CHAPTER 1:

Introduction

This book is a guide to how we do product development at Basecamp. It's also a toolbox full of techniques that you can apply in your own way to your own process.

Whether you're a founder, CTO, product manager, designer, or developer, you're probably here because of some common challenges that all software companies have to face.

Growing pains

As software teams start to grow, some common struggles appear:

- Team members feel like projects go on and on, with no end in sight.
- Product managers can't find time to think strategically about the product.
- Founders ask themselves: "Why can't we get features out the door like we used to in the early days?"

We saw these challenges first-hand at Basecamp as we grew from four people to over fifty.

Basecamp started off in 2003 as a tool we built for ourselves. At the time we were a consultancy designing websites for clients. Information would get lost in the game of telephone between the client, the designer, and the person managing the project. We wanted Basecamp to be a centralized place where all parties could see the work, discuss it, and know what to do next. It turned out lots of companies had this "information slipping through the cracks" problem. Today millions of people across all kinds of industries rely on Basecamp as their shared source of truth.

Three of us built the first version. Jason Fried, Basecamp's founder, led the design. His co-founder, David Heinemeier Hansson, programmed it (and created the well-known web framework Ruby on Rails as a by-product). At the time I was a web designer with a focus on usability and user interfaces. I executed Jason's design direction for key features of the app and collaborated with him to fill in details of the concept.

From the first prototypes in July 2003 to launch in February 2004, David only worked ten hours a week. We knew we wouldn't get anywhere with those ten hours of programming unless we used them very deliberately. Our intense focus on "hammering" the scope to fit within a given time budget was born under these constraints.

As the business grew, I started widening my skills. Working with David and Ruby on Rails made the world of programming accessible to me. I learned the techniques programmers use to tame complexity: things like factoring, levels of abstraction, and separation of concerns. With one foot in the design world and one foot in the programming world, I wondered if we could apply these software development principles to the way we designed and managed the product.

The first test of this idea came in 2009. By then we had hired a few more programmers and offered four separate software-as-a-service products. We wanted to bundle the products together into a seamless suite with single-sign-on and unified billing. It was a massive technical undertaking with treacherous user-facing flows. Besides getting the underlying architecture right, we had to interrupt customers on their way in to the product and make them change their username and password for reasons that weren't easy to explain. I wore the designer and product manager hats on the project and prototyped the breadboarding and scope mapping techniques described in this book to manage the complexity.

We had such good results that we decided to apply the same techniques again in 2012, when we redesigned Basecamp from scratch for version 2.0. Again there was a lot of surface area to manage and again the process was surprisingly smooth.

By 2015, we had a core team that had lived through these experiences and hit an impressive stride. But we found it hard to articulate what we were doing to new hires. Our product team had quadrupled and everyone worked remotely. That made it hard to pass on our intuitions. We needed language to describe what we were doing and more structure to keep doing it at our new scale.

To manage this new capacity, we switched from ad-hoc project lengths to repeating cycles. (It took some experimentation to find the right cycle length: six weeks. More on that later.) We formalized our pitching and betting processes. My role shifted again, from design and product management to product strategy. I needed new language, like the word "shaping", to describe the up-front design work we did to set boundaries and reduce risks on projects before we committed them to teams.

Just as we were getting better at articulating the way we work to ourselves, more and more of our friends and peers started coming to us to ask how we do it. Finally Jason pulled me aside one day and said, I think you should write a book about this.

This is the result. You can think of this as two books in one. First, it's a book of basic truths. I want it to give you better language to describe and deal with the risks, uncertainties, and challenges that come up whenever you do product development. Second, the book outlines the specific processes we're using to make meaningful progress on our products at our current scale.

Here's a short overview of the main ideas in the book.

Six-week cycles

First, we work in **six-week cycles**. Six weeks is long enough to build something meaningful start-to-finish and short enough that everyone can feel the deadline looming from the start, so they use the time wisely. The majority of our new features are built and released in one six-week cycle.

Our decisions are based on moving the product forward in the next six weeks, not micromanaging time. We don't count hours or question how individual days are spent. We don't have daily meetings. We don't rethink our roadmap every two weeks. Our focus is at a higher level. We say to ourselves: "If this project ships after six weeks, we'll be really happy. We'll feel our time was well spent." Then we commit the six weeks and leave the team alone to get it done.

Shaping the work

Second, we **shape the work** before giving it to a team. A small senior group works in parallel to the cycle teams. They define the key elements of a solution before we consider a project ready to bet on. Projects are defined at the right level of abstraction: concrete enough that the teams know what to do, yet abstract enough that they have room to work out the interesting details themselves.

interesting details themselves.

When shaping, we focus less on estimates and more on our **appetite.** Instead of asking how much time it will *take* to do some work, we ask: How much time do we want to *spend*? How much is this idea worth? This is the task of shaping: narrowing down the problem and designing the outline of a solution that fits within the constraints of our appetite.

Making teams responsible

Third, we **give full responsibility** to a small integrated team of designers and programmers. They define their own tasks, make adjustments to the scope, and work together to build vertical slices of the product one at a time. This is completely different from other methodologies, where managers chop up the work and programmers act like ticket-takers.

Together, these concepts form a virtuous circle. When teams are more autonomous, senior people can spend less time managing them. With less time spent on management, senior people can shape up better projects. When projects are better shaped, teams have clearer boundaries and so can work more autonomously.

Targeting risk

At every step of the process we target a specific risk: the risk of not shipping on time. This book isn't about the risk of building the wrong thing. Other books can help you with that (we recommend Competing Against Luck). Improving your discovery process should come after regaining your ability to ship. You can have the best strategy in the world, but if you can't act on it, what good does it do?

This book is about the risk of getting stuck, the risk of getting bogged down with last quarter's work, wasting time on unexpected problems, and not being free to do what you want to do tomorrow.

We reduce risk in the shaping process by solving open questions *before* we commit the project to a time box. We don't give a project to a team that still has rabbit holes or tangled interdependencies.

We reduce risk in the planning process by capping our bets to six weeks. If a project runs over, by default it doesn't get an extension. This "circuit breaker" ensures that we don't invest multiples of the original appetite on a concept that needs rethinking first.

And lastly we reduce risk in the building process by integrating design and programming early. Instead of building lots of disconnected parts and hoping they'll fit together in the 11th hour, we build one meaningful piece of the work end-to-end early on and then repeat. The team sequences the work from the most unknown to the least worrisome pieces and learns what works and what doesn't by integrating as soon as possible.

How this book is organized

Part One is all about **Shaping** — the pre-work we do on projects before we consider them ready to schedule. Each chapter explains a specific step of the process, from setting the appetite on a raw idea, to sketching out a solution, to writing a pitch that presents the potential project. Along the way you'll learn specific techniques — like breadboarding and fat-marker sketching — to keep the design at the right level of abstraction.

Part Two is about **Betting** — how we choose among the pitched projects and decide what to do six weeks at a time.

Part Three is about **Building** — the expectations we place on the teams and the special practices they use to discover what to do. We'll look at how the teams figure out what to do, how they integrate design and programming, how they track what's known versus unknown, and finally how they make the hard calls to finish the project on time.

Lastly the Appendix gives you some help for when it's time to make changes at your company. There's some advice on how to try your first six-week experiment, tips on adjusting the methods to your company's size, and specific guidance for how to implement Shape Up using Basecamp.