Find the Right Problem, Prove Demand, and Launch AI-Built Products People Actually Want

THE CODE WAS *NEVER* THE HARD PART

>_

OSKAR GLAUSER

The Code Was Never the Hard Part

Find the Right Problem, Prove Demand, and Launch AI-Built Products People Actually Want

A FREE EXCERPT

Introduction and Chapter One

OSKAR GLAUSER

neverthehardpart.com

CONTENTS

In bold: the two sections in this excerpt.

INTRODUCTION

Twenty years of the hard part

I once co-founded a company that had half a million users, forty million video views a month, and eight hundred thousand euros in annual revenue.

It failed anyway.

The company was called Videofy, a social video platform for creators, built years before short video conquered the world. It was itself born from a pivot: Robert, my friend and co-founder, and I had been producing our own short web series on a site of ours that we called Dayrobber, selling advertising that sold fine to an audience that never arrived. That taught us that audiences, not content or ads, were the scarce thing, and it pointed us at the people who already had them.

We started Videofy in August 2008 and got it off the ground with almost embarrassingly small means: no outside capital, just a small government loan for building the MVP and the two of us. Only after we saw it working, real users, real growth, did we raise a seed round on the strength of those numbers. I want to be fair to us: that part, start small, prove it, then fuel it, we did right. We had press in the Swedish papers, in The Next Web, in the Huffington Post. We had a revenue-sharing model for creators before that was normal. We had, and I say this without false modesty because it no longer matters, a genuinely well-built product. The code worked. The code was never the problem.

The problem was everything around the code. We had picked a race, consumer social, where the entry fee is measured in hundreds of millions, and we showed up with a seed round of about one and a half million dollars. We were competing with Instagram and YouTube for the same attention with a fraction of a fraction of their resources. No amount of shipping was going to change that arithmetic. As I've put it since: we didn't have the muscles to scale fast enough to matter. It took us four years to find that out.

Hold that number. Four years to learn that the answer was no. It's the most expensive tuition I've ever paid, and the receipt is the book you're holding.

What I did with the lesson

After Videofy I did what burned founders do: I swung to the opposite extreme. With one developer hired on contract, I started Minutemailer, an email marketing tool for people who aren't marketers. No investors. No growth targets set by anyone but us. A deliberately narrow promise, email marketing made simple, aimed at the small businesses that found the big platforms too complicated and too expensive.

Minutemailer will never be written about in the business press. It's also still alive more than a decade later, quietly paying its way, serving customers who chose it precisely because it does less. It taught me the other half of what Videofy taught me: a modest product in the right race beats an impressive product in the wrong one, and "right race" is a decision you make before you build, whether you know you're making it or not.

In between and since, I spent years doing strategy and design for other people's products, from startups to some of the largest companies in the world. One project I'll return to in this book was a social fashion app I helped build for a global fashion retailer. It gathered more than three hundred thousand sign-ups, with a core of real users posting hundreds of thousands of photos and telling their friends. It was then shut down anyway, for reasons that had nothing to do with the product and everything to do with a corporate reorganization. Another lesson the code couldn't have taught: sometimes the market says yes and the answer is still no.

So that's what I bring: one product that died big, one that lived small, one that worked and was killed from above. Twenty years of watching where products actually succeed and fail, from the inside, at every scale.

And in all that time, across all those projects, I can tell you where the fatal wound was inflicted in almost every failure I witnessed. It was never in the code. It was in what came before the code, choosing the wrong problem, the wrong race, the wrong words, and in what came after, launching into silence, misreading the signals, drifting instead of deciding. The building in the middle was usually the part that actually worked.

For most of my career this observation was interesting but not actionable. Building was so expensive that it dominated everything regardless. You couldn't tell a founder "the code is the easy part" when the code cost a year and a funding round.

Then, quite recently, that stopped being true.

The wall comes down

The wall is older than my career. When I was ten, maybe younger, I got an Atari ST, paid half with money I had saved and half by my mother. I played everything I could get my hands on, and then I wanted to know how the games were made, so I tried to teach myself Basic. I got as far as a snake game that actually worked: the snake grew longer for every apple it ate, and it took two players, because a computer opponent was beyond me. I could feel how deep the water was, and the discovery underneath was that the other side interested me more: what the game should be, not how the computer did it. So I went back to paper, where I was already drawing constantly, and filled it with concepts instead. Games I'd never build. Apps that didn't exist. At some point an entire operating system, inspired by an Apple I didn't own, drawn by hand with windows, icons, and menus. I dreamed of one day making the real things.

I grew up, and the dream became a profession with the same wall in it. I'm a designer. I've spent my whole career one wall away from the code, able to describe exactly what should exist but never able to build it myself. My working life happened in Figma, Illustrator, Photoshop, and InDesign. The closest I came to code was editing translations in Minutemailer's codebase and adjusting some simple CSS. Every product I ever made required convincing, hiring, or becoming business partners with the people who could build.

Somewhere around early 2025, that wall started coming down, for me and for a whole generation of people like me. Claude and ChatGPT crossed the line where building real things with code actually worked, and a new world opened. I noticed it first in my consulting. I was designing a mobile app

for an energy company, and instead of handing over a Figma prototype I built an HTML prototype that behaved like the real app, and tested it on users. Then I thought: why not build it for real? So I did, in Swift. Later the whole app moved to Flutter, and I built the design system directly in code, with no Figma in between. Step by step, without quite deciding to, I had gone from designer to someone who could deliver all of it.

Then I built an entire product alone.

It's called Mira, an AI tool that conducts customer interviews, born from a frustration I kept meeting in my consulting work: teams making product decisions on "we did some interviews last quarter" and "the NPS is trending down." I defined it with the help of AI, and then, using AI coding tools, I built it. Me myself and Claude. The first working version existed by the end of one weekend. Refining it has taken longer, the way real products always do, but the last of a wall I had been drawing against since the Atari came down between a Friday evening and a Sunday night.

I want to be precise about what I felt, because it wasn't simple joy. It was joy plus vertigo. Because if I could do this, everyone could. And if everyone could, then everything I had watched decide the fate of products, the choosing, the evidence, the positioning, the distribution, the taste, had just become the entire game. The hard part was no longer optional and no longer hidden behind the expensive part. It was the only part left.

Then I watched it happen. Launch platforms began overflowing with fresh AI-built products, and most of them sank without a trace. Well-built, fast, polished, and doomed, exactly the way Videofy was doomed, but at a thousand times

the volume and a hundredth of the cost. A whole generation arriving at the moment where I lost four years, with no map.

This book is the map I wish someone had handed me. It's also a book I want to read myself. That's why I wrote it.

What this book is

It's a method: how to find a problem worth solving, gather real evidence that anyone cares, compress it into a concept, position it, make it feel real, build it in a short sprint, launch it without an audience, and then, the part almost nobody teaches, read the result honestly enough to know whether to kill it or double down.

It's written for the person who can suddenly build: the designer, the marketer, the developer moving upstream, the founder without a technical co-founder, anyone with an idea and access to the tools who can't yet tell which ideas deserve their weeks. If that's you, this book is yours.

It isn't a promise of a hit. I don't have a unicorn to point at, and I've come to see that as a strange kind of qualification. Books by lottery winners teach you about lightning. I can teach you about weather, because I've paid full price for every kind there is: the big failure, the small survival, the success that got shot by its own side. The method in this book exists so that your failures cost you weeks instead of years, which is the only guarantee anyone honest can offer.

One more thing, in the spirit of eating my own dog food. This book was validated using its own method. Before finishing the manuscript, I put its positioning in front of strangers, collected signals, and let real people raise their hands at

neverthehardpart.com. The templates from these chapters live there too.

You can build now. So can everyone. What follows is the part that was always hard, and how to get good at it.

• • •

PART I

Choose

“Make something people want.”

Paul Graham, Y Combinator’s motto

CHAPTER ONE

Anyone can build now

There has never been a better time to build software. You can describe a product in plain language and watch a working version appear the same afternoon. Things that used to require a funded team and six months now require a laptop and a weekend.

So where are all the great products?

Look around. App stores are flooded. Launch platforms list hundreds of new AI tools every week. Most of them are dead within months. Not because they were built badly. Many of them are built well. They're fast, polished, and functional. They flop anyway.

The uncomfortable part: the ability to build has stopped being an advantage. When everyone can build, building is the entry fee. The products that win are separated by everything that happens before and after the code.

That's what this book is about.

The constraint moved

For most of software history, code was the bottleneck. It was expensive, slow, and scarce. If you could build, you had power. Whole business models were designed around that scarcity. Agencies charged for it. Startups raised money for it. "Technical co-founder" became the most hunted role in the startup world.

So we all learned to treat building as the hard part. The idea person with a napkin sketch was a cliché. The builder was the hero.

Then the cost of building collapsed.

This has happened before. In 1888 Kodak sold the first easy camera with the slogan "You press the button, we do the rest," and photography stopped being a craft you needed a darkroom for. More than a century later the smartphone put a serious camera in every pocket. Both times the same thing followed: the world filled with photographs. It didn't fill with great photographs. The scarce skill just moved, from operating the camera to knowing what to point it at.

When a constraint disappears, it doesn't make everything easy. It moves the pressure somewhere else. Remove the dam and the water finds the next narrow point in the river. In software, the next narrow points are these:

- Knowing which problem is worth solving
- Understanding the people who have that problem
- Positioning the product so those people instantly get it
- Reaching them at all
- Having the judgment to say no to everything else

None of these got easier when code got cheap. If anything they got harder, because now you compete with everyone else who can also build in a weekend.

I watch this from the inside every week, consulting for startups, scale-ups, and large enterprises trying to put AI to work. The pattern repeats at every size: the models are ready and the humans are not. Engineers who don't push the tools anywhere near capacity. Managers uneasy about what their role becomes. Leadership enthusiastic in speeches and hesitant about changing how the company is actually shaped. The tech was never the blocker. The willingness to change shape was. And for one person building alone, that willingness decides everything.

The code was never the hard part. It just looked like the hard part, because it was the expensive part. Those aren't the same thing.

Why building feels like progress

Here is the trap, and almost everyone falls into it.

Building feels productive. You start the day with nothing and end it with a working screen. You can show it to people. You can touch it. Every session produces visible output, and the tools make that output arrive faster than ever.

Validation work feels like the opposite. Talking to potential customers is awkward. Researching a market gives you documents, not demos. Sharpening your positioning produces a sentence, and a sentence doesn't feel like a day's work even when it's worth more than a month of code.

So people do the thing that feels like progress instead of the thing that's progress. They build first and ask questions later.

AI has made this trap far more seductive, because the cost of skipping the questions seems so low. Why validate an idea when you can just build it and see?

Because "just build it and see" almost never includes the "see" part. Shipping into silence teaches you very little. The product gets no users, and you can't tell whether the idea was wrong, the positioning was wrong, or nobody ever found out it existed. You've spent your motivation and learned close to nothing.

I fell into this trap at scale. My first startup, Videofy, a social video platform you'll hear more about in this book, spent its final years building. New features, new releases, visible output every week. Meanwhile the actual problem sat unattended: Instagram and YouTube were improving fast, the iPhone was swallowing the market, and we no longer had a sharp answer to why us. What we needed was a new proposition. What we produced was more product. Building felt like progress right up to the end, and the end came anyway.

That was the old version of the trap. Let me confess to the new one. In the past twelve months I've built more than twenty apps. I've released two. The unreleased pile includes location-aware social photos, longevity and habit tracking, a branding design tool, an AI-native collaborative text editor, a multiplayer 3D world of small planets somewhere between a game and an art piece, an app where everyone answers the same question every day, and an app for documenting everything you own so you can sell any of it the moment you want to.

Most were fun experiments, built over weekends because building is a joy now, and I dropped them early. The real reason: I know what the hard part costs, and at the time I

didn't believe any of them was worth paying it. As learning builds, all of them earned their keep. But a few began as genuine ideas I thought I wanted to pursue, and they died the same quiet death as the toys. That's the trap with the friction removed. When building is this easy, building becomes the way you avoid deciding. I wrote this chapter for you and, evidently, for me.

The new math

Put rough numbers on the shift.

The old process looked something like this: one month figuring out what to build, six months building it, then launch and pray. Building dominated the calendar, so building dominated everyone's attention.

Now the build can take a week or two. The first working version of Mira took a single weekend. If you keep the same total effort, you suddenly have months available for everything else. The question is what you do with them.

Most people do nothing with them. They keep their old ratio and just finish faster, shipping the same unvalidated ideas at higher speed. The result is more flops per year. Speed amplifies whatever process you already have. If your process is "build first, think later," AI helps you fail more often.

The opportunity is to flip the ratio. Spend the bulk of your time on the parts that decide the outcome: choosing the problem, talking to real humans, nailing the positioning, planning distribution. Then run the build as a short, sharp sprint at the end. Same total time as before, radically different odds.

That reordering is the core method of this book.

What the hard part is made of

Concretely, what fills the space code used to occupy? Five things.

Judgment. When you can build anything, the scarce skill is deciding what is worth building. This is a filtering problem. Ideas are infinite and cheap. Most of them deserve a fast no. The chapters ahead give you a filter you can run in days, not months.

Evidence. An opinion about what people want is a guess, including your own opinion, especially your own opinion. Evidence comes from watching what people actually do and struggle with. Getting evidence used to be slow. It isn't anymore, and I'll show you how to gather it quickly without fooling yourself.

Positioning. People decide in seconds whether your product is for them. If they can't answer "what is this and why should I care" from one screen, you've already lost them, no matter what the product can do. Positioning is slow thinking work, and it decides more launches than the feature list does.

Distribution. A product nobody finds is indistinguishable from a product that doesn't exist. Distribution can't be an afterthought you get to after launch. It shapes what you should build in the first place.

Taste. When everyone uses the same tools, output converges on the same average. The same layouts, the same tone, the same generated look. Taste is how you escape the average, and it's one of the few things the tools don't hand you. This gets its own treatment later in the book, because it deserves it.

Notice that none of these are technical. The playing field didn't disappear. It moved.

Who wins now

The winners in the AI era aren't the best builders. They're the people who combine decent building skills with strong judgment about problems, people, and markets.

This is good news if you come from the non-technical side. Your understanding of customers, of an industry, of how to sell, of what looks and feels right, all of that just became the differentiating half of the equation, while the half you were missing became accessible.

It's also good news if you're technical, with a catch. Your building skills still matter, but they no longer carry you alone. The developers who thrive are the ones who move upstream into the questions of what and why, not just how.

The shift is coming from every direction at once. I went from designer toward full stack. Developers I know are moving the other way, into product and customers. Others arrive from product management, marketing, finance, even the CEO chair, all discovering the same power to build. Where you start from matters less than it ever has.

Either way, the profile that wins looks the same: someone who can move from a fuzzy signal to a sharp concept, prove there's real demand, and then build and launch fast. One person can now cover that whole span. That has never been true before.

What this book will do

This is the path we'll walk, in order:

First, choosing your problem. How to spot signals worth chasing and drop weak ideas quickly, before they eat your months.

Second, evidence. Fast, honest customer research that tells you whether anyone actually cares, and how to use AI to do in days what used to take weeks.

Third, the concept. Compressing what you've learned into a one-page concept and a rough business case, so you know what you're building and why before you build it.

Fourth, the packaging. The price, the words, the name, and the look: making the product instantly understandable and real-feeling, because trust is decided before anyone clicks a button.

Fifth, the build. A short, focused sprint using AI tools, scoped to the smallest thing that can produce a real market signal.

Sixth, the launch. Getting your first users without a big audience or a budget, and reading the early signals clearly enough to know whether to keep going or move on.

You don't need to be a developer to follow any of it. You don't need to quit your job or raise money. You need a real problem, a few weeks of focused effort, and the discipline to do the unglamorous parts before the fun parts.

One promise before we start. I won't pretend this method guarantees a hit. Nothing does, and anyone who says otherwise is selling something worse than nothing. What a good process does is change your odds and your cost of being wrong. You'll still miss sometimes. You'll miss faster, cheaper, and with lessons you can actually use on the next attempt.

The tools took away your best excuse. You can build now. The question that remains is the one that always mattered:

Should you?

Let's find out. Pick a problem, and turn the page.

• • •

THE REST OF THE BOOK

That was the opening.

Thirteen more chapters follow, in four parts. Weeks of thinking, a short sharp build at the end, and a verdict you can trust, whatever it says.

Choose	Pick your race, spot the signals other people scroll past, and gather evidence from real conversations.
Shape	Compress what you learned into a one-page concept, a price with a reason behind it, and words a stranger understands.
Ship	Plan with AI, build in a focused sprint, launch without an audience, and read the response honestly.
Run	Operate as a one-person company, with a loop that gets sharper every time you run it.

Every worksheet from the book will be free on the site: the problem filter, interview guide, one-page concept, build brief, and launch checklist.

neverthehardpart.com