GROWTH
WIZARDS

THE SCALE SPELLBOOK

BEGINNER MOVES, INTERMEDIATE TACTICS, AND THE EXACT PROMPTS THAT GET CLAUDE TO BUILD THEM WITH YOU.

WHY THIS GUIDE WORKS

Most guides ask you to read, nod, and eventually go build something. This one skips the middle step.

Every framework here comes with a CLAUDE PROMPT block: text you copy, paste into Claude (Claude Code if you're technical, claude.ai if you're not), and hand over. Claude asks you the two or three questions it actually needs, then does the work. It writes the file, builds the workflow, drafts the playbook, sets up the loop. You review, adjust, and ship.

This guide is split into two speeds on purpose. Each part opens with the foundation: the move you need before anything else works. Then it goes one level deeper, into the tactics that separate someone who's automated a task from someone who's automated a function. If you're new to this, start at the foundation and don't skip ahead. If you've already got the basics running, jump straight to the "Level Up" sections and the prompts underneath them.

Scale comes down to three repeatable moves. Automate what a machine can do. Standardize what a person can repeat. Crowdsource what a group can do better than either. This guide gives you the mental model for each, at both levels, then hands the execution to Claude.

How to use this: Read a section. Copy the prompt. Paste it into Claude. Let it build. Move to the next section only once the last one is actually running.

PART I: THE MENTAL OPERATING SYSTEM

Scale starts as a mindset shift before it's a system.

FOUNDATIONS

Know what role you're actually playing. An entrepreneur builds a repeatable, scalable, profitable business model, and is allowed to be wrong 99 times if the 100th attempt pays for all of them. An executive reduces risk and optimizes what already exists. A manager exists only because communication breaks down past a certain headcount. If you think your six-person team needs one, you're not a startup anymore. You're a small business. The test: at any moment, exactly one person should be unambiguously in charge.

The artisan trap. Doing everything yourself feels good. You see the result of your labor, you control the details. It's also a ceiling. You have 24 hours in a day, and a business that depends entirely on your hours isn't a business. It's a job you built for yourself. The fix isn't heroic effort. It's delegating things you're not the best at first, accepting the early dip in quality, and trusting the curve: one person is linear, a team is exponential.

Action beats perfection. Compare two founders. One makes 20 decisions a day at 80% accuracy: 16 good calls and 4 lessons learned, daily. The other makes one "perfect" decision a day at 95% accuracy: less than one good call and barely any lessons. After a month, the first founder has made roughly 480 good calls and absorbed 120 lessons. The second has made 28 good calls and learned from a mistake and a half. Speed of iteration beats theoretical correctness almost every time, in a business that hasn't found its ceiling yet. The only feedback that counts is the market's. Strong negative reactions at launch often correlate positively with eventual success, because people only get angry enough to email you when they feel the problem intensely.

Pick your game, then be honest about it. Status is zero-sum. For one person to rise, another falls. Value creation isn't zero-sum. It's a pie you grow, not divide.

Wanting money is a fine reason to build a company. Wanting to change the world is too. The only real mistake is claiming the second when you mean the first.

LEVEL UP

The three ingredients, and only one is irreplaceable. Every business decision draws on the same three resources: money, people, and time. Money and people can both be replaced. You can raise more, hire more, swap a vendor. Time can't. Once a week is gone, it's gone. That asymmetry should decide where you spend your own hours: on the handful of tasks that only you can do, and that create the most value per hour, right now. What counts as "highest value" for you will change as the business changes. Revisit the question monthly instead of assuming this month's answer still holds next quarter.

Happiness or glory, and it changes your whole strategy. There are two legitimate paths as an operator, and most people never explicitly choose between them, which causes a lot of quiet misery. One path optimizes for a good life: enough income, real time freedom, low stress, a business sized to serve you rather than the other way around. The other optimizes for building something big: more risk, more reinvestment, more of your identity tied up in the outcome, in exchange for a shot at outsized results. Neither is wrong. Trying to run the second path's playbook while actually wanting the first is where founders burn out chasing a version of success they never really wanted.

CLAUDE PROMPT: RUN MY TIME AND ROLE AUDIT

I want you to run a scale audit on how I actually spend my time.

1. Ask me to paste or describe everything I did in the last 5 working days: meetings, tasks, decisions, anything that took real time.
2. Classify every item into:
   - A: only I can do this
   - B: someone else could do this with training
   - C: this could be automated or systematized
3. Calculate what percentage of my time fell into each bucket.
4. If B plus C is over 50%, tell me plainly that I have a scale problem, and rank the top 5 items I should delegate or automate first, by time consumed times ease of handoff.
5. Then ask me two questions: "Are you optimizing for a good life or for building something big right now?" and "What monthly number would cover the lifestyle you actually want?" Use my answers to sketch one realistic small offer I could ship in under a week.

Be blunt. Don't soften the numbers if it's bad news.

PART II: AUTOMATION WITH CLAUDE CODE AND N8N

Automation is the most intuitive lever. Build the system once and it runs forever. Two tools cover almost everything a solo operator needs: Claude Code for anything code-shaped, n8n for anything workflow-shaped.

FOUNDATIONS: CLAUDE CODE

Claude Code is an agentic coding environment, not a chatbot. It reads your files, runs commands, makes changes, and fixes its own errors while you watch, redirect, or walk away. Six things separate people who get real leverage from it and people who get frustrated.

1. **Give it a way to verify its own work.** Tests, screenshots, curl commands. Without a way to check itself, Claude can produce work that looks plausible but doesn't actually function, and you become the only feedback loop.
2. **Explore, then plan, then build.** Don't skip straight to code for anything non-trivial. Use Plan Mode to separate "understand the codebase" from "write the change."
3. **Be specific.** "Add tests for foo.py" gets generic tests. "Write a test for foo.py covering the case where the user is logged out, and avoid mocks" gets the test you actually wanted.
4. **Feed it rich context.** Reference files directly, paste screenshots, hand it URLs, pipe data in from the command line.
5. **Manage your context window like a scarce resource.** Performance degrades as it fills up. Clear the conversation between unrelated tasks. If you've corrected Claude twice on the same problem, stop, clear it, and write a better initial prompt that bakes in what you just learned.
6. **A project instructions file gets read at the start of every session.** Put in it only what Claude can't infer by reading the code: commands it can't guess, house style rules that differ from defaults, repo etiquette, architectural decisions. A bloated instructions file gets skimmed, not followed.

Failure patterns to watch for: mixing unrelated tasks in one session, correcting the same mistake twice and trying a third time instead of restarting, an instructions file so long that important rules get lost in the noise, trusting output without verifying it, and asking Claude to "investigate" something with no scope or boundary.

FOUNDATIONS: N8N

n8n is an open-source, self-hostable workflow automation platform. Think of it like an alternative to the big-name automation tools, where you own the data and there's no per-task pricing ceiling. A workflow is a trigger followed by a chain of nodes and an action. Nodes come in five flavors: Trigger (a webhook, a schedule, an inbound email), Action (send an email, create a record), Logic (branching, merging), Transform (reshape or clean the data), and Integration (chat apps, spreadsheets, your CRM). Data moves between nodes as structured records, and you reference specific fields with simple expressions.

Five patterns cover most real business automation: bidirectional sync between two systems, data aggregation from multiple sources into one report, conditional branching on the shape of the data, retry with backoff for flaky APIs, and batch processing for volume. Whatever you're automating is almost certainly a combination of these five.

LEVEL UP: CLAUDE CODE

Give Claude specialized knowledge and specialized helpers. Beyond the base instructions file, you can define reusable domain knowledge files for things like API conventions or coding standards you use repeatedly, so Claude doesn't have to relearn them every session. You can also define subagents: narrowly scoped assistants Claude can delegate isolated jobs to, each with its own instructions and its own restricted set of tools. A security reviewer that only reads and searches code, and never edits it, is a common first one to build.

Session hygiene, in four moves. You can stop Claude mid-action without losing context. You can open a menu to restore an earlier state. You can ask Claude in plain language to revert its own last change. And you can wipe the conversation clean between unrelated tasks. If you've corrected the same problem twice, that's the signal to clear and restart with everything you just learned folded into the first prompt, rather than patching a conversation that's already gone sideways.

Run Claude without watching it. Once you trust a task, you can run Claude headless, meaning from a script or a scheduled job with no one at the keyboard, for one-off or batch work. A common intermediate move is a fan-out loop: point Claude at a list of files and have it apply the same transformation to each one, reporting success or failure per file, so a migration that would take you a day of manual editing runs unattended overnight.

Run a writer and a reviewer as two separate sessions. One session implements a feature. A second, completely separate session, with no memory of how the first one thought about the problem, reviews the change for edge cases and inconsistencies before you ever look at it yourself. This catches the "looks right but isn't" failure mode that a single session reviewing its own work tends to miss.

LEVEL UP: N8N

Build an agent, not just a pipeline. Beyond simple trigger-to-action workflows, n8n lets you build an AI agent node that can reason and choose which tool to use next. A common pattern for a support function looks like this: a message comes in, the agent decides whether to search a knowledge base for an answer, create a support ticket, or escalate to a human, and it only escalates when the customer is frustrated or the issue is genuinely complex. Give the agent a short, explicit rulebook (search before creating a ticket, escalate on frustration or complexity, always respond professionally) rather than a vague instruction to "help the customer."

Four rules that separate a workflow that survives production from one that doesn't. Always attach an error handler and log failures somewhere centralized. Store credentials in the platform's own credential system, never hardcoded inside a node. Scope every API key to the minimum permission it needs. And cache anything you fetch repeatedly instead of hitting the same endpoint on every run. On top of that, name every node clearly, document non-obvious logic with a note directly on the canvas, and keep one workflow responsible for one job. A workflow that tries to do five things becomes the one nobody dares touch six months from now.

CLAUDE PROMPT: SET UP MY PROJECT FOR CLAUDE CODE

I want to start using Claude Code properly on this project. Do the following:

1. Read my repo structure, package files, and existing docs.
2. Ask me anything you can't infer: preferred code style, test command, deploy process, any hard rules I have.
3. Write a short project instructions file covering only: commands you can't guess, style rules that differ from language defaults, repo and branch conventions, and any quirks of this environment. Do not restate things a competent engineer would already know from reading the code.
4. Propose 1 to 3 specialized subagents worth creating for this project, for example a security reviewer or a test writer, each with a name, a description, and a restricted set of tools it's allowed to use.
5. Show me the final files before writing them, so I can approve.

CLAUDE PROMPT: DESIGN MY FIRST AUTOMATION WORKFLOW

Act as an n8n automation consultant. I want to automate a repetitive task in my business.

1. Ask me: what's the task, what triggers it, what tools or systems are involved (CRM, chat, email, spreadsheet, etc.), and what should happen when something goes wrong.
2. Once you understand it, design the workflow as a labeled sequence of nodes, including an error handling branch.
3. Write the full n8n workflow as importable JSON if you can infer the node types and parameters, plus a plain-English, numbered setup guide I can follow inside the n8n editor for anything that needs manual configuration, like API keys or credentials.
4. Call out anywhere I'll need to plug in a credential or API key, and what to test before I turn the workflow on.

CLAUDE PROMPT: RUN A WRITER AND REVIEWER LOOP ON MY CODE

I want to catch "looks right but doesn't work" bugs before they ship. Run a two-pass review on the change I'm about to describe.

1. First, as the writer: implement or show me what you'd implement for [describe the feature or fix].
2. Then switch roles completely and act as an independent reviewer who did not write this code. Actively look for edge cases, race conditions, inconsistency with existing patterns in this codebase, and anything that looks correct but was never actually verified.
3. List every issue found with a specific line reference and a suggested fix.
4. Only after that review is clean, tell me it's ready to ship.

CLAUDE PROMPT: BUILD MY AI SUPPORT AGENT WORKFLOW

Help me design an n8n workflow with an AI agent that handles a chunk of my customer support or inbound questions.

1. Ask me: what are the 3 to 5 most common questions or requests I get, where does the answer to each one currently live (a doc, my own head, a spreadsheet), and what should trigger a handoff to a real person.
2. Design the workflow: trigger, the AI agent node, and the tools it can call (search a knowledge base, create a ticket, escalate to a human).
3. Write a short, explicit rulebook for the agent covering when to search versus escalate, and how to phrase a handoff so the customer doesn't feel dropped.
4. Flag anything in my current process that's too inconsistent to hand to an agent yet, and what I'd need to standardize first.

PART III: STANDARDIZATION

Automation replaces your hands. Standardization replaces your judgment. It turns what only you know how to do into something anyone can execute at your quality bar.

FOUNDATIONS

Take the case of a founder who's a brilliant closer and thinks the job is to sell as much as possible personally. That's short-term thinking. It caps the business at one person's bandwidth. When that founder finally hires salespeople, all of them underperform, because nothing was ever written down.

The delegation method that actually works has four phases, and none of them get skipped. Observation, for about two weeks: the new hire shadows every call, takes notes, doesn't intervene. Guided participation, another two weeks: they start executing under supervision, with immediate feedback after each interaction. Supervised autonomy, about a month: they run solo but you debrief daily and review recordings. Full autonomy: weekly check-ins, tracked metrics, continuous playbook improvement. The one fatal error is taking the task back, even for a minute, just this once. Every mistake you shield someone from is a mistake they'll make later, at higher stakes.

What to standardize first: your sales playbook (ideal customer profile, buyer personas, objections with rebuttals, scripts, pricing, competitive positioning), your visual design system, and your checklists.

LEVEL UP

Checklists come in three distinct types, and mixing them up is why most checklists fail. A do-confirm checklist is for verifying work that's already been done, after the fact, so nothing important got skipped in the rush. A read-do checklist is followed step by step, in order, while doing the task for the first time, most useful for onboarding. A communication checklist coordinates several people through something time-sensitive, like a launch, where the point isn't just completing tasks

but making sure the right person knows at the right moment. Build the wrong type for the job and people either ignore it or follow it too rigidly.

A design system isn't just colors and buttons. It has three layers. Foundations are the raw ingredients: your color palette, your type scale, your spacing system, your icon style. Components are what you build from those foundations, organized by complexity: simple elements like a button or an input, small combinations like a form field or a card, and full sections like a header or a navigation bar. Patterns are the reusable solutions built from those components for recurring situations: how navigation works across your product, how forms handle errors, how loading and empty states look. Most people stop at foundations. The leverage is in documenting the patterns, because that's what actually keeps a second designer or a contractor from reinventing your product's feel from scratch every time.

Templates worth having on hand, so nobody has to invent a format from scratch every time:

TEMPLATE	WHAT IT'S FOR
Project Brief	Defines scope and objectives before work starts
User Story	Specifies a feature from the user's point of view
Post-Mortem	Analyzes what happened after an incident
One-on-One	Structures a regular check-in with a team member
Retrospective	Drives continuous improvement after a project or sprint
Proposal	Standardizes how you pitch and price your offer
Onboarding	Gets a new hire or client running without you in the room

---

CLAUDE PROMPT: WRITE MY SALES PLAYBOOK

Help me write a standardized sales playbook I can hand to a new hire on day one.

1. Interview me first: who is my ideal customer, who are the buyer personas involved in a deal, what are my 5 most common objections and how do I currently answer them, what does my pricing and packaging look like, and who are my main competitors.
2. Write a playbook with these sections: ICP, Personas, Objections and Rebuttals, Call and Email Scripts, Demo Structure, Pricing and Negotiation Guardrails, Competitive Positioning, Tools and CRM Setup.
3. Flag any section where my answers were too thin to write a useful playbook entry, and tell me exactly what info you still need from me to fill the gap.

CLAUDE PROMPT: BUILD MY DESIGN SYSTEM, ALL THREE LAYERS

Build me a design system I can reuse across every product or page I ship, structured in three layers: foundations, components, and patterns.

1. Ask me for my brand's primary color or colors, or generate an accessible palette if I don't have one yet.
2. Foundations: define a color palette, 3 type sizes, a spacing scale, and an icon style.
3. Components: define 3 button variants, a form field, and a card, built from the foundations above.
4. Patterns: write short rules for how navigation, form errors, and loading states should behave, so they're consistent everywhere.
5. Output the foundations and components as a CSS file with custom properties, plus a one-page markdown reference explaining the patterns, so anyone I hire can apply this without asking me first.

CLAUDE PROMPT: TURN ANY PROCESS INTO THE RIGHT KIND OF CHECKLIST

I'm going to describe a process in my business that currently only works because I'm doing it or supervising it closely.

1. Ask me to describe the process end to end, including anything I do automatically that I might forget to mention.
2. Ask me whether this is something someone will verify after the fact, follow step by step the first time, or coordinate across several people during a time-sensitive event.
3. Based on my answer, build the matching checklist type: do-confirm, read-do, or communication, in markdown with checkboxes, organized by phase.
4. Tell me which single step in this checklist is most likely to go wrong in someone else's hands, and why.

CLAUDE PROMPT: BUILD MY CORE TEMPLATE LIBRARY

I want a starter set of templates I can reuse across my business instead of reinventing the format every time.

1. Ask me what kind of work I do and who I do it for.
2. Build me templates for: a Project Brief, a User Story format, a Post-Mortem, a One-on-One structure, a Retrospective, a client or sales Proposal, and an Onboarding checklist, adapted to my type of business.
3. Deliver each as its own short markdown file, plain enough that I could fill one out in under 10 minutes.
4. Tell me which of these 7 I'm most likely missing right now, based on what I told you about my business.

PART IV: CROWDSOURCING

The most counter-intuitive lever, and the one most solo operators skip entirely, is mobilizing a group of people, often strangers, to do what no individual or small team could do alone. It isn't magic. It's incentive engineering: aligning what a group of people want with an outcome you both benefit from.

FOUNDATIONS

It takes five main shapes. Crowd labor breaks a task into small pieces distributed across many workers. Crowd funding lets many small backers finance something before it exists. Crowd creation has users build part of the product itself, through templates, tutorials, or contributions. Crowd voting lets the group decide what rises to the top. Crowd wisdom aggregates many independent judgments into a better answer than any one person would give.

A community you build around your product typically moves through three stages. Pioneers: a small, high-touch group with direct access to you and fast iteration on their feedback. Growth: structure becomes necessary and culture can dilute if you don't manage it deliberately. Ecosystem: the group starts to organize itself, sub-groups form, and members create more value for each other than you do directly.

The five most common mistakes: treating the community as a marketing channel (people can tell), neglecting your first members, growing faster than your ability to maintain quality, letting conflict fester instead of addressing it, and trying to centralize everything instead of letting a healthy group run part of itself.

LEVEL UP

If you're building anything with two sides, know which kind of network effect you're actually relying on. A direct effect means more users simply create more value for everyone, like in a messaging app. An indirect effect means more of one side, say suppliers, creates more value for the other side, buyers. A data effect means more usage makes the product itself smarter over time. A local effect only matters within a specific geography or niche. Knowing which one applies changes

what you should measure: a direct-effect product lives or dies on total active users, while an indirect-effect product lives or dies on the ratio between its two sides.

The chicken-and-egg problem has five standard fixes. Subsidize one side to get it moving. Make the product genuinely useful on its own before any network exists. Target a narrow niche first instead of the whole market. Accept a slower start while you build up real density. Or sequence deliberately which side you recruit first, and don't try to launch both at once.

Once a community exists, it needs real reasons to keep contributing, not just a place to post. Structured mechanisms that work: recognizing top contributors publicly, giving early or active members access to something exclusive, involving your most engaged users in real product decisions, and running events, virtual or in person, that give the group a reason to show up together instead of just scrolling past each other.

If you're running anything platform-shaped, four numbers tell you if it's actually working. Liquidity is the probability that a listing, a request, or an offer actually results in a completed transaction. Match quality measures whether the connections your platform makes are good ones, not just frequent ones. Trust is the level of confidence participants have in each other, which you can influence through reviews, verification, and guarantees. Volume is simply the total value moving through the platform. Most people only track volume. The other three usually explain why volume is stuck.

CLAUDE PROMPT: DESIGN MY CROWDSOURCING STRATEGY

Help me figure out if and how crowdsourcing could apply to my business.

1. Ask me what my product or service is, what repetitive or large-scale task currently requires me or my team's direct labor, and whether I have any existing users or customers who could plausibly contribute: content, feedback, referrals, templates, code, reviews, or anything else.
2. Identify which type fits best, crowd labor, crowd creation, crowd voting, crowd wisdom, or crowd funding, and explain why in 2 or 3 sentences.
3. Design a minimal first version: who are the first 10 contributors, what incentive gets them to participate, how do you guarantee quality through redundancy or moderation, and what's the smallest possible pilot I could run in the next 2 weeks.
4. Flag the single biggest risk of this approach for my specific business. It's usually quality control or misaligned incentives.

CLAUDE PROMPT: DESIGN MY COMMUNITY ENGAGEMENT PROGRAM

Help me design a real engagement program for my community or user base, not just a place for people to post.

1. Ask me: how big is my community right now, what platform does it live on, and what have I tried so far to keep people active.
2. Propose a mix of mechanisms from these four: public recognition for top contributors, exclusive access for active members, involving members in real product or content decisions, and a recurring event that gives people a reason to show up together.
3. For each mechanism you recommend, tell me exactly what it would look like in my specific community and what it would cost me in time or money to run monthly.
4. Tell me which single mechanism would have the highest payoff for the least effort, given what I told you about my community.

PART V: INTEGRATION, THE GROWTH LOOP

The three paradigms compound rather than stack. Understanding a function well enough to delegate it leads to a good team. A good team leads to growth. Growth leads to more revenue, which funds more delegation, which builds an even better team, which drives more growth. Skip a step, delegating what you haven't mastered or automating what you haven't standardized, and you hit a plateau you can't diagnose, because you never learned the nuances that would tell you what's actually broken. Master before you delegate. Delegate before you automate. Automate before you crowdsource.

FOUNDATIONS

Measure the right thing at each stage. Retention before product-market fit. Growth after product-market fit. Efficiency, meaning acquisition cost against lifetime value, once you're scaling. Profitability at maturity. Watch out for vanity metrics: downloads without retention, followers without engagement, traffic without conversion. Numbers that feel good and predict nothing.

Think in flow, not stock. "We made $1,000 this week" is a snapshot: flat, satisfying, and uninformative on its own. "We doubled last week's number" is a trend: it tells you direction. Reframe every metric you track as a rate of change, not a static total.

Retention isn't a nice-to-have. It's the whole point. The amount of work it takes to train someone to your standard is the amount of value you lose if they leave. Pay well, offer real growth and ownership, and run one-on-ones that actually ask how someone is doing rather than just checking status.

Your job shrinks to three things as the company scales: never run out of money, define the vision, and recruit the right people. Everything else should be delegable.

LEVEL UP

Not every process deserves the same amount of attention, and sorting them takes two minutes. Plot every recurring process in your business on two axes: how much impact fixing it would have, and how much effort that fix would take. Anything

high impact and low effort gets done this week. Anything high impact and high effort gets planned as a real project. Anything low impact gets ignored or batched for later, regardless of how easy it looks, because easy wins that don't matter are still a distraction. Most people default to fixing whatever's most annoying that day. This matrix forces you to fix what's actually expensive instead.

Decisions should sit with whoever has the most relevant information, not whoever has the most seniority. As you delegate and standardize more of the business, information naturally spreads out across more people. The habit that keeps decision quality high through that spread is pushing decisions down to whoever is closest to the relevant facts, and reserving your own judgment for calls that require seeing across the whole business at once. A founder who insists on making every call personally, long after other people have better information than they do, becomes the actual bottleneck they were trying to avoid by staying hands-on.

Your team's rules change by stage, and applying the wrong stage's rules is its own kind of failure. Before product-market fit, stay small, five to seven people, skip titles, skip formal hierarchy. Every extra person is fixed cost against a shrinking runway. After product-market fit, run on total transparency, make sure everyone actually understands the plan and their specific role in it, and review pay more often than once a year, since a startup's market rate moves faster than an annual cycle. Once you're scaling, accept that the people who got you here aren't automatically the people who get you further. That's not disloyalty. It's a different skill set, and the honest move is being transparent about what's changing while offering real growth paths to anyone willing to grow with the company.

CLAUDE PROMPT: RUN MY WEEKLY SCALE REVIEW (SAVE THIS ONE AND REUSE IT EVERY WEEK)

Act as my weekly scale review copilot. This is a recurring prompt I'll paste in every week. Treat each run as a checkpoint, not a one-off.

Each time I run this:

1. Ask me for this week's key numbers, whatever I'm tracking: revenue, retention, pipeline, or anything else that applies to my stage.
2. Ask me: what's one thing I did myself this week that someone else could have done? What's one process I repeated manually that could become a checklist or an automation?
3. Tell me plainly whether any of my tracked metrics look like vanity metrics, meaning they feel good but don't predict outcomes, given my current stage.
4. Reframe my top 3 numbers as a rate of change compared to last week, not a static total.
5. Plot the top 3 problems I mention against impact and effort, and tell me which one actually deserves this week's attention.
6. Give me exactly one action per paradigm, automate, standardize, and crowdsource, to do before next week's review. Make each one concrete enough that I could start on it in the next 10 minutes.

Keep this short. I want a 5-minute reality check, not an essay.

PART VI: STAYING STEADY IN A DOWNTURN

Growth means speed, and speed means the occasional accident. A mistake that would have cost you $10,000 last year at $1 million in revenue can cost you $500,000 this year at $30 million. Downturns aren't an anomaly of growth. They're a natural consequence of it.

FOUNDATIONS

The mindset that gets you through it has three parts. Absolute pragmatism: zero margin for optimism you haven't verified. Constructive attention: actively asking what you can do today to make sure the business is still standing tomorrow, rather than sitting with passive anxiety. And a shorter planning horizon: weekly plans and frequent cash checks instead of quarterly roadmaps.

Pricing works differently under pressure than it feels like it should. A downturn is often the right moment to hold or even raise your prices rather than discount, because money moves less and slower across the market as a whole, but the people who still have it end up with relatively more of it. Focus your offer on the customers who can still say yes, rather than chasing the shrinking middle with a lower price.

Marketing in a downturn has one job: don't disappear, and don't be tone-deaf. Everyone floods the market with content when things get hard, and everyone burns out at roughly the same rate, which means quality marketing matters more, not less. Four ingredients carry it: authenticity, tact, sensitivity to the moment, and a return to your actual core value instead of an opportunistic pivot.

LEVEL UP

Watch for three levels of guilt that quietly sabotage good marketing decisions during hard times. Guilt about making money at all. Guilt-driven distraction into unrelated "helping" that isn't your actual business. And a broader doubt about whether your work matters right now. The honest answer to all three is the same: the best thing you can do in a downturn is keep building something that works and solves a real problem for the people who need it.

Shift your marketing weight toward the customers you already have. Acquisition marketing gets more expensive and less reliable in a downturn, because everyone's budget is tighter and everyone's attention is more guarded. Retention marketing, meaning the effort you put into keeping and growing existing customers, gets relatively cheaper and more reliable at the same time, because those people already trust you. Before cutting spend across the board, check whether the honest move is simply rebalancing it: less toward cold acquisition, more toward the accounts and customers who already know your name.

CLAUDE PROMPT: BUILD MY DOWNTURN MARKETING PLAYBOOK

Help me build a marketing playbook I can activate fast if revenue drops. This is about marketing and positioning only, not legal or headcount decisions.

1. Ask me: what would a 30%, 50%, and 70% revenue drop look like for my business.
2. For each scenario, help me draft: whether my pricing should hold steady, rise, or shift focus to a specific segment and why, how my marketing budget should shift between acquisition and retention, which of my current marketing activities to double down on versus pause, and how I'd keep communicating with existing customers so I don't go quiet at the worst time.
3. Write me a one-page plan I could act on within a day if revenue drops suddenly, focused entirely on marketing moves I can make myself.
4. Tell me the single blind spot most businesses like mine miss when planning for a downturn.

YOUR 30-DAY SCALE PLAN

Three questions before you close this guide. Where are you right now: before product-market fit, after it, or already scaling? What's your actual bottleneck? Which of the three paradigms have you been quietly neglecting?

Pick one concrete action per paradigm, due this week, not this quarter. Then don't run this guide once. Run it as a loop.

CLAUDE PROMPT: BECOME MY SCALE CO-PILOT (THE MASTER LOOP, SAVE THIS ONE)

I want you to act as my ongoing scale co-pilot, using the three paradigms of scale: automation, standardization, and crowdsourcing.

First session:
1. Ask me where I am (before product-market fit, after it, or scaling) and what my single biggest bottleneck is right now.
2. Ask which paradigm I've been neglecting and why.
3. Help me pick one action per paradigm to complete in the next 7 days, each specific enough to start in the next 10 minutes.
4. Write these 3 actions, plus this week's key metric, into a short markdown file called scale-log.md, dated today.

Every time I come back and say "weekly check-in":
1. Read scale-log.md, or ask me to paste last week's version.
2. Ask what happened with last week's 3 actions: done, partial, or skipped, and why if skipped.
3. Update the log with this week's outcome and 3 new actions.
4. Once a month, step back and ask whether my actual bottleneck has changed. If it has, help me re-diagnose before assigning new actions.

Be direct. If I keep skipping the same type of action, call it out by name instead of quietly reassigning easier ones.

Scale isn't a finish line. It's a loop you run on purpose, on a schedule, with Claude doing the parts that don't need to be you.

Questions, or want a hand implementing any of this directly? Reach me at alex@growthwizards.biz.