Member-only story

How to Vibe-Code a Profitable App

A comprehensive guide to essential parts of both the business and the app implementation.

29 min read · Just now

Gunnar Östberg

Listen Share More

This is a long variant of the article, a detailed guide for readers who want the full reasoning, not a summary. If you would rather read a shorter version with a conceptual overview, read this story instead.

Introduction

You may have tried vibe coding yourself and have already developed one or a few apps, dreaming of selling them and earning a side income. You might think you do not have to do much planning or work now that AI technology has become so good. In this story, you'll see that it might not be quite so easy, although you may have great help from AI.

Context

Andrej Karpathy coined "vibe coding" on the 1. Feb. 2025:

> *There's a new kind of coding I call 'vibe coding', where you fully give in to the vibes, embrace exponentials, and forget that the code even exists. It's possible because the LLMs are getting too good.*

With the rise of vibe coding, where AI and natural-language prompts are used to generate working applications, a long-standing bottleneck has been removed. Programming skill is no longer the primary limiting factor for turning an idea into something that actually runs.

This changes where the real work lies

When implementation becomes easier, the difficulty does not disappear. It moves. It moves upstream, toward problem selection, needs analysis, and deliberate design decisions. It also moves downstream, toward marketing, positioning, sales, and operations.

In other words, coding is no longer where most projects fail. Judgment is.

Quality, therefore, matters more, not less. Almost anyone can now generate something that kind of works. Very few can build something that works reliably, explains itself clearly, and feels intentional enough for people to pay for.

Do the work. Right.

The door is open. You should absolutely walk through it. If you don't already know software development, you will learn. Use AI wherever it is relevant and helpful. Be curious and inquisitive. But throughout all of this, focus on quality. And above all, start in the right order.

Method

What many people do, especially when excited by new AI tools, is skip needs analysis, requirement definition, architectural thinking, and business validation, and jump straight into building the app.

That order is wrong.

To use vibe coding to build a revenue-generating, ultimately profitable app that can sustain a business, you need to follow a sequence of deliberate steps. Each step transforms the idea into a more defined and more valuable state. Each step is guarded by a gate that asks a concrete question. If the answer is no, you stop, pivot, or discard the idea before investing more time and energy.

The early gates are intentionally analytical. Their purpose is not to slow you down, but to prevent you from wasting months building and marketing ideas that should have been killed much earlier.

The gate sequence

1. **Problem selection (founder–problem fit)**
   Gate: Have you selected a real, valuable problem, and are you personally motivated and well-positioned to pursue it?

2. **Quality of problem understanding**
   Gate: For your defined ideal customer profile, is your understanding of the customer's practical, internal, and external needs deep enough that your solution choices are deliberate rather than accidental?

3. **Desk validation**
   Gate: Is the app business idea feasible, differentiated, and economically plausible?

4. **Market validation (demand validation)**
   Gate: Are real customers actually paying for it?

5. **MVP build**
   Gate: Does the product deliver real value in real use?

6. **Go-to-market scaling**
   Gate: Can you acquire customers in a repeatable way?

7. **Product scaling**
   Gate: Do users stay, and does the product improve over time?

8. **Re-architecture**
   Gate: Can the system sustain growth without collapsing under its own weight?

9. **Process management**
   Gate: Can sales, support, and delivery run without heroics?

10. **Product management**
    Gate: Is continuous improvement built into the business's structure?

Step 1. Problem Selection

Finding and defining a good business idea is not mysterious, but it does require effort and honesty. A viable app business is always based on a real problem experienced by real people.

The first step, therefore, is not about features or technology. It is about identifying a genuine need, understanding who experiences it, and deciding whether it is a problem you are willing and able to work on over the long term.

Finding a problem can be done online using AI or by talking to people. Ensure you understand the user well by using empathy and “walking in their shoes”.

A useful signal is existing competition. If no one is attempting to solve a problem, it is often because the problem is not valuable enough. Competition does not invalidate an idea. On the contrary, it often confirms that demand exists. Differentiation comes later.

The strongest starting point is when you experience the problem or pain point yourself and understand the domain firsthand. That gives you insight, empathy, and motivation, all of which matter far more than clever ideas.

The extraction mindset

Do not begin by inventing ideas and then trying to convince the world they are important. Instead, extract existing demand.

Look for explicit signals in forums, communities, reviews, and social media. People frequently say things like “I wish there was an app for…” or “Why isn’t there a tool that…”. These are not idle complaints. They are expressions of unmet demand.

You can also study what investors and accelerators publicly seek. These lists are the result of prior market research and are another form of demand extraction.

AI is particularly useful here. It can scan hundreds of discussions, reviews, or complaint threads and surface recurring patterns far faster than manual reading.

Founder–problem fit

Even if a problem is real, you should not pursue it unless you are genuinely motivated to do so.

Ask yourself, honestly, whether you can imagine working on this problem for years rather than weeks. Ask whether you have the time, energy, and persistence required. Ask whether you can clearly envision an app-based solution, and whether you have some insight, experience, or access that others lack.

The distance between “I have an idea” and “I have a profitable business” is measured in sustained effort, not inspiration. Success is about purpose and passion, inner drive, grit, and resilience. Enthusiasm or excitement is not enough. The idea and problem you chose to work with should align with your values, what you’re curious about, and what you’re willing to work for.

B2C and B2B problem discovery

For business-to-consumer ideas, scale matters. The economics usually require large numbers of paying users. Use AI-assisted search to identify many potential problems, then evaluate them based on their severity and prevalence.

For business-to-business ideas, problems are rarely discussed openly. They must be uncovered through conversations. That means interviewing decision-makers, running structured discovery meetings, or facilitating workshops where subject-

matter experts can surface and prioritize real issues. AI can help you design interview guides and synthesize notes, but it cannot replace access to people who actually make decisions.

Choosing one idea

Once you have several candidates, choose one. Only one.

This feels risky because it seems safer to keep options open. In practice, spreading your attention across multiple ideas prevents depth. You cannot validate several ideas with the same rigor you can apply to one.

Choose the idea that combines real pain, a sufficiently large market, your motivation, and your unique position to solve it.

Gate 1

At this point, ask yourself plainly whether you have selected a real and valuable problem, and whether you are personally motivated and well-positioned to pursue it. If the answer is yes, continue. If not, stop and start again.

Step 2. Problem Understanding

This step is about the quality of your understanding. If you do not understand the problem deeply, you will build the wrong solution, even if you build it quickly.

The output of this step is clarity. Clarity about who the product is for, what problem it solves, and what conditions must be true for the solution to work.

Defining the ICP

Begin by defining your ideal customer profile. This is not “everyone with the problem”. It is the specific group that feels the pain most strongly, has the ability and willingness to pay, and is reachable through channels you can realistically access. Starting broadly means you please no one deeply.

A narrow ICP simplifies everything. It sharpens marketing, clarifies feature decisions, and accelerates learning. You can always expand later.

Describe your ICP concretely. For consumers, this includes behaviors, habits, and where they spend time. For businesses, it includes company size, industry, roles involved, and buying cycles.

Modeling the user's process

If the solution is obvious, state it. If it is not, you need to model the user's current process.

A proper needs analysis focuses on the user's reality, not on the app. Map the current process step by step. Identify where information is used, where decisions are made, and where friction occurs. This is the "as-is" model.

Then design the improved process, the "to-be" model, where your app removes friction and adds value. The gap between these two models defines what you actually need to build.

Also consider stakeholders, goals, constraints, and the domain vocabulary. At this stage, no technology is involved. AI can be extremely helpful for structuring and documenting these models.

Ensure that an app product will solve the user's entire problem, not just part of it. A partial solution often has zero value, because the user still has to do the hard part manually.

Understanding the full problem

Solving the problem means more than addressing a surface need. It also means understanding how the problem feels to the user and why it matters to them.

In practical terms, this often involves three levels. There is the external problem, which is the practical task the user struggles with. There is an internal problem, which concerns frustration, anxiety, or uncertainty. And there is a deeper philosophical dimension that relates to values and identity.

You do not need to use this language explicitly in the product, but you should understand it. When you do, your later marketing becomes precise rather than generic.

Examples:

- External problem: "I need to track my expenses."
- Internal problem: "I feel anxious because I don't know where my money goes."
- Philosophical: "I should be in control of my financial life."

Data and feasibility

No solution is viable if its data requirements are impossible.

Ask where the data comes from, whether it is legally accessible, whether it can be obtained reliably and at acceptable cost, and what happens if the source changes or disappears. An idea that depends on fragile or inaccessible data is not feasible, regardless of how attractive it sounds.

PESTLE check

A brief feasibility scan across political, economic, social, technological, legal, and environmental dimensions helps surface risks early. This is not an academic exercise. It is a way to identify deal-breakers before they become expensive mistakes.

Use AI to research and surface risks that kill businesses:

- **Political**: Will regulation help or hurt you? Is the political climate stable?
- **Economic**: Can your target customers afford to pay? What happens in a recession?
- **Social**: Are cultural attitudes moving toward or away from your solution?
- **Technological**: Do the required technologies exist, and are they mature enough?
- **Legal**: Are there compliance requirements, licensing needs, or liability issues?
- **Environmental**: Are there any sustainability concerns affecting your business model?

The PRD

Document what you have learned in a product requirements document. This is not a detailed technical specification. It is a thinking tool that captures the problem, the ICP, the solution concept, user journeys, success criteria, constraints, assumptions, and open questions.

A useful practice: Have your PRD reviewed by multiple AI models. Ask each to identify gaps, question assumptions, and suggest what a skeptical engineer would want to know. Then iterate.

A well-written PRD becomes valuable context for AI coding later. The clearer your thinking, the better the output you get from AI tools.

Gate 2

Now ask whether your understanding of the problem is deep enough that your solution choices are deliberate rather than accidental. If the answer is yes, proceed. If not, deepen your analysis.

Step 3. Desk Validation

Desk validation is about plausibility. Before investing in development, you want to know whether the idea could work as a business.

Product definition

Translate the PRD into a concrete product concept. Define what is included in the first version and, just as importantly, what is excluded. Discipline here means subtraction. Your first instinct will be to include too much. A focused product that does one thing excellently beats a bloated product that does many things adequately.

The product is not just the software app; it is also the promises that come with the solution. Outline them, as they are part of your value proposition and your competitive differentiation. They also define your operational requirements. More about the promise in the next step.

Competition analysis

Study competitors and alternatives, including manual workarounds and the option of doing nothing. Understand pricing, positioning, and customer complaints.

Market analysis

Estimate market size realistically.

- **TAM** (Total Addressable Market): Everyone who could theoretically buy your product
- **SAM** (Serviceable Addressable Market): The portion you can realistically reach
- **SOM** (Serviceable Obtainable Market): The portion you can realistically capture in the near term

Be honest. Large theoretical markets mean little if you cannot reach them. Small markets can support great businesses if unit economics are strong.

Pricing

Set pricing based on customer value rather than your costs. Then estimate unit economics, including customer acquisition cost and lifetime value. These numbers will be imperfect, but writing them down forces clarity.

Common pricing models for apps:

- **Subscription**: Monthly or annual recurring fee
- **Usage-based**: Pay per use, transaction, or resource consumed
- **Freemium**: Free tier with paid upgrades
- **One-time purchase**: Single payment for perpetual access
- **Tiered**: Multiple packages at different price points

Choose a pricing model that aligns with how customers get value. If the value is continuous, subscription makes sense. If the value is episodic, usage-based may work better. The choice of pricing model is also connected to the go-to-market model.

Unit economics

Ensure the numbers work:

- **Customer Acquisition Cost** (CAC): How much does it cost to get one customer?
- **Lifetime Value** (LTV): How much revenue does one customer generate over time?
- **LTV/CAC ratio**: Should be at least 3:1 for a healthy business

If it costs you $100 to acquire a customer who pays $10/month and churns after 5 months, your LTV is $50, and your ratio is 0.5:1. That business loses money on every customer.

Run these numbers with realistic assumptions. If you don't have data yet, estimate CAC by researching industry benchmarks, calculating expected ad costs (cost per click × the number of clicks needed per conversion), and factoring in your time for direct outreach. For LTV, be conservative and estimate LTV based on your pricing and realistic assumptions about customer retention. Note your assumptions explicitly. You'll validate these estimates in the next step.

Positioning and differentiation

Finally, formulate a simple positioning statement that immediately clarifies who the product is for, what it does, and why it matters.

Differentiation can come from many angles:

- **Feature differentiation**: You do something competitors don't
- **Experience differentiation**: You do the same thing, but better or easier
- **Price differentiation**: You offer similar value at a lower cost
- **Niche differentiation**: You serve a specific segment that competitors ignore
- **Integration differentiation**: You connect with tools your users already use
- **Speed differentiation**: You deliver faster results
- **Trust differentiation**: You offer guarantees or transparency that others don't

You need at least one clear differentiator. "We're like X but better" is *not* a form of differentiation. Specificity matters.

A simple kill sheet

At this stage, you should stop if you cannot identify a clear differentiator, if the data dependency is fragile, if the economics do not work even under optimistic assumptions, or if the scope cannot be reduced without destroying value.

Gate 3

If the idea is feasible, differentiated, and economically plausible, continue. Otherwise, stop.

Step 4. Market Validation

Market validation asks a single question: Are real customers willing to pay?

The form of evidence depends on context. For B2B, this often means paid pilots, deposits, or letters of intent. For B2C, it may involve conversions, pre-orders, or early subscriptions, sometimes supported by a thin but usable prototype.

The important thing is evidence, not enthusiasm.

Create a simple sales or landing page that communicates the problem, the solution, and a clear promise. Be honest about what exists and what does not. Drive relevant traffic and observe behavior.

If people sign up, talk to them. Listen more than you speak. Ask about their situation, what they have tried before, and what would happen if they did nothing. Then ask directly whether they are willing to move forward.

Low response is also data. Pivot quickly.

Try the following approach to sell before you build

Securing orders from real customers is the only way to know that the design and development won't be a waste and that the app can be profitable.

This step separates winning approaches from losing ones. The winners go: Money → Product → Scale. The common but losing approach is: Product → Money → Scale. Selling before building feels uncomfortable. Do it anyway.

Prepare for lead collection

To collect email addresses, create a sales page using an AI-powered web page builder. Ask it to build an app to solve the problem, but limit it to a mock-up and include features for a combined promise-and-squeeze page. It looks like the product demonstrates its value, but it aims to capture people's interest; only those who value the app sign up.

Your page should communicate:

- The problem you solve (speak to the internal and external problems)
- The solution you offer (show enough to be credible, not everything)
- Social proof, if you have any (testimonials, logos, numbers)
- Value communication
- A clear call to action (sign up, book a call, pre-order)
- What happens next (set expectations)

Share the link to your squeeze page on all your social media platforms, relevant communities, and anywhere your ICP spends time.

Checklist before launch:

- You have an attractive and credible mockup that demonstrates value
- The squeeze page is well-designed and conversion-focused
- You have a plan to drive relevant traffic
- There is a clear agreement model and communication with the customer about the development process and delivery

The promise

When selling an app using a mockup and a squeeze page, you must be clear and honest in your communication with the customer to build trust and reduce buyer uncertainty. The wording of the promise should focus on what the customer can expect, how the process will work, and the results or value the app will deliver upon completion. The promise should be concrete and realistic, not unrealistically exaggerated, and preferably describe the entire development and delivery process.

Examples of promise wording could be:

“With our app, you will get a digital tool that facilitates X and Y. We will start with a mockup and ongoing presentations so that you are involved in the development. Delivery will take place according to the agreed schedule, and we offer [guarantee/customer satisfaction policy] to ensure your peace of mind.”

Having a guarantee or some form of security measure is common and recommended, so customers can feel confident buying even though the app isn’t fully developed yet. The guarantee needs to be carefully defined in writing so that both parties know what applies in the event of any misunderstandings or problems.

To encourage the customer to buy, you can use several methods to provide reassurance:

1. **Clear description of the process and terms**: Describe each step, the timeline, what is included, and what may be additional services. This provides transparency and clear expectations.

2. **Written agreement**: Always have a clear agreement that defines responsibilities, payment terms, delivery, and how issues such as bugs or delays will be handled. This provides legal protection for both parties.

3. **Warranty or satisfaction guarantee:** You can offer a warranty, a voluntary commitment to maintain a certain level of quality or functionality for a specified period. The warranty should be clearly stated in the agreement with terms and conditions and a time period. This can also serve as a "satisfaction guarantee," giving the customer the opportunity to receive compensation or corrective action if there are problems.

4. **Payment model**: Many use a partial payment at the time of advance booking, with the remainder due upon delivery, to reduce the customer's risk.

5. **References or social proof**: If possible, show previous work, reviews, or customer cases to strengthen credibility.

Make ads

Create simple ads that speak directly to your ICP's pain points. AI can help generate ad copy quickly.

The process:

1. Use AI to generate multiple versions of ad copy focused on the problem and promise
2. Use AI to generate or select images that evoke the emotional state of your customer (before) and the desired state (after)
3. Set a small test budget to see what resonates
4. Track which messages drive sign-ups or calls, not just clicks

Test several angles, A/B tests. You don't know what will work until you try. Let the data decide, not your assumptions.

Collect leads

Drive traffic to your page through:

- Social media posts in relevant communities
- Direct outreach to people who match your ICP
- Content that demonstrates your expertise on the problem
- Paid ads with small test budgets

The goal is to get enough interest to validate demand, not to build a huge list yet.

Discovery calls

If you get sign-ups, book discovery calls. These conversations are where you learn about the prospect's specific situation and confirm that your solution is a good fit.

On a discovery call:

- Listen more than you talk
- Ask about their current situation and pain
- Ask what they've tried before
- Ask what would happen if they don't solve this
- Explain your solution and how it addresses their needs
- Be honest about what's built and what isn't
- Ask if they're ready to move forward

What constitutes enough validation to proceed?

For B2C, aim for a landing page conversion rate of 5–10% (sign-ups from visitors) and, ideally, several preorders or paid commitments.

For B2B: three discovery calls that lead to commitments or letters of intent are a strong signal. If your landing page converts at less than 2%, or if nobody books calls, or if nobody commits after calls, that is also a signal.

Pivot or iterate within days, not months. The specific thresholds matter less than the principle: real evidence of willingness to pay, not just polite interest.

Build and maintain a community

Pre-sales support is part of validation. Create a space (e.g., an email list, community forum, or chat group) where interested prospects can stay connected throughout the development process.

This community becomes:

- A source of feedback and feature requests
- A test group for early versions

- Your first promoters when you launch
- A moat against competitors

Keep them engaged with updates on your progress. Make them feel like insiders. Their investment of attention now increases their likelihood of paying later.

Order handling

When someone commits:

- Send a clear confirmation of what they're buying
- Collect payment (full or partial, depending on your model)
- Set expectations for timeline and next steps
- Document everything

Keep your promises. Under-promise and over-deliver. Your first customers are your most important references.

Gate 4

Have customers made a real economic commitment, cash paid, deposit made, or binding agreement, such that building the product is no longer based on hope or assumptions?

Step 5. MVP Build

Now you build a real product. Not a proof of concept or a mockup, but something that delivers value in real-world use. A PoC only proves something is technically possible. An MVP proves that something is valuable enough for people to use.

AI will accelerate development, but it does not replace responsibility. You must act as product owner and decision-maker. You decide what to build, how it should work, and what quality is acceptable.

Specify requirements clearly. Everything you leave unspecified will be decided implicitly by the AI. Design a simple architecture using well-documented, mainstream technologies. Build incrementally. Test continuously. Use version control (e.g., git) from the start.

Deployment and basic operational setup should be simple but real. Monitoring, backups, and security are not optional.

AI assistance

AI will assist you at every step of the project, dramatically improving both the process and the results. Regarding system development, there are different strategies depending on your programming and system development competence, as well as the capabilities of the tools you use.

- **Vibe coding** is the entry-level approach that lets users generate code purely from natural-language prompts to LLMs, without reviewing, editing, or understanding the code — ideal for non-programmers building prototypes. You describe the "vibe" or desired outcome, evaluate results via execution, and iterate by feeding back errors, as popularized by Andrej Karpathy in 2025.
- **AI agentic coding** is when autonomous AI agents take on more tasks, access codebases, run commands, and iterate independently toward goals, often using tools such as CLI agents (e.g., Claude Code, aider). It builds on vibe coding but adds structure, such as version control and basic testing, to produce more reliable outputs, making it suitable for intermediate users.
- **AI-assisted manual coding** lets experienced developers leverage AI for targeted help— such as autocomplete, refactoring, or chat-based generation—while retaining full control, review, and oversight over production code. This highest skill level combines human expertise with AI-enhanced IDEs.

Choose the type of tool and specific tools carefully. This article doesn't tell you which tools to use because the choice depends on many factors, such as the app's target platform (e.g., web, Android, iOS), your skills, and your app requirements. The best tools are also continually changing, almost from week to week.

The role you must play

Going to a programmer (or an AI coding assistant) with an idea and expecting them to figure everything out is like going to a carpenter with a house idea and expecting them to build it. It rarely works for anything beyond the simplest projects.

You need to be the product owner. You need to be the architect of decisions. The AI handles syntax; you handle semantics. The AI generates code; you specify what code to generate.

This means:

- You define what to build (requirements)
- You decide how it should work (design)
- You verify that it works correctly (testing)
- You determine what's good enough (quality)

The AI accelerates execution. It does not replace thinking.

Requirement Specification for Design and Development

For vibe coding, it is very important to specify well what you want. Everything you don't specify carefully, the AI will choose for you based on statistical assumptions. Sometimes you want that, most often not.

For instance, you may have no clue what a modern and popular graphical user interface with an attractive visual appearance looks like — you can trust the AI for that. However, you should explicitly specify the features and all other aspects of the user experience.

Specify with:

- **User stories**: "As a [type of user], I want to [action] so that [benefit]."
- **Use cases**: Detailed descriptions of how users interact with the system
- **Feature descriptions**: What each feature does, including edge cases
- **Acceptance criteria**: How you will know when something is done correctly
- **Non-functional requirements**: Performance, security, accessibility, etc.

Use AI to work out an information model, perhaps a class model, a data structure, and more from your PRD.

Ensure that you build with data security and all laws and regulations in mind from the outset. This is very important for a successful, profitable app. GDPR, data retention, user consent, encryption at rest and in transit — these are not afterthoughts.

Design

Design involves sketching and modeling, transforming requirements into goals for realization, functions into experiences, utility into joy, and usability into a desire to use. This is where, for example, user interfaces and database design (based on the information model) are created. This is where information architecture, system architecture, and technology choices are made.

Before coding, design the architecture:

- **Database model**: ER diagram showing entities, relationships, and cardinalities
- **System architecture**: Frontend, backend, database, external services, and how they connect
- **Technology choices**: Programming languages, frameworks, hosting platforms, third-party services

Your architectural choices affect everything downstream: development speed, maintenance costs, scalability, security, and what's possible overall.

For vibe-coded apps, simpler architectures are better. Fewer moving parts mean fewer things that can break and fewer things you need to explain to the AI.

When selecting technologies for vibe coding, consider whether they are well-documented and have numerous online examples. (AI assistants perform better with widely-used technologies that appear frequently in training data.)

- Is there an active community where common problems have been solved and discussed?
- Does the framework have clear conventions that reduce the number of decisions you need to make?
- Can you deploy easily with minimal DevOps complexity?

Start with established, mainstream technologies rather than cutting-edge options — you want boring and reliable, not exciting and fragile.

Document your design decisions and the reasoning behind them. You will need this context later.

Development

Work step-by-step. Break the build into small, testable increments.

Consider text-driven development:

1. Write a description of what you want to build
2. Feed it to the AI with relevant context (PRD, models, existing code)
3. Review what the AI generates
4. Test it
5. Iterate until it works
6. Move to the next increment

Never ask the AI to build the entire app at once. Large prompts produce incoherent results. Small, focused prompts produce code you can understand and verify.

Keep the context updated. As your codebase grows, the AI needs to understand what already exists to generate consistent new code. Maintain documentation that serves as context for future prompts.

Version control everything from the start. Commit frequently. Write meaningful commit messages. You will thank yourself later.

Test

Testing is not optional. Without tests, you don't know if your app works. You only know it hasn't visibly failed yet.

Levels of testing:

- **Unit tests**: Individual functions work correctly
- **Integration tests**: Components work together correctly
- **End-to-end tests**: Complete user flows work correctly
- **User acceptance testing**: Real users confirm it meets their needs

AI can help write tests. Tell it what you want to verify, and it can generate test code. Then run the tests to see which pass and which fail.

Test with real data and real scenarios, not just happy paths.

- What happens when the user enters invalid input?

- What happens when the network fails?
- What happens when the database is empty?

Deployment

Deployment is the process of moving your app from development to production, where users can access it.

Modern deployment options make this much easier than it used to be. Platforms exist that let you deploy with a few commands or even automatically when you push code.

Key deployment considerations:

- **Environment configuration**: Separate settings for development, staging, and production
- **Secrets management**: API keys, passwords, and sensitive data stored securely
- **Domain and SSL**: A real domain name with HTTPS encryption
- **Monitoring**: Visibility into whether the app is running and how it's performing

Start simple. You can add complexity later when you need it.

DevOps

DevOps is the practice of automating and streamlining development and operations.

For an MVP, keep it minimal:

- Automated deployment when you push code
- Basic monitoring and alerting
- Logs you can access when things go wrong
- Backups of your database

As you grow, you'll add more: continuous integration, automated testing pipelines, infrastructure-as-code, and more sophisticated monitoring. But for now, focus on having something that works and can be updated quickly.

Gate 5

If the product delivers real value in real use, you have a product.

Step 6. Go-to-Market Scaling

With a working product and paying customers, the next question is whether customer acquisition can be repeated.

Analyze where your first customers came from and which messages resonated with them. Focus on channels that work. Document your sales process. Track the metrics that matter, including conversion rates, acquisition costs, retention, and lifetime value.

Scale cautiously. Prove things manually before automating them.

Finding what works

Your first customers came from somewhere — direct outreach, social media, ads, referrals, or something else. Analyze what worked:

- Where did your paying customers come from?
- What message or angle resonated?
- What was the conversion rate at each step?
- How much did it cost to acquire each customer?

Double down on what works. Ignore what doesn't.

Channels

Different customer types require different channels:

- **Content marketing:** Write or create content that demonstrates your expertise and attracts people searching for solutions to the problem you solve
- **Paid advertising:** Test platforms where your ICP spends time — this could be social platforms, search engines, or niche sites
- **Direct outreach:** Personally contact potential customers through email, social media, or professional networks
- **Partnerships:** Work with complementary businesses or influencers who can refer customers

- **Product-led growth:** Make your product so good that users refer others organically

You don't need all channels. You need one or two that work consistently.

Sales Process

Document your sales process:

- How do prospects find you?
- What happens when they first engage?
- How do you qualify them?
- How do you demonstrate value?
- How do you close the sale?
- How do you onboard new customers?

As you do this repeatedly, patterns emerge. Systematize what works, so you (or others) can replicate it.

Metrics that matter

Track the numbers:

- **Traffic:** How many people visit your site or landing page
- **Conversion rate:** What percentage takes the desired action
- **Cost per acquisition:** How much you spend to get one customer
- **Revenue per customer:** How much each customer pays
- **Churn rate:** What percentage of customers leave each period
- **Lifetime value:** Total revenue from an average customer

These numbers tell you whether your go-to-market approach is sustainable. If CAC exceeds LTV, you lose money on every customer. If conversion rates are too low, you need better messaging or targeting.

Scaling carefully

Scale what's proven, not what you hope will work.

Before spending heavily on ads, prove the messaging and targeting with small tests. Before hiring salespeople, prove you can sell yourself. Before automating outreach, prove it works manually.

Premature scaling is a common killer of startups. It burns cash on approaches that don't work yet.

Gate 6

If customer acquisition is repeatable, you have a business.

Step 7. Product Scaling

Product scaling is about improving and extending the product based on real usage and feedback.

This requires systematic learning. That means deliberately testing hypotheses, analyzing how users actually behave, complementing quantitative data with qualitative feedback, and paying attention to changes in the competitive landscape. Even if you work alone, these learnings should be made explicit and documented so that decisions are based on insight rather than memory.

Prioritize improvements based on impact rather than novelty. Work in short cycles of building, measuring, and learning. Retention matters more than acquisition. A small improvement in retention often has a larger impact on long-term value than a comparable improvement in acquisition.

Customer feedback loops

Systematically collect feedback:

- What do users love?
- What frustrates them?
- What do they wish the product did?
- What would make them recommend it to others?
- Why do users who leave decide to leave?

Sources of feedback: support requests, user interviews, surveys, usage analytics, churn analysis, and social media mentions.

Turn feedback into insights. Not every request should be built, but patterns reveal priorities.

Prioritizing features

You will have more ideas than you can build. Prioritize ruthlessly.

Useful frameworks:

- **Impact vs. effort**: High impact, low effort first
- **ICE scoring**: Impact × Confidence × Ease
- **Jobs to be done**: What job is the user hiring the product to do?
- **Revenue impact**: What features would reduce churn or increase pricing power?

Build what moves the metrics that matter, not what sounds cool.

Iteration cycles

Work in short cycles following the build-measure-learn loop:

1. Identify the most important improvement (based on the hypothesis about what users need)
2. Build it quickly (the minimum needed to test the hypothesis)
3. Release it
4. Measure the impact (did it move the metrics that matter?)
5. Learn and repeat (what does the data tell you about your hypothesis?)

This is the core of lean product development. Speed matters. Faster cycles mean faster learning. Faster learning means a better product. Each cycle should teach you something about your users that informs the next cycle.

Retention

Acquiring customers costs money. Keeping them generates profit.

Focus on retention by:

- Delivering consistent value
- Reducing friction in the user experience
- Engaging users before they drift away
- Solving problems quickly when they arise
- Continuously improving based on feedback

Monitor retention metrics closely. A small improvement in retention often has a larger impact on business value than a small improvement in acquisition.

Expansion revenue

Grow revenue from existing customers through:

- **Upselling**: Higher tiers with more features or capacity
- **Cross-selling**: Additional products that complement the core offering
- **Usage growth**: Pricing that increases as customers use more

Happy customers who get more value should pay more. Design your pricing to make this natural.

Gate 7

If users stay and the product improves over time, you have sustained product–market fit. Product–market fit, in the classical sense, requires actual usage, retention, and repeated value delivery.

Step 8. Re-Architecture

As usage grows, earlier technical choices may become limiting. Performance issues, increasing complexity, or rising costs may signal the need for re-architecture.

Approach this carefully. Avoid large rewrites. Identify specific constraints, design a target state, and migrate incrementally. AI is helpful here, but experienced engineers may be necessary when systems become complex.

However, at some point, the technical choices you made for your MVP may limit your ability to grow. The code that worked for 100 users might not work for 10,000.

The architecture that was simple to build quickly might be hard to maintain or extend.

Re-architecture means reworking the technical foundation to support the next phase of growth.

Signs you need re-architecture

- Performance degradation as usage grows
- An increasing number of bugs that are hard to fix
- New features take longer and longer to build
- Critical functions depend on code nobody understands
- Security vulnerabilities that require structural fixes
- Scaling costs are growing faster than revenue

Approach

Re-architecture is risky. You're changing a working system. Approach it carefully:

1. **Identify the constraint**: What specific problem are you solving?
2. **Design the target state**: What does the improved architecture look like
3. **Plan the transition**: How do you get there without breaking what works
4. **Execute incrementally**: Migrate piece by piece, testing at each step
5. **Verify the improvement**: Measure that the problem is actually solved

Avoid the "big rewrite." Projects that try to rewrite everything at once often fail or take far longer than expected. Incremental improvement is slower but safer.

When to involve professional engineers

If your app has grown to a significant scale and complexity, this may be the point at which human engineers complement AI.

AI coding assistants are excellent at generating code from specifications. They are less good at diagnosing complex system interactions, making architectural trade-offs, or debugging subtle production issues.

For significant re-architecture, consider bringing in experienced developers who can:

- Assess your current system objectively
- Recommend architectural improvements
- Implement critical changes safely
- Transfer knowledge so you can maintain the system

This is an investment. Weigh it against the cost of not making necessary improvements.

Gate 8

If the system can sustain growth, move on to operations.

Step 9. Process Management

Process management means making marketing, sales, order handling, delivery, and support systematic and, where possible, automatic.

A business that depends on heroic effort doesn't scale. The founder becomes the bottleneck. Process management removes that bottleneck.

Document your processes

Start by writing down how things work today:

- How do leads come in and get tracked?
- How do sales conversations happen?
- How are orders processed?
- How do new customers get onboarded?
- How are support requests handled?
- How are bugs reported and fixed?
- How are invoices sent and payments collected?

This documentation reveals inefficiencies and creates the foundation for improvement.

Automate where possible

Identify repetitive tasks that follow consistent rules. These are candidates for automation:

- Email sequences triggered by user actions
- Lead routing based on criteria
- Onboarding workflows that don't require human judgment
- Invoice generation and payment reminders
- Basic support responses for common questions
- Reporting and analytics dashboards

Automation tools exist for the most common business processes. Use them.

Create systems, not dependencies

A system is a documented, repeatable way of doing something that doesn't depend on any single person's knowledge.

For each key process:

- Document the steps clearly enough that someone new could follow them
- Define the inputs and outputs
- Identify decision points and who makes those decisions
- Set quality standards and how they're verified
- Create checklists and templates where helpful

Systems enable delegation. You can bring in help (human or AI) because the knowledge isn't just in your head.

Support that scales

As you get more customers, support volume grows. Without systems, it drowns you.

Build support that scales:

- **Self-service**: Documentation, FAQs, knowledge bases, video tutorials
- **Community**: Users helping users in forums or chat groups
- **Tiered support**: Automated or low-effort responses for simple issues, human attention for complex ones
- **Feedback loops**: Support insights flowing back to product improvement

The goal is to reduce support volume through better product and documentation, while handling necessary support efficiently.

Gate 9

If the business can operate without heroics, continuous improvement becomes possible.

Step 10. Product Management

Product management ensures that improvement is deliberate rather than accidental. Products have lifecycles, and strategies must adapt accordingly.

Product management involves establishing mechanisms for continuous product optimization. It's the discipline that ensures your product keeps getting better systematically, not randomly.

Maintain a roadmap to provide direction, not promises. Establish regular review rhythms to surface issues early. Build learning into how you work, through experiments, data analysis, and user research.

Product Lifecycle Management

Products have lifecycles: introduction, growth, maturity, and eventually decline. Managing this lifecycle means:

- Recognizing what phase you're in
- Adapting your strategy to the phase
- Planning for what comes next

In the introduction phase, you're finding product-market fit. In growth, you're scaling what works. In maturity, you're optimizing and defending a position. Eventually, you may need to innovate again or sunset the product.

Roadmap

Maintain a product roadmap:

- **Short-term (weeks)**: Specific features and improvements are being built
- **Medium-term (months)**: Themes and priorities for the next phases
- **Long-term (quarters/years)**: Strategic direction and vision

The roadmap is a planning tool, not a promise. It changes as you learn. But without a roadmap, you make decisions without context.

Metrics and review cadence

Establish regular rhythms:

- **Weekly**: Review key metrics, address immediate issues
- **Monthly**: Assess progress against goals, adjust short-term plans
- **Quarterly**: Evaluate strategic direction, plan the next quarter

These reviews create accountability and surface problems early.

Continuous learning

Build learning into how you work:

- Run experiments to test hypotheses about what users want
- Analyze data to understand user behavior
- Conduct regular user research to stay close to customers
- Monitor competitors and market trends
- Share learnings across the team (even if the team is just you)

A product that stops learning starts dying. Markets change, customers change, competitors change. Continuous learning keeps you ahead.

Planning for the future

Think beyond the current product:

- What adjacent problems could you solve?
- What would threaten your current position?
- What new technologies might change the market?
- What would a 10x better version of your product look like?

You don't have to act on these thoughts immediately. But having them keeps you from being blindsided.

Gate 10

If continuous improvement is built into the structure, you have a sustainable product business.

Conclusion

Building a profitable app with Vibe Coding is not primarily about coding. AI has removed much of the implementation bottleneck. What remains is judgment: understanding real problems, validating demand, making deliberate design choices, proving willingness to pay, and systematizing operations.

Most app ideas fail not because they cannot be built, but because they should not have been built. No one wanted them enough to pay for them. The economics did not work. The founder lost motivation. The data dependency was fragile.

The gates exist to enforce discipline. By working through them in order, you kill bad ideas early, validate demand before building, focus effort where it matters—to deliver value — and create systems you know will scale and don't depend on heroic effort.

The approach feels slower at first. You want to start building. That feeling is a trap. The fastest path to a profitable app is not the shortest path to a working app. It's the path that validates every step before taking the next one.

AI can handle syntax. Strategy, judgment, and persistence remain human responsibilities. Build something people want enough to pay for. And do it in the right order.

Vibe Coding | Product Management | Product Development | Software Development | Software

Edit profile

Written by Gunnar Östberg

196 followers · 143 following

CEO Business Clarity. I write enjoyable, rewarding articles that retain their value. Please check my profile to see the various story topics!

No responses yet

Gunnar Östberg

What are your thoughts?

More from Gunnar Östberg