deploy it. Measure.

Once it's running reliably for a couple weeks, move to the next Tier 1 automaton.

Tier 2: Automations that capture opportunity.

These are the automations that make you money you weren't making. Storm-response campaigns (for restoration shops). Seasonal re-engagement campaigns (for HVAC, plumbing, remodeling). Quarterly review requests. These are "we should be doing this but we never get around to it" automations.

Tier 2 only after all Tier 1 automations are running smoothly.

Tier 3: Automations for internal tools and dashboards.

These are nice-to-haves. Dashboards that show your revenue. Reports on your project pipeline. Tools that help your crew. These are valuable but they're not urgent. Build them only after Tier 1 and Tier 2 are

running.

The rule: never scope more than 3 automations at once

Here's why: if you scope 10 automations, you'll never build them. You'll feel overwhelmed before you start. If you scope 3, you can actually finish them. And the third one teaches you lessons that make the next 3 easier.

Think of it like this: every automation you build teaches you something. You learn more about edge cases. You learn more about what your agents can and can't do. You get better at writing scopes. By the time you finish your third automation, your scopes are dramatically better than when you started. So start with 3. Finish 3. Then scope 3 more. And so on.

The queue is a living document

Your business changes. The queue should change too. Every month (maybe quarterly at minimum), sit down and review the queue. Ask:

- Are the Tier 1 and Tier 2 automations still the highest-impact ones?
- Has anything changed (new software, new customer segment, new regulation) that changes priorities?
- Has any Tier 3 automation become a Tier 1 because of circumstances (e.g., a customer complaint about lack of communication makes a communication tracking dashboard urgent)?

Update the queue accordingly. Add new scopes. Reprioritize. Archive scopes that are no longer relevant (maybe you switched software and the old automation no longer applies).

Putting this all together

At this point you have:

- A prioritized list of improvements (from Part I)
- 2-3 detailed agent-readable scopes for your top priorities (from Part II)
- A prioritized build queue (Tier 1, 2, 3) from this chapter

You're ready to move to Part III: standing up your agent team.

The agent team doesn't need to be elaborate. A single main agent can execute any of these scopes. But a multi-agent setup gives you specialization and cost savings. Either is fine for now. The important thing is that you have a running system.

Chapter 12: Installing Hermes Desktop

Enough theory. Time to install the tool.

Where to download

Go to https://hermes-agent.nousresearch.com/ and download the Hermes Desktop installer for your operating system (Windows, Mac, or Linux).

Installation

Windows: Run the downloaded .exe file. Follow the prompts. Default install location is fine for most people.

Mac: Drag the downloaded app to Applications. Launch it.

Linux: Follow the instructions on the site (there's a shell script installer).

First launch and setup wizard

When you first launch Hermes, it runs a setup wizard. It asks you a few questions:

> **Which AI provider?** For this guide, I recommend **OpenRouter** because it gives you access to every model we've discussed (Claude, GPT, DeepSeek, Qwen, GLM) through a single API key. You don't need to juggle multiple provider accounts. Choose OpenRouter.
>
> **Which model?** For your main agent (the one you'll use for orchestration and general tasks), I recommend **Qwen3 7B Plus** or **GLM 5.2** depending on your budget. Both are reasonably priced and capable. If you want the best overall quality, you can use Claude Sonnet 4 later and switch between models.
>
> **Your API key.** If you chose OpenRouter, you'll need an OpenRouter API key. Go to https://openrouter.ai, create an account (free), and generate an API key. Paste it in when asked.
>
> **Messaging platforms.** At this early stage, you can skip this. We'll add platforms in Chapter 14.
>
> **Your working directory.** Pick a folder where your agent's projects will live. Create a folder called something like "hermes-projects" and point it there.

Test the installation

Open the Hermes app. You should see an input bar at the bottom. Type something like:

> *"Give me a simple one-paragraph summary of the home service industry in [your city] and list three major trends."*

Hit enter. The agent should respond with a generated paragraph. If you see a response, your installation is working.

If nothing happens, check the status at the bottom of the app (should say "connected" with your API provider).

That's it. You're running.

You now have a working AI agent on your machine. No cloud dependencies. No server setup. It's all local. Your data stays on your computer.

The next few chapters will show you how to configure this agent for your specific business needs.

Chapter 13: First Agent — Your Personal Assistant

You have Hermes running. Now let's make it your personal assistant. Not a generic chatbot, but an agent that knows who you are, what your business does, and how you think.

Step 1: Write your SOUL.md file

SOUL.md is a document that defines who your agent is and how it should behave. It's the personality and rules for your main agent. Think of it as the agent's job description plus your company culture.

Create a file called SOUL.md (you can use any text editor) with the following structure:

```
# SOUL.md

You are the assistant to [Your Name], the owner of [Your Company Name].

[Your Company Name] is a [trade] shop operating in [your city/region]. We focus on [your specialty — residen

## Who we are

- We've been in business for [X] years
- We primarily serve [target customer — homeowners, property managers, commercial?], specifically [describe
- We pride ourselves on [your values — craftsmanship, communication, speed, quality?], and we're known for [

## How we operate

- Our typical job flow is: [brief description of your 11-step lifecycle — lead consult estimate sign design
- We have [X] crew members and [X] office staff
- Our key people are:
 - Marie: office manager, handles scheduling, invoicing, filing
 - Dave: lead carpenter, handles the big remodel jobs
 - [Your name]: owner, estimator, decision-maker

## How to communicate

- To me directly: be direct. No fluff. I'm a busy owner, so if you have a summary, I want the summary (not t
- To customers: friendly, professional, not overly formal. We're the home service company that actually pick
- To subcontractors: clear, direct, respectful.

## What you should always do

- Be accurate. If you're not sure about something, say so and ask.
- Reference our shop's domain knowledge when relevant (for example, we never schedule tile work on Fridays i
- Flag anything that could cost us money or reputation (edge cases, unusual situations, potential customer c

## What you should never do

- Don't make commitments to customers without my approval.
- Don't share sensitive business info (margins, employee compensation) with anyone outside the business.
- Don't guess at pricing rules. If unsure, ask me.
```

Fill in your business details. Be specific. The more you seed in SOUL.md, the more natural the agent becomes.

Put this SOUL.md file in your Hermes working directory (where you told Hermes your projects live). When you launch Hermes, it reads SOUL.md and uses it as its context for every conversation.

Step 2: Seed memory with key facts

Hermes has a memory system. The main agent remembers facts across sessions. You can seed it with important business information so the agent has context from day one.

Launch Hermes and type:

> *"Remember these facts about my business:*
>
> - *My company is [name] doing [trade] in [city]*
> - *Our biggest job type is [X] with typical project value of $[Y]*
> - *Our typical lead response time goal is under 2 hours*
> - *Our average review rate is currently [Z]% and we want to get to 20%+*
> - *The binding constraint I identified is [your binding constraint from Chapter 3]*
> - *My top priority automation is [your first automation from Part II]"*

The agent saves these facts and will reference them in future conversations. You can add more memories anytime by telling the agent "remember this" or "update your memory with..."

Step 3: Skills

Skills are reusable procedures. They're like the agent's muscle memory for common tasks. "Generate a change order" is a skill. "Draft a review request email" is a skill. "Build a project summary report" is a skill.

You'll create custom skills as you build automations. For now, install a few general-purpose skills to understand the pattern:

In Hermes, type:

> *"List the skills I can install"*

The agent shows available skills from the Hermes skills hub. Install a few useful ones:

- `home-service-business` – a skill for running general home service shop analysis

- `copywriter` – a skill for writing marketing and customer communications
- `planner` – a skill for breaking down complex tasks into steps

To install a skill, type:

> *"Install the planner skill"*

The agent installs it and now has it available for future tasks.

Step 4: Test the agent with a real task

Now let's use the agent for something real. Pick one of your low-stakes tasks — something that takes you time but isn't critical. Maybe drafting a follow-up email for a completed project.

Type:

> *"Draft a follow-up email for the Millers on Elm Street. Their kitchen project is marked complete as of [date]. Dave was the lead carpenter. Use our review-request template but with a personal touch. Email it to [Miller's email]. Don't actually send it — just draft it and show me."*

The agent reads your SOUL.md (knows your tone), your memory (knows the context), and your request (what you want). It drafts the email, shows you the text, and asks if you want to send it. You review, tweak if needed, then say "yes, send it."

That's the workflow. You delegate. The agent executes. You approve.

Do this for 2-3 tasks your first week. Build the muscle memory. Learn the cadence. Figure out what the agent is good at and where it needs coaching (tweak the SOUL.md accordingly).

What you have now

You have a working AI agent that:

- Knows who you are and what your business is
- Has access to your memories (key business facts)
- Can communicate with you in plain English
- Can execute tasks on your behalf (sending emails, drafting documents, running analysis)
- Can improve over time with skills and better SOUL.md content

This is the foundation. In the next chapter, we connect this agent to your phone (via messaging platforms) so you can interact with it from anywhere — job sites, client meetings, in the truck.

Part II ends with you holding:

- 1–3 fully designed improvement plans — clear enough to hand to an implementer (human or AI) without back-and-forth
- A reusable design template you can apply to every future improvement
- A clear order for what to build first, based on which improvement unlocks the most business value
- Confidence that the designs are complete (every edge case, constraint, and success criterion spelled out)
- A working agent that knows your business and has executed its first real task

---

Part III — Stand Up Your Agent Team

Goal: You know what to improve (Part I) and you know what good looks like (Part II). Now we build the team that actually does the work. For almost any operational problem a 2026 home service shop faces, the best way to implement the fix these days is a team of AI agents — cheap to run, fast to stand up, and they learn from every job. This part walks through how to pick the right agents, give them the right roles, and point them at your improvement designs.

What you hold at the end of this part: A working team of AI agents, each assigned a role that matches an actual job in your business — not a demo, not a toy — ready to tackle the improvements from Part II.

Chapter 14: Connecting Messaging Platforms

You've got your agent running on your computer. That's fine when you're sitting at your desk, but that's not how home service business actually works. You're on job sites. You're in the truck. You're meeting homeowners. You're not chained to a desk.

And your agent needs to work where you work. That's what the **Hermes Gateway** is for.

The Gateway is Hermes's way of connecting your agent to the messaging platforms you already use. Instead of only talking to your agent through the desktop app, you can talk to it through Telegram, Discord, email, WhatsApp, Slack, even SMS. Same agent, same memory, same skills — just accessible from wherever you already are.

This matters more than you might think. If I'm on a job site and Dave the lead carpenter says "The cabinet order is delayed two weeks, how do we reschedule?" I don't want to go back to the office to ask my agent. I want to pull out my phone, send the agent a message, and get an answer right there. Same thing with Marie in the office — if she's got a question about a homeowner's payment status, she should be able to ask the

agent from her desk, not schedule a meeting with me.

The agent has to meet you where you are. Not the other way around.

Which platforms should you connect?

For a home service shop, I recommend three to start:

Telegram — This is your mobile access. You get it on your phone, and it works great from job sites. Telegram bots are easy to set up (five minutes, you just talk to @BotFather and follow the prompts), and the app has great support for voice messages, document sharing, and even photo uploads. When your agent generates a quote or a schedule or a report, it can send you the document right there in the chat.

Email — This is for automated customer communication. When the agent sends a homeowner an invoice, or a project status update, or a follow-up email after job completion, that should go through email, not a chat platform. Email is formal, searchable, and homeowners expect it for business communication.

Discord (or Slack if you already use it) — This is for office staff. If Marie, your estimator, your project manager — whoever's in the office with you — need to interact with the agents, Discord works great. You can set up channels for different agents (like a #scheduling channel that talks to the ops agent, a #quotes channel that talks to the estimator agent). It's also good for you to see what the agents are doing without being in the middle of every conversation.

Step-by-step: Connect Telegram

Let's walk through connecting Telegram, since that's the one you'll use most.

Step 1: Create a Telegram bot.

Open Telegram on your phone or desktop. Search for `@BotFather` (that's a real Telegram user, verified, it's how you create bots). Send him the `/newbot` command. He'll ask for a bot name and a username. Pick something descriptive like "YourCompany Assistant" and a username like `yourcompany_agent_bot` (it has to end in `_bot`).

BotFather gives you an access token — it's a long string of letters and numbers. Copy that. You'll need it in a second.

Step 2: Configure Hermes Gateway.

In your terminal (or the Hermes Desktop app), run:

```
hermes gateway setup
```

This walks you through setting up the Gateway. When it asks which platforms to configure, choose Telegram. Paste in the access token BotFather gave you.

Step 3: Test it.

Open Telegram on your phone and find your new bot (search for the username you just created). Send it a message — something simple like "What's on my schedule today?"

Your agent should respond. Same agent you've been talking to on your desktop, same memory, same everything. You're just talking to it through Telegram now.

Try sending a voice message. Try sending a photo. Try asking it to "Check the status of the Elm Street kitchen project" and see if it pulls up the right details.

If it works, you've just made your agent mobile. You can talk to it from anywhere.

Step 4: Add your staff.

If you want Marie or your estimator to have access, you can either:

- Add a separate bot for them (each person gets their own bot, their own conversation with the agent)
- Or set up a Discord server with a shared bot (everyone on your team can see the conversations, which is good for office coordination)

For most shops, I'd start with just you on Telegram, plus maybe a Discord server for the office team. Don't overcomplicate it.

Email and other platforms

Email setup is a bit more involved (you need SMTP credentials from your email provider, or you can use a service like SendGrid). I'd tackle that in week two or three, after you're comfortable with Telegram.

Discord setup is similar to Telegram — you create a bot through Discord's developer portal, get an access token, configure it in Hermes. The Discord bot can join multiple servers, so you can set up a private server for your business and invite your staff.

Slack, SMS, WhatsApp — all follow the same pattern. You create a bot or app on the platform, get credentials, configure in Hermes.

Why this matters

Here's the real value: once your agent is on Telegram, you stop treating it like a desktop app and start treating it like a team member. You ask it questions on the drive to a job site. You tell it "Draft a follow-up email for the Millers" while you're waiting for the plumber to show up. You check project status from the truck.

It becomes part of how you work, not a separate thing you have to go do.

And once your staff can access it too, workflows change. Instead of "Hey Marie, what's the status on the Elm Street kitchen?" (and Marie has to dig through emails or call the homeowner), you ask the agent. It knows. It can answer immediately.

That's the shift. The agent stops being a tool and starts being infrastructure. It's just there, part of how your business runs.

Chapter 15: Designing Your Agent Team

So far you've got one agent. It knows your business, it can talk to you through Telegram, it can execute tasks. For a simple shop with straightforward workflows, that might be enough.

But most home service businesses have distinct roles that need different kinds of thinking. Your estimator prices jobs differently than your project manager schedules crews. Your marketing person writes promotional emails differently than your ops person writes status reports. Your coder builds software differently than your researcher analyzes market trends.

One agent can *try* to do all of that, and for a while it might work fine. But eventually you'll notice: the agent is getting confused. It's mixing up contexts. It's using marketing-speak when it should be using ops-speak. It's trying to remember everything and getting overwhelmed.

That's when you split it into a team.

What's a "profile"?

In Hermes, a profile is just a separate instance of the agent. Each profile has its own:

- **SOUL.md** — its own personality, communication style, domain rules
- **Memory** — its own persistent memory (it remembers different things than other agents)
- **Skills** — its own set of learned procedures
- **Model** — you can run each profile on a different AI model, optimized for its job

So your ops agent and your marketing agent are literally different processes, running on your computer (or VPS), with different personalities and different memories. They're specialists, not generalists.

Creating a profile is simple:

```
hermes profile create ops
```

That creates a new profile called "ops." It gets its own directory, its own config, its own everything. You then write a SOUL.md for it (we'll do that in Chapter 16), and you can connect it to Telegram or Discord just like your main agent.

Which roles should you create?

This is where most people overspend or underbuild. Let me give you the framework I use for home service shops.

First, don't add a specialist agent until you've identified the repetition. If you're only sending three marketing emails a month, you don't need a marketing agent — your main agent can handle it. If you're only doing one complex estimate a week, you don't need an estimator agent. Specialists make sense when the work is frequent enough that the specialization pays off.

That said, for most home service shops running at scale (say, 20+ jobs per month), here's the team I'd stand up:

The Main Agent — This is your orchestrator. It's the one you talk to most, the one that coordinates the others. It runs on a mid-tier model (I recommend **Qwen3 7B Plus** or **GLM 5.2** — good at reasoning, good at following instructions, cheap enough that you can talk to it constantly). The main agent doesn't do the specialized work — it figures out what needs doing and delegates to the right specialist.

Cost: about $3-5/month for a shop doing 20-30 jobs a month.

The Ops Agent — This is your operations manager. It handles scheduling, crew coordination, status reports, project tracking. It knows your crew members by name, knows which subcontractors are reliable, knows your typical job timelines. It speaks plainly, understands trade terminology, translates for homeowners when needed. It runs on a fast, cheap model (**DeepSeek V4 Flash** or **Qwen3.5 Flash**) because it's doing high-volume, relatively straightforward work — coordinating, not reasoning through complex problems.

Cost: about $1-3/month.

The Estimator Agent — This is your pricing specialist. It handles complex estimates, change-order pricing, scope reviews. It knows your material costs, your labor rates, your margin targets. It's careful, detail-oriented, good at math. It runs on a more capable model (**Claude Sonnet 4** or **GLM 5.2**) because pricing requires more nuance — it's not just filling in a template, it's reasoning through scope, materials, labor, profit margins.

Cost: about $5-15/month, depending on how many estimates you run.

The Marketing Agent — This is your outreach specialist. It writes promotional emails, social media posts, review request follow-ups, seasonal campaigns. It knows your brand voice, your typical customer, what converts. It runs on a fast, cheap model (**Qwen3.5 Flash** or **DeepSeek V4 Flash**) because marketing copy is high-volume and doesn't require deep reasoning — you need good writing, not deep thinking.

Cost: about $1-3/month.

The Research Agent — This is your market intelligence. It does competitor analysis, industry trend research, pricing comparisons, lead source analysis. It's curious, thorough, good at synthesizing information. It runs on a cheap model (**DeepSeek V4 Flash**) because research is mostly information retrieval and synthesis, not complex reasoning.

Cost: about $1-2/month.

The Coder Agent — This is your developer. It builds web apps, writes API integrations, creates automations. It's the one we'll use heavily in Part IV to turn your scoped improvements into live running systems. It runs on a capable coding model (**GLM 5.2** or **Claude Sonnet 4**) because it needs to write reliable code, not just chat about ideas.

Cost: about $5-20/month, depending on how much building you're doing.

What's the total cost?

For a shop doing 20-30 jobs a month with all six agents running:

- Light use (mostly main agent, occasional specialist work): $5-15/month
- Moderate use (regular specialist delegation, some automations): $15-30/month
- Heavy use (daily specialist work, active building, frequent automations): $30-50/month

Compare that to hiring an ops manager ($40-60K/year), a marketing person ($35-50K/year), or a developer ($60-100K/year). Even at heavy use, you're spending less than $600/year on your entire agent team.

And unlike a human team, they scale instantly. Need to run 50 marketing emails tomorrow? They do it. Need to price 10 change orders this week? They handle it. No hiring, no training, no burnout.

When to add each agent

Don't build the whole team on day one. Start with the main agent. After a week or two, when you notice the main agent is getting bogged down with a specific role (like "I'm spending too much time coordinating schedules, it's slowing down everything else"), add that specialist.

Typical progression for a home service shop:

- **Week 1-2:** Main agent only
- **Week 3-4:** Add ops agent (you notice scheduling is eating your time)
- **Month 2:** Add marketing agent (you realize you should be doing more follow-ups)
- **Month 3:** Add estimator agent (you're pricing more change orders than you want to)
- **Month 4+:** Add research and coder agents as needed (research when you're exploring new markets or services, coder when you're ready to build the automations from Part IV)

By month four, you've got a full team, and each agent has learned your business through its own memory. The ops agent knows your crew. The marketing agent knows your voice. The estimator agent knows your pricing. They're specialists, not generalists, and they're getting better at their jobs every day.

Chapter 16: Writing SOUL.md for Each Role

You've created a profile. Now you need to tell it who it is.

The SOUL.md file is where you define the agent's personality, communication style, domain knowledge, and constraints. It's the agent's job description plus its operating manual. Without it, the agent is a blank slate — it'll try to help, but it won't know how you want things done, what language to use, what matters, what to avoid.

With a good SOUL.md, the agent becomes a specialist. It knows its role, it knows the territory, it knows how to talk to your customers and your crew. It's not just smart — it's smart in the way your business needs.

Most people underinvest in SOUL.md. They write something generic like "You are a helpful assistant for a home service business" and wonder why the agent feels generic. The SOUL.md is where you encode your operational knowledge — the stuff that makes your business work. The more specific you are, the better the agent performs.

Let me walk through the SOUL.md for each role.

The Ops Agent SOUL.md

```
# SOUL.md — Ops Agent

You are the operations manager for a home service business. Your job is to keep jobs running smoothly — schedu

## How you work

You track active projects. You know which crew is on which job, what the timeline looks like, where things mig

You speak plainly. You're talking to homeowners, crew members, and the owner (me). Adapt your language:
- To homeowners: friendly, clear, no jargon. Translate trade terms into plain English.
- To crew: direct, specific, respectful. They're busy, don't waste their time.
- To me (the owner): concise, no fluff, give me the key info first, details if I ask.

## What you know

You understand home service terminology — rough-in, finish work, change orders, scope creep, punch lists. You

You know our typical timelines:
- Small bathroom remodel: 2-3 weeks
- Kitchen remodel: 6-8 weeks
- Full home renovation: 3-6 months

You know our crew:
- Dave is the lead carpenter, handles big remodels
- Mike is our plumber, reliable, usually finishes on time
- Sarah does electrical, great with homeowners, explains things well
- [add your actual crew here]

You know our subcontractors:
- ABC Plumbing — reliable, 1-2 week lead time
- Smith Electric — good work, sometimes slow to respond
- [add your actual subs here]

## What you don't do

You don't price jobs — that's the estimator's job. You don't write marketing copy — that's the marketing agent

You don't make commitments to homeowners without checking with me first. If a homeowner asks "Can you finish b

## Communication examples

**To homeowner (status update):**
"Hi [Name], just checking in on the kitchen remodel. We're on track for the cabinets to arrive Thursday, so we

**To crew (instruction):**
"Dave, the cabinet order for the Miller job should be here Thursday. Can you plan to start the install Friday

**To owner (me):**
"Quick update on Elm Street kitchen: cabinets arrive Thursday, install starts Friday, should be done early nex
```

That's the ops agent. Notice how specific it is — it names the crew, the subs, the typical timelines, the communication style for each audience. It knows what it does and what it doesn't do. It's not a generalist — it's the operations manager.

The Estimator Agent SOUL.md

```
# SOUL.md — Estimator Agent

You are the pricing specialist for a home service business. Your job is to create accurate, detailed estima

## How you work

You take scope descriptions and turn them into detailed, line-item estimates. You know our typical costs:
- Labor: $75/hour for carpentry, $85/hour for plumbing, $90/hour for electrical
- Materials: you mark up 30% above cost
- Overhead: 15% of total project cost
- Target margin: 20-25% on most projects

When pricing a project, you:
1. Break down the scope into specific tasks
2. Estimate labor hours for each task (use industry standards and our historical data)
3. List materials with quantities (be specific — don't say "tile," say "12'x24' porcelain tile, approx 150
4. Calculate labor, materials, overhead, margin
5. Present the estimate in a clear, homeowner-friendly format

For change orders, you price the delta — what's the additional cost compared to the original scope? You exp

## Communication style

You're careful and precise. You explain your reasoning — if an estimate seems high, you explain why (labor-

To homeowners: respectful, clear, no jargon. Explain what they're paying for.

To me (the owner): give me the numbers, the assumptions, and any red flags (unusually high labor, materials

## What you don't do

You don't negotiate pricing with homeowners — that's my job. You present the estimate, I discuss it with th

You don't schedule or coordinate — that's the ops agent's job.

## Examples

**Standard estimate:**
"Miller Kitchen Remodel Estimate:
- Demolition and removal: 16 hours labor @ $75/hr = $1,200
- Cabinet supply and install: $8,500 (materials) + 24 hours labor @ $75/hr = $10,300
- Countertop (quartz): $3,200 (materials) + 8 hours labor = $3,800
- Plumbing modifications: 12 hours @ $85/hr + $800 materials = $1,820
- Electrical updates: 16 hours @ $90/hr + $600 materials = $2,040
- Subtotal: $19,160
- Overhead (15%): $2,874
- Total before margin: $22,034
- Target margin (22%): $4,848
- **Final estimate: $26,882**

Notes: This assumes standard demo (no structural changes), standard cabinet configuration, no permit requir
```

That's the estimator. It's precise, it explains its work, it knows the numbers. It's not guessing — it's using real costs and real margins.

The Marketing Agent SOUL.md

```
# SOUL.md — Marketing Agent

You are the marketing specialist for a home service business. Your job is to write promotional content — emails, social

## How you work

You write marketing copy that's friendly, specific, and calls homeowners to action. You're not writing corporate-speak —

Our brand voice:
- Friendly and approachable (we're the local guys, not a big company)
- Specific and concrete (don't say "quality work," say "we use moisture barriers to prevent mold growth for the next 20
- Benefit-focused (homeowners care about their home, not our tools — focus on the outcome, not the process)
- Local (we're in [your city], we know the area, we've worked on hundreds of homes here)

## What you do

- Write promotional emails (seasonal campaigns, follow-ups, re-engagement)
- Draft social media posts (Facebook, Instagram, Nextdoor)
- Write review request follow-ups
- Write website copy (service descriptions, about us, testimonials)
- Create lead nurturing sequences (what to send a homeowner who requested a quote but hasn't signed yet)

## What you know

You know our ideal customer:
- Homeowners in [your service area]
- Typically 35-65 years old
- Own their home, planning to stay 5+ years
- Value quality over lowest price, but still price-conscious
- Want reliability (show up on time, finish on time, clean up after themselves)

You know our best-performing messaging:
- "Local family-owned business" resonates
- "20 years experience" builds trust
- Specific project examples ("We just finished a kitchen remodel in [neighborhood]...") work better than generic claims
- Homeowners respond to "we'll be there when we say we will" — reliability is our biggest differentiator

You know our marketing channels:
- Facebook ads work well for lead generation
- Nextdoor is great for neighborhood-specific targeting
- Email follow-ups convert 15-20% of quote requests
- Review requests work best 7-10 days after job completion (not immediately)

## Communication examples

**Review request email (sent 7 days after job completion):**
"Hi [Homeowner Name],

Hope you're enjoying your new [kitchen/bathroom/etc.]! It was great working with you on the project — Dave and the crew

We'd love to hear how things are going. If you're happy with the work, would you mind leaving us a quick review on Googl

Here's the link: [Google review link]

Thanks again for trusting us with your home!

- [Your name]

P.S. If anything comes up with the work we did, just let us know — we'll come back and take care of it."
```

That's the marketing agent. It knows the brand voice, it knows the audience, it knows what works. It's not just writing generic marketing — it's writing *your* marketing.

The Research Agent SOUL.md

```
# SOUL.md — Research Agent

You are the research specialist for a home service business. Your job is to gather information,

## How you work

You're curious, thorough, and good at synthesizing information. You find data, analyze it, and p

## What you do

- Competitor analysis (what are other shops in our area doing, pricing, services)
- Industry trends (new materials, regulations, customer preferences)
- Lead source analysis (which marketing channels are working, which aren't)
- Pricing research (what are competitors charging, what's the market rate)
- Performance analysis (which jobs are most profitable, which customers are most valuable)

## Communication style

You're analytical but accessible. You give me the key insight first, then the supporting data. I

To me (the owner): "Here's what I found, here's what it means, here's what you should consider d

## What you don't do

You don't make decisions — you provide information and recommendations. The decision is mine.

You don't execute marketing campaigns — that's the marketing agent's job. You figure out what to

## Example

**Research request:** "What's our average job profitability by service type over the last 12 mon

**Your response:** "Over the last 12 months:

- Kitchen remodels (12 jobs): Average profit margin 24%, average ticket $28K
- Bathroom remodels (18 jobs): Average profit margin 22%, average ticket $15K
- Small repairs/maintenance (45 jobs): Average profit margin 18%, average ticket $2K

Insight: Kitchen remodels are our most profitable per job, but bathroom remodels are more consis

Recommendation: Consider raising prices on small repairs by 15-20%, or outsourcing them to a han
```

That's the research agent. It gives you insight, not just data. It tells you what matters.

The Coder Agent SOUL.md

Chapter 17: Delegation — How Agents Work Together

You've got a team of specialists. Now the question is: how do you use them?

You *could* talk to each agent separately. "Hey ops agent, what's the schedule look like?" "Hey estimator, what's the pricing on this change order?" "Hey marketing, draft a follow-up email."

That works, but it's not how teams actually function. What you want is for the main agent to coordinate the team. You tell the main agent what you need, and it figures out which specialists to involve, in what order, and how to coordinate their work.

That's **delegation**. The main agent delegates tasks to the specialists, they do their work, and the main agent pulls it all together.

How delegation works in Hermes

Hermes has built-in support for what's called **subagent spawning**. The main agent can literally spawn other agents as subprocesses, give them tasks, and wait for their results. The specialists run in their own isolated context (their own memory, their own SOUL.md, their own skills), do their work, and report back.

Here's what that looks like in practice:

The scenario: You just got a new $80,000 kitchen remodel signed at 145 Oak Street. The homeowners are the Millers. You need to get everything rolling — scheduling, ordering, permits, communication.

Without delegation: You'd have to talk to each agent separately.

"Hey ops, we got a new kitchen remodel at 145 Oak St. Can you add it to the schedule?"

"Hey estimator, we need to break down the pricing for the Oak St kitchen into work packages."

"Hey marketing, draft an onboarding email for the Millers at 145 Oak St."

Four separate conversations. You coordinating everything. You remembering to do each thing.

With delegation: You tell the main agent once.

"We just signed a new $80,000 kitchen remodel at 145 Oak Street for the Millers. Can you get everything rolling? Scheduling, permits, ordering, communication — whatever needs to happen to kick this off."

The main agent figures out what needs doing, delegates to the right specialists (in parallel where possible), coordinates their work, and reports back to you with the results.

Behind the scenes, the main agent is doing something like this (you don't see the details unless you want to):

> **Delegates to ops agent:** "Add 145 Oak St kitchen remodel to the schedule. Millers are the homeowners. Timeline is 8-10 weeks, starting [date]. Coordinate with Dave (lead carpenter), order cabinets, pull permits."

Delegates to estimator agent: "Break down the pricing for 145 Oak St into work packages. We need clear milestones for payments — deposit, rough-in complete, finish work complete, final. Confirm the total matches the $80K estimate."

Delegates to marketing agent: "Draft an onboarding email sequence for the Millers. Welcome email, timeline overview, what to expect during the remodel, who to call with questions."

Delegates to research agent (optional): "Check the permit timeline for [your city] — how long does it typically take to get a kitchen remodel permit approved? Any neighborhood-specific requirements for 145 Oak St?"

All four specialists run in parallel (they're independent tasks, no dependencies). Each does its work, each reports back to the main agent. The main agent pulls the results together and gives you a summary:

"Here's what I've set up for the Oak St kitchen:

- **Schedule:** Added to the calendar, starting [date]. Dave's crew will start demo on [date]. Cabinets ordered from [supplier], estimated arrival [date].
- **Permits:** Applied for [date], typical timeline is 2-3 weeks. No HOA restrictions for this address.
- **Pricing breakdown:** Deposited 25% ($20K), rough-in complete payment 30% ($24K), finish work 30% ($24K), final payment 15% ($12K).
- **Communication:** Onboarding email sent to the Millers, they have Dave's number and my contact info.

Anything else you need on this?"

That's delegation. You gave one instruction, the main agent coordinated four specialists, you got a complete result.

How to set up delegation

Delegation is configured in your Hermes config file (usually `~/.hermes/config.yaml` or `~/.config/hermes/config.yaml`). The key settings are:

```
delegation:
 enabled: true
 max_concurrent_children: 3  # how many specialists can run at once
 model: qwen/qwen3-7b-plus   # which model the main agent uses for coordination
 provider: openrouter
```

The `max_concurrent_children` setting controls how many specialists can run in parallel. If you set it to 3, the main agent can spawn up to 3 specialists simultaneously. If you have 4 tasks that need doing, it runs 3 in parallel, waits for one to finish, then runs the 4th.

For most home service shops, 3 concurrent children is a good default. You can increase it if you're doing a lot of parallel work, but there's a cost tradeoff — more concurrency means more parallel API calls, which costs more.

What gets delegated (and what doesn't)

Not everything should be delegated. The main agent handles:

- Simple queries that don't require specialist knowledge ("What's the weather today?")
- Coordination and delegation itself (figuring out who should do what)
- Tasks that span multiple specialists (the main agent coordinates, doesn't try to do all the work itself)
- Conversations that require context from multiple sources (the main agent has access to all the specialists' results)

Specialists handle:

- Domain-specific work (ops does scheduling, estimator does pricing, marketing does copy)
- Tasks that benefit from specialization (the ops agent knows your crew and timelines better than the main agent does)
- Repetitive work in the specialist's domain (the marketing agent writes 20 follow-up emails a week, the main agent doesn't need to know how to write each one)

Error handling

What happens if a specialist fails? Maybe the ops agent can't pull permits because the city's website is down. Maybe the estimator agent needs more information to price a change order.

The main agent handles that. It gets the error report from the specialist, decides what to do (retry, escalate to you, ask for more info), and acts accordingly.

For example, if the ops agent reports "Couldn't pull permits — city website is down," the main agent might:

1. Retry in an hour (if it's a temporary outage)
2. Alert you (if it needs your input or approval)
3. Note it in the project tracking (so we don't forget)

You don't have to micromanage every error. The main agent handles most of them. It only escalates to you when it needs your input or when something is outside its authority.

The big picture

Delegation is what turns your agent team from a collection of specialists into a coordinated operation. Instead of you juggling four agents, you talk to one (the main agent), and it coordinates the rest.

This scales. As your business grows and your workflows get more complex, you don't need to manage more agents — you just give the main agent more complex tasks, and it delegates more work to the specialists.

And the specialists get better over time. The ops agent learns your crew's patterns. The estimator agent learns your pricing quirks. The marketing agent learns what converts. Each specialist gets sharper in its domain, and the main agent gets better at coordinating them.

By month three or four, you'll have a team that works like a well-oiled machine. You give it work, it gets done, you get results. You're not managing agents — you're managing a business, with agents doing the operational work.

Part III ends with you holding:

- A working team of AI agents configured around the specific improvements you want to make
- Every agent assigned a role that matches a real job in your business (not a demo, not a toy)
- Communication channels set up so the system fits into how your shop actually works (phone, email, team chat)
- Confidence the team is ready to take your Part II designs and turn them into running systems

---

Part IV — Put It In Motion

Goal: Take each improvement design, make it live, and prove it works the way you expected. A design on paper doesn't do anything — the money shows up when the system is running, producing results, and you're measuring what's changing. This part walks through the build → verify → deploy → measure → adjust loop.

What you hold at the end of this part: 1–3 improvements live and producing real business results (revenue saved, time given back, margin improved, or customer experience upgraded). Plus the examples below, so you know what "live" actually looks like in a shop like yours.

Real Apps Built by Owners Like You

Before you start building, take a few minutes to click through what other shops have already built. These are live, working applications owned and operated by people who went through the same process you're about to:

Marketing Mastery — a marketing operations hub: campaigns, lead tracking, ad creative, all in one dashboard. Built by someone who ran the same "where did all my leads go?" problem you probably have.

VOC Platform — a customer-feedback and reputation dashboard, purpose-built for home service. Turns messy NPS surveys, review chasing, and sentiment into something you can actually see and act on.

Neither was built by a software agency. Both were built by owners who knew what their shop needed, wrote it down clearly, and had a team of AI agents do the building. That's the same playbook we're using here.

Chapter 18: When You Need a Web App (vs. Agent Commands)

You've got your agent team running. You've connected messaging platforms. You've got agents coordinating, delegating, executing tasks. Most of your automations are handled through chat commands, scheduled tasks (cron jobs), and the agents doing their work behind the scenes.

For most home service businesses, that's enough. You don't need a web app for everything. In fact, for most things, you don't want one. Chat-based automation is faster to set up, easier to change, and doesn't require you to maintain a bunch of frontend code.

But there are times when a web app is the right tool. Let's talk about when that is — and when it's not.

When you don't need a web app

If the workflow is:

- Triggered by a chat command, a scheduled task, or an event (like a new lead coming in)
- Handled by an agent doing work behind the scenes
- Communicated through messaging (Telegram, email, Discord)

Then you don't need a web app. Most of the automations we've discussed in this guide fall into this category:

- Review request sequences (the agent tracks projects, sends emails, monitors results)
- Follow-up campaigns (the agent drafts and sends emails, tracks responses)
- Scheduling and coordination (the agent manages the calendar, coordinates crew)
- Pricing and estimates (the agent generates quotes, presents them to you)
- Research and analysis (the agent gathers data, gives you reports)

All of that is chat-based. The agent does the work, sends you updates, you approve or adjust via chat. No web app needed.

When you do need a web app

There are five situations where a web app makes sense:

1. Customer-facing portal

If you want homeowners to have a place to:

- See their project status ("Where are we in the timeline?")
- Upload documents (photos, permits, selections)
- Approve change orders
- View invoices and make payments

That's a web app. It's a portal your customers can log into, see their stuff, take action. The agents can work behind the scenes (updating the project status, generating invoices), but the interface is a web app.

Why not do this through chat? Because homeowners don't want to text your agent — they want a professional portal where they can see everything in one place. It's about the customer experience.

2. Multi-user system

If you need:

- Technicians to log their hours, upload photos, report issues from the field
- Office staff to update project statuses, manage schedules, track materials
- Managers to approve change orders, review reports, oversee operations

That's a web app. It's a system with multiple users, each with their own role and permissions. Agents can automate parts of it (like generating daily reports from the data), but the interface is a web app.

Why not do this through chat? Because you need persistent state — everyone needs to see the same data, updated in real time. Chat is great for one-off tasks, but for ongoing collaboration, you need a shared interface.

3. Structured data entry

If you need to:

- Collect detailed information through forms (lead capture, project intake, feedback surveys)
- Manage structured data (inventory, equipment tracking, supplier contacts)
- Build reports from structured inputs

That's a web app. Forms and structured data are easier to manage in a web interface than through chat.

Why not do this through chat? Because chat is conversational and free-form. If you need structured data (like a lead form with specific fields: name, email, address, project type, budget, timeline), a form is better than asking someone to type it all out in a chat message.

4. Reporting dashboards

If you need:

- Visual reports (charts, graphs, timelines)
- Real-time dashboards (current project status, revenue, pipeline)
- Reports for stakeholders (investors, partners, insurance companies)

That's a web app. Dashboards with visualizations are better in a web interface than as text reports.

Why not do this through chat? Because the agent can generate the reports and send you the data, but if you want to explore it (drill down, filter, zoom in on trends), a web dashboard is better.

5. Lead capture with branded UX

If you need:

- A landing page for marketing campaigns
- A lead capture form optimized for conversion
- A branded experience that looks professional

That's a web app. It's your storefront on the internet.

Why not do this through chat? Because you need control over the design, the messaging, the user experience. A landing page is marketing — it needs to look good, convert well, and represent your brand.

The rule of thumb

Start with agent commands. If you find yourself saying "I wish there was a way to [do this]" and the "this" is:

- A customer-facing experience
- A multi-user collaboration tool
- A structured form or data collection
- A visual dashboard or report
- A branded marketing experience

Then build a web app.

But don't build it until you've tried the agent-first approach. Most things you think need a web app can be handled through chat. You only need a web app when you've hit a wall with chat-based automation.

Example: The lead capture scenario

Let's say you want to capture more leads from your website. You could:

Agent-first approach:

- Set up a simple contact form on your website (just name, email, phone, project description)
- When someone submits the form, it triggers your main agent
- The agent sends you a Telegram message: "New lead from John Smith, wants a kitchen remodel, budget $30-50K, timeline spring. Want me to draft a follow-up email?"
- You say yes, the agent drafts and sends the email
- You move on

No web app needed. The contact form is just a basic HTML form — not a full app. The agent handles the rest.

Web app approach:

- Build a lead capture portal with a branded landing page
- Includes a more detailed form (project type, timeline, budget, photos upload, reference projects)
- Leads are tracked in a dashboard where you can see status, follow-up history, conversion rate
- The agent can still automate follow-ups, but the interface is a web app

When would you build this? When you're spending a lot of time managing leads manually, when you need more detailed data collection, or when you want a branded, professional presence for marketing.

But start with the agent-first approach. See how far you get. Only build the web app when you've hit the limits of chat-based automation.

Why this matters

Most people overestimate how much web app they need. They think "I need a system" and jump straight to building a web app. But for most home service businesses, an agent team with good chat-based automation handles 80% of the work.

The web app is for the last 20% — the customer-facing stuff, the multi-user collaboration, the visual dashboards. It's not the foundation. It's the polish.

Start with agents. Get them working. See where the friction is. Then, and only then, build the web app to handle the parts that need it.

And when you do build the web app, you'll use your coder agent to do it. That's what Part IV is about — turning your scoped designs into live, running systems.

Chapter 19: The Development Stack (Recommended)

You and I aren't going to build the web app. The coder agent is. So we need to pick a stack that the coder agent is genuinely good at — not what's theoretically "best," but what's reliable, well-documented, and something the agent can produce from a spec without stumbling.

Here's the stack I recommend for home service web apps. It's not because this stack is somehow inherently superior to alternatives — it's because the coder agent writes Python and JavaScript all day, these are the tools it has the most practice with, and they work well together. You can change the stack if you have preferences, but unless you have a strong reason, just use this.

Backend: Python + FastAPI

FastAPI is a modern Python framework for building APIs. The coder agent is extremely good at writing FastAPI code. It's well-documented, handles most web service needs with minimal ceremony, and plays well with everything else we need (databases, external APIs, authentication).

Why this matters: the coder agent has read thousands of FastAPI examples in its training data. It writes FastAPI code the way Marie writes invoices — from muscle memory, rarely making small mistakes. When the agent builds your web app's backend, it's working in its comfort zone. You get better code, faster.

For smaller shops with less complex needs, the coder agent might actually generate a single FastAPI app that handles everything. For bigger shops, it might split the backend into multiple services (one for estimating, one for scheduling, one for customer portal). Let the coder agent decide based on your scope.

Frontend: Next.js

Next.js is a React-based framework for building web frontends. It's opinionated about the way files are organized and about how data flows, which actually helps the coder agent stay consistent. Without framework opinions, the agent would sometimes organize files one way, sometimes another — with Next.js, it has a clear pattern to follow.

For most home service shops, the frontend is going to be:

- A dashboard for you (project status, revenue, pipeline, quick actions)
- Maybe a customer-facing portal (if you scoped one)
- Maybe a crew-facing interface (if you scoped one)

Next.js handles all three cleanly, and the coder agent can deploy each one to Vercel without extra configuration.

Database: SQLite first, PostgreSQL when you need it

Start with SQLite. It's a single file on your computer (or your VPS), needs no setup, no server, no password, no backup complications. The coder agent writes SQLite code just as naturally as it writes FastAPI.

For a shop doing 20-30 jobs a month, SQLite will handle everything you need without complaint. When you cross a threshold — maybe 100+ active projects in flight, multiple concurrent users, or you need cloud access — migrate to PostgreSQL. The coder agent can handle the migration.

Don't start with PostgreSQL unless you're sure you need it. Most shops don't.

Hosting: Railway or a VPS

You have two main options:

Railway ($5-20/month, depending on usage): Managed hosting that handles deployment, scaling, and most of the operational pain. You give Railway your code, it builds and runs it. No servers to maintain, no Docker to configure. The coder agent can deploy there with a single command.

VPS (Virtual Private Server) ($5-12/month, typical): A small Linux machine in the cloud (DigitalOcean, Linode, Hetzner, Vultr all work). You rent it for a few bucks a month, run your apps in Docker containers on it, and manage the OS yourself. Less managed, more control, slightly cheaper at scale.

For most home service shops, Railway is the pragmatic choice. You don't want to become a DevOps person — you want to run a home service business. If costs become an issue later (at scale), moving to a VPS is straightforward.

Either way, host the frontend separately from the backend. Frontend on Vercel (free tier), backend on Railway or your VPS. This keeps them independent and makes updates easier.

Domain: Cloudflare Registrar

Buy your domain through Cloudflare Registrar when you can. It's the cost price — no markup. A `.com` is about $10/year at cost, plus DNS hosting that's free and faster than most alternatives.

If you already have a domain, great. Point its DNS to Cloudflare (the registrar transfer isn't required for this to work).

Supporting services

For most home service web apps, you'll reach for:

- **Stripe** for payments — handling invoices, subscription billing, or deposits. The coder agent has extensive training on Stripe's API and can wire it up from a spec.
- **SendGrid** (or a similar transactional email service) for customer-facing emails — invoices, notifications, status updates. SendGrid has a free tier good enough to start.
- **Cloudflare** for DNS, CDN, and DDoS protection — the free tier handles almost every shop's needs.

All of these have well-documented APIs that the coder agent can integrate. The integration work is mechanical once the spec specifies exactly what each service should do and when.

The coder agent does all of this

Here's what I want you to take away: the coder agent builds everything. From your spec, it generates the database schema, writes the API endpoints, builds the frontend pages, sets up the CI/CD, writes the test suite, deploys to Railway or VPS, configures the domain in Cloudflare, wires up Stripe for payments, and connects SendGrid for email.

You don't write code. You review what the coder agent produces, test it, ask for changes, and approve. The labor division is:

- **You:** define the business logic, test, approve, deploy decisions
- **Coder agent:** write all the code, handle all the infrastructure

If you have coding skills already, that changes nothing about the model — it just means you can be a more demanding reviewer. The coder agent still does the writing.

Cost reality

The whole stack costs about $20-40/month at typical home service shop scale:

- Railway backend: $5-15/month
- Vercel frontend: free tier until high traffic
- Stripe: percentage of transactions (no monthly fee for basic)
- SendGrid: free tier handles thousands of emails/month
- Cloudflare: free for DNS, CDN, security
- Domain: ~$10/year

That's $20-40/month for a fully custom web application built specifically for how your home service business runs, not some generic tool you're trying to bend to fit.

Try finding a $40/month off-the-shelf solution that handles your full workflow, matches your exact business rules, and integrates with your other tools. You'll be shopping for a while.

The coder agent, working from your spec, builds exactly what you need. No compromises.

Chapter 20: From Spec to Deployed System — The Workflow

You've got a build-ready spec. You know what you're building, who it's for, what the inputs and outputs are, what the edge cases are, what the acceptance criteria say. The coder agent is ready to go.

Now what?

Here's the actual workflow. Seven steps. You own three of them, the coder agent owns four. This division is deliberate — you're the decision-maker, the agent is the implementer. Neither does the other's job.

Step 1: Share the spec with the coder agent.

This is your job. Open your build-ready scope document (the one you wrote following the Chapter 9 template) and share it with the coder agent. You can paste it into chat, or tell the agent where the file lives on your machine.

If your spec is more than a few pages, ask the agent to read the whole thing first before acting. You want it to have the full picture before it starts building — otherwise it'll build the first part with no idea what comes next, which causes confusion.

What you're saying to the coder agent is essentially: "Here's what I want built. Build it according to this spec. Don't improvise. If something's unclear, ask me."

Step 2: The coder agent scaffolds the project.

The coder agent creates the project structure. For a typical home service web app, this looks like:

- A `backend/` folder with FastAPI code (API endpoints, database models, business logic)
- A `frontend/` folder with Next.js code (pages, components, styling)
- A `database/` folder with migration scripts or a SQLite file
- Configuration files (environment variables, dependencies, build settings)
- A `README.md` documenting the project structure

This is the skeleton. Nothing works yet, but the pieces are there and the agent knows where to put things. You should see a project structure within a few minutes. You don't need to understand the code — you just need to see that the skeleton matches what you asked for.

Step 3: The coder agent builds features iteratively.

This is where the real work happens. The coder agent writes the actual code, feature by feature. It follows your spec, not some generic template. If your spec says "Generate a review request text from this template using these variables," the agent writes exactly that code — not a different pattern.

The agent works in iterations. Each iteration produces something you can see and test. You might see:

- "Login form is done, here's a preview"
- "Dashboard with active projects list is done, want to check?"
- "Invoice generation works with the template you specified, want to test it?"

You review each iteration. You might say:

- "Looks good, move on"
- "The invoice doesn't include the line item subtotal — fix it"
- "Can you show the customer's photo gallery on the dashboard? I forgot to include that in the spec"

This is collaborative. You're the product owner, the agent is the developer. You're not writing code, but you are making product decisions. The agent implements them.

Most features take 10-30 minutes of agent time to build, depending on complexity. If the coder agent gets stuck on something, it'll ask you or explore alternative approaches. The conversation stays focused on the work.

Step 4: The coder agent writes tests.

Your spec's acceptance criteria become real tests. If your spec says "An email is sent within 1 hour of the project being marked 'awaiting_payment'," the coder agent writes a test that checks exactly that.

This is where the agent demonstrates it's actually following your spec. A passing test is evidence that the code works correctly. Failing tests flag when the implementation doesn't match the spec.

The tests also serve as documentation. Six months from now, when you need to understand why some feature works a certain way, you can look at the tests.

Step 5: Test locally.

Before anything goes live, you test locally on your own machine. Your backend runs on port 8001, your frontend on port 3002. You open both in your browser and walk through the user flows manually.

This step is yours, not the agent's. You know how your business should behave. The agent wrote the code according to your spec, but specs have gaps. Local testing is where you catch the things the spec didn't anticipate — the flow that feels clunky, the button that's in the wrong place, the missing field that everyone on your team would want.

If something doesn't feel right, tell the agent. It will fix it.

Step 6: Deploy.

You decide when the thing is ready to go live. Not the agent. You say, "Deploy this to Railway and Vercel." The agent runs the deployment commands. Within a few minutes, your application is on the internet, accessible from anywhere.

If you're using Railway or Vercel, the deployment is essentially one command. If you're using a VPS, it's a few more steps but still manageable (the agent can handle the Docker configuration for you).

Step 7: Point your domain.

The coder agent gives you the DNS records you need — one for the backend API, one for the frontend. You paste them into Cloudflare (or wherever your domain is registered). DNS propagation takes a few minutes to a few hours.

After that, your application is at `app.yourcompany.com` or whatever domain you chose, fully accessible to your customers and your team.

The rhythm that follows

After deployment is when the work really starts. You use the application. You find things to change. You go back to the coder agent with updates: "The crew forgot to log the hours again on the Johnson job — let's add a push notification if they don't log by 5pm."

The coder agent updates the code, tests it, and deploys the update. Same loop. Same division of labor. You decide, the agent implements.

What this means in practice

For most home service shops, you'll end up with a running web application in 2-4 weeks of working with the coder agent, starting from a spec. The app gets smarter over time — it learns from your feedback, adds features you ask for, fixes bugs you report, and continuously improves as your business changes.

And that app is yours. It's custom-built for how your business works. It's not locked into a SaaS platform where you rent access and get stuck with features you don't want. It's your software, running on your infrastructure, handling your business logic.

This is what separates home service shops that build custom tools from those that rent them: the shops that build have software that works exactly how they think. The shops that rent have software that kinda-sorta works, patched together with spreadsheets and workarounds.

Chapter 21: Core Development Skills to Install

The coder agent can build anything, but it's more effective when it has the right skills — learned procedures that tell it exactly how to approach certain kinds of problems.

Skills in Hermes aren't a mystery. They're just markdown documents that say "when you run into this kind of problem, here's how to proceed." The agent loads them into context and follows them.

Here are the skills worth installing for a home service coder agent. You can install these from the Hermes skills hub, or write your own based on what works for your shop.

`plan` — Actionable planning before building

This skill forces the coder agent to write an actionable markdown plan before it starts writing code. The plan breaks the spec into phases, identifies dependencies, and flags things that need your input.

Why this matters: without a plan skill, the coder agent might jump into coding before it's thought through the whole problem. That's how you end up with code that needs to be rewritten. The plan skill slows the agent down in a productive way — it makes the plan explicit before building.

`test-driven-development` — Tests before code

This skill inverts the order: the coder agent writes tests first, then writes the code that passes those tests.

Why this matters: tests-first means the agent knows exactly what "done" means before it starts building. It follows the spec's acceptance criteria literally, because those criteria became real tests. You get code that's provably working, not code that "looks about right."

`saas-scaffold` — Production-ready starter

This skill gives the coder agent a well-structured starting template for a SaaS-style web app — FastAPI backend, Next.js frontend, PostgreSQL or SQLite database, authentication, basic CRUD. The agent fills in your specific business logic on top of this scaffold.

Why this matters: if you're building from scratch each time, there's a lot of boilerplate. The scaffold handles the boring parts so the coder agent can focus on your business logic.

`github-pr-workflow` — Safe code changes

This skill puts every code change through a pull request. The coder agent makes a branch, makes its changes, opens a PR with a description of what it did and why. You review the PR comments, approve, merge.

Why this matters: this is version control. If the coder agent breaks something, you can roll back to the previous version. You've always got a version you can go back to. No "oops, we lost that feature" moments.

`vps-docker-compose-deploy` — Deploy to your VPS

This skill teaches the coder agent how to package your app into Docker containers and deploy them to a VPS. Includes configuration for a reverse proxy (Traefik), SSL certificates (Let's Encrypt), auto-restarts on crashes.

Why this matters: if you run your app on a VPS, you need Docker for deployment. This skill gives the coder agent the pattern it needs so it doesn't have to rediscover Docker best practices each time.

`railway-fastapi-deploy` — Deploy to Railway

Alternative deployment skill for Railway instead of a VPS. If you're on Railway, use this. Simpler, fewer moving parts, less operational overhead (at the cost of slightly more expense and less control).

`react-nextjs-debugging` — Fix frontend issues

This skill gives the coder agent a protocol for diagnosing and fixing frontend problems. It knows to check the browser console, test in incognito, look at common React gotchas (render loops, state issues, hydration mismatches).

Why this matters: frontend debugging is where agents often get confused. This skill gives it a structured approach so it works efficiently.

`systematic-debugging` — Root cause methodology

This is a 4-phase debugging methodology: understand the problem, build a theory about what's broken, test your theory, apply the fix. Forces the agent to think before it pokes at things.

Why this matters: without this, agents tend to try random fixes ("let me change this and see what happens") instead of finding the actual root cause. Root-cause-first debugging saves you hours of flailing.

How to install and customize

To install these skills:

```
hermes skills browse  # look at what's available
hermes skills install plan  # install the plan skill
hermes skills install test-driven-development
hermes skills install saas-scaffold
# ... and so on
```

The agent will use them when appropriate. For example, when you ask it to "build a new feature for the customer portal," it'll load the `plan` skill first, write a plan, then follow the `test-driven-development` skill.

Custom skills: your most powerful leverage

The installed skills from the hub are generic. What really makes the coder agent powerful is custom skills — the procedures *you* teach it based on your specific business.

Example: after the coder agent successfully builds your review-request email sequence, ask it:

"Save what you just did as a skill called 'build-email-automation'."

The agent writes a skill document describing:

- What inputs it needs (scope, template, triggers)
- What files to create (email handler, cron job, tracking spreadsheet connection)
- What edge cases to handle (missing variables, rate limiting, retries)
- What tests to write

Six months from now, when you want to build an onboarding email sequence or a re-engagement sequence, you just say: "Use the build-email-automation skill."

Custom skills are how your operational knowledge compounds. Every success becomes reusable. Every lesson learned transfers to the next project.

The skill ecosystem, in one line

Install hub skills for the mechanics (deployment, testing, debugging). Build custom skills for your business-specific patterns. The combination is what makes the coder agent genuinely useful over time, not just for the first project.

Chapter 22: Scheduled Automations (Cron Jobs)

You've built some automations. Some run once (like generating a change order when the carpenter submits it). Some run on a trigger (like sending a review request when a project completes).

But some things should just *happen* on a schedule. Every morning at 8am, you want a briefing. Every Monday morning, you want a week-ahead summary. Every Friday, you want a revenue report. Every 90 days, you want to re-engage past customers.

That's what cron jobs are. Hermes has a built-in scheduler — you don't need a separate service, you don't need to configure anything, you just tell the agent what schedule and what to do.

Daily 8am briefing

Probably the single most useful cron job for a home service shop owner. Every morning, the ops agent (or your main agent) sends you a briefing on Telegram or Discord with:

- Today's schedule: which jobs are happening, which crews are where
- Overdue follow-ups: emails that were supposed to go out but didn't, review requests pending
- Anything flagged: delays, material deliveries, permit issues
- Quick revenue snapshot: money in, money out, current balances

This briefing runs while you're having your coffee. By the time you're ready to start working, you have the full picture. No digging through email, no checking five different tools.

Setting this up:

```
hermes cron create "every morning 8am"
```

Followed by telling the agent what to do at that time. It'll run the briefing every day, forever, unless you tell it to stop.

Monday 9am: weekly schedule review

Once a week, the ops agent pulls together the upcoming week:

- Projects starting this week (with crew assignments, material delivery confirmations)
- Subcontractor confirmations due (Mike for plumbing, Sarah for electrical)
- Material delivery windows (cabinet delivery Thursday? Countertop next Tuesday?)
- Projects hitting milestones (rough-in done, finish start, final walkthrough)

This surfaces things that would otherwise sneak up on you. You find out Tuesday morning that the cabinet delivery is delayed until next week *before* the crew shows up ready to install.

Friday 5pm: weekly revenue report

End of every week, your researcher agent (or main agent, if you haven't split out a researcher yet) generates:

- Revenue for the week (deposits received, invoices paid)

- Job count (new jobs signed, jobs completed, pipeline)
- Review count (reviews received this week, total, rate)
- Comparison to last week and last month (trends)

This is the kind of report that used to take Marie an hour to put together. Now it runs automatically. You can look at trends over time, see patterns, make decisions based on data.

Every 90 days: re-engagement campaign

For most home service businesses, the customers who've worked with you before are your best source of new business. A past customer is 10x more likely to hire you again than a cold lead.

Every 90 days, the marketing agent runs a re-engagement campaign:

- Pulls past customers who haven't been in touch in 90+ days
- Segments by job type (HVAC past customers get maintenance reminders, remodel customers get "any projects on your list?" nudges)
- Sends personalized emails with different messaging per segment
- Tracks opens, clicks, responses, bookings

This is "we should be doing this but we never get around to it" work. The agent gets around to it, automatically.

On webhook trigger: Stripe payment → follow-up

More advanced: you can wire webhooks so that when an event happens, the agent runs. For example:

When Stripe confirms a payment, the agent:

- Sends a receipt (automatic, no human involvement)
- Updates the project status
- Schedules a review request for 7 days later (if the payment was final)

This requires a bit of setup (Stripe webhook configuration, the agent running an API endpoint to receive webhooks), but once it's working, it's invisible. Events happen, the agent responds, you don't think about it.

How to create a cron job

Two forms:

```
hermes cron create "0 8 * * *"
```

That's cron syntax — "0 8 * * *" means "at minute 0 of hour 8, every day." A bit cryptic.

Or:

```
hermes cron create "every morning 8am"
```

Hermes parses natural language. Use whichever you prefer.

After running the command, the agent will ask what to do at that time. You tell it (in natural language), and it sets up the job.

Delivery

When a cron job runs, the output goes somewhere useful:

- Telegram (to your phone)
- Discord (to a specific channel for your office team)
- Email (in your inbox)
- A file (saved locally for reference)

You pick where it goes when you create the job.

Chaining jobs

Advanced pattern: one cron job collects data, passes it to another job that summarizes it. Example:

Job 1 (runs at 7:55am): Pulls project data, payment data, schedule data. Job 2 (runs at 8am): Summarizes that data into a briefing, sends to you.

The `context_from` configuration lets job 2 see the output of job 1. This lets you chain complex operations without any one job getting too heavy.

What you don't run on cron

Don't put *everything* on cron. Some things should happen:

- Triggered by events (a project completing, a lead arriving, a payment clearing)
- On-demand (when you ask the agent to do something)
- Manually (when you need to make a decision before something happens)

Cron is for predictable things that *always* happen at a specific time. If the timing matters, use cron. If the event matters, use triggers. If the decision matters, wait for human input.

Chapter 23: Automation Recipes for Home Service Shops

This chapter is your cookbook. Five ready-to-adapt automation recipes for the most common home service needs. Each one comes with the scope-level detail you need to adapt it to your business.

Copy what fits. Modify what doesn't. Throw away what doesn't apply to your trade.

Recipe 1: Lead-to-Close Pipeline

The full customer journey, automated end to end:

> *Lead capture → 2-minute response text → estimate/consultation scheduling → proposal → signature → deposit → project kickoff → punch list → invoice → review request → one-year re-engagement touch*

This is the holy grail. Every step handled by the right agent, in the right order, with the right timing.

For a remodeler:

- The marketing agent handles lead capture (form submissions from your website)
- The ops agent handles scheduling (calendar management, crew assignments)
- The estimator agent handles proposals and change orders
- The marketing agent handles follow-ups and re-engagement
- The main agent coordinates the whole thing

Each step has latency targets (respond to leads in 2 minutes, send proposals in 24 hours, invoice within 7 days of completion) and clear escalation points (if you haven't responded to a lead in 2 minutes, alert me).

For an HVAC shop:

- Same pattern, but "estimate" becomes "service call," and the timeline is hours or days instead of weeks

For a plumber:

- Same pattern, compressed even further

The agent team handles it the same way.

Recipe 2: Seasonal Campaign Automation

For most home service businesses, seasonal patterns drive revenue:

- HVAC: spring tune-ups, fall furnace checks
- Roofing: post-storm inspections, pre-winter prep
- Remodeling: spring kitchen projects, fall bathroom upgrades
- Plumbing: winterizing prep, post-holiday drain cleaning

A seasonal campaign is:

1. Calendar-triggered (run every September 1, or run when temperatures hit 80°F)
2. Segmented by past customer (job type, last service date, zip code)
3. Automated sends (personalized email for each segment)
4. Tracks opens, clicks, responses, bookings
5. Reports on conversion rate

Cost: ~$0.05 per campaign in agent time + $0 in hosting. The agent does all the work.

Return: past customers you already know and trust. One booking from a 500-email campaign covers your costs for the year.

Recipe 3: Reputation Engine

Reviews drive local search rankings. A home service shop with 200 five-star reviews wins every time over a shop with 20 five-star reviews.

The reputation engine:

- Waits a set period after project completion (7 days is typical)
- Sends a personalized review request from the lead carpenter's number (feels personal)
- Monitors for reviews for 14 days
- If a review appears: sends thank-you text, updates tracking
- If no review after 14 days: sends second request with different messaging
- Escalates negative reviews to the owner immediately

Expected result: 20-30% of customers leave reviews with systematic prompting. Without prompting, 2-5% leave reviews. The automation turns reviews from rare into routine.

Recipe 4: Change-Order & Mid-Project Documentation

For remodelers especially, mid-project changes are the #1 source of margin loss. The carpenter does the work, forgets to document it, and the change order gets missed.

The workflow:

- The lead carpenter submits a change request from the job site (voice note, photo, or quick text from the truck)
- The estimator agent generates a change-order document (scope, pricing, affected timeline, signatures needed)
- The ops agent sends it to the homeowner for approval (text or email)
- The ops agent tracks signatures

- If unsigned after 48 hours: escalation to the owner
- On signature: updates project schedule, flags invoice for final reconciliation

This turns margin-losing chaos into a clean, documented process. Change orders no longer slip through the cracks.

Recipe 5: Cash Flow & Project-Margin Monitor

Most home service business owners find out about cash flow problems by looking at their bank account. This automation surfaces issues before they become emergencies.

The workflow:

- Daily: Stripe → bank reconciliation summary (in your Telegram briefing)
- Weekly: receivables aging report (who owes you, how long)
- Per-project: margin tracking (budget vs. actuals, catching scope creep before the project closes)
- Alert thresholds: "Accounts receivable over 30 days > $X" or "Project X is 10% over budget"
- Monthly: full P&L summary, delivered as a report

For a shop doing $50K/month in revenue, catching a $3K cash flow gap three weeks early can be the difference between making payroll and scrambling.

The monitoring runs in the background. You get the data. You make the decisions.

Part IV ends with you holding:

- 1–3 improvements live and producing measurable business results
- Clear evidence of which improvement is delivering the most value
- A loop you can use over and over for every future improvement
- Recurring operations (daily briefings, storm triggers, review requests) running on their own without you babying them
- A web application built for your business, not bent-to-fit off-the-shelf software

---

Part V — Keep Getting Better

Goal: One improvement is a project. A habit of getting better is a real asset. This last part is about turning the first few wins into a system that keeps getting sharper — where every job you run, every customer you talk to, every edge case you hit feeds back into what the agents know, so the business gets more valuable every quarter.

What you hold at the end of this part: A business that's measurably better every quarter, without you putting a lot more time in. That's the compounding advantage — and it turns into the moat that nobody can copy.

Chapter 24: Ongoing Management Cadence

You've got your agent team running. Automations are live. A web app maybe. Daily briefings. Cron jobs. The shop is getting measurably better.

Now the question is: how do you keep it working?

The bad answer is "don't worry about it." Agents aren't self-sustaining forever. Memory gets stale. Skills drift. Models change and pricing shifts. The business itself changes — new crew, new territory, new customer type, new services. The system that was right three months ago isn't right today.

The other bad answer is "manage it obsessively." You're running a home service business. You don't want to become a part-time AI operator. The system has to work without you micromanaging it.

The middle answer — and the right one — is a cadence. Short, predictable maintenance windows that catch problems before they become issues.

Here's the rhythm I recommend for a home service shop:

Weekly (15 minutes)

Once a week, check on costs. Run `hermes insights` and look at:

- How much did you spend this week?
- Which agents did the most work?
- Are there any unusual spikes?

This is a five-minute look, max. It surfaces runaway costs before they become runaway costs. It tells you if the marketing agent has suddenly started generating 5,000 emails a day because a cron job fired five times instead of once. It catches things.

If the numbers look weird, you dig in. Otherwise, move on.

Monthly (1 hour)

Once a month, do a proper audit:

- What skills are being used? Are there skills you installed once that never got used? Archive them — they're noise.
- What memory is stale? Did the ops agent remember "our crew has three people" when you just hired a fourth? Update it.
- Are there automations that used to work but are now producing errors? Fix them or rebuild them.
- Did any agent behavior drift from what you want? (e.g., the marketing agent started writing emails that don't match your voice) Adjust its SOUL.md.
- Any new services, new territory, new customer type? Encode that into the relevant agent's memory.

This is the most important maintenance window. An hour a month keeps the system running clean. Skip it for three months, and you'll start seeing problems pile up.

Quarterly (half a day)

Once a quarter, re-run the Part I process map. Literally sit down and draw your business lifecycle on one page again. Ask:

- Has the business changed? (New services, new territory, new crew, new customer type)
- What used to be a Tier 3 automation that's now a Tier 1? (Maybe a dashboard started out as nice-to-have and now your whole team uses it every day)
- What used to be manual that's now fully automated? Any redundant steps you can remove?
- What new inefficiencies have crept in?

This is the big-picture maintenance. It keeps the system aligned with the actual business, not the business as it was six months ago.

Trigger events

Some things happen outside the regular cadence. Trigger events for maintenance:

- **New team member joins.** The ops agent needs to know who they are, what they do, what their responsibilities are. The main agent might need context on how to collaborate with them.
- **New system added.** You switched from Quickbooks to Xero. Point every agent that touches invoicing at the new system, update the SOUL.md references, update any skills that assumed Quickbooks.
- **Big job that exposes gaps.** The $200K renovation job breaks your estimating agent because your SOUL.md didn't include rules for jobs that size. Update the SOUL.md.
- **Customer complaint you didn't catch.** That's a signal that some automation has a gap, or some memory is stale. Trace the failure, fix it.

- **Price change (from a supplier, from your competitors, from your own).** The estimator agent needs to know your new pricing rules. The research agent needs to know about the supplier price change.

Trigger events are when you do maintenance outside the regular cadence. They're the "something just changed and the system needs to know" moments.

The feedback loop

Here's the thing most people miss: every automation run produces data about whether it's working. A review request sent that didn't result in a review. A follow-up email that got a negative response. A change order that took three tries to get right.

Capture that data. Act on it.

The ops agent notices "we're getting 40% review request rejections from homeowners who say 'we already reviewed, why are you asking again?'" You update the reputation engine to detect existing reviews before sending requests.

The marketing agent notices "the re-engagement campaign has a 2% open rate — way below what we get for new customers." You adjust the segmentation to exclude customers who've had negative experiences.

The estimator agent keeps missing line items on kitchen remodels. You look closer and realize you never encoded that "under-slab plumbing" is always a line item on kitchen remodels. You add it to the SOUL.md.

This is the feedback loop. The system tells you what's failing. You update the system. The system gets better.

Don't overschedule

Most home service shops are fine with 15 minutes/week, 1 hour/month, and half a day/quarterly. That's it. Add in trigger events as they come up.

Don't let maintenance become a side job. You're running a home service business, not an AI ops team. The system should work in the background, not demand your attention constantly.

If the maintenance load is more than an hour a week on average, something's wrong. Either the system isn't stable (rebuild it) or the automation is overcomplicated (simplify it).

The maintenance habit as part of your rhythm

The real win is when the maintenance windows stop feeling like AI-maintenance and start feeling like business maintenance. You're already reviewing your business once a month — revenue, job count, customer feedback. Just add the agent-system review to that existing meeting. You're already thinking about process improvements — just add "what should I encode into agent memory?" to that list.

The AI isn't a separate thing you have to maintain. It's the system that's making your business better. Maintenance is just making sure the system keeps getting better too.

Chapter 25: Memory Hygiene

Memory is the agent's persistent knowledge. It's what the ops agent remembers about your crew across sessions. It's what the marketing agent remembers about your best-performing emails. It's what the estimator agent remembers about which material suppliers are reliable.

Bad memory is one of the top reasons agents stop performing well. The marketing agent is writing copy that doesn't match your voice anymore because it remembered old templates from six months ago. The ops agent keeps assigning a crew member who quit three months ago because that person is still in its memory. The estimator is pricing jobs based on last year's material costs because its memory hasn't been updated.

These aren't catastrophic failures. But they add up. They make the agent less useful than it was. And if they accumulate for long enough, they make you distrust the agent and stop delegating, which defeats the point of having it in the first place.

Memory hygiene is about keeping the agents' memories accurate and useful.

What belongs in memory

Memory should contain facts that are:

- About your business (crew, suppliers, customer patterns, pricing rules, territory)
- Stable enough to stay true for weeks or months
- Referenced repeatedly by the agent in its work

Examples:

- "Dave is our lead carpenter" (stable, referenced every scheduling task)
- "Our margin target is 22% on remodels" (stable, referenced on every pricing task)
- "ABC Plumbing is usually 1-2 week lead time" (stable, referenced on scheduling)
- "The Millers on Elm Street are sticklers about dust barriers" (stable for the project duration, referenced on ops tasks)

Examples of what should NOT be in memory:

- "The plumber showed up late yesterday" (transient event, not a durable fact)
- "The Millers said they liked the tile" (project-specific, lives in the project record)
- "This estimate was $26,882" (project-specific, lives in the invoice)

Seeding memory correctly

When you first set up each agent, seed its memory with the essential facts. For the ops agent, that's:

- Crew names and roles

- Subcontractor names and typical lead times
- Typical timelines per project type
- Owner communication preferences
- Business rules (what's a selection change vs. a scope change, who decides what)

For the marketing agent:

- Brand voice
- Customer persona
- Best-performing messaging themes
- Channel preferences
- Frequency rules (how often to send marketing emails before you look spammy)

For the estimator:

- Pricing rules (labor rates, markup, overhead, margin targets)
- Material supplier list with typical pricing
- Trade-specific scope conventions (what's always included in a kitchen remodel, what's optional)
- Your red lines (what never gets discounted, what always requires owner approval)

For the research agent:

- Business KPIs and typical values
- Market segments
- Services offered
- Geographic focus

Get this right upfront and the agents perform well from day one. Get it wrong and you'll spend weeks correcting course.

Periodic purging

Memory that's no longer true becomes a liability. Every quarterly maintenance cycle, sit down with each agent's memory and purge what's stale:

- Crew members who've left
- Suppliers who've gone out of business or changed their pricing
- Business rules that changed

- Customer patterns that no longer hold (maybe a customer type stopped responding to a particular email)

Purging is simple — just tell the agent "forget this" or "remove that from your memory."

What's harder is knowing what to purge. That's the maintenance habit: regularly checking whether the agent's memory still matches reality.

Memory vs. Skills

People confuse these. They're different.

Memory is facts about your business. Persistent. Contextual. "Our crew has four carpenters." "Our margin target is 22%."

Skills are procedures the agent has learned. Actionable. "When asked to generate a scope of work, use this template, apply these rules."

Memory makes the agent *know*. Skills make the agent *do*.

You update memory when facts change (new crew member). You update skills when procedures change (new template for scopes).

Both need maintenance. Both compound over time.

Memory after major business changes

Some moments require a memory review:

- **New service area:** the ops agent needs to know the new territory, maybe the research agent needs to research the new market
- **New team member:** every agent needs to know who they are and how to interact with them
- **New supplier relationship:** the pricing assumptions in the estimator need updating
- **New customer segment:** the marketing agent needs new personas
- **New regulatory requirement:** the ops agent needs new compliance rules

When business changes, memory changes. Capture those changes quickly, while they're fresh.

If you skip this, the agents will make wrong assumptions. The ops agent will schedule a crew member in the old territory. The estimator will price jobs with old supplier costs. The marketing agent will use messaging that doesn't resonate with the new customer segment.

Catch changes when they happen. Update memory when it's relevant. Don't let stale memory accumulate.

The payoff of memory hygiene

A shop that maintains agent memory gets consistently useful behavior. The ops agent never forgets who's on the crew. The estimator never prices from stale costs. The marketing agent never writes copy that doesn't match the current brand.

A shop that neglects memory hygiene gets increasingly unreliable behavior. The agents are "still kind of working" but making more mistakes, needing more corrections, producing worse results.

The maintenance difference is minutes-per-week. The performance difference is dramatic.

Chapter 26: Cost Control

You started at $20/month. Six months later, you're looking at the invoices: $47. $63. $89.

Cost creep happens. It's not usually catastrophic — it's a gradual drift. Agents run more often than intended. A cron job fires five times instead of once. A specialist agent gets used for tasks that don't need specialist-level pricing. You're running on a premium model when a cheap one would do.

Cost control isn't hard. It's just consistent.

Model selection strategy: cheap for 90%, expensive for 10%

This is the single biggest cost lever. Most home service shops' work is straightforward, repetitive, rule-based stuff:

- Draft a follow-up email
- Schedule a crew
- Send a review request
- Generate a report
- Check a calendar

These tasks don't need Claude Sonnet. They need a cheap fast model. DeepSeek V4 Flash, Qwen3.5 Flash, or similar. They can handle 90% of your workload for a fraction of the cost.

The expensive models (Claude Sonnet 4, GPT-4o) matter for specific tasks:

- Pricing a complex change order
- Negotiating language for a tricky customer
- Architecting a multi-service web app

Keep expensive models on the specialist agents that need them. Run the ops agent, research agent, and marketing agent on cheap models.

Prompt efficiency

The agent reads every word you send it. Every extra sentence you add, every bit of context it carries, that's tokens. Tokens are money.

Be terse. Be specific. Don't over-explain things the agent already knows from its SOUL.md or memory.

Instead of: "Can you please draft a follow-up email for the Millers on Elm Street? Their kitchen project is marked as complete as of last Thursday. Dave was the lead carpenter. The project was originally priced at $26,882. They had a pretty standard remodel, no major changes. The review link is [url]. We've had good success with past customers leaving reviews within the first week. Please draft something that sounds personal and friendly, not corporate. And can you send it to [email]?"

Just say: "Draft a review request for the Millers on Elm St, send from Dave's number, use the standard template, send to [email]."

The agent has the context. You don't need to repeat it. Every word you type is a token the agent processes. Keep it tight.

Batching

Don't spawn five agents for five tiny related questions. Put all five questions in one prompt, get all five answers, done.

Instead of: "Ops agent, who's on the schedule today? Marketing agent, what's the review rate this week? Estimator, how many estimates did you draft yesterday? Research, what's the competitive pricing look like this week?"

Just say: "Daily briefing: today's schedule, this week's review rate, yesterday's estimate count, current competitive pricing comparison."

One agent. One conversation. One set of tokens. Way cheaper.

Credential pools

If you see rate-limit errors (which can spike your costs when the agent retries), set up credential pools — multiple API keys rotating so no single key hits the rate limit.

```
hermes auth add openrouter
hermes auth add openrouter   # adds a second key
```

The agent rotates between keys, never hitting the limit. You avoid the cost spike and the errors.

Cost ceilings

Every provider has a way to set spend alerts. OpenRouter lets you set a monthly cap. Anthropic lets you set spend notifications. Do this on day one.

If you don't, a misconfigured cron job can rack up hundreds of dollars overnight. I've seen it happen to small shops. Not common, but catastrophic when it does.

Monthly cost breakdown

Tier	Use case	Estimated cost
Light	Solo operator, 3-4 automations	$5–15/month
Medium	Small team, 10+ automations, some web app	$20–50/month
Heavy	Franchise with multiple locations, active marketing, full pipeline	$50–150/month

At the heavy tier, you're still at less than $2000/year. That's cheaper than one part-time admin ($36,000/year). The cost-benefit math doesn't work the other way even at heavy use.

What to watch for

- Unexpected spikes in one agent's spend — usually a cron job firing too often, or an agent stuck in a retry loop
- A cheap-model agent taking too long to do something an expensive model would do faster — maybe switching models is the cheaper choice when you count time
- Running the same agent on an expensive model when the tasks don't need it

Watch the weekly numbers. They tell you everything.

Chapter 27: Security & Best Practices

Home service businesses handle customer data. Names, addresses, phone numbers, payment info, project details. Some of it is sensitive — financial information, insurance details, maybe photos of a homeowner's property. Some could be embarrassing if it leaked — before renovation photos, say.

AI agents processing this data need real security. Not theoretical security. Practical security that keeps your data safe while still letting the agents do useful work.

API key management

Your providers (OpenRouter, Anthropic, OpenAI, etc.) authenticate you with API keys. Those keys give whoever has them access to your account — and to billing. Don't leak them.

Never commit API keys to git. If they're in a `.env` file (the standard place), add `.env` to your `.gitignore`. If they're in a config file, make sure that file is ignored too.

If you accidentally commit a key to git, rotate it immediately through your provider's dashboard. A leaked key is a leaked key until you rotate it, regardless of how fast you caught it.

Use a secrets manager for production (HashiCorp Vault, or a simple solution like Doppler). Don't put keys in source code anywhere, ever.

Command approval modes

Sometimes agents run shell commands. Sometimes those commands are risky — deleting files, running deployments, editing production data.

Hermes has three approval modes:

- `manual` (safest): prompt for every risky command. Nothing runs without you explicitly approving. Best for shops just starting out, or shops with sensitive environments.
- `smart` (recommended middle ground): an auxiliary AI auto-approves commands it judges as low-risk (read file, check status), prompts on high-risk ones (delete file, deploy code). Balances safety with speed.
- `off` / `--yolo` (dangerous): never prompt. Everything runs. Use only if you're really confident in what you're doing, and only in environments where the blast radius is contained.

For most home service shops, `smart` is the right choice. You're not running nuclear launch codes. But you don't want the agent deleting files without asking, either. The middle ground catches the important stuff.

Secret redaction

Hermes redacts things that look like API keys, tokens, and other secrets in tool output by default. Keep this on (`security.redact_secrets` in config).

The default is `true`. Don't turn it off unless you have a specific reason and you know what you're doing. Leaving it on means if an agent accidentally encounters a key, it doesn't echo it back into the conversation where it could leak through a screenshot or a log export.

VPS hardening

If you host the agent on a VPS, harden the VPS itself. The minimum:

- **UFW firewall** open only on ports 22 (SSH), 80 (HTTP), 443 (HTTPS). Everything else blocked by default.
- **SSH key-only access.** Disable password authentication. SSH is the single most targeted service on the internet — password auth gets brute-forced within hours of going live.
- **fail2ban** running to block repeated failed login attempts.
- **Automatic security updates** enabled (via `unattended-upgrades` on Debian/Ubuntu).
- **Regular backups** of anything you can't afford to lose.

Most VPS providers make this straightforward. A couple hours of setup, ongoing security that mostly just works.

Customer data handling

Decide what goes into agent memory and what stays out:

- **Patterns, not names.** "We have a lot of repeat customers in the 90210 zip code" is a useful pattern. "John Smith at 123 Main St last hired us in March" is PII that probably shouldn't persist indefinitely.
- **No payment info in memory.** Stripe handles payments. The agent handles everything *around* payments — status updates, invoice generation — but not the payment info itself.
- **Photos can be sensitive.** Some customers don't want before/after photos stored long-term. Respect that.

The agents don't need to remember everything about every customer. They need to remember the patterns that help them do their job.

If you're in a regulated industry (medical restoration, say, or anything touching health records), the memory hygiene chapter matters even more. The rules are similar but the stakes are higher.

Rate limiting: preventing runaway costs

A misconfigured cron job can fire hundreds of times a minute. That's runaway agent calls. That's hundreds of dollars in minutes.

Set rate limits on everything that can fire multiple times:

- Cron jobs: configure a minimum interval between runs
- Webhook handlers: validate the payload before processing
- Agent-spawned tasks: cap how many can run in parallel

And set cost ceilings at your provider. Even if something goes wrong, the cost ceiling catches you.

What could go wrong without these

The realistic worst case: a misconfigured cron job fires 1000 times an hour for a week. You get a $2,000 bill. No data breach. No exposed customer info. Just a bad bill.

The really bad worst case: an API key leaks from your git history, someone pulls it, runs a bunch of stuff on your account, racks up a ton of cost. Same basic problem, different flavor.

Neither is an existential risk for a small shop. But both are preventable with basic hygiene. And both will happen eventually if you skip the hygiene.

Do it right from day one. It's a couple hours once, a few minutes a month ongoing.

Chapter 28: The Compounding Advantage

Here's what actually happens over time with a home service shop running on an AI agent team.

Month 1: You set things up. You have the main agent, one specialist (the ops agent). You've built 1–2 automations from your scoped designs. Maybe a review request sequence. A daily briefing. Things work, but it's still kind of manual. You're getting used to delegating to the agent. You're learning to write better scopes.

Month 2: You add a second specialist, probably marketing. You've learned the rhythm. Your scopes are sharper — you catch edge cases before the build phase now. Your first automations are reliably producing results. You're adding a third automation. Maybe a web app is in early stages.

Month 3: You add the estimator agent. You've got three automations running, all three producing measurable value. You're writing scopes in 30 minutes where it used to take you 2 hours. Your team is getting used to working with the agents.

Month 6: You have the full team. The coder agent has built 2-3 web apps. You've built 6-10 automations total. Every agent has six months of operational memory. The ops agent *knows* your crew the way Marie once did — only more completely, and it never forgets.

Year 1: Your system is now mature. The automations are producing compound results. The review engine is generating reviews you can measure. The re-engagement engine is booking repeat customers you wouldn't have reached otherwise. The daily briefing tells you things you used to have to dig for.

The compounding that happens

Here's the thing that nobody tells you about when you start: the knowledge compounds.

Skills build on each other. The first email automation you built becomes a skill. The second email automation uses that skill. The third uses it and two more you've built. By the tenth automation, you're not starting from scratch — you're building on the foundation of nine prior automations.

Memory deepens. The ops agent has 12 months of operational memory. It knows your crew better than you remember them sometimes. It knows when things go wrong, why, and what worked. The estimator has a year of pricing data in its memory. It's not guessing at job costs — it's extrapolating from real history.

Scope quality improves. You're better at writing scopes now because you've written 20 of them. You catch the edge cases upfront. You encode the business rules explicitly. You write specs the agents can run on first try, without back-and-forth.

You're not the same owner you were at month 1. The agents aren't the same agents. Neither of you are starting from scratch anymore.

Costs decrease per output. Each automation reuses the skills and memory of all the previous ones. The tenth automation is cheaper to build than the second because it builds on the foundation. The agent runs faster because it's working from learned patterns, not figuring things out from scratch each time.

The end state

Six months in you have a shop that's measurably more efficient. A year in you've moved from "we've always done it this way" to "here's our operating system" — documented, codified, running.

Two or three years in, you have something that's genuinely valuable. A shop with:

- Documented operational knowledge that doesn't live in anyone's head
- Running automations that handle 80% of the operational work
- An agent team that gets sharper every week
- Custom web apps built for how your business actually works

This is the franchise value. When a buyer looks at your shop, they're no longer buying "the business John runs." They're buying "the business plus the system that runs it, plus the operational knowledge that's documented in it."

The shop becomes more valuable not just because you built it well, but because every piece of operational knowledge is documented, reusable, transferable. The buyer inherits a running system, not tribal knowledge.

The moat that nobody can copy

Here's the deepest insight: this system takes time to build. The operational knowledge compounds. The memory deepens. The skills interlock. You can't get there overnight unless you've done three years of work.

A competitor can buy the same software (Hermes). They can sign up for the same models. They can read this same guide. But they can't replicate your three years of accumulated operational knowledge, codified in your agents' memory and skills.

That's the real moat. It's not technology. Anyone can get the technology. It's the three years of you encoding how your business actually works, in a form that's reusable and transferable.

This is the whole point of the guide

The point wasn't to teach you AI. The point was to show you the skill of extracting your operational knowledge and encoding it in a way that lets running systems do the work for you.

The AI is the implementation tool. Not the foundation. The foundation is the operational knowledge.

You had the operational knowledge all along. The guide just showed you how to make it compound. That's the whole point.

Now: go build something. Start with Part I. Map your business. Find the binding constraint. Write the scope. Stand up the team. Put it in motion. Keep getting better.

The window is open.

Part V ends with you holding:

- A maintenance rhythm that fits into your normal week — not a separate "AI project" hat you have to put on
- Cost controls that keep your AI spend sane while the system keeps growing
- A measurable compounding advantage: your business is demonstrably better every quarter
- A transferable business asset — your operational knowledge lives in documented, running systems instead of in someone's head
- The confidence that what you've built compounds, not decays
- The understanding that the AI is the tool; the operational knowledge is the asset

---

Getting Started — Your First 30 Days

A day-by-day plan. Weeks 1-2 are business thinking; Weeks 3-4 are implementation. This ordering is deliberate — the implementation in Weeks 3-4 only works because the thinking in Weeks 1-2 happened first.

Week 1: Map & Diagnose (Days 1-7) — Business thinking, no software yet

- [ ] Day 1-2: Map your business lifecycle (Chapter 2 exercise) — one page, every touchpoint
- [ ] Day 3: Audit every function against the Four Losses (Chapter 3)
- [ ] Day 4: Mark each inefficiency E/S/A (Chapter 4)
- [ ] Day 5: Run the Extraction Interview on your TOP priority (Chapter 8)
- [ ] Day 6-7: Write a build-ready scope for that priority (Chapter 9 template)

Week 2: Stand Up Your Agent Team (Days 8-14)

- [ ] Day 8: Install Hermes Desktop, run setup wizard (Chapter 12)
- [ ] Day 9: Write your SOUL.md — encode how your business actually works (Chapter 13)
- [ ] Day 10: Connect Telegram so the system reaches you from job sites (Chapter 14)
- [ ] Day 11-12: Complete one real task end-to-end with your agent, validate it matches how you think

- [ ] Day 13-14: Install domain-relevant skills (permitting rules for your trade, estimating patterns, seasonal campaign cadences, etc.)

Week 3: First Implementation & Verification (Days 15-21)

- [ ] Day 15-16: Create your first specialist agent aligned with a real business function (recommend: `ops`; Chapter 15)
- [ ] Day 17: Seed the specialist SOUL.md with the domain rules from your scope
- [ ] Day 18-19: Point the agent at your first improvement design and run it end-to-end
- [ ] Day 20-21: Measure — did the improvement deliver the expected business result?

Week 4: Put It In Motion & Keep Improving (Days 22-30)

- [ ] Day 22: Schedule the first recurring operation (daily briefing, Chapter 22)
- [ ] Day 23-24: Implement second improvement from your prioritized queue
- [ ] Day 25-26: Wire in additional communication channels as the team needs
- [ ] Day 27: Review costs vs. business value delivered — target under $20/month
- [ ] Day 28: Re-run the Part I process map — has anything changed now that the system is live?
- [ ] Day 29-30: Plan Month 2 — what's next on the queue, based on the results you measured

---

Appendices (Planned)

Appendix A: Model Selection Guide (Implementation Reference)

- Current pricing for all relevant LLM providers
- Decision tree: which model for which business function
- Cost calculator template (monthly spend by usage tier)

Appendix B: Glossary

- Home service terms: trade lingo you'll see in examples (change orders, punch lists, rough-in, selections, scope of work, etc.)

- Agent terms: Skill, Memory, Delegation, Cron, Gateway, SOUL.md, MCP, Profile
- Restoration-specific example terms: IICRC, Xactimate, S500, S520, Cat 1/2/3, DASH, mHelpDesk (referenced in worked examples)

Appendix C: Troubleshooting

- "My agent forgot something" — memory troubleshooting
- "My agent is too expensive" — cost reduction checklist
- "My deploy failed" — common web app deployment errors
- "The automation isn't doing what I expected" — scope-vs-implementation diagnosis

Appendix D: Template Library

- SOUL.md templates (one per role)
- Scope template (full version with home service examples filled in; remodeling as the primary walkthrough)
- Extraction Interview question list
- config.yaml templates (basic, multi-agent, full orchestration)

Appendix E: Recommended Resources

- Trade-specific standards references (IICRC for restoration, building code for remodeling, EPA for HVAC — for skill authoring)
- OpenRouter docs
- Hermes documentation: hermes-agent.nousresearch.com/docs
- Community: Hermes Discord, home service industry forums
- Legal: AI advice disclaimer template (this guide is educational, not legal/insurance advice)

What This Guide Does NOT Cover

- Enterprise-grade platforms (Salesforce, ServiceTitan integration)

- Building AI models from scratch
- Native iOS/Android mobile development
- HIPAA/compliance-heavy workflows (medical restoration)
- Advanced ML operations (fine-tuning, custom RAG beyond retrieval basics)
- Large team coordination (50+ employees — different architecture)

Estimated Deliverable Specs

Attribute	Target
Total length	25,000-35,000 words
Format	Markdown (exportable to PDF/ePub)
Code examples	15-25 runnable examples
Diagrams	20-30 (process maps, architecture, agent flows)
Reader time	8-12 hours total, or pick parts à la carte
Prerequisites	Basic computer literacy for Parts I-III; no coding required to use the systems built

---

Open Questions for You to Decide

1. **Delivery format:** single document or modular (each part standalone)?
2. **Companion materials:** video walkthroughs? pre-built profile packages for sale?
3. **Pricing model:** free lead magnet? paid course? consulting upsell?
4. **Maintenance cadence:** model pricing changes quarterly minimum — who owns updates?
5. **Scope depth on web apps:** full build tutorial or overview + pointers?
6. **Restoration sub-niches:** (if we do a restoration-specific spinoff later, cover water-only or also fire/mold/biohazard equally?)
7. **Legal disclaimer:** needed? (Recommended — "not legal/insurance advice")

Next Steps

1. You review this scope, confirm the thesis and sequence feel right
2. Finalize table of contents — any chapters to add/remove/merge?
3. Begin writing Part I (Chapters 1-6) — this is the "why" and sells the thesis
4. Build the template library (Appendix D) in parallel — immediately useful on its own
5. Create one working example end-to-end before generalizing anything
6. Set up quarterly update cadence (model pricing, new Hermes features, new home service use cases)

This scope document defines boundaries and structure. The central thesis — knowledge-to-specification beats code-to-product — runs through all 28 chapters. Content decisions within each section are made during drafting. You retain final authority on all positioning.