

Ryan Singer  
这里有很多女人

# Shape Up 振作起来

Stop Running in Circles and  
别再兜圈子了  
Ship Work that Matters  
重要的船舶工作

# Contents

# 目录

Foreword by Jason Fried	前言作者: Jason Fried	8
Acknowledgements		
鸣谢		10
Introduction		
引言		11
Growing pains	成长的烦恼	11
Six-week cycles	六周周期	14
Shaping the work	塑造工作	14
Making teams responsible	让团队负责	15
Targeting risk	瞄准风险	15
How this book is organized	这本书是如何组织起来的	16

## Part 1: Shaping

## 第一部分: 塑造

Principles of Shaping	塑造的原则	20
Wireframes are too concrete	线框图太具体了	20
Words are too abstract	文字太抽象了	21
Case study: The Dot Grid Calendar	案例研究: Dot 网格日历	21
Property 1: It's rough	属性 1: 粗糙	24

Property 2: It's solved 特性 2: 问题解决了	25
Property 3: It's bounded 特性 3: 它是有界的	25
Who shapes 谁塑造了	25
Two tracks 两首歌	26
Steps to shaping Steps to shapes 塑形的步骤	27
<hr/>	
Set Boundaries 设定界限	28
Setting the appetite 设置食欲	28
Fixed time, variable scope 固定时间, 可变范围	29
“Good” is relative ”好”是相对的	30

Responding to raw ideas	
回应原始的想法	31
Narrow down the problem	
缩小问题范围	31
Case study: Defining “calendar”	
案例研究: 定义”日历”	32
Watch out for grab-bags	
当心抓包	33
Boundaries in place	
界限就位	34
Find the Elements	
Find the Elements	—————
寻找元素	35
Move at the right speed	
以正确的速度前进	35
Breadboarding	
面包板	36
Fat marker sketches	
脂肪标记素描	41
Elements are the output	
元素就是输出	44
Room for designers	
设计师的空间	46
Not deliverable yet	
目前还不能交付	46
No conveyor belt	
没有传送带	47
Risks and Rabbit Holes	
风险与兔子洞	—————
Different categories of risk	
不同类别的风险	49
Look for rabbit holes	
寻找兔子洞	50
Case study: Patching a hole	
案例分析: 修补漏洞	51
Declare out of bounds	
宣布出界	53
Cut back	
削减	54

Present to technical experts 呈现给技术专家	54
De-risked and ready to write up 风险降低，准备写报告	56
Write the Pitch 写推销词	57
Ingredient 1. Problem 材料 1. 问题	58
Ingredient 2. Appetite 配料 2。食欲	59
Ingredient 3. Solution 材料 3. 溶液	60
Help them see it 帮助他们看到它	60
Embedded Sketches 嵌入式草图	61
Annotated fat marker sketches 带注释的脂肪标记草图	62
Ingredient 4. Rabbit holes 材料 4: 兔子洞	64

Ingredient 5. No Gos	
原料 5: 没有 Gos	64
Examples	
例子	64
Ready to present	
准备展示	67
How we do it in Basecamp	
我们在 Basecamp 是怎么做的呢	67

## Part 2: Betting

### 第二部分: 博彩

Bets, Not Backlogs	———
下注, 而不是积	
压	72
No backlogs	
没有积压文件	72
A few potential bets	
一些潜在的赌注	73
Decentralized lists	
去中心化的列表	73
Important ideas come back	
重要的想法又回来了	74
The Betting Table	———
投注表	75
Six-week cycles	
六周周期	75
Cool-down	
冷静一下	76
Team and project sizes	
团队和项目规模	77
The betting table	
赌桌	77
The meaning of a bet	
赌博的意义	78
Uninterrupted time	
不间断的时间	79

The circuit breaker 断路器	80
What about bugs? 那么 bug 呢 ?	81
Keep the slate clean 保持过去的清白	83
<hr/>	
Place Your Bets	
下注吧	84
Look where you are 看看你在哪里	84
Existing products 现有产品	84
New products 新产品	84
R&D mode R & d 模式	85
Production mode 生产模式	86
Cleanup mode Cleanup 模式	87
Examples 例子	88

Questions to ask 要问的问题	90
Does the problem matter? 这个问题重要吗？	90
Is the appetite right? 胃口好吗？	90
Is the solution attractive? 解决方案有吸引力吗？	91
Is this the right time? 现在合适吗？	92
Are the right people available? 有合适的人选吗？	92
Post the kick-off message 发布启动信息	93

## Part 3: Building 第三部分：建筑

Hand Over Responsibility —————— 移交责任	96
Assign projects, not tasks 分配项目，而不是任务	96
Done means deployed 完成意味着部署	97
Kick-off 开始	98
Getting oriented 开始定位	100
Imagined vs discovered tasks 想象任务 vs 发现任务	101
Get One Piece Done————— 完成一件事	103
Integrate one slice 整合一个部分	103
Case study: Clients in projects 案例分析：项目中的客户	105

Programmers don't need to wait 程序员不需要等待	107
Affordances before pixel-perfect screens 在像素完美的屏幕前提供支持	107
Program just enough for the next step 程序刚好够下一步使用	110
Start in the middle 从中间开始	111
<hr/>	
Map The Scopes 绘制瞄准镜地	
图	113
Organize by structure, not by person 按照结构组织，而不是按照个人	113
The scope map 范围图	114
The language of the project 项目的语言	117
Case study: Message drafts 案例研究: 消息草稿	117
Discovering scopes 发现范围	122

How to know if the scopes are right 如何知道范围是否正确	123
Layer cakes 分层蛋糕	124
Icebergs 冰山	125
Chowder 巧达	126
Mark nice-to-haves with ~ 马克与……相处愉快	126
Show Progress 显示进步	127
The tasks that aren't there 不存在的任务	127
Estimates don't show uncertainty 估计不会显示不确定性	129
Work is like a hill 工作就像一座山	129
Scopes on the hill 山上有望远镜	133
Status without asking 未经询问的状态	134
Nobody says "I don't know" 没人会说“我不知道”	136
Prompts to refactor the scopes 提示重构范围	137
Build your way uphill 建设你的方式上坡	139
Solve in the right sequence 按正确的顺序解决问题	139
Decide When to Stop 决定何时停止	142
Compare to baseline 与基线比较	143
Limits motivate trade-offs 限制激发权衡	144

Scope grows like grass	
范围像草一样生长	144
Cutting scope isn't lowering quality	
缩小范围并不会降低质量	145
Scope hammering	
瞄准镜敲击声	145
QA is for the edges	
质量保证是针对边缘的	147
When to extend a project	
何时扩展项目	148
Move On	
继续	150
Let the storm pass	
让暴风雨过去吧	150
Stay debt-free	
还清债务	150
Feedback needs to be shaped	
需要塑造反馈	151

Conclusion	———	
结论		152
Key concepts		
关键概念		152
Get in touch		
保持联系		153

## Appendices

### 附录

How to Implement Shape Up in Basecamp	———	
如何在 Basecamp 中实现 Shape Up		156
A Basecamp Team for shaping		
一个用于塑形的大本营团队		156
Basecamp Projects for the cycle		
projects		
Basecamp 项目的周期性项目		158
To-Do Lists for scopes		
作用域的待办事项清单		160
Track scopes on the Hill Chart		
在山图上追踪范围		161

### Adjust to Your

#### Size

调整你的尺寸	———	
Basic truths vs. specific practices		165
基本真理 vs 具体实践		165
Small enough to wing it		
小到可以即兴发挥		166
Big enough to specialize		
大到可以专攻		167

### How to Begin to Shape Up

如何开始塑造自己	———	
Option A: One six-week experiment		169
选项 a: 一个为期六周的实验		169

Option B: Start with shaping 备选方案 b: 从塑造开始	170
Option C: Start with cycles 选项 c: 从循环开始	170
Fix shipping first 先解决运输问题	170
Focus on the end result 专注于最终结果	171
Glossary 词汇表	172
About the Author 关于作者	176

# **Foreword by Jason Fried**

## **前言作者: Jason Fried**

The way a team works has an enormous influence on what it can do. The process, the methods, the practices, the approach, the discipline, the trust, the communication style, the pace. The way—the how—is foundational and fundamental.

一个团队的工作方式对它能做什么有着巨大的影响。过程，方法，实践，方法，纪律，信任，沟通风格，节奏。方式-方法-是基础和根本的。

You'll often hear people say “execution is everything,” but that's not quite right. In fact, it's often quite wrong.

你会经常听到人们说“执行就是一切”，但这种说法并不完全正确，事实上，这种说法经常是错误的。

When it comes to project work, and specifically software development, executing something the wrong way can destroy morale, grind teams down, erode trust, crunch gears, and wreck the machinery of long-term progress. So yeah, it's “done,” but at what cost? By doing, what have we done to ourselves? Do we really have to do that again, over and over month after month, year after year?

当涉及到项目工作，特别是软件开发时，以错误的方式执行某些东西会破坏士气，打击团队，削弱信任，磨损齿轮，并破坏长期进展的机器。所以，是的，它已经“完成”了，但代价是什么呢？通过做，我们对自己做了什么？我们真的需要一遍又一遍，一个月又一个月，一年又一年？

How many projects have you been a part of that you'd want to do over? How many projects have gone long, piled up at the end, and burned people out? How many projects were essentially collections of unreasonable expectations? How many projects turned teams against

each other, frustrated everyone from builder to stakeholder, and ultimately would have been better off dying than delivering?

你曾经参与过多少项目，想要重做？有多少项目已经进行了很长时  
间，最后堆积如山，把人们都累垮了？有多少项目本质上是不合理  
期望的集合？有多少项目让团队相互对立，让从建设者到利益相关  
者的每个人都感到沮丧，最终还是死了比交付更好？

Sometimes execution is everything—everything that's wrong. So what does executing right look like?

有时候执行就是一切——一切错误。那么执行正确是什么样子  
的呢？

Over the last few years, there's been a heightened curiosity about how we work at Basecamp. People often ask us how we get so much done so quickly at such a high level of quality with such a small team. And how we keep our teams together for years and years.

在过去的几年里，人们对我们在 Basecamp 的工作方式越来越好奇。  
人们经常问我们，在这么小的团队中，我们是如何在如此高的质量  
水平下如此迅速地完成这么多工作的。以及我们是如何让我们的团  
队年复一年地团结在一起的。

For one, we're not into waterfall or agile or scrum. For two, we don't  
首先，我们不喜欢瀑布，不喜欢敏捷，也不喜欢混乱，其次，我们  
不喜欢

line walls with Post-it notes. For three, we don't do daily stand ups, design sprints, development sprints, or anything remotely tied to a metaphor that includes being tired and worn out at the end. No backlogs, no Kanban, no velocity tracking, none of that.

在墙上贴上便利贴。第三，我们不做每天的单口相声，设计冲刺，开发冲刺，或任何远程联系到一个隐喻，包括疲惫和疲惫的结束。没有积压，没有看板，没有速度跟踪，什么都没有。

We have an entirely different approach. One developed in isolation over nearly 15 years of constant trial and error, taking note, iterating, honing in, and polishing up. We've shaped our own way.

我们有一个完全不同的方法。一种是在近 15 年的不断尝试和错误中孤立地发展起来的，记笔记，重复，磨练，完善。我们已经形成了自己的方式。

Blog posts, workshops, and occasional conference talks have provided glimpses of our own unique process, but we've never laid it bare for all to see. This book does just that.

博客文章、研讨会和偶尔的会议演讲让我们瞥见了我们自己独特的过程，但我们从未让所有人都看到它。这本书就是这样做的。

Now that our process is fully formed, documented, and ready to go, we're here to share it with all those curious enough to listen to a new way of doing things. Explorers, pioneers, those who don't care what everyone else is doing. Those who want to work better than the rest. 现在，我们的过程已经完全形成，文档化，并准备就绪，我们在  
这里与所有好奇心足够强的人分享它，聆听一种新的做事方式。  
探险家，先驱者，那些不在乎别人在做什么的人。那些想要比其他人工作得更好的人。

Don't think of this as a book. Think of it as a flashlight. You and your team have fumbled around in the dark long enough. Now you've got something bright and powerful to help you find a new way.

不要把这当成一本书。把它想象成一个手电筒。你和你的团队在黑暗中摸索的时间已经够长了。现在你有了一些光明而强大的东西来帮助你找到新的方法。

We hope you find it interesting, enlightening, and, most of all, helpful.

我们希望你觉得它有趣，有启发性，最重要的是，有帮助。

Thanks for reading.  
感谢阅读。

# Acknowledgements

## 鸣谢

Jason Fried and David Heinemeier Hansson, Basecamp's founders, planted many of the seeds for this book. It is informed by their values, Basecamp's culture, and fifteen years of collaborative trial-and-error. 大本营的创始人杰森弗里德和达维德海涅迈尔汉松为这本书埋下了许多种子。它是由他们的价值观, Basecamp 的文化, 和 15 年的合作试错。

Bob Moesta and Chris Spiek made pivotal contributions. This book wouldn't have come together without their help.

Bob Moesta 和 Chris Spiek 做出了关键性的贡献, 如果没有他们的帮助, 这本书不会走到一起。

Yaneer Bar-Yam's lectures at the New England Complex Systems Institute helped me structure the method.

Yaneer Bar-Yam 在新英格兰复杂系统研究所(New England Complex Systems Institute)的讲座帮助我构建了这种方法。

The expert designers and programmers at Basecamp tried, tested, and improved these techniques over the years to ship real projects. Their efforts make this a book of practice, not theory.

Basecamp 的专家设计师和程序员多年来尝试、测试和改进了这些技术来运送真正的项目。他们的努力使这本书成为一本实践书籍, 而非理论书籍。

**10**  
10 个

# Introduction

## 引言

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.

这本书是指导我们如何在大本营进行产品开发。它也是一个充满技术的工具箱，你可以用你自己的方式应用到你自己的过程中。

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.

无论你是创始人、首席技术官、产品经理、设计师还是开发人员，你来到这里很可能是因为所有软件公司都必须面对一些共同的挑战。

### Growing pains

#### 成长的烦恼

As software teams start to grow, some common struggles appear:  
随着软件团队开始成长，一些常见的问题出现了：

- Team members feel like projects go on  
and on, with no end in sight.

团队成员感觉项目一直在进行，没有结  
束的迹象。

- Product managers can't find time to think  
strategically about the product.

产品经理没有时间对产品进行战略性思考。

- Founders ask themselves: “Why can’t we get features out the door like we used to in the early days?”

创业者们问自己：“为什么我们不能像早期那样把特性推向市场呢？”

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

我们在 Basecamp 上亲眼目睹了这些挑战，因为我们从 4 个人成长到了 50 多岁。

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

Basecamp 在 2003 年开始作为我们自己构建的一个工具。当时我们是一家为客户设计网站的咨询公司。Information 会在客户、设计师和项目管理者之间的电话游戏中迷失。我们希望 Basecamp 成为一个集中的地方，在那里所有人都可以看到工作，讨论它，并知道下一步该做什么。事实证明，很多公司都存在这种“信息漏洞百出”的问题。今天，各行各业数以百万计的人依赖

on Basecamp as their shared source of truth.  
作为他们共同的真相来源。

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.

我们三个人建立了第一个版本。Jason Fried, Basecamp 的创始人, 领导了设计。他的联合创始人达维德海涅迈尔汉松(david heinemeier hansson)为其编写了程序(并创建了著名的 web 框架 Ruby on Rails 作为副产品)。当时我是一个专注于可用性和用户界面的网页设计师。我执行了杰森的设计方向, 为应用程序的关键特性, 并与他合作, 填补了概念的细节。

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.

从 2003 年 7 月的第一个原型到 2004 年 2 月的发射, David 每周只工作 10 个小时。我们知道, 除非我们非常谨慎地使用这十个小时的编程, 否则我们不会有任何进展。在这些限制条件下, 我们集中精力“锤炼”范围, 使之适应给定的时间预算。

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.

随着业务的发展，我开始拓展自己的技能。在 Rails 上与 David 和 Ruby 一起工作使得编程的世界对我来说变得更加容易接近。我学习了程序员用来驯服复杂性的技术：比如分解、抽象层次和关注点分离。我一只脚踏入了设计世界，一只脚踏入了编程世界，我想知道我们是否可以将这些软件开发原则应用到我们设计和管理产品的方式上。

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 seam-less 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

这个想法的第一次测试是在 2009 年。到那时，我们已经雇佣了更多的程序员，并提供了四个独立的软件即服务产品。我们想把这些产品捆绑在一起，形成一个单点登录和统一计费的无缝套件。这是一项巨大的技术工程，面向用户的流程充满了危险。除了让底层架构正确之外，我们还必须在客户进入产品的途中打断他们，让他们改变他们的

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.

用户名和密码的原因很难解释。我在项目中扮演了设计师和产品经理的角色，在这本书中描述了面包板和范围映射技术的原型来管理复杂性。

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.

我们获得了如此好的结果，以至于我们决定在 2012 年再次应用同样的技术，当时我们为 2.0 版重新设计了 Basecamp。再一次，我们需要管理大量的表面积，而且整个过程出人意料的顺利。

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.

到 2015 年，我们的核心团队经历了这些考验，取得了令人印象深刻的进步。但是我们发现很难说清楚我们对新员工做了什么。我们的产品团队扩大了四倍，每个人都远程工作。这使得我们很难传递我们的直觉。我们需要语言来描述我们正在做的事情，并且需要更多的结构来保持我们新的规模。

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.

为了管理这种新的能力，我们从临时的项目长度转变为重复的周期。(需要一些实验才能找到正确的周期长度: 六周。稍后再详细说明我们正式确定了我们的投球和下注流程。我的角色再次转变，从设计和产品管理到产品策略。我需要一种新的语言，比如“塑造”这个词，来描述我们在把项目交付给团队之前为了设定边界和减少风险而做的前期设计工作。

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.

正当我们越来越善于表达自己的工作方式时，越来越多的朋友和同事开始来问我们是如何做到的。终于有一天杰森把我拉到一边说，“我觉得你应该写一本关于这个的书。

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  
这就是结果。你可以把它想象成两本书合二为一。首先，这是  
一本关于基本真理的书。我希望它能给你更好的语言

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.

描述和处理产品开发过程中出现的风险、不确定性和挑战。其次，这本书概述了我们正在使用的具体过程，以我们目前的规模在我们的产品上取得有意义的进展。

Here's a short overview of the main ideas in the book.  
以下是本书主要观点的简要概述。

## Six-week cycles 六周周期

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.

首先，我们以六周为周期工作。六个星期的时间足够长，可以从头到尾构建一些有意义的东西，而且时间足够短，每个人都能从一开始就能感觉到最后期限即将到来，所以他们明智地利用这段时间。我们的大多数新功能都是在一个六周的周期内构建和发布的。

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.

我们的决定是基于在接下来的六周内推进产品，而不是微观管理时间。我们不会计算时间，也不会质疑每一天是如何度过的。我们没有每天的会议。我们不会每两周重新思考我们的路线图。我

们的关注点在更高的层次上。我们对自己说：“如果这个项目在六周后发布，我们会非常高兴。我们会觉得我们的时间花得很值。”然后我们投入六周的时间，让团队自己完成任务。

## Shaping the work

### 塑造工作

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.

其次，我们在把工作交给一个团队之前，先对工作进行定型。一个小型的高级团队与自行车团队平行工作。在我们考虑一个可以下注的项目之前，他们定义了解决方案的关键要素。项目是在正确的抽象层次上定义的：具体到团队知道要做什么，抽象到他们有足够的空间自己解决有趣的细节。

When shaping, we focus less on estimates and more on our  
在塑形时，我们较少关注估计，而更多地关注我们的

**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.

胃口。我们不会问做一些工作需要多少时间，而是问：我们想要花多少时间？这个想法值多少钱？这就是塑造的任务：缩小问题范围，设计出符合我们胃口的解决方案的轮廓。

## Making teams responsible 让团队负责

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.

第三，我们将全部责任交给一个由设计师和程序员组成的小型集成团队。他们定义自己的任务，对范围进行调整，一起构建产品的纵向切片。这与其他方法完全不同，在其他方法中，管理者分割工作，程序员扮演检票员的角色。

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.

总之，这些概念形成了一个良性循环。当团队更加自主的时候，高层人员可以花更少的时间管理他们。在管理上花费更少的时间，高级人员可以形成更好的项目。当项目形成得更好时，团队有更清晰的界限，因此可以更自主地工作。

## Targeting risk 瞄准风险

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?

在这个过程的每一步，我们都针对一个特定的风险：不能按时发货的风险。这本书不是关于建造错误的东西的风险。其他书籍可以帮助你解决这个问题(我们推荐《与运气竞争》)。改善你的发现过程应该在你重新获得发布的能力之后。你可以拥有世界上最好的策略，但是如果你不能付诸行动，它又有什么用呢？

This book is about the risk of getting stuck, the risk of getting bogged down with last quarter's work, wasting time on unexpected

这本书是关于陷入困境的风险，关于陷入上季度工作的风险，关于在意想不到的事情上浪费时间

problems, and not being free to do what you want to do tomorrow.  
问题，以及不能自由地做你明天想做的事。

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.

在我们把项目交付给一个时间框之前，我们通过解决开放性问题来降低成形过程中的风险。我们不会把一个项目交给一个仍然存在兔子洞或纠缠不清的相互依存关系的团队。

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.

我们通过将赌注限制在六周来降低计划过程中的风险。如果一个项目超时了，默认情况下它不会得到延期。这个“断路器”确保了我们不会在一个需要首先重新思考的概念上投入原始欲望的倍数。

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.

最后，我们通过尽早整合设计和编程来降低建造过程中的风险。我们没有构建大量互不相关的部分，并希望它们能在最后时刻拼凑在一起，而是尽早构建一个有意义的工作部分，然后重复。团队将工作从最不知名的部分排序到最不令人担忧的部分，并通过尽快整合来了解哪些工作可行，哪些不可行。

## How this book is organized 这本书是如何组织起来的

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

第一部分是关于塑造——在我们认为项目已经准备就绪之前，我们对项目所做的前期工作。每一章都解释了这个过程中的一个具体步骤，从设置一个原始想法的胃口，到勾勒出一个解决方案，再到写出一个展示潜在项目的推介。在这个过程中，你会学到一些具体的技巧

— like breadboarding and fat-marker sketching — to keep the design at the right level of abstraction.

就像面包板和脂肪标记草图一样，使设计保持在正确的抽象水平上。

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

第二部分是关于博彩——我们如何在项目中进行选择，并决定一次花六周时间做什么。

Part Three is about **Building** — the expectations we place on the teams and the special practices they use to discover what to do.

第三部分是关于建立——我们对团队的期望以及他们用来发现要做什么的特殊实践。

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.

我们将看到团队如何弄清楚要做什么，他们如何集成设计和编程，他们如何跟踪已知的和未知的，以及最终他们如何做出艰难的决定来按时完成项目。

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.

最后，附录给了你一些帮助，当你的公司需要做出改变的时候。这里有一些关于如何尝试第一个六周实验的建议，关于如何根据公司规模调整方法的提示，以及如何使用 Basecamp 实现 Shape Up 的具体指导。

CHAPTER 1 - Introduction

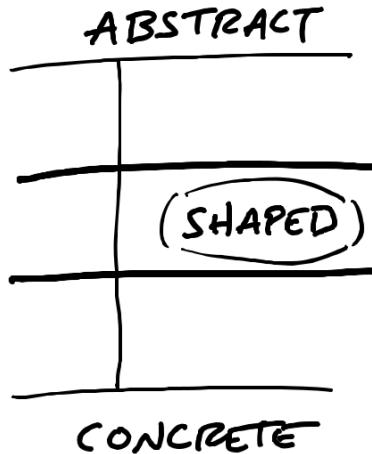
17

Chapter1-inttutn17



PARTONE  
Partone

# Shaping 塑形



## Principles of Shaping 塑造的原则

When we shape the work, we need to do it at the right level of abstraction: not too vague and not too concrete. Product managers often err on one of these two extremes.

当我们塑造工作时，我们需要在正确的抽象层次上进行：不要太模糊，也不要太具体。产品经理经常在这两个极端之一上犯错误。

### Wireframes are too concrete

线框图太具体了

When design leaders go straight to wireframes or high-fidelity mockups, they define too much detail too early. This leaves design-ers no room for creativity. One friend put it this way:

当设计领导者直接进入线框图或高保真模型，他们定义太多的细节太早。这使得设计师们没有创造性的空间。一个朋友这样说：

*I'll give a wireframe to my designer, and then I'm saying to her: "I know you're looking at this, but that's not what I want you to design."*

*I want you to re-think it!” It’s hard to do that when you’re giving them this concrete thing.*

我会给我的设计师一个线框图，然后我对她说：“我知道你在看这个，但这不是我想让你设计的。我希望你重新考虑一下！”当你给他们这个具体的东西的时候，很难做到这一点。

Over-specifying the design also leads to estimation errors. Counter-intuitive as it may seem, the more specific the work is, the harder it can be to estimate. That’s because making the interface *just so* can过度指定设计也会导致估计错误。虽然看起来有点违反直觉，但是工作越具体，估算就越困难。这是因为设计界面就是这么简单

require solving hidden complexities and implementation details that weren't visible in the mockup. When the scope isn't variable, the team can't reconsider a design decision that is turning out to cost more than it's worth.

需要解决隐藏的复杂性和实现细节，这些在模型中是看不到的。

当范围不是可变的，团队不能重新考虑一个设计决策，结果是成本超过了它的价值。

## Words are too abstract

文字太抽象了

On the other end of the spectrum, projects that are too vague don't work either. When a project is defined in a few words, nobody knows what it means. "Build a calendar view" or "add group notifications" sound sensible, but what exactly do they entail? Team members don't have enough information to make trade-offs. They don't know what to include or leave out. A programmer who worked in that situation said: 另一方面，那些过于模糊的项目也不起作用。当一个项目用几个词来定义时，没有人知道它意味着什么。“建立日历视图”或者“添加组通知”听起来很合理，但是它们到底意味着什么呢？团队成员没有足够的信息进行权衡。他们不知道应该包括或者忽略什么。

一位在这种情况下工作的程序员说：

*You're solving a problem with no context. You have to be a mind reader. It's like: "we'll know it when we see it."*

你正在解决一个没有上下文的问题。你必须成为一个读心者。

就像是：“当我们看到它的时候，我们就会知道它。”

Concerning estimation, under-specified projects naturally grow out of control because there's no boundary to define what's out of scope.

关于评估，指定不足的项目自然会失去控制，因为没有界限来定义范围之外的东西。

## Case study: The Dot Grid Calendar 案例研究: Dot 网格日历

Let's look at an example of how to shape a project at the right level of detail.

让我们来看一个如何在正确的细节层次上塑造项目的示例。

We launched version three of Basecamp without a calendar feature. It had a “schedule” feature that just listed events one after the other without any kind of monthly, weekly or daily grid.

我们发布了没有日历功能的 Basecamp 第三版。它有一个“日程表”功能，只是列出一个接一个的事件，没有任何形式的每月，每周或每日网格。

Soon after launch, customers started asking us to “add a calendar” to Basecamp. We had built calendars before and we knew how complex they are. It can easily take six months or more to build a proper calendar.

发布后不久，客户就开始要求我们在 Basecamp 中“添加一个日历”。  
我们之前已经建立了日历，我们知道它们有多复杂。建立一个合适的日历可能需要 6 个月或更长的时间。

These are the kinds of things that make a calendar complicated:  
就是这些事情让日历变得复杂：

- Dragging and dropping events between cells to move them  
拖放单元格之间的事件来移动它们
- Wrapping multi-day events around the edge of the screen  
围绕屏幕边缘包装多日事件
- Different views for monthly, weekly, or daily time scales  
每月、每周或每日时间尺度的不同视图
- Dragging the edge of an event to change its duration  
拖动事件的边缘以更改其持续时间
- Color coding events for different categories  
不同类别的颜色编码事件
- Handling different expectations for  
desktop vs. mobile interactions  
处理桌面与移动交互的不同期望

Past versions of Basecamp had calendars, and only about 10% of customers used them. That's why we didn't have the appetite for spending six months on a calendar. On the other hand, if we could do something to satisfy those customers who were writing us in one six week cycle, we were open to doing that.

过去版本的 Basecamp 有日历，只有 10% 的客户使用它们。这就是为什么我们没有兴趣在日历上花费六个月的时间。另一方面，如果我们可以做一些事情来满足那些在一个六周周期内给我们写信的客户，我们愿意这样做。

With only six weeks to work with, we could only build about a tenth of what people think of when they say “calendar.” The question became: which tenth?

只有六个星期的工作时间，我们只能建立大约十分之一的什么人们认为当他们说“日历”问题变成了：哪十分之一？

We did some research (discussed in the next chapter) and narrowed down a use case that we wanted to solve. We eventually arrived at a promising concept inspired by calendars on phones. We could build a two-month, read-only grid view. Any day with an event would have a dot for each event. A list of events would appear below the calendar, and clicking a day with a dot would scroll the events for that day into view. We called it the Dot Grid.

我们做了一些研究(在下一章中讨论)，缩小了我们想要解决的用例范围。我们最终得到了一个很有前途的概念，灵感来自于手机上的日历。我们可以建立一个为期两个月的只读网格视图。任何有事件的日子都会有一个点来表示每个事件。一个事件列表会出现在日历的下方，点击一个点将当天的事件滚动到视图中。我们称之为 Dot Grid。

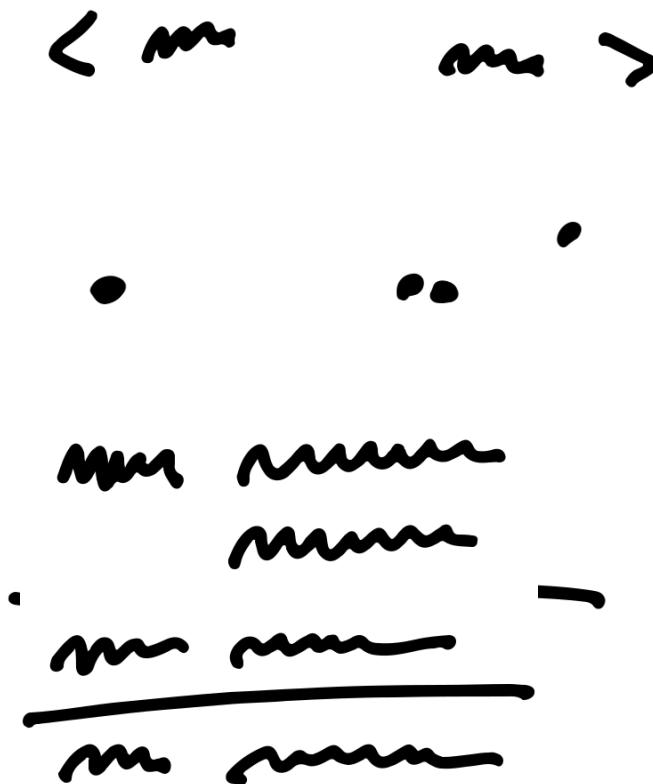
The Dot Grid wasn't a full-featured calendar. We weren't going to allow dragging events between days. We weren't going to span Dot 网格并不是一个功能齐全的日历。我们不允许在两天之间拖动事件。我们不会跨越时间

multi-day events across the grid; we'd just repeat the dots. There'd be no color coding or categories for events. We were comfortable with all these trade-offs because of our understanding of the use case.

多天的事件，我们只是重复那些点。事件没有颜色编码或分类。

由于我们对用例的理解，我们对所有这些权衡都很满意。

This is the level of fidelity we used to define the solution:  
这是我们用来定义解决方案的保真度级别：



*Rough sketch of the Dot Grid concept*  
*Dot 网格概念的粗略草图*

Note how rough the sketch is and how many details are left out. The designer had a lot of room to interpret how this should look and feel.

注意草图有多粗糙，有多少细节被遗漏。设计师有很大的空间来解释这个应该看起来和感觉起来怎么样。

At the same time, note how specific the idea is. It's very clear how it works, what needs to be built, what's in and what's out.

同时，注意这个想法是多么具体。很清楚它是如何运作的，需要建造什么，什么是进来的，什么是出来的。

At the end of the project, the finished work that the designers and programmers created looked like this:

在项目结束时，设计师和程序员完成的工作看起来是这样的：

August							September						
SUN	MON	TUE	WED	THU	FRI	SAT	SUN	MON	TUE	WED	THU	FRI	SAT
		1	2	3	4	5						1	2
6	7	8	9	10	11	12	3	4	5	6	7	8	9
13	14	15	16	17	18	19	10	11	12	13	14	15	16
20	21	22	23	24	25	26	17	18	19	20	21	22	23
27	28	29	30	31			24	25	26	27	28	29	30

**Wed, Aug 23** Interview with The TaTa Top 🎙  
9:30am - 10:00am  
Interview with Tim Krepp 🎙  
12:00pm - 12:30pm  
[Add an event](#)

**Mon, Aug 28** Interview with Amy Trager 🎙  
11:00am - 11:30am

**Wed, Aug 30** Interview with Jenifer of Colorstock 🎙  
10:00am - 10:30am

*Screenshot of the Dot Grid when it launched*  
网格的屏幕截图

This small example highlights a few properties of shaped work.  
这个小例子突出了成形工作的一些属性。

**Property 1: It's rough**

属性 1: 粗糙

Work in the shaping stage is rough. Everyone can tell by looking at it that it's unfinished. They can see the open spaces where their contributions will go. Work that's too fine, too early commits

在成型阶段的工作是粗糙的。每个人都可以通过观察它来判断它是否未完成。他们可以看到他们的贡献将会流向哪里。太精细的工作，太早的承诺

everyone to the wrong details. Designers and programmers need room to apply their own judgement and expertise when they roll up their sleeves and discover all the real trade-offs that emerge.

每个人都有错误的细节。当设计师和程序员卷起袖子，发现所有真正的权衡时，他们需要空间来运用他们自己的判断和专业知识。

## Property 2: It's solved

### 特性 2: 问题解决了

Despite being rough and unfinished, shaped work has been thought through. All the main elements of the solution are there at the macro level and they connect together. The work isn't specified down to individual tasks, but the overall solution is spelled out. While surprises might still happen and icebergs could still emerge, there is clear direction showing what to do. Any open questions or rabbit holes we could see up front have been removed to reduce the project's risk.

尽管是粗糙和未完成的，成形的工作已经通过深思熟虑。解决方案的所有主要元素都存在于宏观层面，并且它们连接在一起。这项工作并没有具体到单个任务，但是整体的解决方案已经阐明。

虽然意外仍然可能发生，冰山仍然可能出现，但是有一个清晰的方向告诉我们该怎么做。为了降低项目的风险，我们已经移除了所有可能出现的问题或漏洞。

## Property 3: It's bounded

### 特性 3: 它是有界的

Lastly, shaped work indicates what *not* to do. It tells the team where to stop. There's a specific appetite—the amount of time the team is allowed to spend on the project. Completing the project within that fixed amount of time requires limiting the scope and leaving specific things out.

最后，有形的工作表明什么是不该做的。它告诉团队在哪里停下来。有一个特定的需求——团队被允许在项目上花费的时间。在规定的时间内完成项目需要限制范围，忽略特定的事情。

Taken together, the roughness leaves room for the team to resolve all the details, while the solution and boundaries act like guard rails. They reduce risk and channel the team's efforts, making sure they don't build too much, wander around, or get stuck.

总的来说，粗糙度为团队解决所有细节留下了空间，而解决方案和边界就像护栏一样。他们降低了风险，引导团队的努力，确保他们不会建造太多，四处游荡，或者陷入困境。

## Who shapes 谁塑造了

Shaping is creative and integrative. It requires combining interface ideas with technical possibilities with business priorities. To do that you'll need to either embody these skills as a generalist or collaborate with one or two other people.

塑形是创造性和综合性的。它需要将界面概念、技术可能性和业务优先级结合起来。要做到这一点，你需要要么将这些技能体现为一个通才，要么与其他一两个人合作。

Shaping is primarily design work. The shaped concept is an interaction design viewed from the user's perspective. It defines what the feature does, how it works, and where it fits into existing flows.

整形主要是设计工作。成形的概念是一个从用户的角度来看的交互式设计。它定义了功能是什么，如何工作，以及它在现有流程中的位置。

You don't need to be a programmer to shape, but you need to be technically literate. You should be able to judge what's possible, what's easy and what's hard. Knowledge about how the system works will help you see opportunities or obstacles for implementing your idea.

你不需要成为一个程序员来塑造，但是你需要有技术素养。你应该能够判断什么是可能的，什么是容易的，什么是困难的。了解这个系统是如何工作的，可以帮助你看到实现你想法的机会或障碍。

It's also strategic work. Setting the appetite and coming up with a solution requires you to be critical about the problem. What are we trying to solve? Why does it matter? What counts as success? Which customers are affected? What is the cost of doing this instead of something else?

这也是一项战略性的工作。设置胃口和提出解决方案需要你对问题持批判态度。我们试图解决什么问题？为什么这很重要？什么是成功？哪些客户受到影响？这样做而不是做其他事情的代价是什么？

Shaping is a closed-door, creative process. You might be alone sketching on paper or in front of a whiteboard with a close collaborator. There'll be rough diagrams in front of you that nobody outside the room would be able to interpret. When working with a collaborator, you move fast, speak frankly and jump from one promising position to another. It's that kind of private, rough, early work.

整形是一个封闭的、创造性的过程。你可能独自在纸上或者在白板前画草图，还有一个亲密的合作者。在你面前会有一些粗略的图表，在房间外面没有人能够解释。当与合作者一起工作时，你行动迅速，说话坦率，从一个有前途的职位跳到另一个。这是一种私人的，粗糙的，早期的工作。

## Two tracks 两首歌

You can't really schedule shaping work because, by its very nature, unshaped work is risky and unknown. For that reason we have two separate tracks: one for shaping, one for building. During any six week cycle, the teams are building work that's been previously shaped and the shapers are working on what the teams might potentially build in a future cycle. Work on the shaping track is kept private and not shared with the wider team until the commitment

你不能真的安排塑形工作，因为从本质上来说，不成形的工作是有风险和未知的。出于这个原因，我们有两条独立的轨道：一条用于塑造，一条用于建造。在任何六个星期的周期中，团队正在构建先前形成的工作，而形成者正在研究团队可能在未来周期中构建的内容。塑造轨道的工作是保密的，在承诺之前不会与更广泛的团队分享

has been made to bet on it. That gives the shapers the option to put work-in-progress on the shelf or drop it when it's not working out. 已经被迫下注了。这使得塑造者可以选择把正在进行的工作放在架子上，或者在工作不顺利的时候放弃。

## Steps to shaping

### Steps to shapes 塑形的步骤

Shaping has four main steps that we will cover in the next four chapters.

Shape 有四个主要步骤，我们将在接下来的四章中讨论。

**1. Set boundaries.** First we figure out how much time the raw idea is worth and how to define the problem. This gives us the basic boundaries to shape into.

设置边界。首先，我们要弄清楚原始想法值多少时间，以及如何定义问题。这给了我们一个基本的界限。

**2. Rough out the elements.** Then comes the creative work of sketching a solution. We do this at a higher level of abstraction than wireframes in order to move fast and explore a wide enough range of possibilities. The output of this step is an idea that solves the problem within the appetite but without all the fine details worked out.

粗略处理一下这些元素。然后是草拟解决方案的创造性工作。我们在一个比线框图更高的抽象层次上做这件事，以便快速移动并探索足够广泛的可能性。这个步骤的输出是一个在需求范围内解决问题的想法，但是没有所有的细节。

**3. Address risks and rabbit holes.** Once we think we have a solution, we take a hard look at it to find holes or unanswered questions that could trip up the team. We amend the solution, cut

things out of it, or specify details at certain tricky spots to prevent the team from getting stuck or wasting time.

解决风险和漏洞。一旦我们认为我们有了一个解决方案，我们就会仔细研究它，找出可能会让团队陷入困境的漏洞或未解决的问题。我们修改解决方案，删除其中的内容，或在某些棘手的地方指定细节，以防止团队陷入困境或浪费时间。

4. **Write the pitch.** Once we think we've shaped it enough to potentially bet on, we package it with a formal write-up called a pitch. The pitch summarizes the problem, constraints, solution, rabbit holes, and limitations. The pitch goes to the betting table for consideration. If the project gets chosen, the pitch can be re-used at kick-off to explain the project to the team.

写推销词。一旦我们认为我们已经形成了足够的潜在赌注，我们包装它与正式的写作称为推销。推销总结了问题、约束、解决方案、漏洞和局限性。这个投注将被放到投注台上进行考虑。如果项目被选中，球场可以在开球时重复使用，向球队解释项目。



## Set Boundaries

## 设定界限

The first step of shaping is setting boundaries on what we're trying to do. The conversations we have are going to be entirely different if people think we're talking about a small improvement or a major redesign.

塑造的第一步是设定我们正在尝试做的事情的界限。如果人们认为我们谈论的是一个小的改进或者一个重大的重新设计，那么我们的谈话将会完全不同。

The conversation about building a feature always starts with a raw idea, like “customers are asking for group notifications.” Before we all go down the rabbit hole discussing ways we can solve it, we should first set some broad terms on the discussion to make it productive.

关于构建特性的讨论总是从一个原始的想法开始，比如“客户要求组通知”在我们陷入讨论如何解决这个问题的兔子洞之前，我们首先应该在讨论中设置一些宽泛的术语，使其更有效率。

## Setting the appetite

## 设置食欲

Sometimes an idea gets us excited right away. In that case we need to temper the excitement by checking whether this is really something we're going to be able to invest time in or not. If we don't stop to think about how valuable the idea is, we can all jump too quickly to either committing resources or having long discussions about potential solutions that go nowhere.

有时候一个想法会让我们立刻兴奋起来。在这种情况下，我们需要通过检查这是否真的是我们能够投入时间的东西来缓和兴奋。如果我们不停下来思考一下这个想法有多么有价值，我们可能都会过于匆忙地投入资源，或者就可能的解决方案进行长时间的讨论，结果却毫无进展。

Other ideas are less exciting and feel more like a challenge we didn't ask for. The customer wants a calendar; we don't particularly want to build one, but we do feel we need to do something about the request. 其他的想法不那么令人兴奋，感觉更像是一个我们没有要求的挑战。客户想要一个日历；我们并不特别想要建立一个，但是我们确实觉得我们需要对这个请求做些什么。

Whether we're chomping at the bit or reluctant to dive in, it helps to explicitly define how much of our time and attention the subject deserves. Is this something worth a quick fix if we can manage?

无论我们是迫不及待还是不情愿，它都有助于明确定义我们应该花多少时间和注意力在这个主题上。如果我们能处理好这些问题，那么这些问题是否值得快速解决呢？

Is it a big idea worth an entire cycle? Would we redesign what we already have to accommodate it? Will we only consider it if we can implement it as a minor tweak?

这个主意值得一整个周期吗？我们会重新设计我们已经不得不适应它的东西吗？我们是否只会考虑将其作为一个小的调整来实现？

We call this the appetite. You can think of the appetite as a time budget for a standard team size. We usually set the appetite in two sizes:

我们称之为欲望。你可以把胃口想象成一个标准团队规模的时间预算。我们通常将胃口设置为两种规模：

- Small Batch: This is a project that a team of one designer and one or two programmers can build in one or two weeks. We batch these together into a six week cycle (more on that later).

小批量：这是一个由一个设计师和一两个程序员组成的团队可以在一到两周内完成的项目。我们把这些项目组合在一起，形成一个六周的周期(稍后详细介绍)。

- Big Batch: This project takes the same-size team a full six-weeks.  
大批量: 这个项目需要相同规模的团队整整六个星期。

In rare cases where the scope is so big that a six-week project isn't conceivable, we'll try to hammer it down by narrowing the problem definition. If we still can't shrink the scope, we'll break off a meaningful part of the project that we can shape to a six-week appetite.

在极少数情况下, 范围是如此之大, 以至于六周的项目是不可想象的, 我们将尝试通过缩小问题定义来敲定它。如果我们仍然不能缩小范围, 我们将中断项目的一个有意义的部分, 我们可以塑造一个六个星期的胃口。

## Fixed time, variable scope 固定时间, 可变范围

An appetite is completely different from an estimate. Estimates start with a design and end with a number. Appetites start with a number and end with a design. We use the appetite as a creative constraint on the design process.

食欲与估计是完全不同的。估计从一个设计开始, 以一个数字结束。Appetite 以数字开始, 以设计结束。我们把欲望作为设计过程中的创造性约束。

This principle, called “fixed time, variable scope,” is key to successfully defining and shipping projects. Take this book for an example. It’s hard to ship a book when you can always add more, explain more, or improve what’s already there. When you have a dead-line, all of a sudden you have to make decisions. With one week left, I can choose between fixing typos or adding a new section to a chapter. That’s the tension between time, quality, and scope. I don’t want to release a book with embarrassing typos, so I’ll choose to reduce the scope by leaving out the extra section. Without the pressure of the fixed deadline, I wouldn’t make the trade-off. If the scope wasn’t variable, I’d *have* to include the extra section. Then there’d be no time to fix the quality issues.

这个原则被称为“固定时间，可变范围”，是成功定义和运送项目的关键。以这本书为例。当你总是可以添加更多，解释更多，或者改进已经存在的内容时，就很难发布一本书。当你有一个截止日期时，突然间你不得不做出决定。在剩下的一周时间里，我可以选择是修正打字错误还是在章节中增加一个新的章节。这就是时间、质量和范围之间的紧张关系。我不想发行一本有着令人尴尬的拼写错误的书，所以我会选择省略额外的部分来缩小范围。没有固定截止日期的压力，我不会做出取舍。如果范围不是可变的，我就必须包括额外的部分。然后就没有时间去解决质量问题了。

We apply this principle at each stage of the process, from shaping potential projects to building and shipping them. First, the appetite constrains what kind of a solution we design during the shaping process. Later, when we hand the work to a team, the fixed time box pushes them to make decisions about what is core to the project and what is peripheral or unnecessary.

我们在过程的每个阶段都应用这个原则，从塑造潜在的项目到建造和运输它们。首先，欲望约束了我们在设计过程中设计的解决方案的类型。后来，当我们把工作交给一个团队时，固定的时间框会推动他们做出决定，什么是项目的核心，什么是外围的或不必要的。

“Good” is relative

”好”是相对的

There's no absolute definition of “the best” solution. The best is relative to your constraints. Without a time limit, there's always a better version. The ultimate meal might be a ten course dinner. But when you're hungry and in a hurry, a hot dog is perfect.

“最佳”解决方案没有绝对的定义。最好的是相对于你的约束。没有时间限制，总会有更好的版本。最终的晚餐可能是一顿十道菜的晚餐。但是当你饿的时候，赶时间的时候，热狗是最好的选择。

The amount of time we set for our appetite is going to lead us to different solutions. We could model a whole set of database columns in the fancy version, or just provide a flat textarea in the simple version. We could redesign the main landing page to accommo-date a new feature, or we could push it back to a screen with fewer

我们为自己的胃口设定的时间长度将引导我们找到不同的解决方案。我们可以在花哨的版本中建立一个完整的数据库列模型，或者在简单的版本中提供一个扁平的文本区域。我们可以重新设计主着陆页面以适应新的功能，或者我们可以把它推回到一个更少的屏幕上

design constraints. We can only judge what is a “good” solution in the context of how much time we want to spend and how important it is.  
设计限制。我们只能根据我们想花多少时间和它有多重要来判断什么是“好”的解决方案。

## Responding to raw ideas 回应原始的想法

Our default response to any idea that comes up should be: “Interesting. Maybe some day.” In other words, a very soft “no” that leaves all our options open. We don’t put it in a backlog. We give it space so we can learn whether it’s really important and what it might entail.

我们对出现的任何想法的默认回应应该是：“in-interesting。也许有一天。”换句话说，一个非常温和的“不”，留给我们所有的选择。我们不会把它放在积压文件中。我们给它空间，这样我们就能知道它是否真的重要，以及它可能需要什么。

It’s too early to say “yes” or “no” on first contact. Even if we’re excited about it, we shouldn’t make a commitment that we don’t yet understand. We need to do work on the idea before it’s shaped enough to bet resources on. If we always say “yes” to incoming requests we’ll end up with a giant pile of work that only grows.

在第一次接触的时候说“是”或“不是”还为时过早。即使我们对此感到兴奋，我们也不应该做出一个我们还不理解的承诺。我们需要在这个想法形成之前就开始工作，以便在它上面投入足够的资源。如果我们总是对收到的请求说“是”，我们最终会得到一大堆工作，而这些工作只会越来越多。

It’s important to keep a cool manner and a bit of a poker face. We don’t want to shut down an idea that we don’t understand. New information might come in tomorrow that makes us see it differently. On the other

hand, showing too much enthusiasm right away can set expectations that this thing is going to happen. We may not be able to commit to it once we've put it into context with everything else we want to do.

保持冷静的态度和一张扑克脸是很重要的。我们不想关闭一个我们不理解的想法。明天可能会有新的信息出现，让我们有不同的看法。另一方面，马上表现出太多的热情会让人期待这件事情会发生。一旦我们把它和我们想做的其他事情联系起来，我们就无法对它做出承诺。

## Narrow down the problem 缩小问题范围

In addition to setting the appetite, we usually need to narrow down our understanding of the problem.

除了设置食欲，我们通常需要缩小我们对问题的理解。

We once had a customer ask us for more complex permission rules. It could easily have taken six weeks to build the change she wanted.

Instead of taking the request at face value, we dug deeper.

我们曾经有一个客户要求我们提供更复杂的许可规则。要做出她想要的改变很容易需要六个星期的时间。我们没有按照请求的表面价值去做，而是更深入地挖掘。

It turned out that someone had archived a file without knowing the file would disappear for everyone else using the system. Instead of creating a rule to prevent some people from archiving, we realized we could put a warning on the archive action itself that explains the impact. That's a one-day change instead of a six-week project.

原来有人在不知道其他所有使用系统的人都会消失的情况下存档了一个文件。我们意识到我们可以对存档操作本身加上一个警告，来解释这种影响，而不是创建一个规则来阻止一些人进行存档。  
这是一个一天的改变，而不是一个六周的项目。

Another example is the “calendar view” from the previous chapter. Everyone knows what a calendar view is. But unpacking it revealed tons of unknowns and decisions that would drastically affect the scope. If we only want to spend six weeks instead of six months building a huge calendar, how do we narrow it down?

另一个例子是前一章中的“日历视图”。每个人都知道什么是日历视图。但是揭开它揭示了大量的未知数和决定，这些都会极大地影响它的范围。如果我们只想花六个星期而不是六个月建立一个巨大的日历，我们如何缩小范围？

In that case we flip from asking “What could we build?” to “What’s really going wrong?” Sure, a calendar sounds nice. But what is driving the request? At what point specifically does someone’s current workflow break down without this thing they’re asking for?

在这种情况下，我们不再问“我们能建造什么？”到“到底出了什么问题？”当然，日历听起来不错。但是是什么驱动了这个请求呢？在什么情况下，人们当前的工作流程会在没有他们所要求的东西的情况下崩溃？

Case study: Defining “calendar”  
案例研究：定义“日历”

In the case of the calendar request, we called a customer who asked for this feature. Instead of asking her why she wants a calendar and what it should look like, we asked her *when* she wanted a calendar. What was she doing when the thought occurred to ask for it?

在日历请求的情况下，我们打电话给一个要求这个特性的客户。我们没有问她为什么需要一个日历以及它应该是什么样子，而是问她什么时候需要一个日历。当她想要的时候，她在做什么？

She told us she worked in an office with a big calendar drawn on a chalkboard wall. Her officemates marked when they were meeting clients in the handful of meeting rooms on the calendar. One day she was working from home. A client called and asked her to schedule a meeting. She had to drive to the office to look at the wall calendar. Traffic was terrible along the way, and in the end there wasn't a free space that worked for her client. She could have saved an hour in traffic and a lot of frustration if she had been able to check for open spots on the calendar from her computer at home.

她告诉我们，她在一间黑板墙上画着大日历的办公室里工作。她的同事们会在日历上的几个会议室里记录他们与客户见面的时间。有一天，她在家工作。一个客户打电话给她，让她安排一个会议。她不得不开车去办公室看挂历。一路上交通状况很糟糕，最后没有一个空闲的地方能让她的客户使用。如果她能从家里的电脑上查看日历上的空位，她本可以在堵车时节省一个小时，也可以减少许多挫折感。

The insight wasn't "computerize the calendar"—that's obvious. What we learned was that "see free spaces" was the important thing for this use case, not "do everything a calendar does."

洞察力不是“电脑化日历”——这是显而易见的。我们学到的是“查看空闲空间”是这个用例的重要内容，而不是“做日历做的所有事情”

This story, and others like it, gave us a specific baseline to design against. Basecamp had an agenda view of events. It worked for listing major deadlines and milestones but it wasn't good for re-source scheduling because you couldn't see empty spaces on it. We narrowed down the need from "do everything a calendar does" to "help me see free spaces so I can figure out when to

这个故事，以及其他类似的故事，给了我们一个设计的基准。

Basecamp 有一个事件的议程视图。它可以列出主要的截止日期和里程碑，但是它不适合资源调度，因为你看不到上面的空白。我们将需求从“做日历上的每一件事”缩小到“帮助我看到空闲的空间，这样我就可以知道什么时候去做”

schedule something."

安排一下。”

We didn't have a solution yet. But now we felt like we had a problem that was specific enough to spark an idea that could fit within our appetite. This led us to the simpler "Dot Grid" concept from the last chapter.

我们还没有一个解决方案。但是现在我们觉得我们遇到了一个问题，这个问题非常具体，足以激发出一个符合我们胃口的想法。这让我们想到了上一章中更简单的“Dot Grid”概念。

What if we can't figure out a specific pain point or use case? Our appetite can also tell us how much research is worthwhile. If it's not critical now and we can't get our hands around the problem, we'll walk

away from it and work on something else. Maybe in the future a new request or story will pop up that gives us better insight into the problem. 如果我们不能找出特定的痛点或者用例呢？我们的胃口也可以告诉我们有多少研究是值得的。如果现在不是关键时刻，我们不能解决这个问题，我们就会离开它，去做其他的事情。也许在未来，一个新的请求或故事会弹出来，让我们更好地洞察问题。

## Watch out for grab-bags 当心抓包

When it comes to unclear ideas, the worst offenders are “rede-signs” or “refactorings” that aren’t driven by a single problem or use case. When someone proposes something like “redesign the Files section,” that’s a grab-bag, not a project. It’s going to be very hard to figure out what it means, where it starts, and where it ends. Here’s a more productive starting point: “We need to rethink the Files section because sharing multiple files takes too many steps.”

当涉及到不明确的想法时，最糟糕的违规者是“重新标记”或“重构”，它们不是由单一的问题或用例驱动的。当有人提出诸如“重新设计文件部分”之类的建议时，那只是一个抓包，而不是一个项目。很难弄清楚它的含义，它从哪里开始，又从哪里结束。这里有一个更有效的起点：“我们需要重新考虑文件部分，因为共享多个文件需要太多的步骤。”

Now we can start asking: What's not working? In what context are there too many steps? What parts of the existing design can stay the same and what parts need to change?

现在我们可以开始问: 什么不起作用? 在什么情况下有太多的步骤?

现有设计的哪些部分可以保持不变, 哪些部分需要改变?

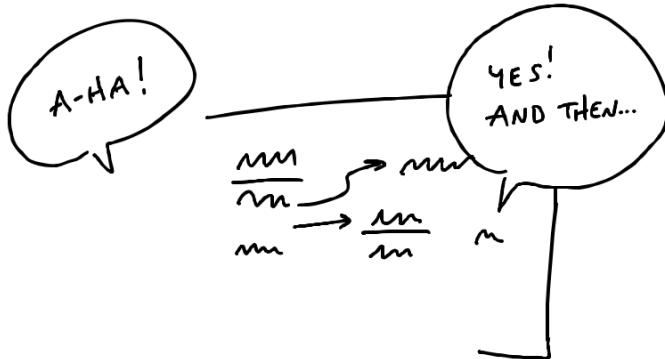
A tell-tale sign of a grab-bag is the “2.0” label. We made the mistake in the past of kicking off a “Files 2.0” project without really considering what that meant. Our excitement about improving a huge part of our app got the better of us. We know there were a lot of problems with our Files feature, but we didn’t ask ourselves what specifically we were going to do. The project turned out to be a mess because we didn’t know what “done” looked like. We recovered by splitting the project into smaller projects, like “Better file previews” and “Custom folder colors.” We set appetites and clear expectations on each project and shipped them successfully.

抓包的一个标志就是“2.0”标签。我们过去犯了一个错误, 在开始“文件 2.0”项目的时候并没有真正考虑它的含义。我们对于改进我们的应用程序的一大部分的兴奋使我们失去了理智。我们知道我们的文件功能有很多问题, 但是我们没有问自己具体要做什么。这个项目最终变得一团糟, 因为我们不知道“完成”是什么样子。我们通过把项目分成更小的项目来恢复, 比如“更好的文件预览”和“自定义文件夹颜色”我们对每个项目都设定了兴趣和明确的期望, 并成功地将它们发布出去。

## Boundaries in place 界限就位

When we have all three things—a raw idea, an appetite, and a narrow problem definition—we’re ready to move to the next step and define the elements of a solution.

当我们拥有了三样东西——一个原始的想法、一种欲望和一个狭隘的问题定义——我们就准备好进入下一个步骤并定义解决方案的要素。



## Find the Elements

## Find the Elements 寻找元素

Now that we have the constraints of an appetite and the problem we're solving, it's time to get from an idea in words to the elements of a software solution. There could be dozens of different ways to approach the solution for a problem. So it's important that we can move fast and cover a lot of different ideas without getting dragged down.

既然我们已经受到欲望和我们正在解决的问题的限制，现在是时候从文字上的想法到软件解决方案的元素了。对于一个问题的解决方案可以有几十种不同的方法。所以重要的是，我们可以快速行动，涵盖很多不同的想法，而不会被拖下水。

### Move at the right speed

### 以正确的速度前进

Two things enable us to move at the right speed at this stage.  
在这个阶段，有两个因素使我们能够以正确的速度前进。

First, we need to have the right people—or nobody—in the room. Either we’re working alone or with a trusted partner who can keep pace with us. Someone we can speak with in shorthand, who has the same background knowledge, and who we can be frank with as we jump between ideas.

首先，我们需要有合适的人——或者没有人——在这个房间里。

要么我们单独工作，要么我们有一个值得信赖的合作伙伴，他可以跟上我们的步伐。一个我们可以用速记方式交谈的人，一个拥有相同背景知识的人，一个我们可以坦诚相待的人。

Second, we need to avoid the wrong level of detail in the drawings

第二，我们需要避免图纸中错误的细节水平

and sketches. If we start with wireframes or specific visual layouts, we'll get stuck on unnecessary details and we won't be able to explore as broadly as we need to.

还有素描。如果我们从线框图或者特定的视觉布局开始，我们就会陷入不必要的细节，我们就不能像我们需要的那样进行广泛的探索。

The challenge here is to be concrete enough to make progress on a specific solution without getting dragged down into fine details. The questions we're trying to answer are:

这里的挑战是要足够具体，在一个具体的解决方案上取得进展，而不是被拖入细节。我们试图回答的问题是：

- Where in the current system does the new thing fit?

在当前的体系中，新事物适用于哪些方面？

- How do you get to it?

你是如何得到它的？

- What are the key components or interactions?

什么是关键组件或交互作用？

- Where does it take you?

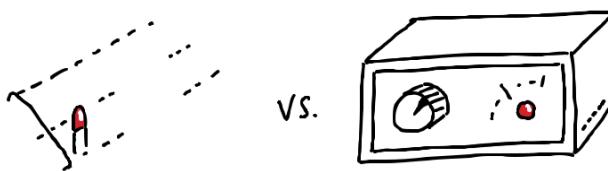
它会把你带到哪里？

To stay on the right level of detail and capture our thoughts as they come, we work by hand using a couple of prototyping techniques: breadboarding and fat marker sketches. These allow us to quickly draw different versions of entire flows so we can debate the pros and cons of each approach and stay aligned with what we're talking about as we go. 为了保持正确的细节水平，捕捉我们的想法，我们使用一些原型技术手工工作：面包板和脂肪标记草图。这些方法允许我们快速绘制整个流程的不同版本，这样我们就可以讨论每种方法的优缺点，并与我们所讨论的内容保持一致。

## Breadboarding 面包板

We borrow a concept from electrical engineering to help us design at the right level of abstraction. A breadboard is an electrical engineering prototype that has all the components and wiring of a real device but no industrial design.

我们从电气工程中借鉴了一个概念，以帮助我们在正确的抽象层次上进行设计。面板是一个电气工程原型，它包含了一个真实设备的所有部件和线路，但没有工业设计。



Deciding to include an indicator light and a rotary knob is very different from debating the chassis material, whether the knob should go to the left of the light or the right, how sharp the corners should be, and so on.

决定包括一个指示灯和一个旋转旋钮是非常不同于辩论底盘材料，旋钮应该去向灯的左边还是右边，如何尖锐的角落，等等。

Similarly, we can sketch and discuss the key components and connections of an interface idea without specifying a particular visual design. To do that, we can use a simple shorthand. There are three basic things we'll draw:

类似的，我们可以草图和讨论关键组件和连接的界面想法，而不指定一个特定的视觉设计。为了做到这一点，我们可以使用一个简单的速记。我们将画三个基本的东西：

1. Places: These are things you can navigate to, like screens, dialogs, or menus that pop up.

地点: 这些是你可以导航到的东西，像屏幕，对话框，或者弹出的菜单。

2. Affordances: These are things the user can act on, like buttons and fields. We consider interface copy to

支持: 这些都是用户可以操作的东西，比如按钮和字段。  
我们考虑界面复制

be an affordance, too. Reading it is an act that gives the user information for subsequent actions.

阅读它是为用户提供后续行动信息的一种行为。

3. Connection lines: These show how the affordances take the user from place to place.

连接线: 这些显示了 afford 舞蹈如何把用户从一个地方带到另一个地方。

We'll use words for everything instead of pictures. The important things are the components we're identifying and their connections. They allow us to play out an idea and judge if the sequence of actions serves the use case we're trying to solve.

我们会用文字代替图片。重要的是我们正在识别的组成部分和它们之间的联系。它们允许我们展示一个想法，判断行动的顺序是否符合我们试图解决的用例。

### *Example*

#### *例子*

Suppose our product is an invoicing tool. We're considering adding a new "Autopay" feature to enable our customers' customers to pay future invoices automatically.

假设我们的产品是一个发票工具。我们正在考虑添加一个新的“自动支付”功能，使客户的客户能够自动支付未来的发票。

How do you turn Autopay on? What's involved? We can pick a starting point and say that the customer landed on an invoice. That's  
如何打开自动转账功能？这涉及到什么？我们可以选择一个起点，  
假设客户开出了一张发票。这就是

our first place. We draw it by writing the name of the place and underlining it.

第一个地方。我们通过写下地方的名字和下划线来画它。

INVOICE

On the invoice, we're thinking we could add a new button to "Turn on Autopay." That's an affordance. Affordances go below the line to indicate they can be found at that place.

在发票上，我们想我们可以在“打开自动转账”上加一个新按钮这是一个启示。它们在线下表示可以在那个地方找到它们。

INVOICE

TURN ON  
AUTOPAY

Where does that button go? Some place for setting up the Autopay. We don't have to specify whether it's a separate screen or a pop up modal or what. From a what's-connected-to-what standpoint (the topology) it's all the same. Let's draw a connection line from the button to the Setup Autopay screen.

这个按钮通向哪里？设置自动转账的地方。我们不需要指定它是一个单独的屏幕还是一个弹出模式或者其他什么。从什么连接到什么立场(拓扑结构)都是一样的。让我们画一条从按钮到安装自动支付屏幕的连接线。

INVOICE

TURN ON  
AUTOPAY

SET UP AUTOPAY

Now we can talk about what belongs on that screen. Do we ask for a credit card here? Is there a card on file already? What about ACH or other payment methods?

现在我们可以谈谈屏幕上的东西了。我们在这里申请信用卡吗？

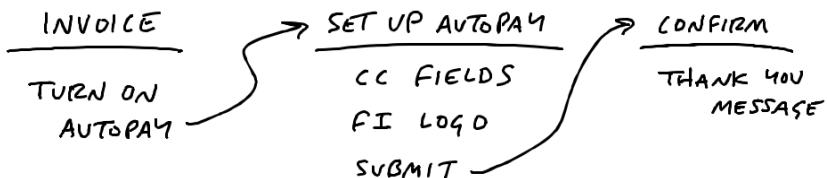
已经存档了吗？ACH 或者其他支付方式呢？

Just figuring out what to write under the bar starts to provoke  
仅仅是弄清楚在酒吧下面写些什么就开始激起人们的兴趣了

debates and discussions about what to build.  
关于建造什么的辩论和讨论。

As we think it through, we decide we should ask for credit card details here and show the logo of the financial institution (an aspect of the domain in this specific product).

当我们想通了，我们决定我们应该要求信用卡的细节在这里，并显示金融机构的标志(在这个特定的产品领域的一个方面)。



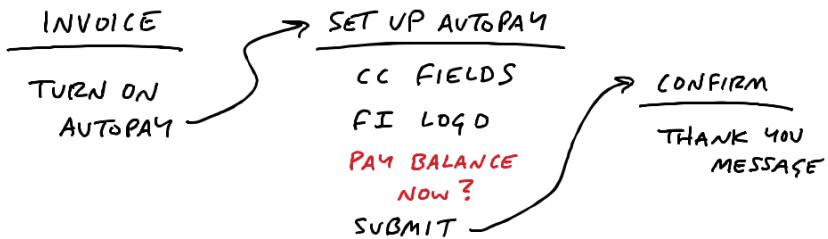
Straightforward enough. But wait — did we actually pay the original invoice or not? Hm. Now we have both functional and interface questions. What does enabling Autopay actually do? Does it apply only for the future or does paying with Autopay the first time also pay the current invoice? And where do we explain this behavior? We're starting to have deeper questions and discussions prompted by just a few words and arrows in the breadboard.

非常简单。但是，等等——我们实际上是否支付了原始发票？嗯。现在我们既有功能问题，也有界面问题。启用自动转账到底能做什么？它只适用于未来还是第一次使用自动转账支付也支付当前的发票？我们该如何解释这种行为呢？我们开始有了更深层次的问题和讨论，而这些问题和讨论仅仅是由面板上的几个词和箭头引起的。

Since we're using such a lightweight notation, and we aren't bogged down with wireframes, we can quickly jump around and entertain different possibilities.

因为我们使用的是如此轻量级的符号，而且我们没有陷入线框图的泥潭，所以我们可以快速地跳跃和娱乐不同的可能性。

We could add an option to the Setup screen...  
我们可以在安装界面添加一个选项..。

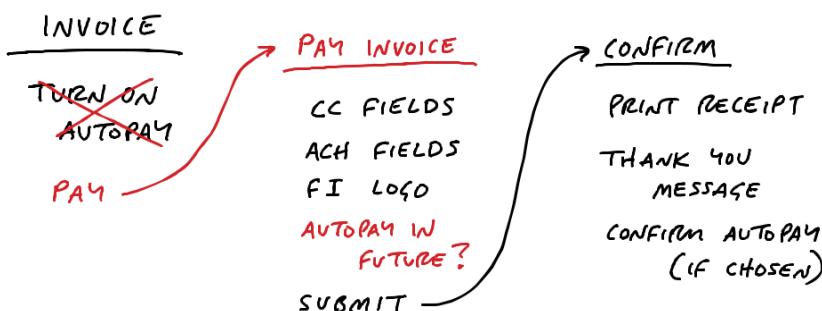


But now we're complicating the responsibilities of the confirmation screen. We're going to need to show a receipt if you pay your balance now. Should the confirmation have a condition to sometimes show a receipt of the amount just paid?

但是现在我们正在复杂化确认屏幕的职责。如果你现在支付余额，我们需要出示收据。确认书是否应该有一个条件——有时需要显示刚刚支付的金额的收据？

How about an entirely different approach. Instead of starting on an Invoice, we make Autopay an option when making a payment. This way there's no ambiguity about whether the current amount is being paid. We could add an extra "Autopay was enabled" callout to the existing payment confirmation page.

不如换个方法。与开发票不同，我们在付款时可以选择自动付款。这样一来，当前金额是否已经支付就不再模棱两可了。我们可以现有的付款确认页面上添加一个额外的“Autopay was enabled”标注。



Sketching this out reminded us that the current payment form supports ACH in addition to credit card. We discuss and confirm that we can use ACH too.

这个草图提醒我们，目前的支付表单除了支持信用卡以外，还支持 ACH。我们讨论并确认我们也可以使用 ACH。

What about after Autopay is enabled? How does the customer turn it off? Up to this point, many customers in the system didn't have usernames or passwords. They followed tokenized links to pay the invoices one by one. One might naturally assume that now that the customer has something like Autopay, they need a username and password and some landing place to go manage it.

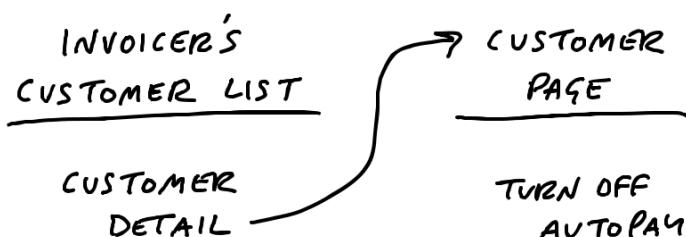
启用自动转帐后会怎样？客户如何关闭它？到目前为止，系统中的许多客户都没有用户名或密码。他们按照标记链接一个一个地支付发票。人们可能会很自然地认为，既然客户已经有了像Autopay这样的东西，他们就需要一个用户名和密码，以及某个着陆点来管理它。

The team in this case decided that adding the username/password  
在这种情况下，团队决定添加用户名/密码

40  
4万

flows was too much scope for their appetite at the time. Reflecting strategically on what they knew about their customers, they thought it would be quite alright if the invoicer's customers had to reach out to the invoicer and ask them to turn off the Autopay. In that case we could add a single option to disable Autopay in the customer detail page that we already offered to invoicees. We drew out the flow like this:

资金流动在当时对他们的胃口来说太大了。从战略上反思他们对客户的了解，他们认为如果发票人的客户不得不与发票人联系并要求他们关闭自动转账，那就没什么问题了。在这种情况下，我们可以已经在提供给发票人的客户详细信息页面中添加一个禁用自动转账的选项。我们画出了这样的流程：



This example illustrates the level of thinking and the speed of movement to aim for during the breadboarding phase. Writing out the flows confronts us with questions we didn't originally think of and stimulates design ideas without distracting us with unimportant visual choices.

这个例子说明了在面包旅行阶段要达到的思维水平和行动速度。

写出这些流程会让我们面对一些我们最初没有想到的问题，并且在不分散我们注意力的情况下激发我们的设计想法。

Once we get to a place where we play through the use case and the flow seems like a fit, we've got the elements we need to move on to start defining the project more clearly. We're getting more concrete while still leaving out a huge amount of detail.

一旦我们到了一个地方，我们通过用例和流似乎是一个合适的，我们已经有了我们需要继续的元素，开始定义项目更清楚。我们在保留大量细节的同时获得了更多的具体信息。

## Fat marker sketches

### 脂肪标记素描

Sometimes the idea we have in mind is a visual one. Breadboarding would just miss the point because the 2D arrangement of elements is the fundamental problem. In that case, we still don't want to

有时候我们脑海中的想法只是一个视觉上的想法。面包板会忽略这一点，因为元素的二维排列是最基本的问题。在这种情况下，我们仍然不想这样做

waste time on wireframes or unnecessary fidelity. Instead we use fat marker sketches.

浪费时间在线框图或不必要的保真度上。相反，我们使用脂肪标记草图。

A fat marker sketch is a sketch made with such broad strokes that adding detail is difficult or impossible. We originally did this with larger tipped Sharpie markers on paper. Today we also do it on iPads with the pen size set to a large diameter.

一个肥胖的标记草图是一个草图与这样广泛的笔触，添加细节是困难的或不可能的。我们最初是用更大的笔尖在纸上做这个。现在我们也在 ipad 上使用笔的大小设置为大直径。

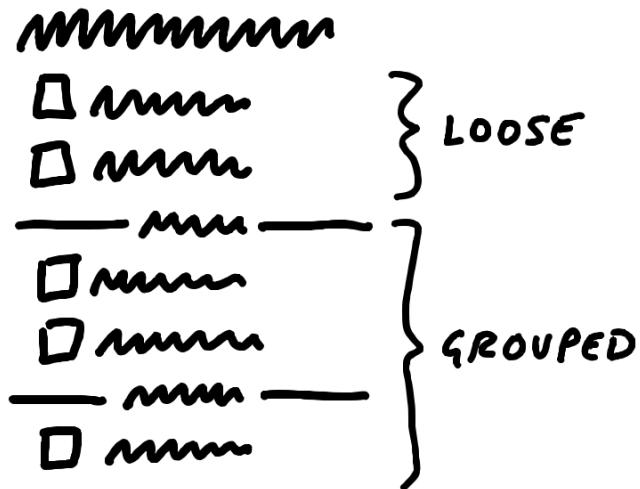
Here's an example. We found ourselves often creating fake to-dos in our Basecamp to-do lists that acted as dividers. We'd create an item like "— Needs testing —" and put items below it. We had the idea to make some kind of official divider feature in our to-do tool to turn the workaround into a first class function of to-do lists.

这里有一个例子。我们发现自己经常在我们的 Basecamp 待办事项列表中创建假的待办事项，作为分割线。我们会创建一个类似“--需要测试--”的项目，然后把这些项目放在下面。我们的想法是在待办事项工具中创建某种官方分割器功能，将工作区转化为待办事项列表的第一类函数。

We had to work out what the implications of adding a divider were. We came up with a rough idea that adding a divider separates the list into “loose” to-dos above the divider and “grouped” to-dos below. Adding subsequent dividers adds more groups below the “loose” items at the top.

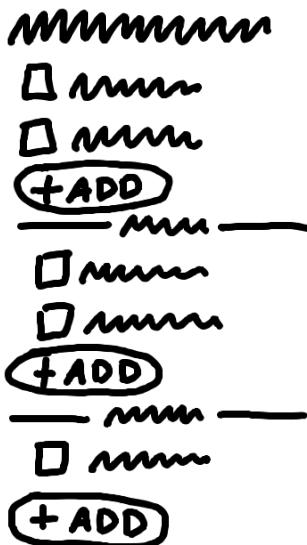
我们必须弄清楚添加一个分隔符意味着什么。我们提出了一个粗略的想法，即添加一个分隔符将列表分为分隔符上方的“松散”待办

事项和下方的“分组”待办事项。添加后续的分隔符会在顶部的“松散”项目下面添加更多的组。



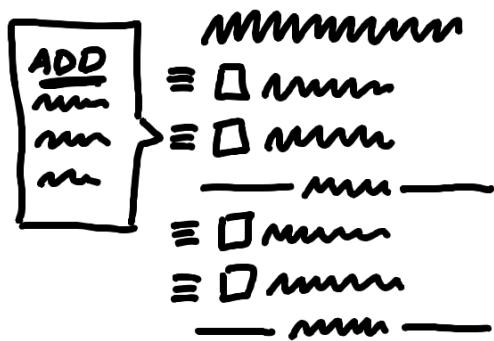
We could add items via some affordance within each group, including the “loose” group on top.

我们可以通过在每个组中添加一些项目，包括在顶部的“松散”组。



We were a little concerned the add buttons might break up the gestalt of the list, and the groups might all separate too much from the lists on the page. We talked about possibilities to place the “add” affordance inside of a menu that we already had to the left of each to-do item.

我们有点担心添加按钮可能会破坏列表的格式塔，组可能都从页面上的列表中分离得太多。我们讨论了将“添加”功能放在菜单中的可能性，我们已经把它放在了每个待办事项的左边。



CHAPTER 4 - Find the Elements  
Chap tr4-findeeeeeeeeets43

43

This notation is much less constraining than breadboards, which has downsides. We might sketch a sidebar and get attached to a layout element like that even though it's not a core element. But as long as we keep an eye on that we're still far better off than if we get sucked into the weeds by creating wireframes too early.

这个符号比面板的约束要少得多，因为它有缺点。我们可以画一个侧边栏，然后附加一个布局元素，即使它不是一个核心元素。但是，只要我们注意到这一点，我们仍然会比过早创建线框图而陷入困境要好得多。

It may seem a little silly to call fat marker sketches a technique or a tool. The reason for calling them out is we too easily skip ahead to the wrong level of fidelity. Giving this rough early stage a name and using a specific tool for it helps us to segment our own creative process and make sure we aren't jumping ahead to detail a specific idea when we haven't surveyed the field enough.

将脂肪标记草图称为一种技术或工具似乎有点愚蠢。之所以把它们叫出来，是因为我们太容易跳到错误的忠实度水平。给这个粗糙的早期阶段一个名字，并使用一个特定的工具，它帮助我们分割我们自己的创造过程，并确保我们没有跳到细节的具体想法，当我们没有调查该领域足够。

## Elements are the output 元素就是输出

In the case of the Autopay example, we ended up with some clear elements:

在自动转账的例子中，我们得到了一些清晰的元素：

- A new “use this to Autopay?” checkbox on the existing “Pay an invoice” screen

在现有的“支付发票”屏幕上有一个新的“用

这个来自动转账?”复选框

- A “disable Autopay” option on the invoicer’s side

For the To-Do Groups project, the elements were:

发票方面的“禁用自动转账”选项对于 To-Do Groups

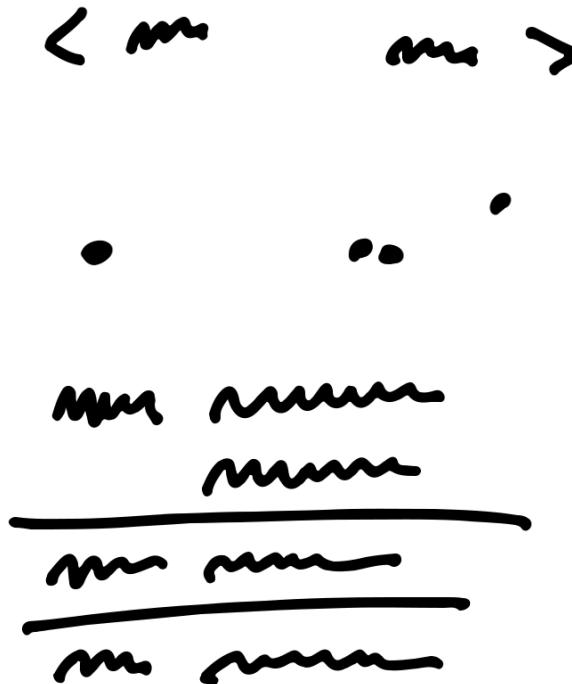
项目，元素如下：

- Loose to-dos above the first group belong directly to the parent  
第一组上面的 Loose to-dos 直接属于父组
- Grouped to-dos appear below the loose to-dos  
分组的待办事项出现在松散的待办事项下面
- We’d like to try an *add* affordance within each section, but if  
that doesn’t work visually, we’re ok with relying on the  
action menu for inserting to-dos into position.

我们想尝试在每个部分中添加一个可行性，但是如果那不能在视觉上工作，我们可以依靠操作菜单插入待办事项到位置。

Similarly, when we sketched the simplified solution for rendering events on a calendar grid, we used the fat marker approach.

同样，当我们在日历网格上绘制事件渲染的简化解决方案时，我们使用了胖标记的方法。



This enabled us to work out the main elements of the solution:  
这使我们能够找出解决方案的主要要素：

- A 2-up monthly calendar grid  
2-up 每月日历网格
- Dots for events, no spanned pills  
短期疗法用于治疗事件，没有延期服药
- Agenda-style list of events below that scrolls  
an event into view when you tap a dot  
议程式的事件列表，当你点击一个点时，下面的事件会滚动到视图中

This list of elements is extremely narrow and specific compared to “monthly calendar.” Exactly the kind of narrowing we hope to accomplish through the shaping process.

与“月历”相比，这个元素列表非常狭窄和具体正是我们希望通过塑造过程实现的那种缩小。

## Room for designers 设计师的空间

Later, when it's time to involve a designer, you don't want to have to say "I know I drew it like this but ignore that...". Regardless of what you say, any specific mockups are going to bias what other people do after you—especially if you're in a higher position than them. They'll take every detail in the initial mockups as direction even though you didn't intend it.

以后，当需要设计师参与的时候，你不想说“我知道我是这样画的，但是忽略它……”。不管你说什么，任何特定的模型都会偏向其他人在你之后所做的事——尤其是当你处于比他们更高的位置时。他们会把初始模型中的每个细节作为方向，即使你并不是故意的。

Working at the right "level of abstraction" not only ensures we move at the right speed, it also leaves this important room for creativity in the later stages.

在正确的“抽象层次”工作不仅确保我们以正确的速度前进，而且在后期阶段留下了创造性的重要空间。

By leaving details out, the breadboard and fat marker methods give room to designers in subsequent phases of the project.

通过省略细节，面板和脂肪标记方法在项目的后续阶段为设计师提供了空间。

This is a theme of the shaping process. We're making the project more specific and concrete, but still leaving lots of space for decisions and choices to be made later. This isn't a spec. It's more like the boundaries and rules of a game. It could go in countless different ways once it's time to play.

这是一个主题的形成过程。我们正在使这个项目更具体，更具体，但仍然留下了很多空间，以便以后做出决定和选择。这不是一个

规范。它更像是游戏的界限和规则。一旦游戏时间到了，它可以以无数种不同的方式进行。

## Not deliverable yet

### 目前还不能交付

This step of shaping is still very much in your private sphere. It's normal for the artifacts at this point — on the wall or in your notebook — to be more or less indecipherable to anybody who wasn't there with you.

这一步的塑造仍然是非常私人的领域。在这个时候，这些艺术品——在墙上或者在你的笔记本上——对于那些没有和你在一起的人来说或多或少都是难以辨认的，这很正常。

We've gone from a cloudy idea, like "autopay" or "to-do groups," to a specific approach and a handful of concrete elements. But the form we have is still very rough and mostly in outline.

我们已经从一个模糊的想法，如“自动付款”或“待办事项小组”，到一个具体的方法和一些具体的元素。但是我们现有的形式仍然非常粗糙，大部分只是轮廓。

What we've done is landed on an approach for how to solve the  
我们所做的就是找到一种方法来解决

problem. But there may be some significant unknowns or things we need to address before we'd consider this safe to hand off to a team to build successfully.

问题。但是在我们认为交给一个团队成功构建之前，可能还有一些重要的未知因素或者事情需要解决。

The next step is to do some stress-testing and de-risking. We want to check for holes and challenges that could hinder the project from shipping within the fixed time appetite that we have in mind for it.

下一步是做一些压力测试和降低风险。我们想要检查那些可能阻碍项目在固定时间内完成的漏洞和挑战。

After that we'll see how to wrap up the shaped concept into a write-up for pitching.

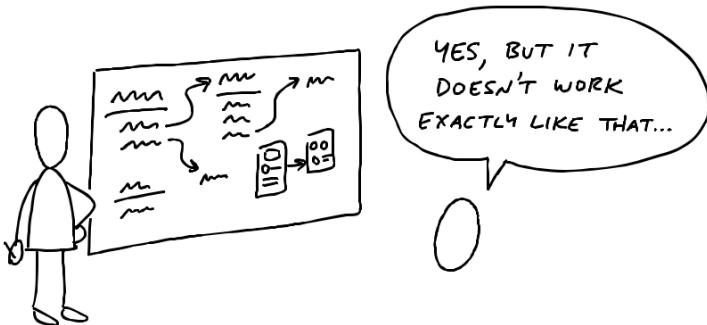
之后，我们将看到如何包装成形的概念成为一个投球的报告。

## No conveyor belt 没有传送带

Also keep in mind that, at this stage, we could walk away from the project. We haven't bet on it. We haven't made any commitments or promises about it. What we've done is added value to the raw idea by making it more actionable. We've gotten closer to a good option that we can later lobby for when it's time to allocate resources.

还要记住，在这个阶段，我们可以放弃这个项目。我们还没有打赌。我们没有做出任何承诺。我们所做的就是通过让这个原始的想法变得更加可行来增加它的价值。我们已经离一个好的选择越来越近了，我们可以在以后分配资源的时候进行游说。

CHAPTER 4 - Find the Elements      **47**  
Chapter4-findtheElements47



## Risks and Rabbit Holes

## 风险与兔子洞

Remember that we're shaping work for a fixed time window. We may trust from our experience that the elements we fleshed out in the previous chapter are buildable within the appetite (six weeks). But we need to look closer, because all it takes is one hole in the concept to derail that. Suppose we bet on the project and a team takes it on. If they run into an unanticipated problem that takes two weeks to solve, they just burned a third of the budget!

记住，我们正在为一个固定的时间窗口塑造工作。我们可以从我们的经验中相信，我们在前一章中充实的元素在我们的胃口(六周)内是可以构建的。但是我们需要更仔细地观察，因为所有这些只需要概念上的一个漏洞就可以让它脱离轨道。假设我们在这个项目上下了赌注，一个团队接手了这个项目。如果他们遇到了一个意想不到的问题，需要两个星期来解决，他们只是烧掉了三分之一的预算！

Even worse, sometimes you run into problems that don't just delay the project—they have no apparent solution. We once bet on a project to redesign the way we present projects with clients on Basecamp's home

screen. We assumed the designer would figure it out; we didn't do the work in the shaping phase to validate that a viable approach existed. Once the project started, it turned out to be a much harder problem than we expected. None of us were able to find a suitable design solution within the six weeks we budgeted. We ended up abandoning the project and rethinking it later.

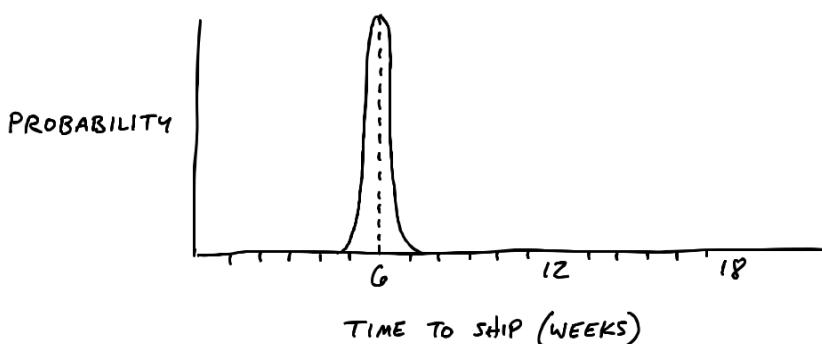
更糟糕的是，有时候你会遇到一些问题，这些问题不仅仅是延迟了项目——它们没有明显的解决方案。我们曾经押注于一个项目，重新设计我们在 Basecamp 主屏幕上向客户展示项目的方式。我们认为设计师会搞清楚这个问题，但是我们没有在设计阶段验证一个可行的方法是否存在。一旦项目开始，它变成了一个比我们预期更难的问题。在我们预算的六个星期内，没有一个人能够找到合适的设计方案。我们最终放弃了这个项目，并在以后重新考虑它。

Of course there will always be unknowns. That's why we apply the many practices in Part Three so that teams tackle the right problems in the right order, leaving room for the unexpected. But that doesn't mean we shouldn't look for the pitfalls we *can* find up front and eliminate them before betting on the project. Before we consider it safe to bet on, a shaped project should be as free of holes as possible.

当然，未知总是存在的。这就是为什么我们应用第三部分中的许多实践，以便团队以正确的顺序处理正确的问题，为意想不到的事情留出空间。但是，这并不意味着我们不应该在项目上下赌注之前寻找那些我们可以提前发现并消除的陷阱。在我们认为下注是安全的之前，一个有形的项目应该尽可能的避免漏洞。

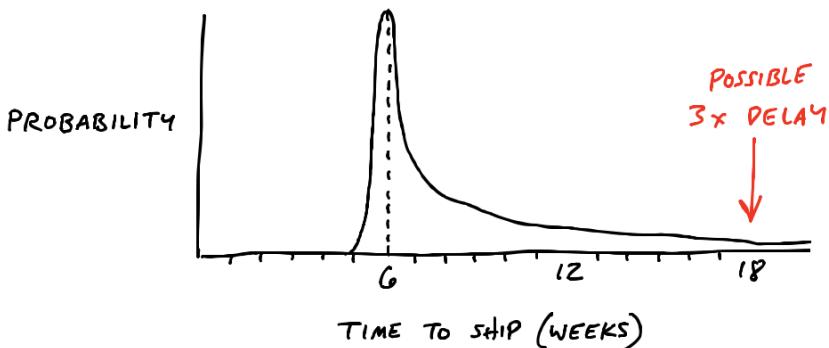
## Different categories of risk 不同类别的风险

In terms of risk, well-shaped work looks like a thin-tailed probability distribution. There's a slight chance it could take an extra week but, beyond that, the elements of the solution are defined enough and familiar enough that there's no reason it should drag on longer than that.  
就风险而言，形状良好的工作看起来像一个薄尾概率分布。有一点可能需要额外的一个星期，但是，除此之外，解决方案的元素已经定义得足够明确，足够熟悉，没有理由拖得比这更久。



However, if there are any rabbit holes in the shaping—technical unknowns, unsolved design problems, or misunderstood interdependencies—the project could take *multiple times* the original appetite to complete. The right tail stretches out.

然而，如果形成过程中有任何漏洞——技术未知、未解决的设计问题或误解的相互依存关系——项目可能需要多倍于原始意愿的时间才能完成。右边的尾巴伸出来了。



We want to remove the unknowns and tricky problems from the project so that our probability is as thin-tailed as possible. That means a project with independent, well-understood parts that assemble together in known ways.

我们希望从这个项目中去除未知和棘手的问题，这样我们的可能性就会尽可能的小。这意味着一个项目要有独立的、众所周知的、以已知方式组装在一起的部分。

## Look for rabbit holes

### 寻找兔子洞

Fleshing out the elements of the solution was a fast-moving, exploratory process. It was more breadth than depth. In this step, we slow down and look critically at what we came up with. Did we miss anything? Are we making technical assumptions that aren't fair?

充实解决方案的要素是一个快速发展的探索性过程。它更多的是广度而不是深度。在这一步中，我们放慢速度，严格审视我们得出的结论。我们错过了什么吗？我们是否做出了不公平的技术假设？

One way to analyze the solution is to walk through a use case in slow motion. Given the solution we sketched, how exactly would a user get from the starting point to the end? Slowing down and playing it out can reveal gaps or missing pieces that we need to design.

分析解决方案的一种方法是以慢动作遍历一个用例。根据我们草拟的解决方案，用户到底是如何从起点到终点的呢？放慢速度，逐步完成可以发现我们需要设计的缺口或者缺失的部分。

Then we should also question the viability of each part we think we solved. We ask ourselves questions like:

然后我们还应该质疑我们认为已经解决的每一个部分的可行性。

我们问自己这样的问题：

- Does this require new technical work we've never done before?  
这是否需要我们从未做过的新技术工作？
- Are we making assumptions about how the parts fit together?  
我们是否在假设零件是如何组合在一起的？

- Are we assuming a design solution exists that we couldn't come up with ourselves?

我们是否假设存在一个我们自己想不出来的设计解决方案？

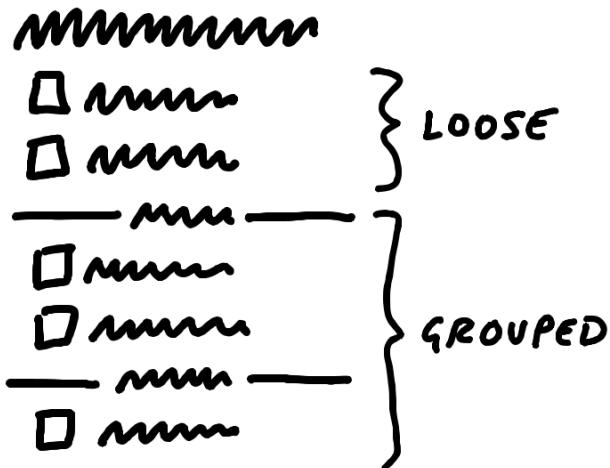
- Is there a hard decision we should settle in advance so it doesn't trip up the team?

是否有一个艰难的决定，我们应该提前解决，这样就不会绊倒球队？

### Case study: Patching a hole 案例分析：修补漏洞

For example, when we defined the To-Do Groups project, we introduced the idea of dividers in the to-do list:

例如，当我们定义 To-Do Groups 项目时，我们在 To-Do 列表中引入了分隔符的概念：



We liked the idea of the dividers, and the logic of loose versus grouped to-dos made sense to us. But when we looked closer we realized that we didn't address how to display completed items. In the pre-existing

design, the latest few completed items displayed below the list. Should we now render completed items at the bottom of each group instead of the list? Or should we continue to show completed items at the bottom, and repeat the same set of dividers within the completed items section? Should we reconsider how we handle completed items entirely?

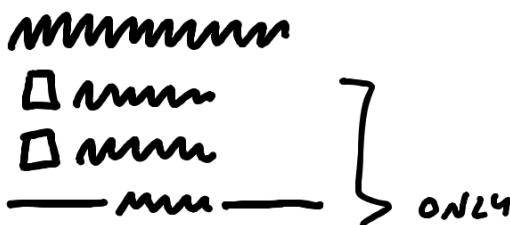
我们喜欢分割的想法，松散和分组的逻辑对我们来说是有意义的。但是当我们仔细观察的时候，我们意识到我们没有解决如何显示完成的项目。在已有的设计中，最新的几个已完成的项目显示在列表下面。我们现在是否应该将已完成的项目放在每个组的底部而不是列表中？或者我们应该继续在底部显示已完成的项目，并在已完成项目部分重复相同的分隔符集合？我们应该重新考虑如何处理完整的项目吗？

This was a hole in the concept. If we didn't address it, we'd be pushing a deep design problem down to the team and unreasonably asking them to find a solution under deadline. It's not responsible to give the team a tangled knot of interdependencies and then ask them to untangle it within a short fixed time window.

这是概念上的一个漏洞。如果我们不解决这个问题，我们就会把一个深层次的设计问题推给团队，毫无理由地要求他们在最后期限内找到解决方案。给团队一个纠结的相互依赖的结，然后让他们在短时间内解开，这是不负责任的。

We knew from experience that changing the way completed to-dos render has lots of complicated implications in user experience, navigation, and performance. To remove uncertainty in the project, we decided to dictate a solution in the shaped concept. We would leave the completed items exactly as they worked previously. Instead of grouping or segmenting them, we would just append the name of the group to each completed item. It would be a little messy, but we justified the trade-off: it drastically simplified the problem, and we could still show completed items from a group on the group's detail page.

我们从经验中了解到，改变完成 to-dos 渲染的方式会在用户体验、导航和性能方面产生许多复杂的影响。为了消除项目中的不确定性，我们决定在成形的概念中指定一个解决方案。我们会让已经完成的项目完全保持之前的工作状态。我们不会对它们进行分组或分段，而只是在每个完成的项目后面添加组的名称。这可能会有点混乱，但我们证明了这种权衡：它大大简化了问题，而且我们仍然可以在组的详细页面上显示组中完成的项目。



**52**

This is the kind of trade-off that's difficult to make when you're working inside the cycle under pressure. There are lots of reasons why a different design or a deeper reconsideration of completed to-dos would be objectively better. Why not try rendering them inside each group? A designer could reasonably think, "Maybe if I experiment with the styling a little more I can make them blend in better." They could easily waste a few days of the very few weeks they have going down a dead end.

当你在压力下的循环中工作时，这是一种很难做出的权衡。有很多原因可以解释为什么不同的设计或者对完成的待办事项进行更深层次的重新考虑客观上会更好。为什么不尝试在每个小组内部渲染呢？一个设计师可以合理地思考，“也许如果我多尝试一些风格，我可以让他们更好地融入其中。”他们可以很容易地在走进死胡同的几个星期里浪费几天时间。

As shapers, we're thinking less about the ultimate design and more about basic quality and risk. With the compromised concept we get to keep all the elements that made the project worth doing—the groups of incomplete items—and we get to cut off a big tail of risk.

作为塑造者，我们考虑的更多的是基本的质量和风险，而不是最终的设计。有了妥协的概念，我们可以保留所有让项目值得做的元素——不完整项目的组合——并且我们可以切断风险的大尾巴。

Next, when we write the pitch for this project, we'll point out this specific "patch" as part of the concept. That way nobody down the line will get tripped up on it.

接下来，当我们为这个项目写推介时，我们将指出这个特定的“补丁”作为概念的一部分。这样就不会有人在这个问题上犯错了。

## Declare out of bounds 宣布出界

Since everyone on the team wants to do their best work, they will of course look for all the use cases to cover and consider them necessary. As the team gets more comfortable with scope hammering (see Decide When to Stop), this improves. But it's still a good idea to call out any cases you specifically *aren't* supporting to keep the project well within the appetite.

因为团队中的每个人都希望做到最好，所以他们当然会寻找所有的用例来覆盖并认为它们是必需的。随着团队越来越适应范围敲击（参见“决定何时停止”），这种情况就会得到改善。但是这仍然是一个好主意，呼吁任何情况下，你特别是不支持，以保持项目良好的胃口。

For example, we worked on an idea for notifying groups of people in Basecamp. Rather than checking off five programmers one by one, you could just click “Programmers” and they'd be selected for notification. As we looked at the product, we saw tons of places where this kind of behavior might make sense. If we let you choose a group when posting a message, why not when assigning a to-do,

例如，我们曾经想过一个办法，通知 Basecamp 中的人群。而不是一个一个的检查五个程序员，你只需要点击“程序员”，他们就会被选中进行通知。当我们看到这个产品的时候，我们看到很多地方这种行为可能是有意义的。如果我们让你在发布信息的时候选择一个组为什么不在分配待办事项的时候，

or mentioning people in the chat room?  
或者在聊天室里提到别人？

We decided for the purpose of the project that the core value was narrowing down who to notify about a message. We explicitly marked off the other cases as “out of bounds” for the project and focused on the win we wanted: a faster flow for posting messages.

出于项目的目的，我们决定核心价值是缩小通知消息的对象范围。我们明确地将其他情况划分为项目的“边界”，并专注于我们想要的胜利：更快的发布消息流。

## Cut back 削减

There may be parts of the solution we got excited about during the sketching phase that aren't really necessary. When we designed the To-Do Groups feature, we thought it would be great to color-code groups. No doubt the page would look more interesting with color-coded group labels, and the feature might be more useful too. But we decided to flag this as unnecessary and cut it from the core of the project. We could mention it to the team as a nice-to-have, but everyone should start from the assumption that the feature is valuable without it.

可能有一部分解决方案，我们在草图阶段感到兴奋，但实际上并不是必要的。当我们设计 To-Do Groups 功能时，我们认为这对于颜色代码组来说是非常棒的。毫无疑问，使用颜色或编码的分组标签，页面看起来会更有趣，而且这个功能可能也会更有用。但是我们决定把它标记为不必要的，并且从项目的中心中删除它。我们可以把它作为一个很好的东西提供给团队，但是每个人都应该从一个假设开始，即如果没有它，这个功能是有价值的。

## Present to technical experts 呈现给技术专家

Up to this point shaping has been a closed-door activity. Before you're ready to write up the idea to share more widely, you might need input on some parts of the concept you aren't completely sure about. There may be a technical assumption that you need to verify with someone who understands the code better. Or perhaps you want to make sure that usage data doesn't contradict an assumption you're making about current customer behavior.

到目前为止，塑形一直是一种闭门造车的活动。在你准备写下这个想法来更广泛地分享之前，你可能需要对这个概念的某些部分进行投入，这些部分你并不完全确定。可能有一个技术上的假设，你需要找一个更好理解代码的人来验证。或者你想确保使用数据不会与你对当前客户行为的假设相矛盾。

This is a good time to grab some technical experts and walk them through the idea. Communicate that this is just an idea. It's something you're shaping as a potential bet, not something that's coming down the pipe yet. The mood is friendly-conspiratorial:

现在是一个很好的时机去找一些技术专家，并且带领他们完成这个想法。告诉他们这只是一个想法。这是你正在形成的一个潜在的赌注，而不是一个即将流入管道的东西。气氛是友好的-阴谋论的：

“Here’s something I’m thinking about... but I’m not ready to show anybody yet... what do you think?”

“我正在考虑一件事... ... 但我还没准备好给任何人看.... ... 你觉得怎么样?”

Beware the simple question: “Is this possible?” In software, ev-erything is possible but nothing is free. We want to find out if it’s possible within the appetite we’re shaping for. Instead of asking “is it possible to do X?” ask “is X possible in 6-weeks?” That’s a very different question. 当心这个简单的问题: “这可能吗?”在软件中, 一切皆有可能, 但没有什么是免费的。我们想知道在我们所塑造的欲望中是否有可能。而不是问“做 x 可能吗?”问“六个星期内可以做到吗?”这是一个非常不同的问题。

Talk through the constraints of how this is a good solution given the appetite, so they’re partners in keeping the project at the size you intend. And emphasize that you’re looking for risks that could blow up the project. It’s not just a “what do you think” conversa-tion—we’re really hunting for time bombs that might blow up the project once it’s committed to a team.

考虑到人们的胃口, 这是一个很好的解决方案, 所以他们是保持项目规模的合作伙伴。并强调你正在寻找可能导致项目崩溃的风险。这不仅仅是一个“你觉得怎么样”的对话——我们真的在寻找一个定时炸弹, 一旦这个项目被交给一个团队, 它可能会爆炸。

Try to keep the clay wet. Rather than writing up a document or creating a slideshow, invite them to a whiteboard and redraw the elements as you worked them out earlier, building up the concept from the beginning. Stick completely to the concept you already worked out to get feedback on the work you’ve already done. Then once you’ve covered the work you already did, open it up and invite them to suggest revisions. Having seen this concept, do they have any insights about how to drastically simplify or approach the problem differently?

尽量使粘土保持湿润。与其写一个文档或者创建一个幻灯片，不如邀请他们到一个白板前，重新绘制你之前设计出来的元素，从一开始就构建概念。完全坚持你已经制定出来的概念，以获得对你已经完成的工作的反馈。然后，一旦你完成了你已经完成的工作，打开它，邀请他们提出修改建议。看过这个概念之后，他们对如何彻底简化或者用不同的方法解决问题有什么见解吗？

Depending on how the conversation goes, you may either have validated your approach or discovered some problems that send you back for another round of shaping.

取决于谈话的进展，你可能已经验证了你的方法，或者发现了一些问题，让你回到了另一轮的形成。

## De-risked and ready to write up

### 风险降低，准备写报告

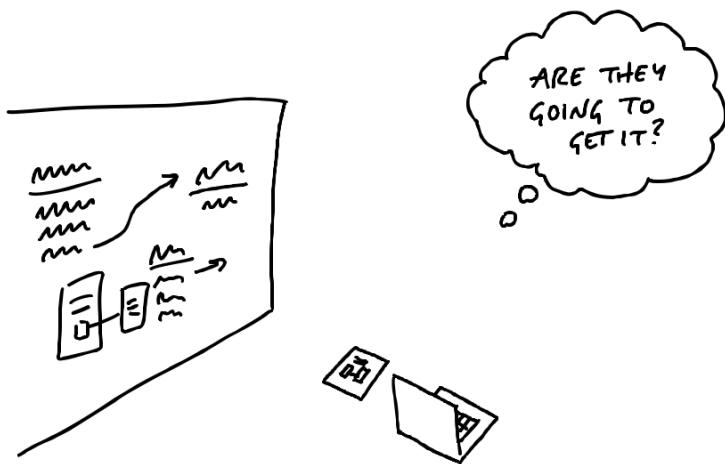
At the end of this stage, we have the elements of the solution, patches for potential rabbit holes, and fences around areas we've declared out of bounds. We've gone from a roughly formed solution with potential risk in it to a solid idea that we now hope to bet on in the future.

在这个阶段的最后，我们有了解决方案的元素，可能出现的兔子洞的补丁，以及我们已经宣布的禁区周围的围栏。我们已经从一个大致形成的具有潜在风险的解决方案，变成了一个我们现在希望在未来押注的可靠想法。

That means we're ready to make the transition from privately shaping and getting feedback from an inner-circle to presenting the idea at the betting table. To do that, we write it up in a form that communicates the boundaries and spells out the solution so that people with less context will be able to understand and evaluate it. This “pitch” will be the document that we use to lobby for resources, collect wider feedback if necessary, or simply capture the idea for when the time is more ripe in the future.

这意味着我们已经准备好从私下形成和从内部圈子获得反馈过渡到在赌桌上展示我们的想法。为了做到这一点，我们以一种传达边界和阐明解决方案的形式来编写它，这样一来，缺乏背景的人们就能够理解和评估它。这种“推销”将是我们用来游说资源的文件，必要时收集更广泛的反馈意见，或者只是捕捉未来时机成熟时的想法。

**56**  
**56**



## Write the Pitch 写推销词

We've got the elements of a solution now, and we've de-risked our concept to the point that we're confident it's a good option to give a team. But the concept is still in our heads or in some hard-to-decipher drawings on the whiteboard or our notebook. Now we need to put the concept into a form that other people will be able to understand, digest, and respond to.

我们现在已经有了解决方案的要素，我们已经降低了我们的概念的风险，以至于我们相信这是一个很好的选择，给一个团队。但是这个概念仍然存在于我们的头脑中，或者存在于白板或笔记本上一些难以辨认的图画中。现在我们需要把这个概念变成一种其他人能够理解、消化和回应的形式。

This is where we say “Okay, this is ready to write up as a pitch.” In this chapter, we'll walk through the ingredients of a pitch and show some fully worked out examples from real projects at Basecamp.

这时我们会说：“好的，这就是我们要写的东西。”在这一章中，我们将逐一介绍推销的要素，并展示一些来自 Basecamp 实际项目的完整实例。

The purpose of the pitch is to present a good potential bet. It's basically a presentation. The ingredients are all the things that we need to both capture the work done so far and present it in a form that will enable the people who schedule projects to make an informed bet.

推销的目的是提出一个很好的潜在赌注。这基本上就是一个展示。要素是我们需要的所有东西，既要捕捉到目前为止已完成的工作，又要以一种能够让安排项目的人做出知情赌注的形式来展示它。

There are five ingredients that we always want to include in a pitch:  
我们总是希望在推销中包括以下五个要素:

1. **Problem** — The raw idea, a use case, or something

we've seen that motivates us to work on this

问题——原始的想法，用例，或者我们已经看到的

能够激励我们去做的事情

2. **Appetite** — How much time we want to spend

and how that constrains the solution

胃口——我们想花多少时间，以及这些时间对

解决方案的制约作用

3. **Solution** — The core elements we came up with, presented in

a form that's easy for people to immediately understand

解决方案——我们提出的核心要素，以一种易于人们立即理解

的形式呈现出来

4. **Rabbit holes** — Details about the solution

worth calling out to avoid problems

兔子洞——为避免问题而提出的解决方案的

细节

5. **No-gos** — Anything specifically excluded from the concept:

functionality or use cases we intentionally aren't covering to  
fit the appetite or make the problem tractable

No-gos -- 任何特别排除在概念之外的东西: 我们故意没有覆

盖的功能或用例，以适应需求或使问题易于处理

## Ingredient 1. Problem

### 材料 1. 问题

It's critical to always present both a problem and a solution together. It sounds like an obvious point but it's surprising how often teams, our

own included, jump to a solution with the assumption that it's obvious why it's a good idea to build this thing.

总是提出一个问题和一个解决方案是至关重要的。这听起来似乎是一个显而易见的观点，但令人惊讶的是，包括我们自己的团队在内的多少团队都会跳到一个解决方案上，并假设构建这个东西的好处是显而易见的。

Diving straight into “what to build”—the solution—is dangerous. You don't establish any basis for discussing whether this solution is good or bad without a problem. “Add tabs to the iPad app” might be attractive to UI designers, but what's to prevent the discussion from devolving into a long debate about different UI approaches? Without a specific problem, there's no test of fitness to judge whether one solution is better than the other.

直接投入“建造什么”——解决方案——是危险的。你没有建立任何基础来讨论这个解决方案是好还是坏而没有任何问题。“在 iPad 应用程序中添加标签”对 UI 设计师来说可能很有吸引力，但是怎样才能防止讨论演变成一场关于不同 UI 方法的长期争论呢？没有一个特定的问题，就没有适合的测试来判断一个解决方案是否比另一个更好。

Establishing the problem also lets us have a clearer conversation later when it's time to pitch the idea or bet on it. The solution might 建立这个问题也可以让我们在以后提出这个想法或打赌的时候有一个更清晰的对话。解决方案可能会

be perfect, but what if the problem only happens to customers who are known to be a poor fit to the product? We could spend six weeks on an ingenious solution that only benefits a small percentage of customers known to have low retention. We want to be able to separate out that discussion about the demand so we don't spend time on a good solution that doesn't benefit the right people.

要做到完美，但是如果问题只发生在那些被认为不适合该产品的顾客身上呢？我们可以花六个星期的时间来寻找一个巧妙的解决方案，这个方案只能使一小部分已知的低保留率的客户受益。我们希望能够分离出关于需求的讨论，这样我们就不会花时间在一个不利于合适人群的好解决方案上。

How far you have to go to spell out the problem will depend on how much context you share with the people reading the write-up. The best problem definition consists of a single specific story that shows why the status quo doesn't work. This gives you a base-line to test fitness against. People will be able to weigh the solution against this specific problem—or other solutions if a debate ensues—and judge whether or not that story has a better outcome with the new solution swapped in. 你需要走多远才能说清楚这个问题，这取决于你与读者分享了多少内容。最好的问题定义包括一个单一的具体故事，说明为什么现状不起作用。这给了你一个测试健康状况的基线。人们将能够权衡解决方案与这个具体问题或其他解决方案，如果一场辩论接踵而至，并判断这个故事是否有一个更好的结果与新的解决方案交换。

## Ingredient 2. Appetite

### 配料 2。食欲

You can think of the appetite as another part of the problem definition. Not only do we want to solve this use case, we want to come up with a

way to do it in six weeks, not three months, or—in the case of a small batch project—two weeks, not the whole six weeks.

您可以将食欲视为问题定义的另一部分。我们不仅想要解决这个用例，我们还想要找到一种在六周内完成它的方法，而不是三个月，或者——在小批量项目的情况下——两周，而不是整个六周。

Stating the appetite in the pitch prevents unproductive conversations. There's always a better solution. The question is, if we only care enough to spend two weeks on this now, how does *this specific solution* look?

在演讲中陈述欲望可以防止没有成效的对话。总有更好的解决办法。问题是，如果我们现在只关心这个问题两周，这个具体的解决方案看起来怎么样？

Anybody can suggest expensive and complicated solutions. It takes work and design insight to get to a simple idea that fits in a small time box. Stating the appetite and embracing it as a constraint turns everyone into a partner in that process.

任何人都可以提出昂贵而复杂的解决方案。这需要工作和设计的洞察力，以获得一个简单的想法，适合在一个小的时间盒。在这个过程中，陈述自己的欲望并将其视为一种约束，每个人都会成为自己的合作伙伴。

## Ingredient 3. Solution

### 材料 3. 溶液

Like solutions with no problems, sometimes companies bet on problems with no solution. “We really need to make it easier to find things on the messages section. Customers are complaining about it.” 就像没有问题的解决方案一样，有时候公司把赌注押在没有解决方案的问题上。他说：“我们真的需要让在信息部分找东西变得更容易。顾客们都在抱怨。”

That's not ready to pitch or bet on. A problem without a solution is unshaped work. Giving it to a team means pushing research and exploration down to the wrong level, where the skillsets, time limit, and risk profile (thin vs. heavy tailed) are all misaligned.

还没准备好投球或下注。没有解决方案的问题是不成形的工作。把它交给一个团队意味着把研究和探索推到错误的水平，在这个水平上，技能集、时间限制和风险概况(瘦尾和重尾)都是不一致的。

If the solution isn't there, someone should go back and do the shaping work on the shaping track. It's only ready to bet on when problem, appetite, and solution come together. Then you can scrutinize the fit between problem and solution and judge whether it's a good bet or not. 如果解决方案不存在，那么应该有人回过头来，在塑造轨道上做塑造工作。只有当问题、欲望和解决方案结合在一起的时候，它才会准备好。然后你可以仔细审视问题和解决方案之间的契合度，判断它是否是一个好的赌注。

Help them see it

帮助他们看到它

During the elements phase, it was critical to sketch ideas at the right level of abstraction so we didn't slow down or lose any of the ideas appearing at the corners of our brains and tips of our tongues.

在元素阶段，在正确的抽象层次上勾勒想法是至关重要的，这样我们就不会放慢或丢失出现在我们大脑角落和舌尖上的任何想法。

We also need to draw at the right level of detail when we write the pitch. Here the challenge is a little different. We have time to slow down and prepare a proper presentation. We need to stay high level, but add a little more concreteness than when we worked alone or with a partner. People who read the pitch and look at the drawings without much context need to "get" the idea.

我们还需要绘制在正确的细节水平，当我们写的推介。这里的挑战有点不同。我们有时间慢下来，准备一个恰当的演讲。我们需要保持高水平，但要比单独工作或与同事一起工作时增加一些具体性。那些没有太多背景知识去阅读推介和看图的人需要“理解”这个想法。

We need more concreteness, but we don't want to over-specify the design with wireframes or high-fidelity mocks. They'll box in the designers who do the work later. We also risk side-tracking the

我们需要更多的具体性，但是我们不想用线框图或者高保真的模仿来过度指定设计。他们会限制那些稍后做这项工作的设计师。

我们也冒着边缘化的风险

discussion into topics like color, proportions, or layout that have nothing to do with the actual shaping work we did.

讨论像颜色、比例、布局这些与我们实际的塑造工作无关的话题。

At the same time, hand-written breadboards have a “you had to be there” quality to them. To people who didn’t watch the breadboard unfold step by step, it can look like a soup of words and arrows.

同时，手写的面包板有一种“你必须在那里”的特质。对于那些没有看着面板一步一步展开的人来说，它看起来就像是一锅由文字和箭头组成的汤。

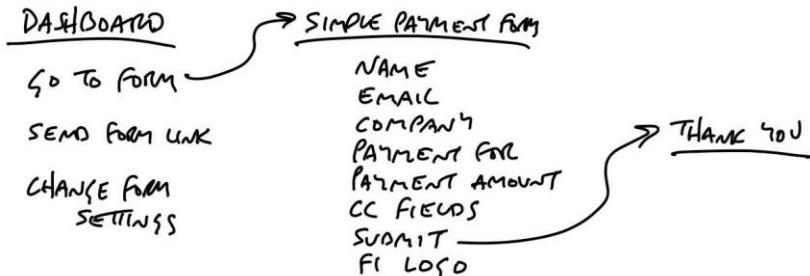
Therefore we need some techniques to help people see the idea while still not going too far into irrelevant details.

因此，我们需要一些技巧来帮助人们理解这个想法，同时不要过于深入无关的细节。

### ***Embedded sketches***

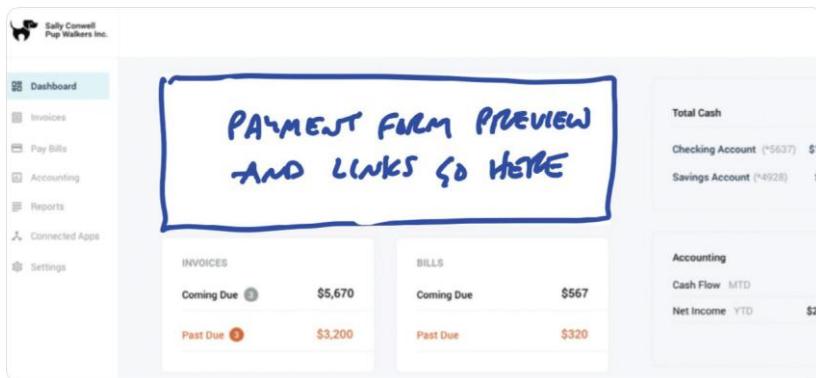
#### **嵌入式草图**

Suppose your breadboard from the shaping session looked like this:  
假设你的面板从造型阶段看起来像这样：



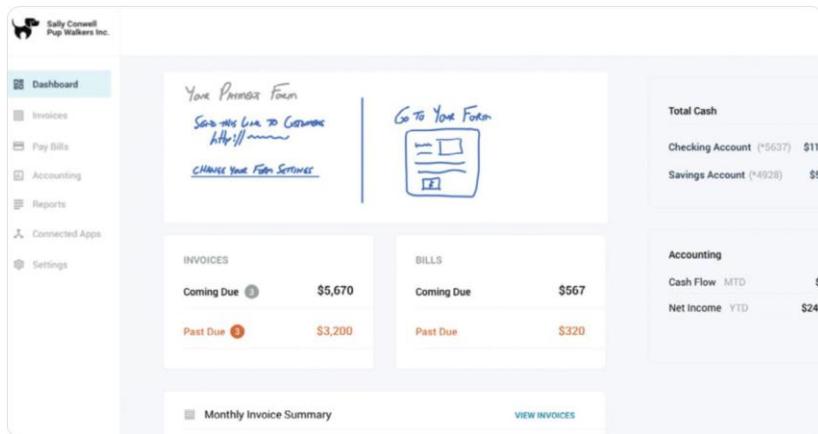
People might have trouble visualizing where these new affordances go on the Dashboard. We could sketch a new box on the Dashboard to make it clearer:

人们可能难以想象这些新功能在仪表盘上的位置。我们可以在仪表板上画一个新的框，让它更清晰：



But we're still asking people to imagine too much. It's worth the trade-off to go one step down into fat-marker detail here.

但我们仍然要求人们想象得太多。在这里，进一步深入到脂肪标记的细节是值得的。



This makes it easier to see what the elements are and evaluate how clearly the feature presents itself on the dashboard. The downside is we've gotten into some layout decisions that would have been nice to avoid. Designers should feel free to find a different design than the box divided with a vertical line. We'd add a disclaimer here in the pitch that reminds designers of the latitude they should take.

这样可以更容易地看到元素是什么，并评估特性在仪表盘上的清晰度。缺点是我们已经进入了一些布局决策，这些决策应该是最好避免的。设计师应该自由地找到一个不同的设计，而不是用垂直直线分隔的盒子。我们会在这里加一个免责声明，提醒设计师他们应该采取的纬度。

This is an example of selectively getting into more visual detail because we need it to sell the concept. Fortunately, we won't need to make as many visual decisions in other parts of the concept. This was a "linchpin" part of the design that everybody had to see concretely in order to "get" it.

这是一个有选择地进入更多视觉细节的例子，因为我们需要它来推销概念。幸运的是，我们不需要在概念的其他部分做太多的视觉决策。这是设计的“关键”部分，每个人都必须看到具体的，以便“得到”它。

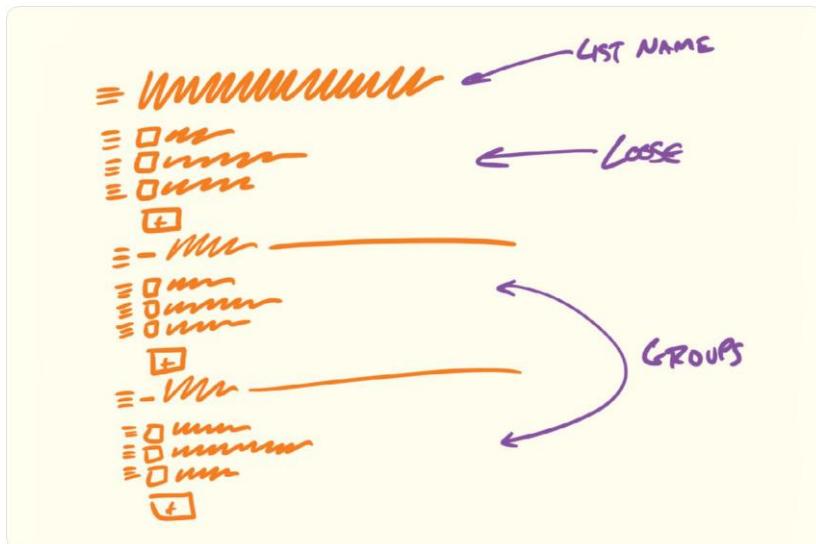
### *Annotated fat marker sketches*

#### **带注释的脂肪标记草图**

Sometimes ideas are inherently visual or a little too complicated to express in a schematic breadboard. Fat marker sketches can be very effective in a pitch; you just need to take more care to label them cleanly. 有时候，想法本身就是视觉的，或者有点太复杂，以至于不能用图示板来表达。Fat 标记草图可以非常有效地表达你的观点；你只需要更加小心地将它们清晰地标注出来。

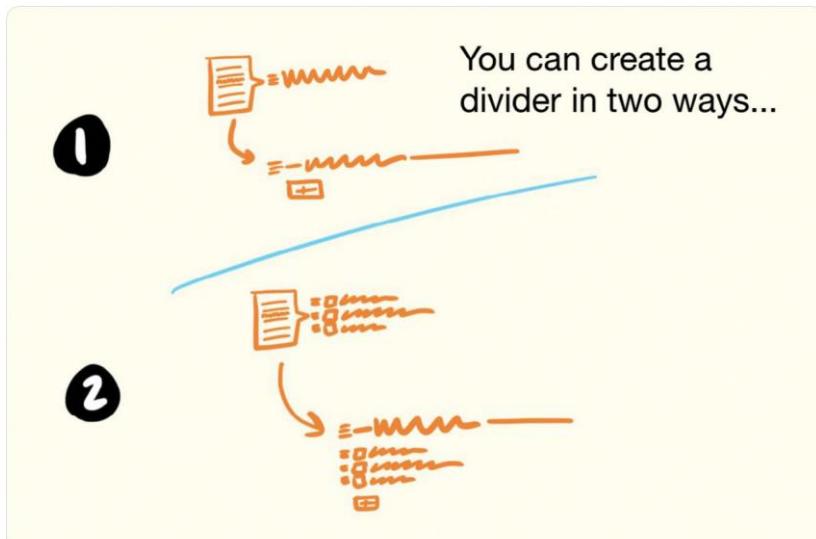
Redrawing the sketch on an iPad—still with a fat brush size—works well. You can use different colors to separate the labels from the material parts of the sketch.

在 ipad 上重新绘制草图——仍然使用粗笔刷大小——效果很好。  
你可以用不同的颜色把标签和素描的材质部分分开。



Or you might add some call-outs to enable discussion of specific elements.

或者你可以添加一些调用，以便能够讨论特定的元素。





## Ingredient 4. Rabbit holes

### 材料 4: 兔子洞

Sometimes addressing a rabbit hole just requires a few lines of text. For example, in the Payment Form project above, the shapers wanted to call out a specific solution for how to create URLs. The URLs would never live on custom domains for v1 of the project. This is the kind of thing that's not central to the concept, but spelling it out patches a potential rabbit hole.

有时候寻找一个兔子洞只需要几行文字。例如，在上面的支付表单项目中，塑造者想要为如何创建 url 调用一个特定的解决方案。Url 永远不会存在于 v1 的自定义域名中。这种事情并不是这个概念的核心，但是把它拼出来可能会补上一个兔子洞。

## Ingredient 5. No Gos

### 原料 5: 没有 Gos

Lastly if there's anything we're *not* doing in this concept, it's good to mention it here. In the case of the Payment Form project, the team decided up front that they wouldn't allow any kind of WYSIWYG editing of the form. Users would only be able to provide a logo and customize the header text on a separate "customize" page. WYSIWYG might be better in some peoples' eyes, but given the appetite it was important to mark this as a no-go.

最后，如果我们在这个概念中还有什么没有做的，最好在这里提一下。在 Payment Form 项目中，团队事先决定不允许对表单进行任何所见即所得的编辑。用户只能在一个单独的“自定义”页面上提供 logo 和自定义标题文本。在一些人看来，所见即所得可能更好，但考虑到用户的胃口，将其标记为禁用是很重要的。

## Examples

### 例子

Here are two examples of real pitches.

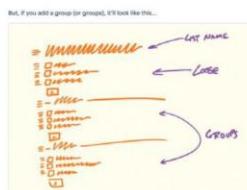
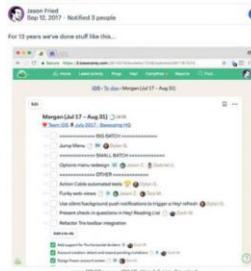
这里有两个真正的投球例子。

6 4  
64

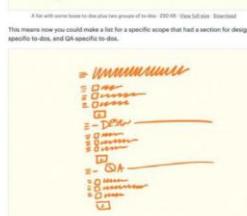
This pitch for grouping to-dos together starts by showing a work-around people are using in the current design. Then it sketches out all the main ideas for how to enable optional to-do groupings.

这种将待办事项分组在一起的方法首先展示了人们在当前设计中使用的工作流程。然后勾勒出所有关于如何实现可选的待办事项分组的主要想法。

### Grouping to-dos



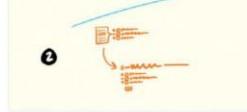
Completed to-dos will still be grouped into a single collection at the bottom of the entire list. If they were part of a group when you checked them off, we'll prepend the group name before the to-do like so...



If you check a completed to-do from a group, it could be stuck up to the group becoming the group divider's last item. 100 KB View full size Download  
Groups will also have their own permssions. So you can click the title of a group/divider, and you'll see a new page with just the to-dos in that group...



What we like about this concept overall is that it's very straightforward. It doesn't change anything you do at all. No new permanent UI - there was no behavior change if you don't use groups.



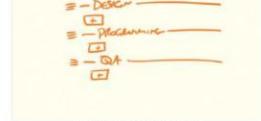
But if you want to level up your organization - and we put the shit out of it - you can add groups to a to-do. No groups within groups, no nesting - one-level groups as you like.



There are open questions. Thing like if you move the divider, do all the items move with it? If you add to a group, but you have to move that out, do you have to start putting groups above lower to-dos, will have to prevent dragging above a certain point, but that's doable as well. I'm sure there are a few other questions as well, but that's out of scope for now.



BONUS! Draw the next wild idea to explore adding to-do templates to B2C. We have project templates, but now we just talking to-do templates. You can imagine creating a template with no to-dos, but with groups in place. This is a process boost - people can organize projects in similar ways with just a little bit of structure like this...



Save and edit in place 100 KB View full size Download

*Two screenshots demonstrate the problem. Fat marker sketches describe the solution. Rabbit holes motivated some of the sketches. See full size at [basecamp.com/shapeup-todo-pitch](http://basecamp.com/shapeup-todo-pitch).*

两张截图展示了这个问题。Fat 标记草图描述了解决方案。

兔子洞激发了一些草图。详见 [basecamp.com/shapeup-todo-pitch](http://basecamp.com/shapeup-todo-pitch)。

CHAPTER 6 - Write the Pitch                    **65**  
Chapter 6-wrrteeeeeeeeeeeeeeeeeeeeeeee  
eeeeeeeeeeeeeeeeeeeeeeeeeeeeeeee  
eeeeeee



两个视频显示了这个问题。一个肥胖的记号笔草图和一块面板描述了解决方案。黑匣子包含支持解决方案中权衡的数据可视化。详见  
*[basecamp.com/shapeup-groups-pitch](http://basecamp.com/shapeup-groups-pitch)*。

**6 6**  
**6.6**

## Ready to present 准备展示

The next step will be to make the case that this pitch describes a bet worth making. This can happen in a couple ways.

下一步要做的就是证明这个说法描述了一个值得下注的赌注。这可以通过几种方式来实现。

We prefer asynchronous communication by default and escalate to real-time only when necessary. This gives everyone the maximum amount of time under their own control for doing real work. That means the first step for presenting a pitch is posting the write-up with all the ingredients above somewhere that stakeholders can read it on their own time. This keeps the betting table short and productive. In ideal conditions everyone has time to read the pitches in advance. And if that isn't possible in some cases, the pitch is ready to pull up for a quick live sell.

默认情况下，我们更喜欢异步通信，只有在必要时才升级为实时通信。这给每个人最大限度的时间在他们自己的控制下做真正的工作。这意味着提出一个推介的第一步是把上面的所有要素都张贴在某个地方，利益相关者可以在他们自己的时间阅读它。这样可以让博彩表简短而富有成效。在理想的情况下，每个人都有时间提前看球。如果在某些情况下无法做到这一点，那么推销活动已经准备就绪，可以进行快速现场销售。

## How we do it in Basecamp 我们在 Basecamp 是怎么做的呢

We post pitches as Messages in Basecamp. We created a Message Category called *Pitch* so we can easily find them. Pitches are posted to a Team called *Product Strategy* that can be accessed by people on the betting table.

我们在 Basecamp 中以消息的形式发布广告。我们创建了一个名为 Pitch 的消息分类，这样我们就可以很容易地找到它们。Pitch 被发布到一个叫做 Product Strategy 的团队中，人们可以通过这个团队进行投注。

The screenshot shows the Basecamp interface with the 'Product Strategy' team selected. The 'Message Board' section displays four messages from a user named Ryan Singer:

- Group notifications**  
Pitch by Ryan Singer • Feb 18 — Last November I wrote a pitch for adding Groups ala BCX to BC3: Groups! - Product Strategy. After looking at the pitch, 1 reply.
- Untangle the people spreadsheet**  
Pitch by Ryan Singer • Oct 11, 2018 — Software gets complex when one thing does two things. Today the people spreadsheet is one interface that does at least 5 replies.
- Sort Messages Setting (revised)**  
Pitch by Ryan Singer • Sep 21, 2018 — Here's a revised second take on the Sort Messages A-Z pitch. After talking to support and a couple customers via phone, 6 replies.
- Follow-up: Client-visible recordings**  
Report by Ryan Singer • Sep 13, 2018 — In April we launched the new client-visible toggle to replace the Clientside. Do people use the new client toggle? How 2 replies.

*Pitches on the Message Board of the Product Strategy team in Basecamp  
在 Basecamp 产品战略团队的留言板上进行推销*

A screenshot of a Basecamp interface showing a message board titled "Sort Messages Setting (revised)". The message was posted by Ryan Singer on Sep 21, 2018, and notified 4 people. It has 6 replies. The message content discusses a revised pitch for sorting messages by latest comment, mentioning a customer's issue with finding work items and the need for a one-week appetite.

**Sort Messages Setting (revised)**

Pitch by Ryan Singer  
Sep 21, 2018 · Notified 4 people

Here's a revised second take on the [Sort Messages A-Z](#) pitch.

After talking to support and a couple customers via phone, the biggest immediate win would be offering to [sort by latest comment](#).

**Example case**

One customer described the problem very well. Her team posts work for clients to review on the Message Board. Sometimes a client is slow to respond. By the time the client responds to a proof, the thread might be 10 items down on the Messages list. This makes it nearly impossible find later when she's trying to determine where the project left off.

Did she try looking at Latest Activity? She said yes, but the problem is Latest Activity has too much stuff on it. All the events about To-Dos and other discussion drown out the feed.

Sorting Messages by latest comment would make it easier to see where the latest response was.

*A pitch as a Message. Note the one-week appetite.*  
作为信息的推销。注意一个星期的胃口。

*This was a Small Batch project.*  
这是一个小批量的项目。

When we need to include a fat marker sketch in a pitch, we'll draw it on an iPad (with Notability) and take a screenshot. Basecamp's text editor makes it easy to insert images and caption them so they make sense in the flow of the pitch.

当我们需要在一个推介中加入一个大的标记草图时，我们会在 iPad 上画出来(使用 Notability)并截图。Basecamp 的文本编辑器使得插入图片和标题变得很容易，这样它们就可以在文章的流程中有意义。

who gets notified about some commentable. That "Who should be notified?" modal could offer Groups at the top. Clicking them checks members of the Groups below.



*A sketch drawn on an iPad in the middle of a pitch*  
一个在 iPad 上绘制的草图

People comment on the pitch asynchronously. Not to say yes or no  
人们在球场上的评论是异步的，不是说是或不是  
— that happens at the betting table — but to poke holes or contribute  
missing information.  
—发生在赌桌上—而是戳洞或提供缺失的信息。

MESSAGES INDEX

ACTIONS

ACTIONS MENU

BOOKMARK

SETTINGS

MESSAGES SETTINGS

"How do you want messages to be organized for your team?"

SORT BY:

- MOST RECENT COMMENT
- A→Z
- DATE MESSAGE WAS POSTED

SAVE

Sort options on a settings screen · 95.8 KB · [View full-size](#) · [Download](#)

### 3 Comments

 David Heinemeier Hansson Sep 21, 2018 · ...  
On the technical side, what this requires is a new column like Message#last\_comment\_at, and then an expiration into how we can use a recordable column to do the filtering permanently on a recording search. All very doable! (Just a caution not to use Recording#updated\_at as this timestamp is touched by all sorts of things, like boosts).

+ explicit too

*Our CTO responds with technical thoughts on the pitch.*  
我们的首席技术官在球场上以技术思维作出回应。

In the next chapter we'll look at the betting process in more detail to see where pitches go and how we turn them into scheduled projects.  
在下一章中，我们将更详细地了解投注过程，看看球场在哪里，以及我们如何将它们转化为计划项目。

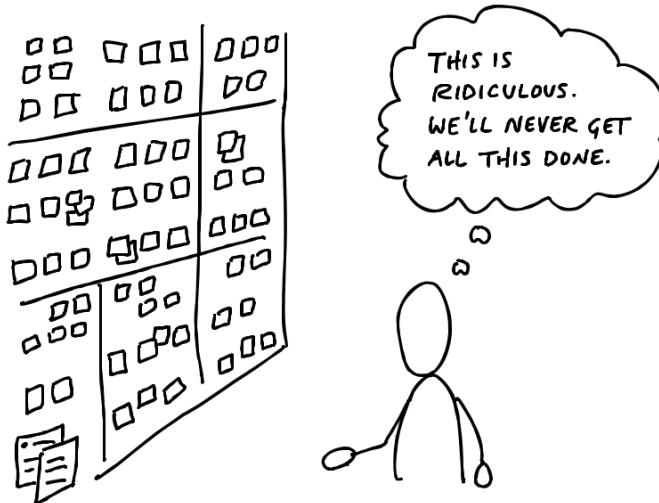
CHAPTER 6-WritethePitch        **69**  
Chapter 6-wrrtttttttttttttttttttt  
eeeeeeeeeeeeeeeeeeeeeeeeeeee  
eeeeeee



PARTTWO  
Parttwo

# Betting

# 赌博



## Bets, Not Backlogs

## 下注，而不是积压

Now that we've written a pitch, where does it go? It doesn't go onto a backlog.

现在我们已经写了一个推介，它去了哪里？它没有进入积压。

### No backlogs

### 没有积压文件

Backlogs are a big weight we don't need to carry. Dozens and eventually hundreds of tasks pile up that we all know we'll never have time for. The growing pile gives us a feeling like we're always behind even though we're not. Just because somebody thought some idea was important a quarter ago doesn't mean we need to keep looking at it again and again.

积压的文件是我们不需要背负的沉重负担。数十个甚至数百个任务堆积起来，我们都知道我们永远没有时间去做。越来越多的事情让我们觉得我们总是落后，即使我们没有落后。仅仅因为一个

人在一个季度前认为某个想法很重要，并不意味着我们需要一遍又一遍地看它。

Backlogs are big time wasters too. The time spent constantly reviewing, grooming and organizing old ideas prevents everyone from moving forward on the timely projects that really matter right now.

积压文件也是很大的时间浪费。花在不断回顾、梳理和组织旧想法上的时间阻碍了每个人及时完成现在真正重要的项目。

## A few potential bets 一些潜在的赌注

So what do we do instead? Before each six-week cycle, we hold a betting table where stakeholders decide what to do in the next cycle. At the betting table, they look at pitches from the last six weeks — or any pitches that somebody purposefully revived and lobbied for again.

那我们该怎么办？在每个六周周期之前，我们举行一个赌桌，让利益相关者决定在下一个周期做什么。在赌桌上，他们看着过去六周的投球——或者任何有人故意再次复活和游说的投球。

Nothing else is on the table. There's no giant list of ideas to review. There's no time spent grooming a backlog of old ideas. There are just a few well-shaped, risk-reduced options to review. The pitches are potential bets.

桌子上没有其他的东西。没有大量的想法需要回顾。我们没有时间去整理积压的旧想法。只有一些形状良好、风险降低的选项需要回顾。这些推销都是潜在的赌注。

With just a few options and a six-week long cycle, these meetings are infrequent, short, and intensely productive.

由于只有几个选择和六周的长周期，这些会议是罕见的，短暂的，强烈的生产力。

If we decide to bet on a pitch, it goes into the next cycle to build. If we don't, we let it go. There's nothing we need to track or hold on to.

如果我们决定在一个推介上下赌注，它就会进入下一个周期去构建。如果我们不这么做，我们就放手。没有什么需要我们去追踪或坚持的。

What if the pitch was great, but the time just wasn't right? Anyone who wants to advocate for it again simply tracks it independently— their own way—and then lobbies for it six weeks later.

如果演讲很棒，但时机不对呢？任何想要再次提倡它的人只需要独立地跟踪它——以他们自己的方式——然后在六周后为它游说。

## Decentralized lists 去中心化的列表

We don't have to choose between a burdensome backlog and not remembering anything from the past. Everyone can still track pitches, bugs, requests, or things they want to do independently without a central backlog.

我们不必在繁重的积压和不记得过去的任何事情之间做出选择。每个人仍然可以在没有中心积压的情况下追踪推销、错误、请求或者他们想独立完成的事情。

Support can keep a list of requests or issues that come up more often than others. Product tracks ideas they hope to be able to shape in a future cycle. Programmers maintain a list of bugs they'd like to fix when they have some time. There's no one backlog or

Support 可以保存一个请求或问题的列表，这些请求或问题出现的频率比其他的更高。产品跟踪他们希望能够在未来周期中形成的想法。程序员会保留一个他们有时间想要修复的 bug 列表。没有一个积压或者

central list and none of these lists are direct inputs to the betting process.

中心名单，这些名单没有一个是直接输入到投注过程。

Regular but infrequent one-on-ones between departments help to cross-pollinate ideas for what to do next. For example, Support can tell Product about top issues they are seeing, which Product can then track independently as potential projects to shape. Maybe Product picks off just one of those top issues to work on now. Then, in a future one-on-one, Support can lobby again for something that hasn't yet gotten attention.

部门之间定期但不频繁的一对一交流有助于交流下一步该做什么的想法。例如，技术支持可以告诉产品他们看到的最重要的问题，哪些产品可以作为潜在的项目独立跟踪。也许 Product 只挑出了其中一个需要解决的问题。然后，在未来的一对一，支持可以再次游说的东西，尚未得到关注。

This approach spreads out the responsibility for prioritizing and tracking what to do and makes it manageable. People from different departments can advocate for whatever they think is important and use whatever method works for them to track those things—or not.

这种方法分散了优先考虑和跟踪要做的事情的责任，使之易于管理。来自不同部门的人可以提倡任何他们认为重要的事情，并使用任何对他们有效的方法来追踪这些事情——或者不追踪。

This way the conversation is always fresh. Anything brought back is brought back with a context, by a person, with a purpose. Everything is relevant, timely, and of the moment.

这样谈话总是很新鲜。任何带回来的东西都会带回一个背景，一个人带回来的，带回来的目的。每一件事都是相关的，及时的，当下的。

Important ideas come back

## 重要的想法又回来了

It's easy to overvalue ideas. The truth is, ideas are cheap. They come up all the time and accumulate into big piles.

想法很容易被高估。事实是，创意是廉价的。它们不断涌现，积累成一大堆。

Really important ideas will come back to you. When's the last time you forgot a really great, inspiring idea? And if it's not that interesting—maybe a bug that customers are running into from time to time—it'll come back to your attention when a customer complains again or a new customer hits it. If you hear it once and never again, maybe it wasn't really a problem. And if you keep hearing about it, you'll be motivated to shape a solution and pitch betting time on it in the next cycle.

真正重要的想法会回到你身上。你上一次忘记一个伟大的、鼓舞人心的想法是什么时候？如果它不是那么有趣——也许是客户时不时遇到的一个 bug ——那么当客户再次抱怨或者新客户访问它时，它就会再次引起你的注意。如果你一次又一次地听到它，也许这并不是一个真正的问题。如果你一直听到这个消息，你就会有动力去形成一个解决方案，并在下一个周期为它投下赌注。



## The Betting Table 投注表

Now that we have some good potential bets in the form of pitches, it's time to make decisions about which projects to schedule.

现在，我们已经有了一些很好的潜在赌注，以推介的形式，现在是时候做出决定，哪些项目的时间表。

### Six-week cycles 六周周期

Committing time and people is difficult if we can't easily determine who's available and for how long. When people are available at different times due to overlapping projects, project planning turns into a frustrating game of Calendar Tetris. Working in cycles drastically simplifies this problem. A cycle gives us a standard project size both for shaping and scheduling.

如果我们不能很容易地确定谁有空，有多长时间，那么投入时间和人力是很困难的。当人们因为项目重叠而在不同时间有空的时候，项目计划就变成了一个令人沮丧的日历俄罗斯方块游戏。周

期性工作极大地简化了这个问题。一个周期为我们提供了一个标准的项目规模，无论是形成还是进度安排。

Some companies use two-week cycles (aka “sprints”). We learned that two weeks is too short to get anything meaningful done. Worse than that, two-week cycles are extremely costly due to the planning overhead. The amount of work you get out of two weeks isn’t worth the collective hours around the table to “sprint plan” or the  
一些公司使用两周的周期(也就是“冲刺”)。我们知道两周太短了，  
没法完成任何有意义的事情。更糟糕的是，由于计划的开销，两  
周的周期是极其昂贵的。你在两周内完成的工作量不值得你花费  
在“冲刺计划”或者“冲刺计划”上的时间

opportunity cost of breaking everyone's momentum to re-group.  
打破每个人重新组合的势头的机会成本。

This led us to try longer cycles. We wanted a cycle that would be long enough to finish a whole project, start to end. At the same time, cycles need to be short enough to see the end from the beginning. People need to feel the deadline looming in order to make trade-offs. If the deadline is too distant and abstract at the start, teams will naturally wander and use time inefficiently until the deadline starts to get closer and feel real. 这导致我们尝试更长的周期。我们想要一个足够长的周期来完成一个从开始到结束的整个项目。同时，周期需要足够短，以便从头看到尾。为了做出权衡，人们需要感觉到最后期限正在逼近。如果最后期限在开始的时候过于遥远和抽象，团队将自然而然地漫游和低效率地使用时间，直到最后期限开始接近和感觉真实。

After years of experimentation we arrived at six weeks. Six weeks is long enough to finish something meaningful and still short enough to see the end from the beginning.

经过多年的实验，我们最终达到了 6 周。六个星期的时间足够完成一些有意义的事情，也足够从头看到尾。

## Cool-down 冷静一下

If we were to run six-week cycles back to back, there wouldn't be any time to breathe and think about what's next. The end of a cycle is the worst time to meet and plan because everybody is too busy finishing projects and making last-minute decisions in order to ship on time.

如果我们连续进行六个星期的周期，就没有时间呼吸和思考接下来会发生什么了。一个周期的结束是最糟糕的会议和计划时间，因为每个人都忙于完成项目和做最后一分钟的决定，以便按时发货。

Therefore, after each six-week cycle, we schedule two weeks for cool-down. This is a period with no scheduled work where we can breathe, meet as needed, and consider what to do next.

因此，在每个六周的周期之后，我们安排两周的时间来冷静下来。这段时间我们没有计划好的工作，我们可以在这段时间里呼吸，在需要的时候见面，并考虑下一步要做什么。

During cool-down, programmers and designers on project teams are free to work on whatever they want. After working hard to ship their six-week projects, they enjoy having time that's under their control. They use it to fix bugs, explore new ideas, or try out new technical possibilities.

在冷静期间，项目团队中的程序员和设计师可以自由地做任何他们想做的事情。在努力完成他们为期六周的项目之后，他们享受着自己掌控的时间。他们利用这些时间来修正错误，探索新的想法，或者尝试新的技术可能性。

## Team and project sizes 团队和项目规模

In addition to standardizing the length of our cycles, we also roughly standardize the types of projects and teams that we bet on.

除了标准化我们周期的长度之外，我们还粗略地标准化了我们押注的项目和团队的类型。

Our project teams consist of either one designer and two programmers or one designer and one programmer. They're joined by a QA person who does integration testing later in the cycle.

我们的项目团队要么由一名设计师和两名程序员组成，要么由一名设计师和一名程序员组成。他们由一名 QA 人员加入，该人员在周期的后期进行集成测试。

These teams will either spend the entire cycle working on one project, or they'll work on multiple smaller projects during the cycle. We call the team that spends the cycle doing one project the big batch team and the team working on a set of smaller projects the small batch team.

Small batch projects usually run one or two weeks each. Small batch projects aren't scheduled individually. It's up to the small batch team to figure out how to juggle the work so they all ship before the end of the cycle.

这些团队要么将整个周期花在一个项目上，要么将在周期中花在多个较小的项目上。我们把在一个周期内做一个项目的团队称为大批量团队，把做一组小项目的团队称为小批量团队。小批量项目通常每个项目运行一到两周。小批量项目不是单独安排的。这取决于小批量团队如何处理这些工作，这样他们就可以在周期结束之前发货。

Now that we have a standard way to think about capacity, we can talk about how we decide what to schedule.

现在我们有了一个标准的方法来考虑容量，我们可以讨论我们如何决定什么计划。

## The betting table 赌桌

The betting table is a meeting held during cool-down where stakeholders decide what to do in the next cycle. The potential bets to consider are either new pitches shaped during the last six weeks, or possibly one or two older pitches that someone specifically chose to revive. As we said last chapter, there's no “grooming” or backlog to organize. Just a few good options to consider.

投注表是在冷静期间举行的会议，利益相关者在会上决定下一个周期要做什么。可以考虑的潜在赌注要么是过去六周内形成的新投标，要么可能是有人特别选择重启的一个或两个旧投标。正如我们在上一章所说的，没有“修饰”或积压需要组织。只有一些好的选择需要考虑。

Our betting table at Basecamp consists of the CEO (who in our case is the last word on product), CTO, a senior programmer, and a product strategist (myself).

我们在 Basecamp 的下注人员包括 CEO (在我们的案例中，他是产品的最终决定者)、CTO、一位高级程序员和一位产品策略师(我自己)。

C-level time is only available in small slices, so there's an atmosphere of "waste no time" and the call rarely goes longer than an hour or two. Everyone has had a chance to study the pitches on their own time beforehand. Ad-hoc one-on-one conversations in the weeks before usually establish some context too. Once the call starts, it's all about looking at the options that made it to the table and making decisions.

C 级时间只有很小的片段，所以有一种“不浪费时间”的氛围，通话时间很少超过一两个小时。每个人都有机会提前在自己的时间里研究这些音高。在开始之前的几个星期进行一对一的特别对话，通常也会建立一些背景。一旦通话开始，所有的一切都是关于看看哪些选项使它成为谈判桌，并作出决定。

The output of the call is a cycle plan. Between everyone present, there's knowledge of who's available, what the business priorities are, and what kind of work we've been doing lately. All of this feeds into the decision-making process about what to do and who to schedule (more on this below).

调用的输出是一个循环计划。在场的每个人都知道谁有空，业务优先级是什么，以及我们最近在做什么样的工作。所有这些都会影响到决策过程，比如要做什么，要安排谁(更多内容在下面)。

The highest people in the company are there. There's no "step two" to validate the plan or get approval. And nobody else can jump in afterward to interfere or interrupt the scheduled work.

公司的最高层都在那里。没有“第二步”来验证计划或获得批准。而且没有其他人可以在事后插手干扰或打断计划的工作。

This buy-in from the very top is essential to making the cycles turn properly. The meeting is short, the options well-shaped, and the headcount low. When these criteria are met, the betting table becomes a place to exercise control over the direction of the product instead of a battle for resources or a plea for prioritization. With cycles long enough

to make meaningful progress and shaped work that will realistically ship, the betting table gives the C-suite a “hands on the wheel” feeling they haven’t had since the early days.

这种来自最高层的支持对于让周期正常运转至关重要。会议时间短，选择良好，员工人数少。当这些标准得到满足时，赌桌就变成了一个对产品方向进行控制的地方，而不是一场资源争夺战或者对优先次序的请求。随着周期足够长，以取得有意义的进展和形成的工作，将实际发布，赌桌给高管“手握方向盘”的感觉，他们从来没有早期的日子。

## The meaning of a bet 赌博的意义

We talk about “betting” instead of planning because it sets different expectations.

我们谈论“赌博”而不是计划，因为它设置了不同的期望。

First, bets have a payout. We’re not just filling a time box with tasks  
首先，赌注是有回报的，我们不仅仅是在一个时间盒里塞满了任务

until it's full. We're not throwing two weeks toward a feature and hoping for incremental progress. We intentionally shape work into a six-week box so there's something meaningful finished at the end. The pitch defines a specific payout that makes the bet worth making.

直到它满了。我们不会花两周的时间来完成一个特性，然后期待渐进式的进展。我们故意把工作放在一个六周的盒子里，这样在最后才能完成一些有意义的工作。推介定义了一个特定的支出，使得这个赌注值得下。

Second, bets are commitments. If we bet six weeks, then we commit to giving the team the entire six weeks to work exclusively on that thing with no interruptions. We're not trying to optimize every hour of a programmer's time. We're looking at the bigger movement of progress on the whole product after the six weeks.

其次，赌注就是承诺。如果我们下六周的赌注，那么我们就承诺给团队整整六周的时间专门做这件事，不会有任何干扰。我们不会试图优化程序员的每一个小时。我们看到的是六周之后整个产品更大的进步。

Third, a smart bet has a cap on the downside. If we bet six weeks on something, the most we can lose is six weeks. We don't allow ourselves to get into a situation where we're spending multiples of the original estimate for something that isn't worth that price.

第三，明智的押注有一个下行上限。如果我们在某样东西上押六周，我们最多只能输六周。我们不允许自己陷入这样一种境地：我们为了一些不值那个价格的东西花费了原来估计数倍的钱。

Let's look at these last two points more closely.  
让我们更仔细地看看最后两点。

Uninterrupted time  
不间断的时间

It's not really a bet if we say we're dedicating six weeks but then allow a team to get pulled away to work on something else.

如果我们说我们要花费六个星期的时间，然后让一个团队抽出时间去做其他的事情，这并不是一个真正的赌注。

When you make a bet, you honor it. We do not allow the team to be interrupted or pulled away to do other things. If people interrupt the team with requests, that breaks our commitment. We'd no longer be giving the team a whole six weeks to do work that was shaped for six weeks of time.

当你打赌的时候，你要信守承诺。我们不允许团队被打断或者被拉去做其他事情。如果有人用请求打断了团队，那就违背了我们的承诺。我们就不会再给团队整整六周的时间去做那些已经形成了六周时间的工作。

When people ask for “just a few hours” or “just one day,” don’t be fooled. Momentum and progress are second-order things, like growth or acceleration. You can’t describe them with one point. You need an uninterrupted curve of points. When you pull someone

当人们要求“只有几个小时”或“只有一天”时，不要被愚弄了。动力和进步是次要的，就像成长或加速。你不能用一个点来描述它们。

你需要一个不间断的点曲线。当你拉某人的时候

away for one day to fix a bug or help a different team, you don't just lose a day. You lose the momentum they built up and the time it will take to gain it back. Losing the wrong hour can kill a day. Losing a day can kill a week.

离开一天去修复一个 bug 或者帮助一个不同的团队，你不会只是浪费一天。你失去了他们建立起来的动力，也失去了重新获得动力所需要的时间。错误的时间会毁掉一天。浪费一天可以消磨一周。

What if something comes up during that six weeks? We still don't interrupt the team and break the commitment. The maximum time we'd have to wait is six weeks before being able to act on the new problem or idea. If the cycle passes and that thing is still the most important thing to do, we can bet on it for that cycle. This is why it's so important to only bet one cycle ahead. This keeps our options open to respond to these new issues. And of course, if it's a real crisis, we can always hit the brakes. But true crises are very rare.

如果那六个星期发生了什么事怎么办？我们仍然不会打断团队，打破承诺。我们需要等待的最长时间是六个星期，然后才能对新问题或新想法采取行动。如果这个循环过去了，那件事仍然是最重要的事情，我们可以在这个循环中赌一把。这就是为什么只在前一个周期下注是如此重要。这使得我们有更多的选择来应对这些新问题。当然，如果这是一个真正的危机，我们总是可以踩刹车。但真正的危机是非常罕见的。

## The circuit breaker 断路器

We combine this uninterrupted time with a tough but extremely powerful policy. Teams have to ship the work within the amount of time that we bet. If they don't finish, by default the project doesn't get an extension. We intentionally create a risk that the project—as

pitched—won't happen. This sounds severe but it's extremely helpful for everyone involved.

我们结合这个不间断的时间与一个强硬但极其强大的政策。团队必须在我们下注的时间内完成工作。如果他们没有完成，默认情况下，项目不会得到延期。我们故意制造一种风险，使得项目——正如我们所说的那样——不会发生。这听起来很严重，但对每个参与其中的人来说都非常有帮助。

First, it eliminates the risk of runaway projects. We defined our appetite at the start when the project was shaped and pitched. If the project was only worth six weeks, it would be foolish to spend two, three or ten times that. Very few projects are of the “at all costs” type and absolutely must happen now. We think of this like a circuit breaker that ensures one project doesn't overload the system. One project that's taking too long will never freeze us or get in the way of new projects that could be more important.

首先，它消除了失控项目的风险。我们在项目形成和定位的一开始就定义了我们的胃口。如果这个项目只值六个星期，那么花费两个，三个或者十个星期是愚蠢的。很少有项目是“不惜一切代价”的类型，而且绝对必须现在就开始。我们认为这就像一个断路器，可以确保一个项目不会让系统超载。一个花费太长时间的项目永远不会冻结我们，也不会妨碍那些可能更重要的新项目。

Second, if a project doesn't finish in the six weeks, it means we did something wrong in the shaping. Instead of investing more time in a bad approach, the circuit breaker pushes us to reframe the problem. We can use the shaping track on the next six weeks to come up with a new or better solution that avoids whatever rabbit hole we fell into on the first try. Then we'll review the new pitch at the betting table to see if it really changes our odds of success before dedicating another six weeks to it.

第二，如果一个项目没有在六个星期内完成，这意味着我们在形成过程中做错了什么。与其花更多的时间在一个糟糕的方法上，断路器会推动我们重新构建问题。我们可以在接下来的六个星期里利用塑形赛道来想出一个新的或更好的解决方案，避免我们在第一次尝试时陷入任何兔子洞。然后我们会在赌桌上回顾一下新的投球方式，看看它是否真的改变了我们成功的几率，然后再花六个星期的时间投入其中。

Finally, the circuit breaker motivates teams to take more ownership over their projects. As we'll see in the next chapter, teams are given full responsibility for executing projects. That includes making trade-offs about implementation details and choosing where to cut scope. You can't ship without making hard decisions about where to stop, what to compromise, and what to leave out. A hard dead-line and the chance of not shipping motivates the team to regularly question how their design and implementation decisions are affecting the scope.

最后，断路器激励团队对他们的项目拥有更多的所有权。正如我们将在下一章看到的，团队被赋予了执行项目的全部责任。这包括权衡实施细节和选择削减范围。如果你不能做出艰难的决定，比如在哪里停止，妥协什么，遗漏什么。一个硬性的截止期限和不发货的可能性促使团队经常质疑他们的设计和实现决策是如何影响范围的。

## What about bugs?

那么 bug 呢？

If the teams aren't interrupted in the six week cycle, how do we handle bugs that come up?

如果团队在六周周期内没有被打断，我们如何处理出现的错误？

First we should step back and question our assumptions about bugs.

首先，我们应该后退一步，质疑我们对错误的假设。

There is nothing special about bugs that makes them automatically more important than everything else. The mere fact that something is a bug does not give us an excuse to interrupt ourselves or other people. All software has bugs. The question is: how severe are they? If we're in a real crisis—data is being lost, the app is grinding to a halt, or a huge swath of customers are seeing the wrong thing—then we'll drop everything to fix it. But *crises are rare*. The

Bug 并没有什么特别之处，让它们自动地比其他任何东西都重要。

仅仅因为某些东西是一个 bug 这一事实并不能给我们一个打断自己或他人的借口。所有的软件都有缺陷。问题是：它们有多严重？如果我们真的遇到了危机——数据丢失，应用程序停止运行，或者大量的客户看到了错误的东西——那么我们就会放弃一切来修复它。但是危机是罕见的。译者：

vast majority of bugs can wait six weeks or longer, and many don't even need to be fixed. If we tried to eliminate every bug, we'd never be done. You can't ship anything new if you have to fix the whole world first.

绝大多数的错误可以等待六个星期或更长的时间，许多甚至不需要修复。如果我们试图消除每一个 bug，我们永远不会完成。如果你必须首先修复整个世界，你就不能运送新的东西。

That said, nobody likes bugs. We still want ways to deal with them.也就是说，没有人喜欢虫子，我们仍然需要对付它们的方法。

Three strategies have worked for us.

有三种策略对我们有效。

1. **Use cool-down.** Ask any programmer if there are things they wish they could go back and fix and they'll have a list to show you. The cool-down period between cycles gives them time to do exactly that. Six weeks is not long to wait for the majority of bugs, and two weeks every six weeks actually adds up to a lot of time for fixing them.

使用冷静。问任何一个程序员是否有什么东西他们希望可以回去修复，他们会给你一个列表。周期之间的冷却期给了他们时间去做这些事情。对于大多数 bug 来说，六周的等待时间并不长，而且每六周等待两周实际上意味着要花费大量的时间来修复它们。

2. **Bring it to the betting table.** If a bug is too big to fix during cool-down, it can compete for resources at the betting table. Suppose a back-end process is slowing the app down and a programmer wants to change it from a synchronous step to an asynchronous job. The programmer can make the case for fixing it and shape the solution in a pitch. Then instead of interrupting other work, the people at the betting table can make a deliberate decision. Time should always be used strategically. There's a huge difference between delaying other

work to fix a bug versus deciding up front that the bug is worth the time to fix.

把它拿到赌桌上来。如果一个 bug 太大而无法在冷却时修复，它可以在赌桌上竞争资源。假设一个后端进程正在减慢应用程序的运行速度，程序员希望将它从同步步骤改为异步作业。程序员可以提出修复它的理由，然后把解决方案形成一个框架。然后，与其打断其他工作，赌桌上的人们可以做一个深思熟虑的决定。时间应该总是战略性地使用。拖延其他工作来修复一个 bug 和事先确定这个 bug 值得花时间去修复是有很大区别的。

3. **Schedule a bug smash.** Once a year—usually around the holi-days—we'll dedicate a whole cycle to fixing bugs. We call it a “bug smash.” The holidays are a good time for this because it's hard to get a normal project done when people are traveling or taking time off. The team can self-organize to pick off the most important bugs and solve long-standing issues in the front-end or back-end.

安排一个 bug 粉碎。每年一次——通常在节假日前后——我们会用一整个周期来修复 bug。我们称之为“除虫”假期是一个很好的时机，因为当人们旅行或休假的时候，很难完成一个正常的项目。团队可以自我组织，挑出最重要的 bug，解决前端或后端长期存在的问题。

## Keep the slate clean 保持过去的清白

The key to managing capacity is giving ourselves a clean slate with every cycle. That means only betting one cycle at a time and never carrying scraps of old work over without first shaping and considering them as a new potential bet.

管理产能的关键是在每个周期都给自己一个清白的过去。这意味着一次只能下注一个周期，如果没有首先形成并考虑将其作为一个新的潜在赌注，就不能将旧作品的碎片带过来。

It is crucial to maximize our options in the future. We don't know what will happen in the next six weeks. We don't know what brilliant idea will emerge or what urgent request might appear.

最大化我们未来的选择是至关重要的。我们不知道接下来的六周会发生什么。我们不知道什么聪明的想法会出现，也不知道什么紧急请求会出现。

Even if we have some kind of road map in our heads at the time scale above cycles, we keep it in our heads and in our side-channel discussions. Each six weeks we learn what's working and what isn't, what's important and what's not. There's no downside to keeping the option open and massive upside from being available to act on the unexpected.

即使我们在头脑中有某种在周期之上的时间尺度上的路线图，我们也会把它保留在头脑中和我们的边道讨论中。每六个星期我们都会知道什么有用，什么没用，什么重要，什么不重要。保持选择的开放性没有任何负面影响，对于意想不到的事情采取行动也有巨大的好处。

What about projects that just can't be done in one cycle? In that case we still only bet six weeks at a time. Suppose we envision a feature that takes two cycles to ship. We reduce our risk dramatically by shaping a

specific six week target, with something fully built and working at the end of that six weeks. If that goes as expected, we'll feel good about betting the next six weeks the way we envisioned in our heads. But if it doesn't, we could define a very different project. Or we could put the multi-cycle thing on pause and do something urgent that came up. The important thing is that we always shape what the end looks like for that cycle and that we keep our options open to change course.

那些不能在一个周期内完成的项目呢？在这种情况下，我们仍然一次只赌六个星期。假设我们设想一个需要两个周期才能发布的功能。我们通过制定一个具体的六周目标来大大降低我们的风险，在六周结束的时候完全建立并运作一些东西。如果这一切如预期的那样进行，我们会很高兴按照我们头脑中设想的方式来押注接下来的六周。但如果不是这样，我们可以定义一个非常不同的项目。或者我们可以暂停多周期项目，做一些突然出现的紧急事情。重要的是，我们总是塑造这个周期的结局，我们保留改变方向的选择余地。

# Place Your Bets

## 下注吧

Look where you are

看看你在哪里

Depending on whether we're improving an existing product or building a new product, we're going to set different expectations about what happens during the six-week cycle.

根据我们是在改进现有产品还是在开发新产品，我们将对六周周期内发生的事情设定不同的预期。

This invites us to reflect on where we are in the arc of our product's development and bet accordingly.

这促使我们反思自己在产品开发过程中所处的位置，并据此下注。

Existing products

现有产品

When we add features to an existing product, we follow the standard Shape Up process: shape the work, bet on it, and give it to a team to build. We expect the team to finish and ship some version of the shaped work by the end of the cycle.

当我们向现有产品添加特性时，我们遵循标准的 Shape Up 流程：塑造工作，押注于它，并将其交给团队进行构建。我们期望团队在周期结束时完成并发布一些定型工作的版本。

On an existing product, all of the existing code and design that *isn't* going to change defines a kind of empty space that the new feature will fit into. Shaping and building is like crafting a piece of furniture for a house that is already built.

在一个现有的产品上，所有不会改变的现有代码和设计都定义了一种新特性可以适应的空间。塑造和建造就像为已经建好的房子制作一件家具。

## New products

### 新产品

New products are different. Whereas adding to an existing product is like buying a couch for a room with fixed dimensions, new product development is like figuring out where the walls and the foundation should go so the building will stand.

新产品是不同的。增加一个现有产品就像为一个有固定尺寸的房间购买一个沙发，而新产品的开发就像弄清楚墙壁和地基应该放在哪里，这样建筑才能屹立不倒。

We've noticed three phases of work when we build a new product from scratch. In each phase, the way that we shape and our expectations for how the team will work together during the cycle are

我们已经注意到从零开始构建新产品的工作分为三个阶段。在每个阶段，我们塑造的方式和我们对团队如何在周期中一起工作的期望是

different. These phases unfold over the course of multiple cycles, but we still only bet one cycle at a time.

这些阶段在多个周期的过程中展开，但我们仍然每次只下注一个周期。

## R&D mode

### R & d 模式

At the very earliest stages of a new product, our idea is just a theory or a glimmer. We don't know if the bundle of features we imagine will hold together in reality, and the technical decisions about how to model them in code are even less clear.

在一个新产品的最初阶段，我们的想法只是一个理论或一线希望。我们不知道我们想象中的一系列特性在现实中是否能够保持一致，关于如何在代码中对它们进行建模的技术决策就更不清楚了。

This means there is a lot of scrapwork. We might decide half-way to standing up a feature that it's not what we want and try another approach instead.

这意味着还有很多的拼凑工作。我们可能会在中途决定放弃一个不是我们想要的功能，然后尝试另一种方法。

In other words, we can't reliably shape what we want in advance and say: "This is what we want. We expect to ship it after six weeks." We have to learn what we want by building it.

换句话说，我们不能事先确定我们想要什么，然后说：“这就是我们想要的。我们希望在六周后发货。”我们必须通过建造它来了解我们想要什么。

We call this stage R&D mode and adjust for it in three ways.  
我们称这个阶段为 r & d 模式，并通过三种方式进行调整。

1. Instead of betting on a well-shaped pitch, we mainly bet the *time* on spiking some key pieces of the new product idea. The shaping is much fuzzier because we expect to learn by building.

我们主要把时间押在新产品创意的一些关键部分上，而不是押注于一个形状良好的推介。形状更加模糊，因为我们希望通过构建来学习。

2. Rather than delegating to a separate build team, our senior people make up the team. David (CTO) takes the program-ming role and works with Jason (CEO and designer) or a senior designer with Jason’s guidance. This is necessary for two reasons. First, you can’t delegate to other people when you don’t know what you want yourself. Second, the architectural decisions will determine what’s possible in the product’s future — they define the “holes” that future features fit into. At this phase the team needs to hold the vision of the product and be able to judge the long-term effects of design decisions.

我们的高级人员组成了团队，而不是授权给一个单独的构建团队。David(首席技术官)担任程序员的角色，与Jason(首席执行官和设计师)合作，或者与Jason指导的高级设计师合作。这有两个必要的原因。首先，当你不知道自己想要什么的时候，你不能委派给别人。其次，架构决策将决定产品未来的可能性——它们定义了未来功能所适合的“漏洞”。在这个阶段，团队需要把握产品的远景，并能够判断设计决策的长期影响。

3. Lastly, we don't expect to ship anything at the end of an R&D cycle. The aim is to spike, not to ship. In the best case we'll have some UI and code committed to serve as the foundation for subsequent work. The goal is to learn what works so we can commit to some load-bearing structure: the main code and UI decisions that will define the form of the product going forward.

最后，我们不希望在研发周期结束时发货。我们的目标是达到峰值，而不是发货。在最好的情况下，我们会有一些用户界面和代码作为后续工作的基础。我们的目标是了解什么可以工作，这样我们就可以承担一些负载结构：主要的代码和 UI 决策，这些决策将定义产品的未来形式。

We can't ship anything to customers with just a single cycle of R&D work. But we still don't commit to more than one cycle at a time. We may learn from the first cycle that we aren't ready to tackle the product yet. Or we may discover that our intuition rang true and the product is coming together. Depending on how it goes, we'll decide cycle-by-cycle whether to continue spending informal time in R&D mode.

只有一个研发周期，我们不能把任何东西发给客户。但是我们仍然不承诺一次只进行一个周期。我们可能会从第一个循环中学到，我们还没有准备好处理这个产品。或者我们可能会发现我们的直觉是正确的，产品正在走到一起。根据它的发展情况，我们将逐步决定是否继续在研发模式中花费非正式的时间。

## Production mode 生产模式

If we continue to get warmer after some R&D cycles, we'll eventually reach a point where the most important architectural decisions are settled. The product does those few essential things that define it, and the foundation is laid for the dozens of other things we'll have to do before we can ship to customers.

如果我们在一些 r & d 周期之后继续变暖，我们最终将达到一个最重要的架构决策确定的点。这个产品做了定义它的一些基本的事情，并且为我们在发货给客户之前必须做的许多其他事情奠定了基础。

With this structure in place, the senior team can bring in other people to contribute. This is the flip to production mode, where we work in formal cycles with clear-cut shaping, betting, and building phases. Production mode is like working on an existing product: the precedent set by the R&D work enables new contributors to identify where new functionality belongs and how it fits into the whole.

有了这个架构，高级团队可以引进其他人员来做出贡献。这是生产模式的翻转，我们按照正式的周期工作，有明确的形成、下注和建设阶段。生产模式类似于在现有产品上工作：研发工作设置的先例使新的贡献者能够确定新功能属于哪里，以及它如何融入整体。

In production mode:  
在制作模式中：

1. Shaping is deliberate again. The shaped work describes  
再次刻意塑形。塑形作品描述

what we expect to see at the end of the cycle.

我们期望在周期结束时看到什么。

2. The team that builds the projects is no longer limited to the senior group. It becomes possible to bet multiple teams in parallel (if you have them) and cover more ground.

构建项目的团队不再局限于高级团队。如果你拥有多个团队，你可以同时押注于多个团队，并且覆盖更多的领域。

3. Shipping is the goal, not spiking. But because the product isn't publicly available to customers yet, we define 'ship-ping' differently. Shipping means merging into the main codebase and expecting not to touch it again.

运输是目标，而不是突破。但是因为产品还没有对消费者公开开放，所以我们对“发货平”的定义是不同的。

Shipping 意味着合并到主代码库中，并希望不要再碰它。

Since we aren't shipping to customers at the end of each cycle, we maintain the option to remove features from the final cut before launch. This means we can still be experimental. We can bet six weeks on a feature without knowing if we'll want it in the final product. That's not a problem as long as we set expectations to the build team: we can't predict what we'll want in the final cut, and we're willing to risk this cycle to take our best swing at the idea.

由于我们不会在每个周期结束时向客户发货，所以我们保留了在发布前从最终切割中删除特性的选项。这意味着我们仍然可以进行试验。我们可以在一个功能上下六个星期的赌注，却不知道最终产品是否需要这个功能。只要我们给构建团队设定了期望值，这就不是问题：我们无法预测我们在最终剪辑中想要什么，而且我们愿意冒着这个周期的风险，在这个想法上做出最好的尝试。

## Cleanup mode

## Cleanup 模式

In the final phase before launching the new product, we throw all structure out the window. We call this cleanup mode. It's a free-for-all. We've built enough new products to learn that there are always things we forget, things we miss, details that aren't right, and bugs that creep in over the course of the R&D and production mode cycles.

在发布新产品之前的最后阶段，我们抛弃了所有的结构。我们称之为清理模式。这是一个混战。我们已经开发了足够多的新产品来了解我们总是会忘记一些东西，我们错过的东西，不正确的细节，以及在研发和生产模式周期过程中不断出现的错误。

There's something about putting your finger near the launch button that makes your hair stand up. Everything suddenly gets "real." Things we dismissed before pop out at us with new importance.

把手指放在发射按钮附近会让你毛骨悚然。一切突然变得“真实”我们忽略的事情突然出现在我们面前，变得更加重要。

That's why we reserve some capacity at the end for the unexpected.  
这就是为什么我们在结束时保留一些容量以备不时之需。

In cleanup mode:

在清理模式下：

1. There's no shaping. The cycle is closer in spirit to the "bug smash" mentioned in the previous chapter. Leadership stands at the helm throughout the cycle, calling attention to what's important and cutting away distractions.

没有成形。这个循环在精神上更接近于前一章提到的“虫子粉碎”。在整个周期中，领导力始终处于领导地位，引导人们关注重要的事情，排除干扰。

2. There aren't clear team boundaries. Everyone jumps in to help however they can.

没有明确的团队界限，每个人都会尽自己所能帮忙。

3. Work is "shipped" (merged to the main codebase) continuously in as small bites as possible.

工作被“运送”(合并到主代码库中)，尽可能地小范围地持续进行。

Discipline is still important. We have to check ourselves to make sure these are must-haves we're working on, not just our cold feet begging us to delay launch. Cleanup shouldn't last longer than two cycles.

纪律仍然很重要。我们必须检查自己，以确保这些都是我们正在努力的必需品，而不仅仅是我们临阵退缩，乞求我们推迟发射。清理工作不应该超过两个周期。

Cleanup is also the phase where leadership makes those "final cut" decisions. A smaller surface area on a V1 means fewer questions to answer, less to support, and less we're committing to maintain indefinitely. Sometimes we need to see all the features working as a whole to judge what we can live without and what might require deeper consideration before shipping it to customers.

清理工作也是领导层做出“最终决定”的阶段。一个更小的 V1 表面积意味着更少的问题需要回答，更少的支持，更少的我们承诺无限期的维护。有时候，我们需要看到所有的特性作为一个整体工作，以判断我们可以生活中没有什么，以及什么可能需要更深入的考虑，然后才能将其发送给客户。

## Examples 例子

### *The Dot Grid Calendar*

#### *Dot 网格日历*

We built the Dot Grid Calendar (see Chapter 2) for Basecamp, an existing product. We shaped the project, bet six weeks on it, a team built it, and then we shipped it straight to customers.

我们为现有的产品 Basecamp 构建了 Dot Grid Calendar (参见第 2 章)。我们设计了这个项目，下了六个星期的赌注，一个团队建立了它，然后我们把它直接发送给客户。

### *A new product: HEY*

#### *一个新产品: 嘿*

In 2020, after two years of development, we launched a new email app and service called HEY. HEY was in R&D mode for the first year

在 2020 年，经过两年的发展，我们推出了一个新的电子邮件应用程序和服务，称为 HEY。HEY 第一年就进入了研发模式

of its development. A team of three, Jason (CEO), David (CTO), and Jonas (senior designer) explored a wide variety of ideas before settling on the core. Nearly a year of production mode cycles followed, where all of Basecamp's teams fleshed out HEY's feature set. We ended with two cycles of cleanup and significantly cut back the feature set to launch in July 2020.

它的发展。一个由三人组成的团队，Jason (首席执行官)，David (首席技术官)和 Jonas (高级设计师)在开始设计核心之前探索了各种各样的想法。接下来将近一年的生产模式周期，Basecamp 的所有团队都充实了 HEY 的功能集。我们结束了两个清理周期，并大幅削减了 2020 年 7 月发布的功能集。

To be precise, there was some overlap between R&D and production mode after that first year. Basecamp was big enough as a company that the senior team could shape and delegate production-mode projects around parts of the app that were settled while continuing to explore new territory in R&D mode themselves.

准确地说，在第一年之后，研发和生产模式之间出现了一些重叠。Basecamp 作为一家公司已经足够大了，高级团队可以围绕已经定型的应用程序部分来塑造和授权生产模式项目，同时继续在研发模式下探索新的领域。

Every bet on HEY was placed one at a time. The betting table didn't know they would be working on HEY for two years during those first few R&D cycles. Gradually they gained confidence in the idea and grew a big-picture appetite for how many cycles they were willing to spend on HEY. But they made no specific commitments about what would go into those cycles. And flipping attention back to Basecamp, our existing product, was always on the table.

每次下注都是一次一个。在最初的几个研发周期中，赌桌并不知道他们会在 HEY 上工作两年。渐渐地，他们对这个想法有了信心，并且对他们愿意在 HEY 上花费多少周期有了更大的兴趣。但是他

们并没有对这些周期做出具体的承诺。把注意力转回 Basecamp，我们现有的产品，总是在桌面上。

### *An experimental feature: Hill Charts*

#### **一个实验性功能: Hill Charts**

A third example shows some grey area. When we built the Hill Charts feature in Basecamp (see Chapter 13), we had no idea if it was going to work out or not. Basecamp was an existing product, and it felt too risky to bet on releasing this experimental feature to customers. So we framed the project more like a production mode bet on a new product. We shaped a first version that was just functional enough to use ourselves. We didn't expect to ship it to customers without doing an additional cycle on it. This was a risk: we bet one cycle, not two. If it didn't work out, we'd scrap it. If something more important came up, we might not ever schedule

第三个例子显示了一些灰色地带。当我们在 Basecamp (参见第 13 章) 中构建 Hill Charts 功能时，我们并不知道它是否会成功。

Basecamp 是一个已经存在的产品，我们觉得把这个实验性的功能发布给客户太冒险了。所以我们把这个项目设计得更像是一个生产模式，押注于一个新产品。我们设计了第一个版本，它的功能刚好能让我们自己使用。我们不指望在没有进行额外循环的情况下就把它发送给客户。这是一个风险：我们赌一个周期，而不是两个。如果不成功，我们就取消它。如果有更重要的事情发生，我们可能永远都不会安排

the second cycle. But we ended up feeling confident after the first cycle. We shaped a project to round it out, decided to bet another cycle, and then shipped it to customers.

第二个周期。但是在第一个周期之后，我们最终变得充满信心。我们制定了一个项目来完成它，决定押注于另一个周期，然后把它发送给客户。

## Questions to ask

### 要问的问题

Here are some common questions you might hear when people at the betting table are debating which bets to place.

下面是一些常见的问题，你可能会听到当人们在赌桌上辩论下哪个赌注。

#### ***Does the problem matter?***

#### **这个问题重要吗？**

Just like in pitch write-ups, we always take care to separate problem and solution. The solution doesn't matter if the problem isn't worth solving.

就像在投标报告中一样，我们总是注意把问题和解决方案分开。如果问题不值得解决，解决方案并不重要。

Of course, any problem that affects customers matters. But we have to make choices because there will always be more problems than time to solve them. So we weigh problems against each other. Is *this* problem more important than *that* problem right now?

当然，任何影响客户的问题都很重要。但我们必须做出选择，因为总会有比时间更多的问题需要解决。所以我们要权衡各种问题。这个问题比那个问题更重要吗？

How the people at the table judge problems depends on their perspective, role, and knowledge. For example, a problem might impact a small segment of customers but put a disproportionate burden on

support. Depending on your exposure to support and which aspect of the business you're focused on, you may weigh that differently.

餐桌上的人们如何判断问题取决于他们的观点、角色和知识。例如，一个问题可能会影响到一小部分客户，但却给支持带来不成比例的负担。取决于你对支持的接触程度，以及你关注的业务的哪个方面，你可能会有不同的权衡。

Sometimes a solution that is too complicated or too sweeping may invite questions about the problem. Do we really need to make so many changes across the app? Have we understood the problem specifically enough? Maybe there's a way to narrow it down so that we get 80% of the benefit from 20% of the change.

有时候，一个过于复杂或过于笼统的解决方案可能会引发问题。我们真的需要在整个应用程序中做这么多改变吗？我们是否已经足够明确地理解了这个问题？也许有办法可以缩小范围，这样我们就可以从 20% 的改变中获得 80% 的好处。

### ***Is the appetite right?***

**胃口好吗？**

It's good when we have a solution shaped to a reasonable time frame, like two or six weeks. But we might still debate whether it's

当我们有一个合理的时间框架的解决方案，比如两到六个星期，这是很好的。但是我们可能仍然会争论这个方案是否可行

worth the time. Suppose a stakeholder says they aren't interested in spending six weeks on a given pitch. The negotiation could go a couple directions from there:

值得花时间。假设一个利益相关者说他们没有兴趣在一个特定的场地上花费六个星期。谈判可以从这里开始有几个方向：

1. Maybe the problem wasn't articulated well enough, and there's knowledge that the shaper can add to the conversation right now to swing opinion. For example, "Yeah it doesn't happen often, but when it does people are so vocal about it that it really tarnishes perception of us." Or "Maybe it sounds trivial, but support has to go through 11 time-consuming steps to get to resolution."

也许这个问题表达得不够清楚，也许塑造者现在可以在谈话中加入一些知识来改变观点。例如，“是的，这种情况并不经常发生，但是当它发生时，人们对它如此直言不讳，以至于它真的玷污了我们的感知。”或者“也许这听起来微不足道，但支持必须通过 11 个耗时的步骤才能得到解决。”

2. Sometimes saying "no" to the time commitment is really saying no to something else. Maybe there's something about the solution or the technical implementation they don't like. Asking "How would you feel if we could do it in two weeks?" can uncover that it's not so much about the time. The CTO might answer, "I don't want to introduce another dependency into that area of the app."

有时候，对时间承诺说“不”实际上是对其他事情说“不”。也许他们不喜欢解决方案或者技术实现。问：“如果我们能在两周内完成，你觉得怎么样？”可以发现这不仅仅是时间的问题。首席技术官可能会回答：“我不想在应用程序的那个区域引入另一个依赖项。”

3. The shaper might just let the idea go if interest is too low.  
如果兴趣太低，塑造者可能会放弃这个想法。

4. The shaper might go back to the drawing table and either work on a smaller version (for a shorter appetite) or do more

research if they believe the problem is compelling but they weren't armed well enough to present it.

塑造者可能会回到绘图桌，或者工作在一个较小的版本(为了更短的胃口)，或者做更多的研究，如果他们认为问题是强迫性的，但他们没有足够的武装，以提出它。

### ***Is the solution attractive?***

#### **解决方案有吸引力吗？**

The problem may be important and the appetite fair, but there can be differences about the solution.

这个问题可能很重要，人们的胃口也很好，但是解决方案可能会有分歧。

For example, adding interface elements to the screen carries an invisible cost: giving up the real estate. A button in the corner of the home page might perfectly solve the problem. But that real estate is valuable. If we give it up now, we won't be able to use it in the future.

例如，将界面元素添加到屏幕上会带来无形的成本：放弃不动产。首页角落的一个按钮可能会完美地解决这个问题。但是房地产是很有价值的。如果我们现在放弃它，我们将来就不能使用它了。

Are we selling it too cheaply to solve this particular problem?  
要解决这个问题，我们是不是卖得太便宜了？

If someone offers an immediate design solution, like “how about we move that button to an action menu instead,” we might discuss it. But generally we’ll avoid doing design work or discussing technical solutions for longer than a few moments at the betting table. If we catch ourselves spending too much time in the weeds we’ll remind ourselves “okay, we’re not doing design here” and move back up to the high level.

如果有人提供了一个立竿见影的设计解决方案，比如“我们把那个按钮移到操作菜单上怎么样”，我们可能会讨论一下。但一般来说，我们会避免做设计工作或讨论技术解决方案的时间超过赌桌上的几分钟。如果我们发现自己花了太多时间在杂草中，我们会提醒自己“好吧，我们不在这里做设计”，然后回到更高的层次。

### *Is this the right time?* 现在合适吗？

The kind of project we want to do next can depend on which projects we’ve done recently. Maybe it’s been too long since we’ve made a splash of news with a new feature. Or perhaps we’ve been building too many new features and feel overdue to fix some long-standing customer requests. Or if the teams spent the last couple cycles in the same area of the app, their morale may dip if we plan yet another project doing the same kind of work.

我们下一步想做什么样的项目取决于我们最近做了哪些项目。也许我们已经很久没有用新功能制造新闻了。或者我们已经开发了太多的新功能，觉得应该修复一些长期存在的客户需求。或者，如果团队在过去的几个周期中都在应用程序的同一个区域，如果我们计划另一个项目做同样的工作，他们的士气可能会下降。

Those are all reasons that we might pass on a project even though it’s perfectly well shaped and valuable. The project’s great; it’s just not the right time.

这些都是我们可能放弃一个项目的原因，即使它形状完美，价值不菲。这个项目很棒，只是时机不对。

### *Are the right people available?*

#### **有合适的人选吗？**

As part of the betting process we choose who specifically will play which role on each team. That is, we'll pair a project with a specific small team of a designer and one or two programmers. We have a "Core Product" team of designers and programmers and we select from that pool when planning teams for each cycle. The team will work with each other for the whole cycle and then the next cycle can be a different combination of people.

作为博彩过程的一部分，我们选择谁在每支球队中扮演特定的角色。也就是说，我们将一个项目与一个由一名设计师和一两名程序员组成的特定小团队配对。我们有一个由设计师和程序员组成的“核心产品”团队，我们在每个周期的规划团队中进行选择。这个团队将在整个周期内彼此合作，然后下一个周期可以是不同的人员组合。

Different projects require different expertise. Maybe we need some  
不同的项目需要不同的专业知识。也许我们需要一些

more front-end programming on this one. Or this other one is going to invite a lot of scope creep so we need someone who's good with the scope hammer.

更多的前端编程。或者另一个会招来很多范围蠕变，所以我们需要一个擅长范围锤的人。

The type of work each person has been doing is another factor. Someone who's done a long string of small batch projects might prefer to take on a big batch, or vice versa.

每个人所从事的工作类型是另一个因素。做过一长串小批量项目的人可能更喜欢承担大批量项目，反之亦然。

And lastly there's always a little Calendar Tetris with peoples' availability. Vacations or sabbaticals affect which projects we can schedule in the coming cycle.

最后，总是有一些小的日历俄罗斯方块与人们的可用性。假期或者休假会影响我们在接下来的周期中安排哪些项目。

We've seen some other companies use a different model where instead of assigning the projects to people, they let the team members choose which projects they want to work on. Culturally, we are too meeting-averse for this extra step. But we've heard it can work well for some teams because the project teams have a little more buy-in.

我们已经看到其他一些公司使用不同的模型，他们不把项目分配给人，而是让团队成员选择他们想从事的项目。从文化角度来说，我们对这个额外的步骤非常厌恶会议。但是我们听说这对一些团队很有效，因为项目团队有更多的支持。

Post the kick-off message  
发布启动信息

After the bets are made, someone from the betting table will write a message that tells everyone which projects we're betting on for the next cycle and who will be working on them.

在下注之后，赌桌上的某个人会写一条信息，告诉每个人我们在下一个周期下注哪个项目，以及谁将从事这些项目。

## Sill: The last cycle of 2017



FYI by Jason Fried  
Nov 17, 2017 · Notified 53 people

It's next cycle time!

First, this is a short cycle. We've got a few major holidays this cycle, plus the end of the year. People are busy with life, travel, snow (!), and all the other stuff that goes with ramping down one year and starting another.

So with that in mind, we're going to do something a bit different this cycle. We're only going to schedule a few core projects focused on making Basecamp feel simpler, a couple of experimental projects, and no dedicated small batch projects.

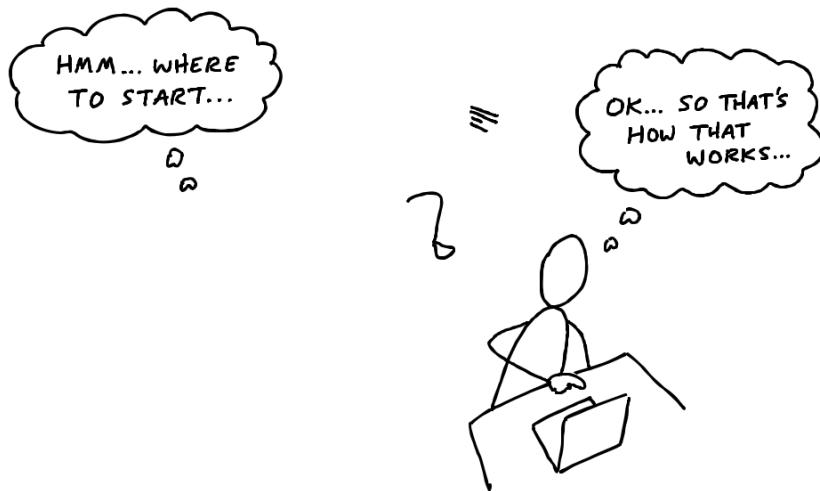
Merge Latest Activity and Reports

*Jason announces the bets for the next cycle with a Basecamp message*  
*Jason 通过一条 Basecamp 消息宣布下一个周期的赌注*



PART THREE  
Patthre

# Building 大楼



## Hand Over Responsibility 移交责任

We've made our bets and now it's time to start the next cycle. How does the team get started?

我们已经下了赌注，现在是时候开始下一个周期了。球队如何开始？

Assign projects, not tasks  
分配项目，而不是任务

We *don't* start by assigning tasks to anyone. Nobody plays the role of the “taskmaster” or the “architect” who splits the project up into pieces for other people to execute.

我们不会一开始就给任何人分配任务。没有人扮演“任务主管”或者“架构师”的角色，他们把项目分解成各个部分让其他人去执行。

Splitting the project into tasks up front is like putting the pitch through a paper shredder. Everybody just gets disconnected pieces. We want the project to stay “whole” through the entire process so we never lose sight of the bigger picture.

提前把项目分解成多个任务就像把推销员扔进碎纸机一样。每个人都只是得到了不连续的碎片。我们希望项目在整个过程中保持“完整”，这样我们就不会忽视大局。

Instead, we trust the team to take on the entire project and work within the boundaries of the pitch. The team is going to define their own tasks and their own approach to the work. They will have full autonomy and use their judgement to execute the pitch as best as

相反，我们相信团队能够承担整个项目，并在规定的范围内工作。

团队将定义他们自己的任务和他们自己的工作方法。他们将拥有完全的自主权，并运用自己的判断力，以最好的方式执行任务

they can.  
他们可以。

Teams love being given more freedom to implement an idea the way they think is best. Talented people don't like being treated like "code monkeys" or ticket takers.

团队喜欢被给予更多的自由，以他们认为最好的方式实现一个想法。有才能的人不喜欢被当作“代码猴子”或者票贩子对待。

Projects also turn out better when the team is given responsibility to look after the whole. Nobody can predict at the beginning of a project what exactly will need to be done for all the pieces to come together properly. What works on paper almost never works exactly as designed in practice. The designers and programmers doing the real work are in the best position to make changes and adjustments or spot missing pieces.

当团队被赋予照顾整个项目的责任时，项目也会变得更好。没有人能够在项目一开始就预测到底需要做些什么才能使所有的部分正确地结合在一起。在纸上行之有效的方法几乎从来没有在实践中完全按照设计的方法行之有效。做实际工作的设计师和程序员处于最佳位置，可以进行修改和调整，或者找出缺失的部分。

When teams are assigned individual tasks, each person can execute their little piece without feeling responsible for judging how all the pieces fit together. Planning up front makes you blind to the reality along the way. 当团队被分配到单独的任务时，每个人都可以执行他们的小任务，而不用感到有责任判断所有的任务是如何组合在一起的。提前计划会让你忽略一路上的现实。

Remember: we aren't giving the teams absolute freedom to invent a solution from scratch. We've done the shaping. We've set the boundaries. Now we are going to trust the team to fill in the outline from the pitch with real design decisions and implementation.

记住：我们不会给团队绝对的自由，让他们从头开始发明解决方案。我们已经完成了设计。我们设定了界限。现在，我们将信任团队，让他们用真正设计决策和实现来填写大纲。

This is where our efforts to define the project at the right level of abstraction—without too much detail—will pay off. With their talent and knowledge of the particulars, the team is going to arrive at a better finished product than we could have by trying to determine the final form in advance.

这就是我们在正确的抽象层次上定义项目的努力——没有太多的细节——将会得到回报的地方。凭借他们的天赋和对细节的了解，团队将会得到一个比我们提前确定最终形式更好的成品。

## Done means deployed 完成意味着部署

At the end of the cycle, the team will deploy their work. In the case of a Small Batch team with a few small projects for the cycle, they'll 在周期结束时，团队将部署他们的工作。如果是一个小批量团队，在这个周期中有几个小项目，他们将

deploy each one as they see fit as long as it happens before the end of the cycle.

只要在周期结束之前发生，他们认为合适就可以部署每一个。

This constraint keeps us true to our bets and respects the circuit breaker. The project needs to be done within the time we budgeted; otherwise, our appetite and budget don't mean anything.

这种约束使我们忠于我们的赌注，并尊重断路器。这个项目需要在我们预算的时间内完成；否则，我们的胃口和预算没有任何意义。

That also means any testing and QA needs to happen *within* the cycle. The team will accommodate that by scoping off the most essential aspects of the project, finishing them early, and coordinating with QA. (More on that later.)

这也意味着任何测试和质量保证都需要在周期内完成。团队将通过确定项目中最重要的方面，提前完成它们，并与 QA 协调来适应这种情况。(稍后详细介绍)

For most projects we aren't strict about the timing of help documentation, marketing updates, or announcements to customers and don't expect those to happen within the cycle. Those are thin-tailed from a risk perspective (they never take 5x as long as we think they will) and are mostly handled by other teams. We'll often take care of those updates and publish an announcement about the new feature during cool-down after the cycle.

对于大多数项目，我们并不严格要求帮助文档、市场营销更新或者向客户发布公告的时间，也不希望这些事情在周期内发生。从风险的角度来看，这些工作都很细微(它们从来不会花费我们预计的 5 倍的时间)，而且大部分都是由其他团队处理的。我们通常会关注这些更新，并在周期结束后的冷却阶段发布有关新功能的声明。

## Kick-off 开始

We start the project by creating a new Basecamp project and adding the team to it. Then the first thing we'll do is post the shaped concept to the Message Board. We'll either post the original pitch or a distilled version of it.

我们通过创建一个新的 Basecamp 项目并将团队添加到其中来开始这个项目。然后我们要做的第一件事就是把这个成形的概念发布到留言板上。我们可以发布原始的推介或者它的精华版本。

Home Pings Hey! Activity My Stuff Find

BC3: Hill Charts > Message Board

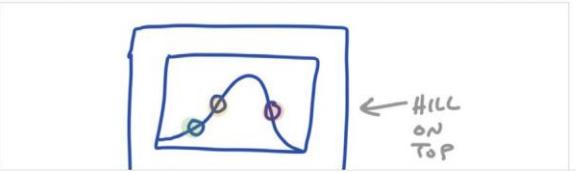
## Hill Charts concept

Announcement by Ryan Singer Nov 17, 2017 · No one was notified. 1

Here's the starting concept for the Hill Chart feature in Basecamp.

**How they relate to to-dos**

The basic idea is there's a hill chart at the top of the todoset. Hill **charts** map one-to-one with todosets, and the **dots** on the hill map one-to-one with todolists.



*The first thing on the Basecamp project is a message with the shaped concept*  
Basecamp 项目的第一件事情是一条带有成形概念的消息

Since our teams are remote, we use the chat room in the Basecamp project to arrange a kick-off call.

由于我们的团队是远程的，我们使用 Basecamp 项目的聊天室来安排一个开始电话。

Home Pings Hey! Activity My Stuff Find

BC3: Hill Charts

### Campfire

Monday, November 20 2017

Hey Scott Pratik I'm free to do a kick off after 12pm today  
got an interview starting now. Bb45

Pratik Naik 9:59am Any time today works for me  
How about 12pm central?

Scott Upton 10:05am That works for me

Scott Upton 10:40am even better

Pratik Naik 10:42am Works for me

10:39am Me Would 12:30pm be ok?

*Arranging a call with the team to walk through the shaped work*

与团队安排一次电话会议，一起走过这些成形的作品

## CHAPTER 10 - HandOverResponsibility

99

The call gives the team a chance to ask any important questions that aren't clear from the write-up. Then, with a rough understanding of the project, they're ready to get started.

电话会议让团队有机会问一些重要的问题，这些问题在报道中并不清楚。然后，对项目有了大致的了解，他们就可以开始了。

## Getting oriented 开始定位

Work in the first few days doesn't look like "work." No one is checking off tasks. Nothing is getting deployed. There aren't any deliverables to look at. Often there isn't even much communication between the team in the first few days. There can be an odd kind of radio silence.

头几天的工作看起来不像是"工作"没有人在检查任务。没有任何东西被部署。没有任何可交付的东西可以看。通常在最初的几天里，团队之间甚至没有太多的沟通。可能会有一种奇怪的无线电静默。

Why? Because each person has their head down trying to figure out how the existing system works and which starting point is best.

Everyone is busy learning the lay of the land and getting oriented.

为什么？因为每个人都低着头试图弄清楚现有的系统是如何工作的，以及哪个起点是最好的。每个人都在忙着了解情况并找到方向。

The screenshot shows a Campfire chat window with the following messages:

- Scott Upton** 3:46pm: We'll need to stub in the Hill Chart Editor, too (i.e. where you can annotate and update positions for each list)
- Pratik Naik** 4:07pm: Scott I can add an empty controller for the editor. But I think it'd help to have some mocks first.
- Scott Upton** 4:15pm: Cool, OK. I'll work something up tomorrow AM.
- Pratik Naik** 4:15pm: Sounds good!

**1 0 0**  
**100**

*The team figuring out where to start*  
团队正在寻找出发点

It's important for managers to respect this phase. Teams can't just dive into a code base and start building new functionality immediately. They have to acquaint themselves with the relevant code, think through the pitch, and go down some short dead ends to find a starting point. Interfering or asking them for status too early hurts the project. It takes away time that the team needs to find the best approach. The exploration needs to happen anyway. Asking for visible progress will only push it underground. It's better to empower the team to explicitly say "I'm still figuring out how to start" so they don't have to hide or disguise this legitimate work.

对于管理者来说，尊重这个阶段是很重要的。团队不能一头扎进代码库，立即开始构建新的功能。他们必须熟悉相关的代码，思考整个过程，然后走进一些短暂的死胡同，找到一个起点。过早地干涉或询问他们的状态会损害项目。团队需要找到最好的方法，这会占用他们很多时间。无论如何，探索都需要发生。要求可见的进展只会把它推向地下。最好是授权团队明确地说“我还在考虑如何开始”，这样他们就不必隐藏或掩饰这些合法的工作。

Generally speaking, if the silence doesn't start to break after three days, that's a reasonable time to step in and see what's going on.  
一般来说，如果三天后沉默还没有打破，那么这是一个合理的时间介入，看看发生了什么。

## Imagined vs discovered tasks 想象任务 vs 发现任务

Since the team was given the project and not tasks, they need to come up with the tasks themselves. Here we note an important difference between tasks we *think* we need to do at the start of a project and the tasks we *discover* we need to do in the course of doing real work.

因为团队被赋予了项目而不是任务，他们需要自己想出任务。

在这里，我们注意到在项目开始时我们认为需要完成的任务和

在实际工作过程中我们发现需要完成的任务之间的一个重要区别。

The team naturally starts off with some imagined tasks—the ones they assume they’re going to have to do just by thinking about the problem. Then, as they get their hands dirty, they discover all kinds of other things that we didn’t know in advance. These unexpected details make up the true bulk of the project and sometimes present the hardest challenges.

团队自然而然地从一些想象中的任务开始——那些他们认为他们仅通过思考问题就能完成的任务。然后，当他们把手弄脏的时候，他们会发现其他各种我们事先不知道的事情。这些意想不到的细节构成了项目的真正主体，有时也会带来最艰巨的挑战。

Teams discover tasks by doing real work. For example, the designer adds a new button on the desktop interface but then notices there's no obvious place for it on the mobile webview version. They record a new task: figure out how to reveal the button on mobile. Or the

团队通过做真正的工作来发现任务。例如，设计师在桌面界面上添加了一个新按钮，但是发现在移动网络视图版本中没有明显的位置。他们记录了一个新的任务：弄清楚如何在移动设备上显示按钮。或者

first pass of the design has good visual hierarchy, but then the designer realizes there needs to be more explanatory copy in a place that disrupts the layout. Two new tasks: Change the layout to accommodate explanatory copy; write the explanatory copy.

设计的第一步有很好的视觉层次结构，但是设计师意识到在一个打乱布局的地方需要更多的解释性复制。两个新的任务：改变布局以适应解释性副本；写解释性副本。

Often a task will appear in the process of doing something unrelated. Suppose a programmer is working on a database migration. While looking at the model to understand the associations, she might run into a method that needs to be updated for a different part of the project later. She's going to want to note a task to update that method later.

通常一个任务会出现在做一些无关的事情的过程中。假设一个程序员正在进行数据库迁移。在查看模型来理解关联的时候，她可能会碰到一个方法，这个方法需要稍后为项目的不同部分进行更新。她需要记录一个任务，以便稍后更新该方法。

**The way to really figure out what needs to be done is to start doing real work.** That doesn't mean the teams start by building just anything. They need to pick something meaningful to build first. Something that is central to the project while still small enough to be done end-to-end—with working UI and working code—in a few days.

真正弄清楚需要做什么的方法是开始做真正的工作。这并不意味着团队从建立任何东西开始。他们首先需要做一些有意义的事情。一些对于项目至关重要的东西，同时仍然足够小，可以在几天内通过工作用用户界面和工作代码进行端到端的处理。

In the next chapters we'll look at how the team chooses that target and works together to get a fully integrated spike working.

在接下来的章节中，我们将看到团队如何选择目标并一起工作，  
以获得一个完全集成的尖峰工作。



## Get One Piece Done 完成一件事

As the team gets oriented, they start to discover and track the tasks they need to do to build the project. It's important at this early phase that they don't create a master plan of parts that should come together in the 11th hour. If the team completes a lot of tasks but there's no "one thing" to click on and try out, it's hard to feel progress. A team can do a lot of work but feel insecure because they don't have anything real to show for it yet. Lots of things are done but nothing is *really* done.

随着团队逐渐确定方向，他们开始发现并跟踪构建项目所需要完成的任务。在这个早期阶段很重要的一点是，他们不能在最后一刻制定一个部分的总体计划。如果团队完成了很多任务，但是没有“一件事”可以点击并尝试，那么很难感觉到进展。一个团队可以做很多工作，但是会感到不安全，因为他们还没有任何实质性的东西来展示。很多事情已经完成了，但是没有一件是真正完成的。

Instead they should aim to make something tangible and demoable early—in the first week or so. That requires integrating vertically on

one small piece of the project instead of chipping away at the horizontal layers.

相反，他们的目标应该是在头一周左右的时间里，尽早做出一些切实可行的东西。这就需要在项目的一小部分上进行垂直整合，而不是在水平层上进行削减。

## Integrate one slice 整合一个部分

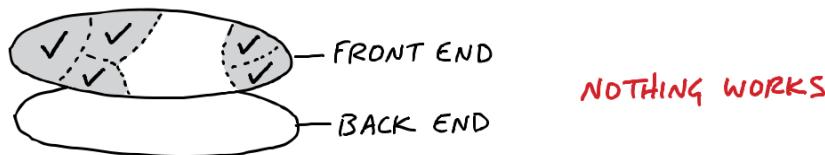
We can think of projects in two layers: front-end and back-end,  
我们可以把项目分为两个层次：前端和后端，

design and code. While technically speaking there are more layers than this, these two are the primary integration challenge in most projects.

设计和代码。虽然从技术上讲，有更多的层次，这两个是主要的集成挑战在大多数项目。

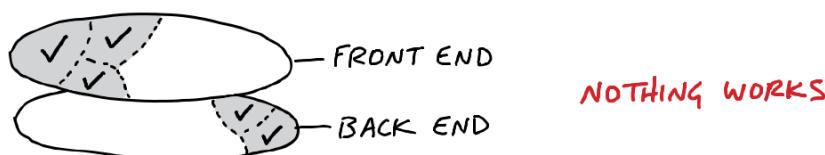
Suppose the project starts with a lot of design. The team could design a variety of screens and even implement them as templates or views. But until they're wired to a backend, nothing does anything. The work remains hypothetical and speculative.

假设项目从大量的设计开始。团队可以设计各种各样的屏幕，甚至将它们实现为模板或视图。但是在他们连接到后端之前，什么都做不了。这项工作仍然是假设性和推测性的。



Same with the backend. A lot of tasks could be checked off, but without any UI—what can you do with it? How do you judge if the work on a specific piece of business logic is really right without interacting with it?

后端也是。很多任务都可以被勾掉，但是没有 ui，你能用它做什么呢？你如何判断在一个特定的业务逻辑上的工作是否真的是正确的，而不需要与它交互？



What we want instead is to pick off one slice of the project to integrate. Then when that's done, the team has something tangible that they've proven to work (or not work and reconsider). Anyone can click through the interaction and see if the feature does what it should and if what it does is what they want.

相反，我们想要的是从项目中挑出一个部分进行整合。然后当这些完成后，团队就有了一些他们已经证明可行(或者不可行并重新考虑)的切实可行的东西。任何人都可以点击这个交互，看看这个功能是否做到了它应该做的，是否做到了他们想要的。



SOMETHING WORKS!

## Case study: Clients in projects

### 案例分析: 项目中的客户

We built a feature in Basecamp 3 that allowed service firms to invite clients to their projects and share chosen documents, messages, or to-do lists with them. The concept, defined in the pitch, had a variety of moving parts:

我们在 Basecamp 3 中构建了一个特性，允许服务公司邀请客户参与他们的项目，并与他们共享选定的文档、消息或待办事项列表。这个概念，在演讲中定义，有很多不同的活动部分：

- Client Access: Before this feature, Basecamp's access model was all or nothing. We needed a way to invite some people to see just some parts of a project. This had major back-end and caching implications.

客户端访问: 在这个功能之前，Basecamp 的访问模型是全有或全无。我们需要一种方法来邀请一些人来看一个项目的一些部分。这对后端和缓存有很大的影响。

- Client Management: We needed a way to add clients to projects and the ability to manage clients separately from team members.

客户端管理: 我们需要一种将客户端添加到项目中的方法，以及独立于团队成员管理客户端的能力。

- Visibility Toggle: Each piece of content in a project should have a toggle to expose it to clients or not.

可见性切换：项目中的每一个内容都应该有一个切换  
来向客户公开或者不公开。

The team had one designer and one programmer. After they got oriented and familiar with how the existing code worked, the designer chose the visibility toggle as the best place to integrate first. This was the most central piece of UI in the project. It's the one that would appear in demo videos and the interaction customers would use most.

这个团队有一个设计师和一个程序员。在他们熟悉了现有代码的工作方式之后，设计师选择了可见性切换作为首先集成的最佳位置。这是项目中最核心的UI部分。它会出现在演示视频中，并且是客户使用最多的交互。

The designer didn't make a pixel-perfect mockup. Instead, he experimented with different affordances and placements in the app's HTML templates. Should the toggle be two radio buttons, a  
设计师没有做出像素完美的模型。相反，他在应用程序的  
HTML 模板中尝试了不同的亮度和位置。如果切换为两个单选  
按钮，

checkbox, or a custom button that changes state?  
复选框，还是改变状态的自定义按钮？

Meanwhile, the programmer wasn't waiting around. He had enough guidance from the pitch to start spiking the access model.

与此同时，程序员并没有坐以待毙。他从球场上得到了足够的指导，开始对接入模型进行调整。

As soon as the designer felt confident in the basic direction of the UI, he pinged the programmer and showed him the stubbed toggle.  
Stepping away from the access problem for a bit, the programmer wired the toggle enough so that it would appear on all the supported content types, change state when clicked, and save its state in the database.  
一旦设计师对用户界面的基本方向有了信心，他就会向程序员发送 ping 信号，并向他展示存根开关。远离访问问题一段时间后，程序员将开关连接到所有支持的内容类型上，单击时改变状态，并将其状态保存到数据库中。

At this point, the toggle didn't actually change the visibility of the content. But it worked from the service firm's point of view. The designer could click it, feel it, and judge how well it worked with live data on a staging server.

此时，切换实际上并没有改变内容的可见性。但是从服务公司的角度来看，它是有效的。设计者可以点击它，感受它，并判断它在临时服务器上处理实时数据的效果如何。

There was still more design work to do on the toggle. But the programmer didn't need to be involved anymore. With the affordance wired up, the designer could continue to experiment with copy, placement, color, mobile view rendering, and more. Meanwhile, the programmer could get back to the access model or whatever else was most important to tackle next.

在这个开关上还有更多的设计工作要做。但是程序员不需要再参与其中了。设计师可以继续尝试复制、放置、颜色、移动视图渲染等等。与此同时，程序员可以回到访问模型或其他任何最重要的解决下一步。

About three days after the start of the project, the designer demoed the working toggle to a manager. Their conversation led to a few more tweaks and then they were able to call the toggle “done.” One important piece of the project was designed, implemented, demoed, and settled. The team felt good about showing tangible progress. And the team and management both felt confidence in the project by seeing a working piece. By clicking through a core interaction early, they were able to validate that what they hoped would make sense in theory did indeed look right and make sense in practice.

大约在项目开始三天后，设计师将工作开关演示给一位经理。他们的谈话导致了一些更多的调整，然后他们能够称之为“完成”该项目的一个重要部分是设计、实施、演示和确定的。团队对于展示实实在在的进展感觉良好。团队和管理层都通过看到一个工作环节而对项目充满信心。通过早期点击核心交互，他们能够验证他们希望在理论上有意义的东西确实看起来是正确的，在实践中也是有意义的。

This short example illustrates a few points about how the teams integrate over short periods to finish one piece of the project at a time.

这个简短的例子说明了关于团队如何在短时间内集成来一次完成项目的一部分的一些要点。

## Programmers don't need to wait 程序员不需要等待

Because the important moving parts were already defined in the shaping process, programmers don't need to sit idle waiting for design when the project starts. There's enough direction in the pitch for them to start working on back-end problems from the start. They won't be able to take a piece of functionality to completion without knowing where it leads on the front-end, but there should be enough information in the pitch to inform foundational modeling decisions.

因为重要的移动部件已经在成型过程中定义，程序员不需要在项目开始时坐等设计。有足够的方向让他们从一开始就开始解决后端问题。如果不知道前端的结果，他们将无法完成一项功能，但是应该有足够的信息来为基础建模决策提供信息。

## Affordances before pixel-perfect screens 在像素完美的屏幕前提供支持

Programmers don't need a pixel-perfect design to start implementing. All they need are endpoints: input elements, buttons, places where stored data should appear. These affordances are the core of a user interface design.

程序员不需要像素完美的设计来开始实现。他们需要的只是端点：输入元素，按钮，存储数据应该出现的地方。这些功能是用户界面设计的核心。

Questions about font, color, spacing, and layout can be resolved after the raw affordances are in place and hooked up in code. Copy-writing, basic affordances, and some wiring are all we need to try a live working version in the browser or on the device. Then we can answer the fundamental questions early: Does it make sense? Is it understandable? Does it do what we want?

关于字体、颜色、间距和布局的问题，可以在原始的支持功能到位并连接到代码之后解决。我们需要在浏览器或设备上尝试一个实时的工作版本。然后我们可以尽早回答这些基本问题：它有意义吗？可以理解吗？它能做我们想做的事吗？

That means the first interface a designer gives to a programmer can look very basic, like the example below. It's more like a breadboard than a visual design or a polished mock-up.

这意味着设计师给程序员的第一个界面可以看起来非常基本，如下面的示例。它更像是一个面板，而不是一个视觉设计或者一个抛光的模型。

The screenshot shows a registration form for a course. At the top, it says "Please choose an option" with a radio button selected for "Private Room". Below that is a section titled "Arrival and departure" with fields for arrival and departure dates. The arrival date is set to "Friday May 24, 2019" and the time is set to "in the Afternoon". The departure date is set to "Monday May 27, 2019". There is also a "Your information" section with a "First name" field.

This screenshot is from a registration app for multi-day courses. The designer made it in HTML by hand. There's barely any style — just enough visual hierarchy to feel confident that the layout is usable and amenable to future layers of styling.

这个截图来自一个多日课程的注册应用程序。设计者用 HTML 手工制作的。几乎没有任何风格 —— 只要有足够的视觉层次感，就可以确信布局是可用的，并且适用于未来的风格层次。

While the design looks simple, a lot of decisions are reflected in it. 虽然设计看起来很简单，但它反映了许多决策。

- The decision to ask for arrival time but not departure time came from detailed discussions about the business logic and pricing model.

要求到达时间而不是离开时间的决定来自于对商业逻辑和定价模型的详细讨论。

- The specific options in the arrival time pulldown correspond to rules that had to be worked out about when to charge for meals and overnight stays. The designer's first sketches used a calendar-style date picker for the arrival and departure days. But that led to UX problems. Some courses were long (multiple weeks) with different phases. There wasn't room in a standard calendar-style date picker to

label the phases on the day boxes. With a pulldown, she could use option groups to label groups of dates when needed. That way users wouldn't need to reference a schedule elsewhere to be sure they were selecting the right dates.

抵达时间下拉表中的具体选项对应于必须制定的关于何时对膳食和过夜住宿收费的规则。设计师的第一个草图使用了一个日历风格的日期选择器来选择到达和离开的日期。但这导致了用户体验问题。有些课程很长(几个星期)，有不同的阶段。在标准的日历风格的日期选择器中没有空间来在日程表上标记每个阶段。通过下拉菜单，她可以在需要的时候使用选项组来标记日期组。这样用户就不需要参考其他地方的日程表来确保他们选择了正确的日期。

Here's another example. This is the first working piece of an app for capturing data from customer interviews.

这是另一个例子，这是一个应用程序的第一个工作部件，用于从客户访谈中获取数据。

Setup	Pushes	Pulls	Habits	Anxieties	Observations
	<ul style="list-style-type: none"><li>• When I just hired another person</li><li>• When I'm going to be on vacation</li><li>• When my staff has been losing track of information</li></ul>	<ul style="list-style-type: none"><li>• So my staff will know what to do</li><li>• So I can feel I'm setting my new employee up for success</li><li>• So I won't be interrupted while I'm on vacation</li></ul>			

[Edit Setup](#) [Add/edit Pushes](#) [Add/edit Pulls](#) [Add/edit Habits](#) [Add/edit Anxieties](#) [Edit Observations](#)

At this early stage the project name (Basecamp) and interview subject (Jan) were hard-coded and most of the links didn't go anywhere.

在这个早期阶段，项目名称(Basecamp)和面试主题(Jan)都是硬编码的，大部分链接都没有任何进展。

Look at how raw this design is. The actions are plain text links in the default blue and purple browser colors. The boxes containing the data points are barely styled with plain black borders. As rough as it is, this design tests some important trade-offs. The designer chose to show as much data as possible above the fold so it would be easy to review interviews. That didn't leave enough room within each section for UI to add, edit, or remove data points. That led the designer to create separate screens for adding and editing data per section.

看看这个设计有多原始。操作是默认的蓝色和紫色浏览器的纯文本链接。包含数据点的框几乎没有纯黑边框的样式。尽管它很粗糙，但是这个设计测试了一些重要的权衡。设计师选择展示尽可能多的数据在折叠之上，以便于审查采访。这样就没有在每个部

分留出足够的空间给 UI 添加、编辑或删除数据点。这导致设计师创建了单独的屏幕来添加和编辑每个部分的数据。



This is the first design for adding and editing “pulls” — a type of data in this interview technique. Again, look at how raw it is. There’s just enough design here to quickly wire it up and test it. The team can click through this to judge whether navigating to a separate screen to record data is acceptable or not. If it works, they can layer on additional styling later. If it doesn’t work, they didn’t waste a lot of time implementing a pixel-perfect design.

这是第一个添加和编辑“拉”的设计，“拉”是这种采访技巧中的一种数据类型。再一次，看看它是多么的原始。这里有足够的设计，可以快速连接并测试它。团队可以通过点击这个来判断导航到一个标准的屏幕来记录数据是否可以接受。如果成功的话，他们可以在以后添加额外的样式。如果不成功，他们也没有浪费太多时间来实现像素级的完美设计。

Beautiful alignment, colors, and typography don’t matter on the first pass. Visual styling is important in the end product, not in the early stages. The biggest uncertainties are about whether it will work, whether it will make sense, and how hard it will be to implement. After the elements are wired up, they can be rearranged, restyled, and repainted to improve the work that’s already done. First make it work, then make it beautiful.

漂亮的对齐、颜色和排版在第一次通过时并不重要。视觉造型在最终产品中很重要，而不是在早期阶段。最大的不确定性在于它是否可行，是否有意义，以及实现起来有多难。在这些元素被连接起来之后，它们可以被重新安排，重新设计，重新绘制以改进已经完成的工作。首先让它工作起来，然后让它变得漂亮。

## Program just enough for the next step 程序刚好够下一步使用

The same is true for back-end work. It doesn’t have to be all or nothing. Sometimes a designer just needs some scaffolding—a couple fields that save data or some code to navigate from one

stubbed screen to another. Other times she needs to populate a variable in the template with a collection of real data so she can iterate on different displays (rows, columns, media boxes, etc) to find the best design.

后端工作也是如此。它不需要全部或者什么都没有。有时候设计师只需要一些脚手架——一些保存数据的字段或者一些代码来从一个屏幕导航到另一个屏幕。其他时候，她需要在模板中用一组真实数据填充一个变量，这样她就可以在不同的显示器（行、列、媒体框等）上迭代以找到最佳设计。

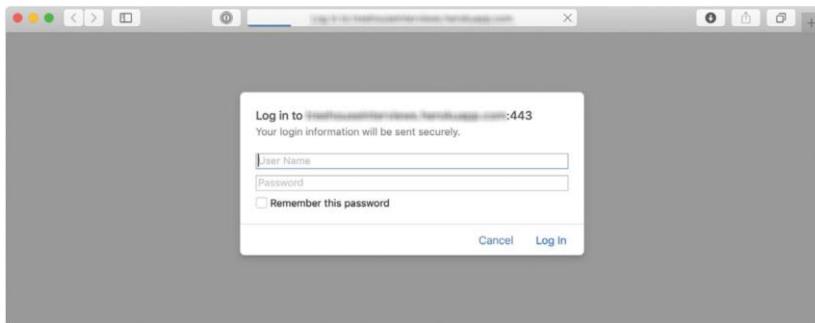
The early back-end work can be strategically patchy. There might be a controller to render templates but no model. Or a controller and bits of a model with mock data but no support for creating or updating the data. Screens that aren't wired yet could at least be connected with routes for navigating between them.

早期的后端工作可能在战略上是零碎的。可能有一个控制器来渲染模板，但是没有模型。或者一个控制器和模拟数据的模型位，但是不支持创建或更新数据。还没有连线的屏幕至少可以连接到它们之间的导航路线。

When it was time to test the first piece of the interview app, the  
当测试面试应用程序的第一部分时，

team knew there would be sensitive data from real interviews going into it. They needed to protect it with some kind of authentication. Rather than building full username and password support—or even integrating a third-party solution—they just used plain HTTPAuth to hard-code a password.

团队知道会有来自真实访谈的敏感数据进入其中。他们需要用某种认证来保护它。他们没有建立完整的用户名和密码支持，甚至没有整合第三方解决方案，他们只是使用简单的 HTTPAuth 来硬编码密码。



This allowed the team to try adding data from real interviews very early in the cycle, without slowing down to hook up some authentication code that wasn't going to teach them anything about the problems they were trying to solve.

这使得团队可以在周期的早期尝试添加来自真实访谈的数据，而不必放慢脚步去连接一些认证代码，这些代码不会教给他们任何关于他们试图解决的问题的东西。

The point is to create a back-and-forth between design and programming on the same piece of the product. Instead of one big hand-off, take turns layering in affordances, code, and visual styling. Step by step, click through the real working feature-in-prog-ress to judge how it's coming and what to do next.

关键是在同一个产品的设计和编程之间建立一个来回。不是一次大的交接，而是轮流在功能、代码和视觉样式上进行分层。一步一步，通过点击正在进行的真正的工作特性来判断它是如何来的以及下一步要做什么。

# Start in the middle

从中间开始

In the examples above, the team didn't build log in first. They didn't build a way to create an interview project and an interview subject before solving the problem of adding interview data. They jumped straight into the middle where the interesting problem was and stubbed everything else to get there.

在上面的例子中，团队没有先建立登录。在解决添加面试数据的问题之前，他们没有建立一个创建面试项目和面试主题的方法。他们直接跳到有趣的问题所在的中间，然后把其他所有的问题都解决掉。

To expand on this, here are three criteria to think about when choosing what to build first:

为了进一步说明这一点，在选择首先建造什么的时候，有三个标准需要考虑：

First, it should be **core**. The visibility toggle was core to the Clients in Projects concept. Without it, the other work wouldn't mean anything. Contrast that with a more peripheral aspect of the project, like the ability to rename a client. Both were “required,” but one was more central and important to prove out early in the cycle. In the interview app, recording interview data was more core—more in the middle—than setting up a new research project.

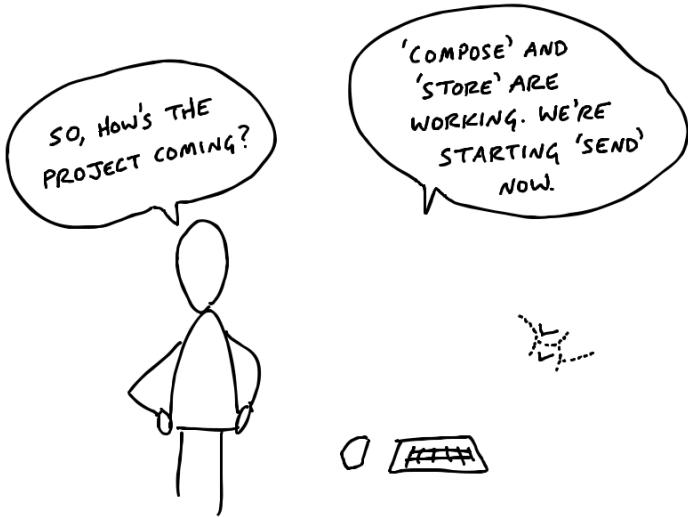
首先，它应该是核心。可见性切换是项目概念中客户端的核心。没有它，其他的工作就没有任何意义。与此形成对比的是这个项目的一个更外围的方面，比如重命名客户的能力。两者都是“必需的”，但其中一个更为核心和重要，需要在周期早期证明。在面试应用程序中，记录面试数据比建立一个新的研究项目更核心-更中间。

Second, it should be **small**. If the first piece of work isn't small enough, there isn't much benefit to carving it off from the rest. The point is to finish something meaningful in a few days and build momentum—to have something real to click on that shows the team is on the right track. 其次，它应该是小型的。如果第一件作品不够小，那么从其他作品中分离出来也没有什么好处。关键是要在几天内完成一些有意义的事情，并建立动力——有一些真实的东西可以点击，以表明团队在正确的轨道上。

Third, it should be **novel**. If two parts of the project are both core and small, prefer the thing that you've never done before. In the Clients in Projects feature, the UI for adding clients was mostly the same as the UI for adding regular users. Starting on that would have moved the project forward, but it wouldn't have taught the team anything. It

wouldn't have eliminated uncertainty. Starting with the visibility toggle boosted everyone's confidence because it proved that a new idea was going to work.

第三，应该是新奇的。如果项目的两个部分都是核心部分和小部分，那么更喜欢你以前从未做过的事情。在项目客户端功能中，用于添加客户端的用户界面与用于添加普通用户的用户界面大致相同。开始这样做可以推动项目向前发展，但是不会教会团队任何东西。它不会消除不确定性。从可见度切换开始，提升了每个人的信心，因为它证明了一个新的想法是可行的。



## Map The Scopes 绘制瞄准镜地图

In the previous chapter, we started the project by finishing one integrated slice early on. That practice belongs to a more general technique that the team can use throughout the project.

在上一章中，我们通过早期完成一个集成片开始了这个项目。

这种做法属于一种更通用的技术，团队可以在整个项目中使用。

### Organize by structure, not by person 按照结构组织，而不是按照个人

When asked to organize tasks for a project, people often separate work by person or role: they'll create a list for Designers and a list for Programmers. This leads to the problem we talked about in the previous chapter—people will complete tasks, but the tasks won't add up to a finished part of the project early enough.

当被要求为项目组织任务时，人们通常会根据个人或角色将工作分开：他们会为设计人员创建一个列表，为程序员创建一个列表。

这就导致了我们在前一章中讨论过的问题——人们会完成任务，但是任务不会在足够早的时候完成项目的一部分。

To take an example outside of software, consider someone organizing a fundraising event. They could create a list of tasks for each of their three volunteers and track the work that way. But then there'd be no way to see the big picture of how the event is coming together—what's done and what's not done at the macro

举一个软件之外的例子，假设有人组织了一个筹款活动。他们可以为他们的三个志愿者创建一个任务列表，并以这种方式跟踪工作。但是那样就没有办法看到事件是如何走到一起的大图景——宏观上什么已经做了，什么还没有做

level. Instead, they should create lists based on the *structure* of the project—the things that can be worked on and finished independently of each other. To do that, they would create lists for Food Menu, Venue Setup, and Light/Sound. Then the organizer can easily see which areas are done and which areas have outstanding work.

水平。相反，他们应该根据项目的结构创建清单——这些事情可以相互独立地工作和完成。为了做到这一点，他们需要为 Food Menu, Venue Setup, and Light/Sound 创建列表。然后组织者可以很容易地看到哪些区域已经完成，哪些区域已经完成。

In product development, the categories aren't pre-cut for us. We usually build things we've never built before. Each project is a wild territory that we have to walk through before we can draw a map. By digging into the work, we figure out where the interdependen-cies are, how things are connected, and what we can slice apart.

在产品开发中，类别不是为我们预先设定的。我们通常制造我们从未制造过的东西。每个项目都是我们在绘制地图之前必须经过的一片荒野。通过深入研究工作，我们可以找出相互依赖的地方，事物之间的联系，以及我们可以分割的部分。

As we saw in the previous chapter, the slices of work integrate front-end and back-end tasks. This allows us to finish one slice of the actual project and definitively move on. That's better than having lots of pieces that—fingers crossed—are supposed to come together by the end of the cycle.

正如我们在前一章中看到的，工作片集成了前端和后端任务。这样我们就可以完成实际项目的一部分，然后继续工作。这总比在周期结束时许多部分——手指交叉在一起——要好得多。

We call these integrated slices of the project scopes. We break the overall scope (singular) of the project into separate scopes (plural) that can be finished independently of each other. In this chapter, we'll see

how the team maps the project into scopes and tackles them one by one.

我们称之为项目范围的整合片段。我们把项目的整体范围(单数)分解成独立的范围(复数)，这些范围可以彼此独立完成。在本章中，我们将看到团队是如何将项目映射到各个范围并逐个处理它们的。

## The scope map

### 范围图

Imagine an overhead view of  
想象一个俯视图

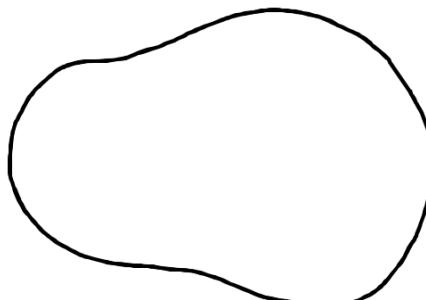
the project. At the beginning,  
这个项目开始的时候，

there's just an outline from  
只有一个轮廓

the shaping work that pre-  
前期的塑造工作

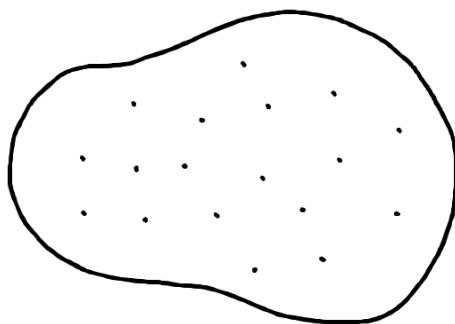
ceded the project. There aren't  
放弃了这个项目。没有

any tasks or scopes yet.  
任何任务或范围。



When the team members take over the project, they start discovering tasks. Tasks are a natural starting point because they're concrete and granular. It's too early to organize them into higher level categories. It would be artificial to try and group them arbitrarily. It's enough at the start just to capture a variety of things that need to happen.

当团队成员接管项目时，他们开始发现任务。任务是一个自然的起点，因为它们是具体的、细粒度的。把它们组织成更高层次的类别还为时过早。试图任意地将它们分组是人为的。一开始，只要捕捉到需要发生的各种事情就足够了。



But we don't want to stay with this picture for long. It's too low-level. There's nothing visible from high altitude.

但是我们不想在这张照片上停留太久，因为它太低了，从高空看不到任何东西。

As the team starts doing real work on the project they learn how the tasks are related and what the structure of the project is really like. Then they become able to factor the project into scopes. This is like dividing the map of the project into separate territories.

当团队开始在项目上进行真正的工作时，他们会了解这些任务是如何相关的，以及项目的结构到底是什么样的。然后他们就能够把项目分解成不同的范围。这就像把项目地图分割成不同的区域。

**C H A P T E R 1 2 - M a p T h e S c o p e s**

**1 1 5**

**C h a p t e r 1 2 - m a p t e e s s 1 1 5**

The scopes reflect the meaningful parts of the problem that can be completed independently and in a short period of time—a few days or less. They are bigger than tasks but much smaller than the overall project.

范围反映了问题中有意义的部分，这些部分可以独立地在短时间内(几天或更短时间内)完成。它们比任务大，但比整个项目小得多。

The map is a mental image. In practice, we define and track the scopes as to-do lists. Each scope corresponds to a list name. Then any tasks for that scope go in that list.

地图是一个心理图像。在实践中，我们定义和跟踪范围作为待办事项清单。每个作用域对应一个列表名称。然后该作用域的任何任务都会进入该列表。

The screenshot shows a web-based to-do list application with a header navigation bar including Home, Pings, Hey!, Activity, My Stuff, Find, and a search bar. Below the header is a breadcrumb trail: Replace Geocode with Location Summary. The main content area is titled "To-dos" with a count of 7/24. It displays four sections, each with a green circular icon and a title, followed by a list of tasks with checkboxes:

- Field on Form**: 0/3 completed
  - Design a location\_summary field below the existing location textarea
  - Add hint text that shows an example of what a good summary looks like
  - Make the summary a required field

Add a to-do
- Preset Locations**: 0/3 completed
  - Stop copying geocode fields
  - Copy location\_summary from center to event on create
  - Make sure location\_summary copies when updating an existing event

Add a to-do
- Display on List**: 0/3 completed
  - Determine if partial list uses same code as widget list
  - Send location\_summary as venue in the widget JS
  - Design a state for truncated summaries if they don't fit in the table column

Add a to-do
- Migrate Events**: 0/2 completed
  - Populate location\_summary field based on city and state geocode
  - Add new column on centers and events

Add a to-do

**116**

## The language of the project 项目的语言

Scopes are more than just slices. They become the language of the project at the macro level. When we were building the *Clients in Projects* feature, the team used the language of the scopes like this: “After *Bucket Access* is done we can implement *Invite Clients*. Then we’ll *Update Recording Visibility* when people on the firm flip the *Visibility Toggle*.”

范围不仅仅是切片。它们在宏观层面上成为项目的语言。当我们在 Projects 特性中构建 Clients 时，团队使用了这样的范围语言：“在 Bucket 访问完成之后，我们可以实现 Invite Clients。然后当公司里的人打开可见性开关时，我们会更新录制可见性。”

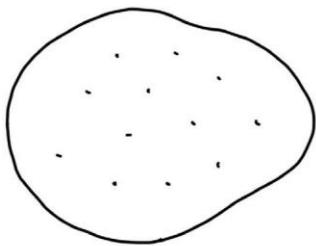
When it’s time to report status, the team uses the language of scopes to explain what’s done and what’s not done. It’s more satisfying to have the conversation at a high level and point to finished pieces of software, instead of going down into the weeds and defending the purposes and status of individual outstanding tasks. (We’ll see more in the next chapter about how to report on scopes using the Hill Chart.)

当报告状态的时候，团队使用作用域的语言来解释什么做了，什么没有做。更令人满意的是，在高层进行对话，指向已经完成的软件部分，而不是深陷其中，为个别未完成任务的目的和状态进行辩护。(我们将在下一章看到更多关于如何使用 Hill Chart 报告作用域的内容)

## Case study: Message drafts 案例研究：消息草稿

A designer and programmer were building a feature to create and save drafts of messages in a new app. After kick-off, they identified a bunch of tasks they would need to do at some point.

一个设计师和程序员正在构建一个功能，在一个新的应用程序中创建并保存消息的草稿。在开始之后，他们确定了一系列他们需要在某个时候完成的任务。



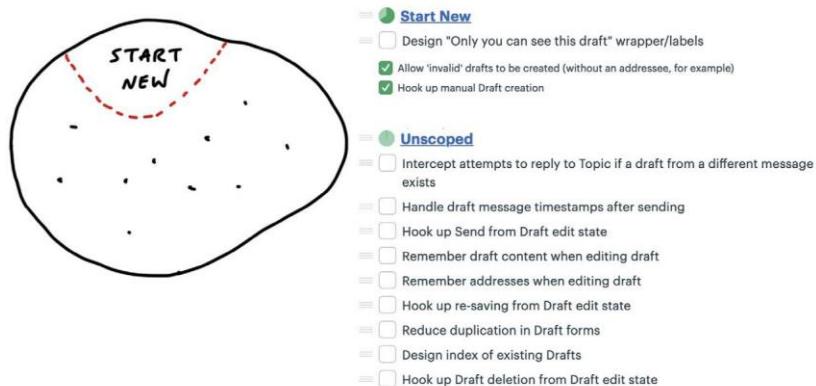
- = **Unscoped**
- =  Intercept attempts to reply to Topic if a draft from a different message exists
- =  Handle draft message timestamps after sending
- =  Hook up Send from Draft edit state
- =  Remember draft content when editing draft
- =  Remember addresses when editing draft
- =  Hook up re-saving from Draft edit state
- =  Reduce duplication in Draft forms
- =  Design index of existing Drafts
- =  Hook up Draft deletion from Draft edit state
- =  Design "Only you can see this draft" wrapper/labels
- Allow 'invalid' drafts to be created (without an addressee, for example)
- Hook up manual Draft creation

As the end of the first week approached, they had completed some of the tasks, but there wasn't anything to show for their work. In the spirit of "get one piece done" they focused on one key interaction they could integrate: creating a new draft.

随着第一周接近尾声，他们已经完成了一些任务，但是他们的工作没有什么可以展示的。本着“完成一件事”的精神，他们专注于一个他们可以整合的关键互动：创建一个新的草案。

They called the new scope "Start New," created a to-do list for it, and moved to-dos into it. There was only one design task left for them to consider this scope finished.

他们将新的作用域称为“Start New”，为其创建了一个待办事项列表，并将待办事项移入其中。他们只剩下一个设计任务来考虑这个范围的完成。



After finishing the one design task, the scope was complete.  
完成一个设计任务后，范围就完成了。



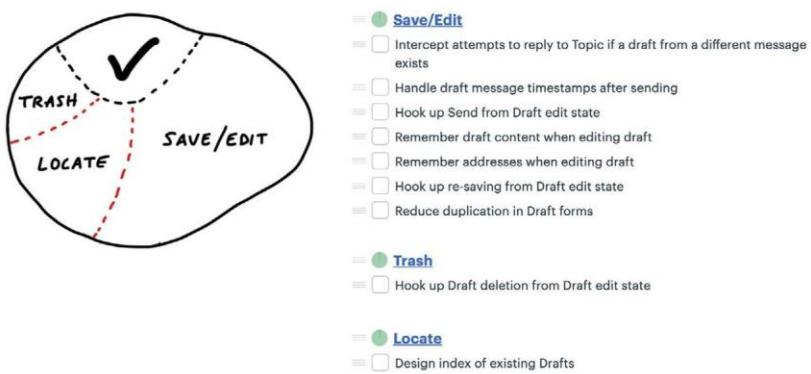
118  
118

The unscoped tasks that are left don't represent all the work that remains. More tasks are going to be discovered as they start working on each of those. Still, there is enough variety in the work to tease out more scopes. The team was motivated to break out the scopes already at this point because they knew they wanted their efforts to add up to another visible piece being finished before long.

剩下的未确定作用域的任务并不代表剩下的所有工作。当他们开始处理这些任务时，会有更多的任务被发现。尽管如此，这些工作还是有足够的多样性来梳理出更多的范围。这个团队在这一点上已经有了突破范围的动力，因为他们知道他们希望他们的努力加起来，不久就能完成另一个可见的部分。

Taking a look at the tasks that were left, they decided to pull out tasks related to finding the drafts into a new scope called Locate and the task for deleting into a scope called Trash. The work that was left all seemed related to saving and editing the draft, so they called that Save/Edit.

看看剩下的任务，他们决定将与查找草稿相关的任务拉出到一个名为 Locate 的新作用域中，并将用于删除的任务拉出到一个名为 Trash 的作用域中。剩下的工作似乎都与保存和编辑草稿有关，所以他们称之为“保存/编辑”。



Take a look at the Locate scope. There's only one task there right now. But surely there will be more work to do than just designing the index. When there are implementation tasks to do, that's where they'll go.

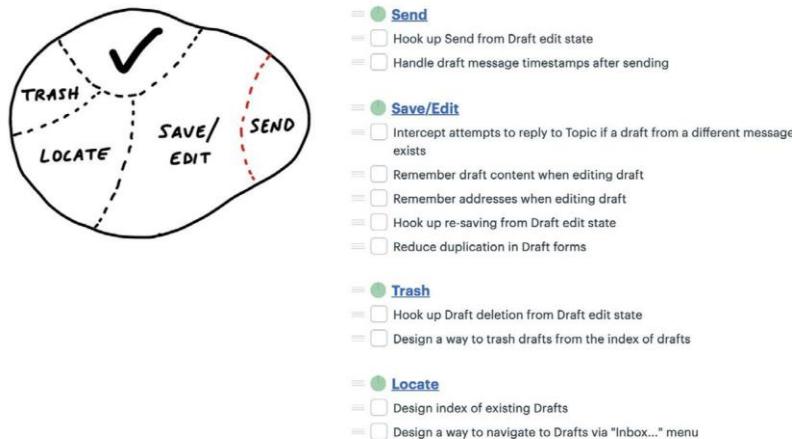
看看 Locate 作用域。现在只有一个任务。但是肯定还有比设计索引更多的工作要做。当有实施任务需要完成时，这就是他们将要去做的。

The designer started some work on Locate while the programmer focused on Save/Edit. As she dug into it, she noticed she could carve off a couple pieces to make more visible progress. There were really three scopes in it.

当程序员专注于 Save/Edit 时，设计师开始在 Locate 上做一些工作。当她深入研究的时候，她注意到她可以分割出一些部分来取得更明显的进展。里面真的有三个望远镜。

First she factored out the work related to sending the drafted message. She called that Send.

首先，她分解出与发送起草的信息有关的工作，她称之为发送。



Finally, some of the remaining Save/Edit tasks were about storing information and one other was actually unrelated—it was a special case for handling drafts when replying to another message. She broke these out into two new scopes: Store and Reply.

最后，剩下的一些 Save/Edit 任务是关于存储信息的，而另一些任务实际上是不相关的——这是在回复另一条消息时处理草稿的特殊情况。她把这些分成了两个新的作用域: Store 和 Reply。



**120**  
**12 万**

At this point the team suddenly felt like they could see the whole of the project at a high level. All the major parts were visible at the macro level as scopes. None of them were so big that important or challenging tasks could hide inside of them unnoticed.

在这一点上，团队突然觉得他们可以在一个高层次上看到整个项目。所有的主要部分在宏观层面上都是可见的。没有一个大到可以忽略其中的重要任务和挑战性任务的地步。

Meanwhile, the designer had made progress on Locate. After a little wiring, they were able to mark that done. Tasks were getting done on Send and Store as well.

与此同时，设计师已经在 Locate 上取得了进展。经过一段时间的布线，他们已经能够标记出这个结果。在 Send 和 Store 上也可以完成任务。



Once Send and Store were finished, just a couple tasks remained for Trash and Reply.

一旦 Send 和 Store 完成，只剩下几个任务留给 Trash 和 Reply。

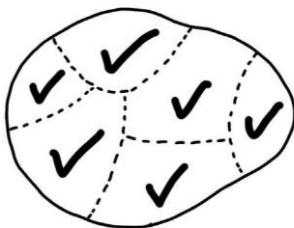


**C H A P T E R 1 2 - M a p T h e S c o p e s**

**1 2 1**

**Chapter12-mapteess121**

And then the project was done.  
然后项目就完成了。



## Discovering scopes 发现范围

Scope mapping isn't planning. You need to walk the territory before you can draw the map. Scopes properly drawn are not arbitrary groupings or categories for the sake of tidiness. They reflect the real ground truth of what can be done independently—the underlying interdependencies and relationships in the problem.

Scope mapping isn't planning 范围映射没有计划。在绘制地图之前，你需要走遍整个地区。正确绘制的 scope 不是为了整洁而随意分组或分类的。它们反映了可以独立完成的事情的真实基础——问题中潜在的相互依赖和关系。

Scopes arise from interdependencies. The way parts depend on each other determines when you can say a given piece of the work is “done.” You don't know what the work and interdependencies actually are in advance. We talked earlier about imagined versus discovered tasks. The same principle applies to scopes. The scopes need to be discovered by doing the real work and seeing how things connect and don't connect.

Scope 产生于相互依赖关系。部件之间相互依赖的方式决定了你什么时候可以说给定的一部分工作已经“完成”你不知道什么样的工作和相互依赖实际上是提前。我们之前讨论过想象的任务和发现的任务。同样的原理也适用于作用域。范围需要通过做实际工作来发现，看看事物是如何连接和不连接的。

That's why at the start of a project, we don't expect to see accurate scopes. We're more likely to see them at the end of week one or start of week two, after the team has had a chance to do some real work and find the natural dividing lines in the anatomy of the problem.

这就是为什么在项目开始的时候，我们不期望看到精确的范围。我们更有可能在第一周结束或第二周开始看到他们，在团队有机会做一些真正的工作并找到解剖问题的自然分界线之后。

It's also normal to see some shuffling and instability in the scopes at first. The lines get redrawn or scopes renamed as the team feels out where the boundaries really are, like in the example above. The一开始在范围内看到一些混乱和不稳定也是正常的。当团队摸清界限的真正位置时，界限会被重新划定或者重命名，就像上面的例子一样。他说

team was focused on specific problems of saving and editing drafts, so it was easiest to identify that scope early. It wasn't until they got into the weeds that they noticed there were tasks specifically about sending the draft and made that a separate scope.

团队专注于保存和编辑草稿的具体问题，因此最容易在早期确定范围。直到他们陷入困境，他们才注意到有一些特定的任务是关于发送草稿，并将其作为一个单独的范围。

## How to know if the scopes are right 如何知道范围是否正确

Well-made scopes show the anatomy of the project. When you feel a pain in your body, you don't have to question whether it's in your arms or your legs or your head. You know the parts and their names so you can explain where the pain is. In the same way, every project has a natural anatomy that arises from the design you want, the system you're working within, and the interdependencies of the problems you have to solve.

制作精良的望远镜显示了这个项目的结构。当你感到身体疼痛的时候，你不必怀疑它是在你的胳膊、腿还是头上。你知道疼痛的部位和它们的名字，这样你就可以解释疼痛在哪里。同样，每个项目都有一个自然的解剖结构，它来自于您想要的设计、您正在使用的系统以及您必须解决的问题的相互依赖性。

Three signs indicate when the scopes are right:  
三个迹象表明何时范围是正确的：

1. You feel like you can see the whole project and nothing important that worries you is hidden down in the details.

你觉得你可以看到整个项目，没有什么重要的，你担心是隐藏在细节。

2. Conversations about the project become more flowing because the scopes give you the right language.

关于项目的对话变得更加流畅，因为范围给了你正确的语言。

3. When new tasks come up, you know where to put them. The scopes act like buckets that you can easily lob new tasks into.

当出现新的任务时，你知道把它们放在哪里。作用域就像一个桶，你可以很容易地将新任务投入其中。

On the other hand, these three signs indicate the scopes should be redrawn:

另一方面，这三个标志表明作用域应该被重绘：

1. It's hard to say how "done" a scope is. This often happens when the tasks inside the scope are unrelated. If the problems inside the scope are unrelated, finishing one doesn't get you closer to finishing the other. It's good in this case to look for something you can factor out, like in the Drafts example.

很难说瞄准镜是怎么做到的。这种情况通常发生在范围内的任务不相关的时候。如果范围内的问题是无关的，完成一个并不能让你更接近完成另一个。在这种情况下，寻找一些你可以分解出来的东西是很好的，就像 draft 的例子一样。

2. The name isn't unique to the project, like "front-end" or "bugs."

We call these "grab bags" and "junk drawers." This suggests you aren't integrating enough, so you'll never get to mark a scope "done" independent of the rest. For example, with bugs, it's better to file them under a specific scope so you can know whether, for example, "Send" is done or if you need to fix a couple bugs first before putting it out of mind.

这个名字对于项目来说并不是唯一的，比如“前端”或者“bug”

我们称之为“抓斗袋”和“垃圾抽屉”这表明你没有完全集成，所以你永远不可能独立于其他部分来标记范围“完成”。例如，对于 bug，最好将它们归档在一个特定的范围内，这样您就可以知道是否已经完成了“Send”，或者您是否需要首先修复一些 bug，然后才能将其抛诸脑后。

3. It's too big to finish soon. If a scope gets too big, with too many tasks, it becomes like its own project with all the faults of a long master to-do list. Better to break it up into pieces that can be solved in less time, so there are victories along the way and boundaries between the problems to solve.

它太大了，不能很快完成。如果一个范围变得太大，任务太多，它就会变得像它自己的项目一样，有一个长长的主任务清单的所有缺陷。最好把它分解成可以在更短的时间内解决的部分，这样在解决问题的过程中就会有胜利和边界。

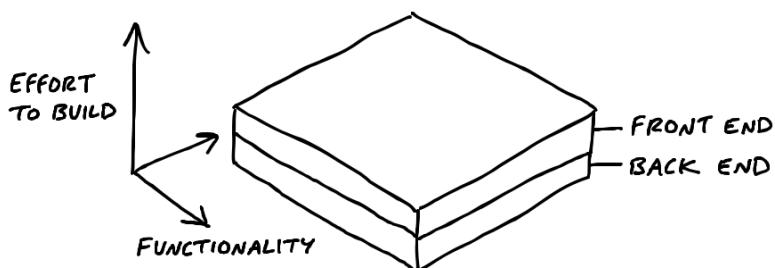
Let's close this chapter with a few tips for dealing with different kinds of tasks and scopes that will come up.

让我们用一些处理将出现的不同类型任务和范围的技巧来结束本章。

Layer cakes  
分层蛋糕

Most software projects require some UI design and a thin layer of code below. Think of a database app where all you need to do is enter information, save it, and display it back. Work like this looks like a layer cake: You can judge the work by UI surface area because the back-end work is thin and evenly distributed. In these cases, you can integrate all design and programmer tasks together in the same scope. This is a good default for most “information system” type apps.

大多数软件项目需要一些 UI 设计和下面的一层代码。想想一个数据库应用程序，你所需要做的就是输入信息，保存信息，然后再显示出来。像这样的工作看起来像一个分层蛋糕：你可以通过 UI 表面积来判断工作，因为后端的工作是薄而均匀分布的。在这种情况下，你可以将所有的设计任务和程序员任务集成在同一个范围内。对于大多数“信息系统”类型的应用程序来说，这是一个很好的默认设置。

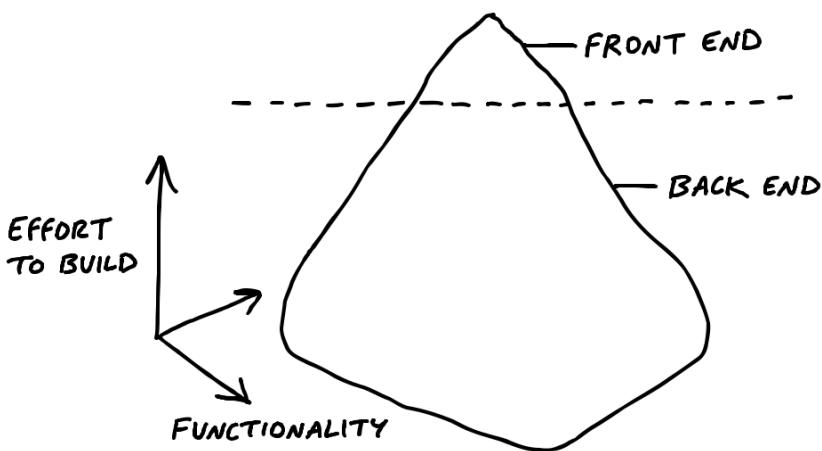


## Icebergs

### 冰山

But sometimes there is significantly more back-end work than UI work or vice versa. For example, a new feature that only requires submitting a form could require very complex business logic to return the right answer. This kind of work is like an iceberg.

但是有时候后端工作比用户界面工作要多得多，反之亦然。例如，一个只需要提交表单的新特性可能需要非常复杂的业务逻辑来返回正确的答案。这种工作就像一座冰山。



For icebergs, it can help to factor out the UI as a separate scope of work (assuming the UI isn't interdependent with the back-end complexity).

If the back-end is complex enough, you can split it into separate concerns and then turn those into scopes as well. The goal in cases like this is to define some different things you can finish and integrate in stages, rather than waiting until the 11th hour with fingers crossed that it will all come together.

对于冰山来说，将 UI 分解成一个独立的工作范围(假设 UI 与后端复杂性不相互依赖)可以帮助解决这个问题。如果后端足够复杂，你可以把它分解成不同的关注点，然后把它们也分解成不同的范

围。在这种情况下，我们的目标是定义一些不同的事情，你可以分阶段完成和整合，而不是等到最后一刻才祈祷一切都会好起来。

You also sometimes see upside-down icebergs, where there is a ton of UI complexity with less back-end complexity. For example, the data model for a calendar isn't complicated, but the interaction for rendering a multiple-day event and wrapping across grid cells could take a lot of time and problem-solving.

你有时也会看到上下颠倒的冰山，那里有大量的用户界面复杂性，而后端的复杂性要少得多。例如，日历的数据模型并不复杂，但是呈现多日事件和包装网格单元的交互可能需要花费大量的时间和解决问题。

For both back-end and front-end icebergs, we always question them before accepting them as a fact. Is the complexity really necessary and irreducible? Do we need that fancy UI? Is there a different way to build that back-end process so it has fewer interdependencies with the rest of the system?

对于后端和前端的冰山，我们总是在接受它们作为事实之前对它们提出质疑。复杂性真的是必要的和不可减少的吗？我们需要那个花哨的 UI 吗？有没有一种不同的方法来构建这个后端过程，以减少它与系统其他部分的相互依赖性？

## Chowder

### 巧达

There are almost always a couple things that don't fit into a scope. We allow ourselves a “Chowder” list for loose tasks that don't fit anywhere. But we always keep a skeptical eye on it. If it gets longer than three to five items, something is fishy and there's probably a scope to be drawn somewhere.

几乎总有一些东西不适合放在一个范围内。我们允许自己列出一个“巧达”清单来处理那些不适合任何地方的松散任务。但是我们总是对它保持怀疑的眼光。如果它超过三到五个项目，一定有什么可疑的地方，可能有一个范围可以画出来。

## Mark nice-to-haves with ~

### 马克与... ... 相处愉快

New tasks constantly come up as you get deeper into a problem. You'll find code that could be cleaned up, edge cases to address, and improvements to existing functionality. A good way to deal with all those improvements is to record them as tasks on the scope but mark them with a ~ in front. This allows everyone on the team to constantly sort out the must-haves from the nice-to-haves.

当你陷入一个更深的问题时，新的任务不断出现。你会发现可以清理的代码，需要处理的边缘案例，以及对现有功能的改进。处

理所有这些改进的一个好方法是将它们记录为范围内的任务，但在前面标记一个~。这使得团队中的每个人都可以不断地从好的东西中分辨出必须拥有的东西。

In a world with no deadlines, we could improve everything forever. But in a fixed time box, we need a machete in our hands to cut down the constantly growing scope. The ~ at the start of an item, or even a whole scope, is our best tool for that. We'll come back to this technique when we talk about making cuts to scope in Chapter 14, Decide When to Stop.  
在一个没有最后期限的世界里，我们可以永远改善一切。但是在  
一个固定的时间框里，我们需要一把大砍刀来缩小不断增长的范围。  
在一个项目的开始，甚至是整个范围，是我们最好的工具。当我们在第 14 章《决定何时停止》中讨论削减范围时，我们会回到这个技巧。



## Show Progress 显示进步

Good-hearted managers don't like asking for status. It's awkward, feels like nagging, and gets even worse when they have to ask follow-up questions to get sufficiently clear about what's going on.

善良的管理者不喜欢要求地位。这很尴尬，感觉就像唠叨，当他们不得不问后续的问题来充分弄清楚到底发生了什么事情时，情况会变得更糟。

Managers would rather be able to see the status themselves when-ever they need to. We saw in the last chapter how organizing to-dos into scopes helps the team to stay on top of the work. But this doesn't help the manager directly. There are a couple problems with to-dos that make them insufficient for judging status.

管理者宁愿能够在需要的时候看到自己的状态。我们在上一章中看到了如何将待办事项组织到范围内来帮助团队保持在工作的顶端。但这并不能直接帮助管理者。在待办事项方面存在一些问题，使得它们不足以判断状态。

The tasks that aren't there  
不存在的任务

Consider a list with a few completed items and no incomplete items left.

This could mean that all the work is done. But it could also mean that the team knows there's more work but hasn't defined tasks yet.

考虑一个列表，其中有几个已完成的项目，没有不完成的项目留下。这可能意味着所有的工作都完成了。但这也可能意味着团队知道还有更多的工作，但是还没有确定任务。

Sometimes a team will define a scope early in the project without populating it with tasks. It marks that some work needs to be done but that actual tasks haven't been discovered yet.

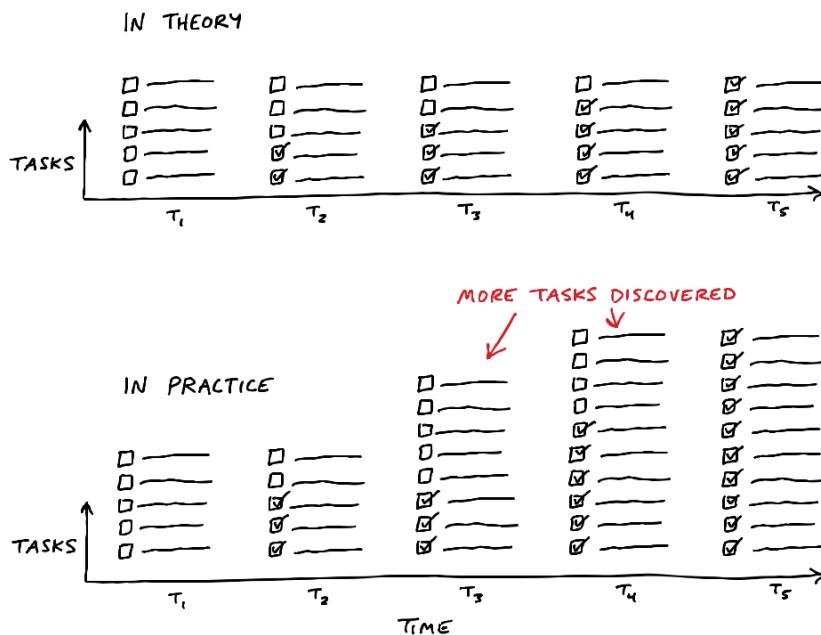
有时候，一个团队会在项目的早期定义一个范围，而不是填充任务。它标志着一些工作需要完成，但是实际的任务还没有被发现。

Or think about doing some QA at the end of a scope. All the tasks are done. There's nothing else to do. Then the act of testing populates the scope with new tasks for the issues found.

或者考虑在范围结束时做一些 QA。所有的任务都完成了。没有别的事可做了。然后，测试 popu 的行为延迟了范围，为发现的问题提供了新任务。

This goes back to the notion of imagined versus discovered tasks. In our naive notion of a list that's planned up-front, somebody populates it with items that are gradually checked off. In real life, issues are discovered by getting involved in the problem. That means *to-do lists actually grow as the team makes progress*.

这可以追溯到想象任务和发现任务的概念。在我们天真的预先计划好的列表中，有些人用逐渐勾掉的项目来填充它。在现实生活中，问题是通过参与到问题中来发现的。这意味着待办事项清单实际上会随着团队的进展而增长。



If we tried to judge at  $t_2$  how far along the project is, we'd be misled. From an outsider's perspective, there's no way to know whether the number of outstanding tasks will go down or up. To know that, you'd need more context on the work inside the scope to understand what it means that those particular tasks are done and whether others might still be coming.

如果我们试图在  $t_2$  时判断项目进展到什么程度，我们就会被误导。

从局外人的角度来看，我们没有办法知道未完成任务的数量是会减少还是增加。要了解这一点，您需要了解范围内工作的更多上下文，以理解这些特定任务已经完成意味着什么，以及其他任务是否可能仍将完成。

## Estimates don't show uncertainty 估计不会显示不确定性

Some teams try to attach estimates to their tasks or scopes to report status. The problem with estimates is they have a very different meaning depending on the nature of the work being estimated.

一些团队试图将估计附加到他们的任务或范围中以报告状态。评估的问题在于，根据被评估工作的性质，它们有着非常不同的含义。

Say you have two tasks, both estimated to take four hours. If one task is something the team has done ten times in the past, you can be confident in the estimate. Suppose the other task is something the team has never done before, or it has unclear interdependen-cies. It could take the four hours if all goes perfectly, but due to the unknowns in it, it could stretch out to two to three days. It's not meaningful to write “4 hours, or maybe 3 days” as the estimate.

假设你有两个任务，估计都需要 4 个小时。如果一个任务是团队在过去已经完成了十次的事情，你可以对这个估计很有信心。假设另一个任务是团队以前从未完成过的，或者它们之间的相互依赖关系不明确。如果一切顺利的话，可能需要四个小时，但是由于其中的未知因素，可能需要两到三天的时间。写“4 小时，或者也许 3 天”作为估计是没有意义的。

Recognizing this, we came up with a way to see the status of the project without counting tasks and without numerical estimates. We do that by shifting the focus from what's done or not done to what's unknown and what's solved. To enable this shift, we use the metaphor of the hill.

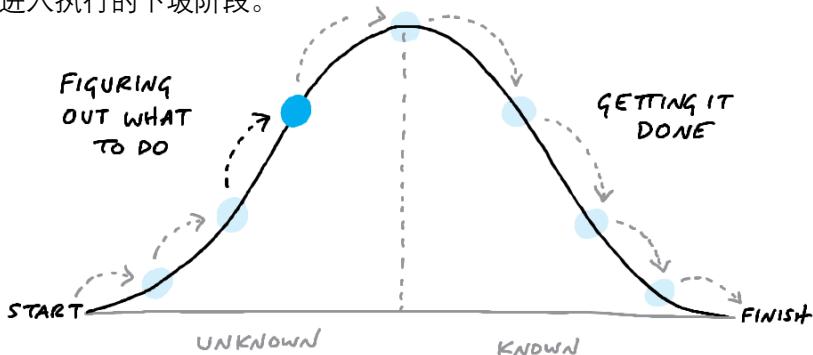
认识到这一点，我们想出了一种方法来查看项目的状态，而无需计算任务和数字估计。我们通过将焦点从已经完成或没有完成的转移到未知的和已经解决的。为了实现这种转变，我们使用了山的比喻。

Work is like a hill

工作就像一座山

Every piece of work has two phases. First there's the uphill phase of figuring out what our approach is and what we're going to do. Then, once we can see all the work involved, there's the downhill phase of execution. --

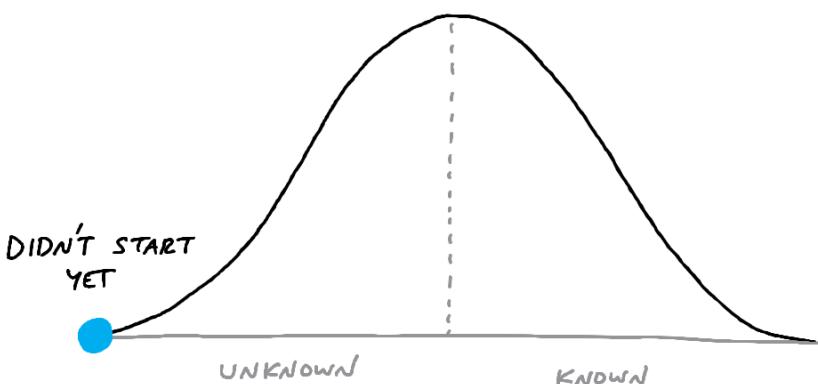
每件作品都有两个阶段。第一个阶段是上坡，弄清楚我们的方法是什么，我们要做什么。然后，一旦我们看到所有涉及的工作，就会进入执行的下坡阶段。



Let's use an everyday example to get the feeling of the hill.  
让我们用一个日常的例子来感受一下这座山。

Suppose you're planning to host a dinner party. You've set the date, but it's still a couple weeks ahead and you haven't thought about what to cook yet. You have no idea what type of cuisine the meal will be or what dish to make. That would place you at the start of the hill on the bottom-left.

假设你计划举办一个晚宴。你已经确定了日期，但是还要提前几个星期，而且你还没有想好做什么菜。你不知道这顿饭会是什么样的菜，也不知道做什么菜。这样的话，你就会在左下角的小山的起点了。

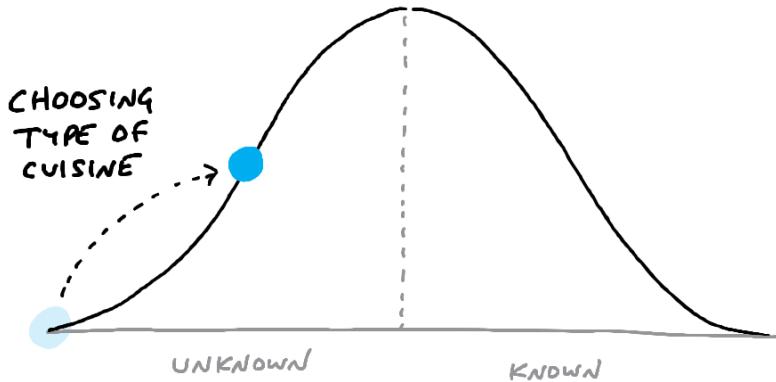


Next you think about who's attending and note that a couple people are vegetarian. That eliminates some options (like grilling out) but still leaves a lot of options open. You consider both Italian and Indian. You think Indian might be more fun to cook, with more interesting vegetarian options. So you decide to look for Indian recipes.

接下来你想想谁会出席，并注意到一些人是素食主义者。这样就排除了一些选择(比如烧烤)，但是仍然有很多选择是开放的。你可以考虑意大利和印度菜。你认为印度菜做起来更有趣，有更多有趣的素食选择。所以你决定寻找印度菜谱。

At this point, the question “What percent complete is the project?” doesn’t even make sense. And if someone asked you to estimate how long the shopping and prep will take, you couldn’t answer that either because you haven’t chosen a dish yet. The answer would be: “I’ve done some work to figure out what kind of cuisine, but I haven’t narrowed it down to a specific dish yet.” We can represent that by putting you halfway up the “figuring it out” side of the hill.

现在，问题是“这个项目完成的百分比是多少?”完全没有意义。如果有人让你估计购物和准备需要多长时间，你也不能回答这个问题，因为你还没有选好菜。答案可能是：“我已经做了一些工作来弄清楚什么样的菜肴，但我还没有把范围缩小到特定的菜肴。”我们可以把你放在半山腰的“弄明白”一侧来表达这一点。

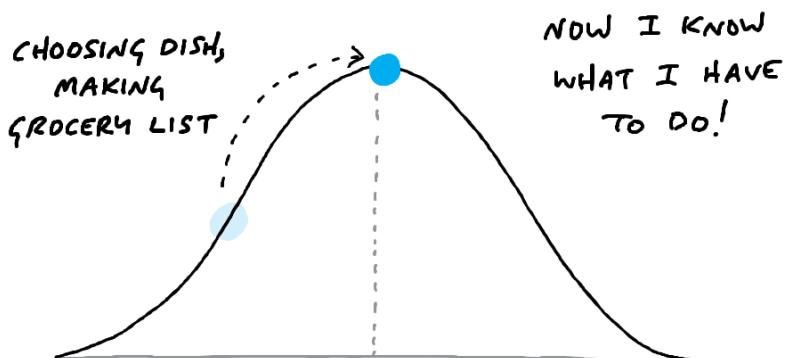


Next you do some searching online and look through your recipe books. You want to find a recipe that will be interesting but doesn't require ingredients that will be too hard to find. You settle on a recipe and prepare a shopping list.

接下来你可以在网上搜索一下，看看你的菜谱。你想找到一个有趣的食谱，但是不需要太难找到的原料。你确定了一个食谱，并准备了一个购物清单。

Now you are in a very different position than before. The feeling changes from “I'm still not sure what I'm doing” to “Now I know what to do.” You're at the top of the hill.

现在你的处境和以前完全不同了。这种感觉从“我还是不确定我在做什么”变成了“现在我知道该做什么了”你在山顶。

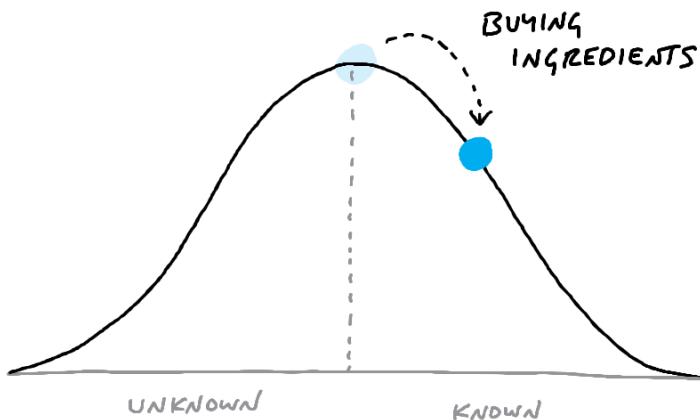


CHAPTER 13 - Show Progress      131  
Chapter13-shopress131

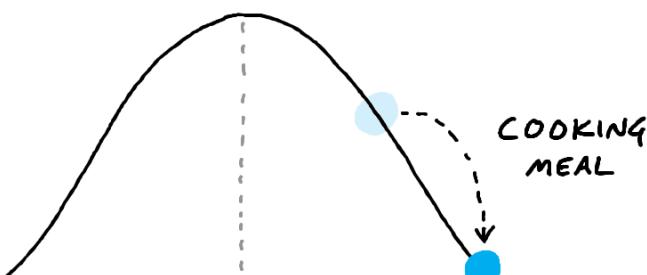
From this vantage point, you can see all of the steps that are left. It's even fair to estimate how long all the work will take ("Let's see...an hour to grocery shop, 30 minutes of prep, cook for 45 minutes...").  
从这个有利位置，您可以看到剩下的所有步骤。我们甚至可以估计所有的工作需要多长时间(“让我们看看……一个小时到杂货店，30分钟的准备，45分钟的烹饪……”。

The day before the dinner party, you go to the grocery store and buy the ingredients. This moves you downhill. You're closer to finishing the task.

在晚宴的前一天，你去杂货店买食材。这会让你走下坡路。你离完成任务更近了。

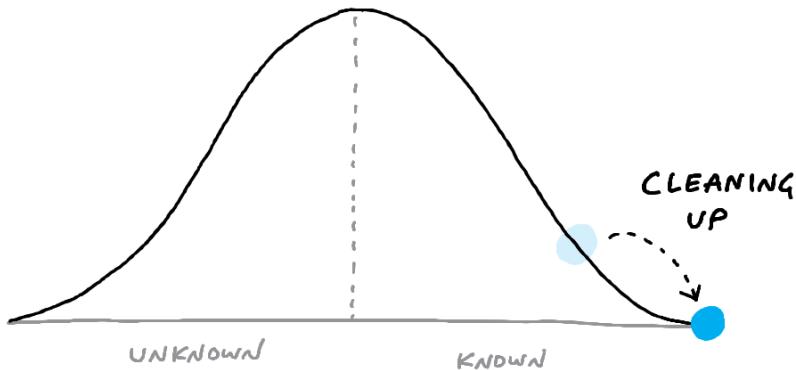


Next comes the work of prepping and cooking the meal.  
接下来是准备和烹饪饭菜的工作。



132  
132

After the meal is over, there's just a little work left: the clean-up.  
用餐结束后，只剩下一点工作：清理房间。



Note how the hill shows how the work *feels* at different stages. The uphill phase is full of uncertainty, unknowns, and problem solving. The downhill phase is marked by certainty, confidence, seeing everything, and knowing what to do.

请注意这座小山在不同阶段是如何显示工作的感觉的。上坡阶段充满了不确定性、未知性和解决问题的能力。下坡阶段的标志是确定性，自信，看到一切，知道该做什么。

## Scopes on the hill 山上有望远镜

We can combine the hill with the concept of scopes from the last chapter. The scopes give us the language for the project (“Locate,” “Reply”) and the hill describes the status of each scope (“uphill,” “downhill”).

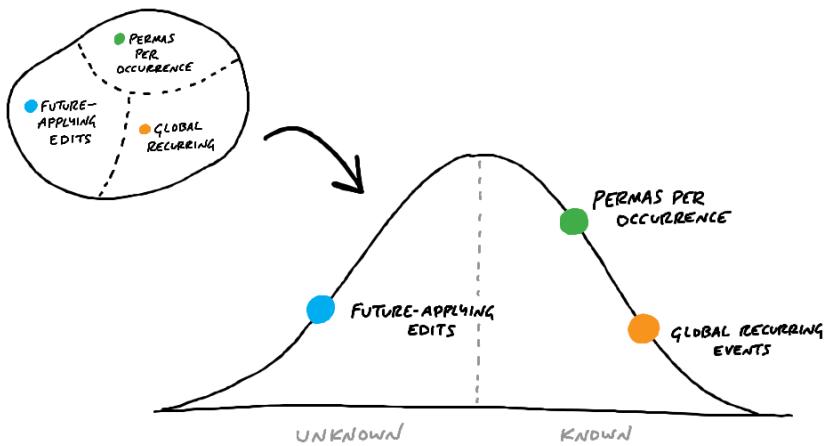
我们可以把山和上一章的瞄准镜概念结合起来。范围为我们提供了项目的语言(“Locate”, “Reply”), 而山则描述了每个范围的状态(“上坡”, “下坡”)。

To see the status of the scopes, we can plot each one as a different color on the hill.

为了查看瞄准镜的状态，我们可以把每个瞄准镜在山上标出不同的颜色。

This is a snapshot from a project to implement recurring events in Basecamp. Here “Future-applying edits” is a scope that is still being worked out, with significant unknowns to solve. The other two scopes have no meaningful unknowns left, and “Global recurring events” is closer to finished.

这是一个在 Basecamp 中实现循环事件的项目的快照。这里的“未来应用编辑”是一个仍在制定的范围，有重大的未知数要解决。另外两个范围已经没有有意义的未知数了，“全局循环事件”也接近完成。



## Status without asking 未经询问的状态

We built a feature exclusive to Basecamp for creating hill charts and updating them with a few clicks. The team members, who have the full context of where the work stands, intuitively drag the scopes into position, and save a new update that's logged on the project (see How to Implement Shape Up in Basecamp).

我们构建了一个 Basecamp 独有的功能，用于创建山形图，并通过几次点击来更新它们。团队成员，他们有工作所处位置的完整上下文，直观地将范围拖到合适的位置，并保存项目上登录的新更新（参见如何在 Basecamp 中实现 Shape Up）。

+ New list
To-dos
View as... ▾

Cancel
Drag each dot to adjust its position on the chart
Save this update

Permas per occurrence

Future-applying edits

Global recurring events

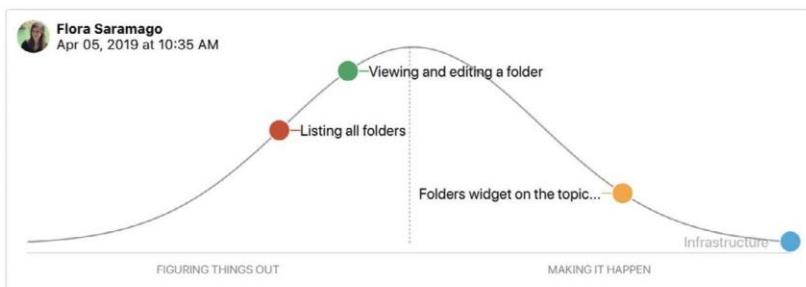
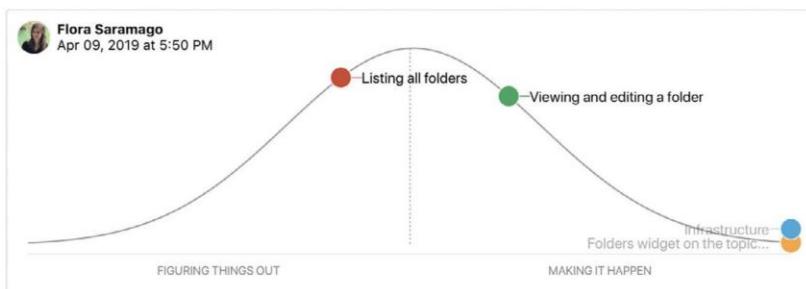
FIGURING THINGS OUT

MAKING IT HAPPEN

1 3 4  
1、3、4

For managers, the ability to compare past states is the killer feature. It shows not only where the work stands but how the work is *moving*.

对于管理者来说，比较过去状态的能力是他们的杀手锏。它不仅显示了工作的状况，还显示了工作是如何进行的。



With this second-order view, managers can judge what's in motion and what's stuck. They can see which problems the team chose to solve and how much time they spent at each stage from unknown to known to done.

通过这种二阶视角，管理者可以判断什么在运转，什么在停滞。他们可以看到团队选择解决哪些问题，以及他们在每个阶段花费了多少时间，从未知到已知要做的事情。

This report becomes the manager's first destination when they feel anxious about a project. Since it's self-serve, there's no need to interrupt the team with the awkward status question. And in cases where something doesn't look right, the manager can jump directly into a conversation about the relevant piece of work. "Looks like 'Autosave' has been uphill for a while. What's the unknown that's holding it back?" The manager can workshop this specific piece of the project without having to first untangle it from all the other things that are moving along as expected.

当经理们对一个项目感到焦虑时，这份报告就成了他们的第一个目的地。因为它是自助式的，所以没有必要用尴尬的状态问题来打断团队。在某些事情看起来不对劲的情况下，经理可以直接进入关于相关工作的对话。他说：“看起来‘自动保存’已经上坡有一段时间了。是什么原因阻碍了它的发展呢？”管理者可以研究项目的这个特定部分，而不必首先将它与其他所有正在按预期进行的事情分开。

## Nobody says "I don't know" 没人会说”我不知道”

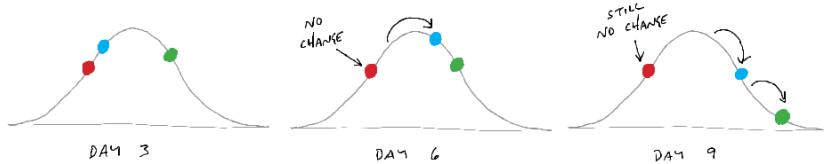
Nobody wants to raise their hand to management and say "I don't know how to solve this problem." This causes teams to hide uncertainty and accumulate risk. The moments when somebody is stuck or going in circles are where the biggest risks and opportunities lie. If we catch those moments early, we can address them with help from someone senior or by reworking the concept. If we don't catch them, the unsolved problems could linger so far into the cycle that they endanger the project.

没有人愿意向管理层举手说“我不知道如何解决这个问题”这会导致团队隐藏不确定性，积累风险。当一个人陷入困境或者在原地打转的时候，才是最大的风险和机会所在。如果我们能及早发现这些时刻，我们就可以通过高层人士的帮助或者改变观念来解决

它们。如果我们没有抓住他们，那些未解决的问题可能会在这个循环中持续太久，以至于危及到项目。

The hill chart allows everybody to see that somebody might be stuck without them actually saying it. A dot that doesn't move is effectively a raised hand: “Something might be wrong here.”

山形图可以让每个人看到，有些人可能卡住了，而他们实际上并没有说出来。一个不动的点实际上就是一只举起的手：“这里可能出了问题。”



Once it's been spotted, the language of uphill/downhill facilitates the conversation. It's less about the person (Looks like you're stuck!) and more about the work. The question is: What can we solve to get that over the hill?

一旦它被发现，上坡/下坡的语言便利了谈话。这不是关于人的(看起来你卡住了)更多的是关于工作。问题是：我们能解决什么问题来让这个问题过去呢？

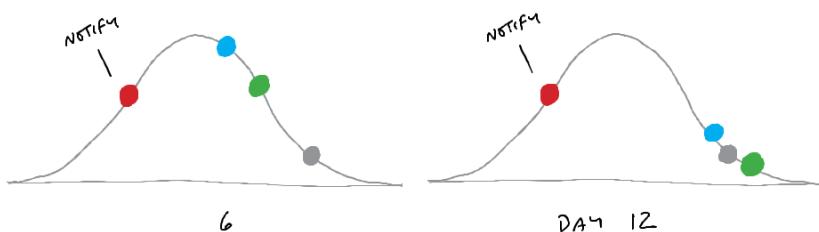
## Prompts to refactor the scopes 提示重构范围

Sometimes probing into a stuck scope reveals that it isn't stuck at all. The problem is in how the lines of the scope were drawn.

有时候，探究一个卡住的范围会发现它根本没有卡住。问题在于范围的界限是如何划定的。

Here's a case where the "Notify" scope was stuck on the hill for too long.

这里有一个例子，“通知”范围被卡在山上太久了。



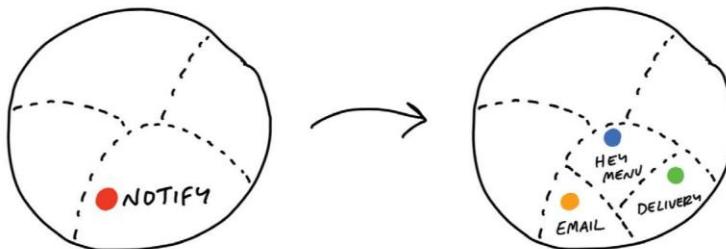
When we checked in with the team, it turned out the work was moving along just fine. The problem was that "Notify" wasn't a single thing. It had three different parts: designing an email, de-delivering the email in the back-end, and displaying the notification in an in-app menu. The team mostly finished the code for delivering the email. The design of the email was nearly figured out. But they hadn't started on the in-app

display. It wasn't possible to say whether "Notify" as a whole was over the hill or not because parts of it were and parts of it weren't.

当我们向团队汇报时，发现工作进展得很顺利。问题是“通知”并不是一件单独的事情。它有三个不同的部分：设计电子邮件，在后端分发电子邮件，以及在应用程序内菜单中显示通知。团队大部分完成了发送邮件的代码。电子邮件的设计也差不多完成了。但是他们还没有开始在应用程序内部显示。我们不可能说“Notify”作为一个整体是否已经过时，因为它的一部分已经过时，而另一部分还没有过时。

The solution in a case like this is to break the scope apart into smaller scopes that can move independently.

在这种情况下，解决方案是将范围拆分为可以独立移动的较小范围。



● **Notify**

- =  Disable read receipts on the delivery service
- =  Use the batch delivery service
- =  Design icon for the Hey notification
- =  Figure out how to title it so it matches the Hey format
- =  Optimize the preview image for email
- =  Push the notification to the Hey menu
- =  Adjust email design so there's room for the document preview
- =  Write copy for the email

● **Hey Menu Notification**

- =  Figure out how to title it so it matches the Hey format
- =  Design icon for the Hey notification
- =  Push the notification to the Hey menu

● **Email Notification**

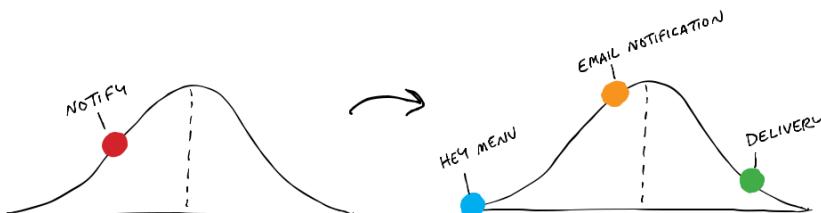
- =  Optimize the preview image for email
- =  Adjust email design so there's room for the document preview
- Write copy for the email

● **Notification Delivery**

- =  Disable read receipts on the delivery service
- Use the batch delivery service

Now the team can move each dot to accurately show where the work stands.

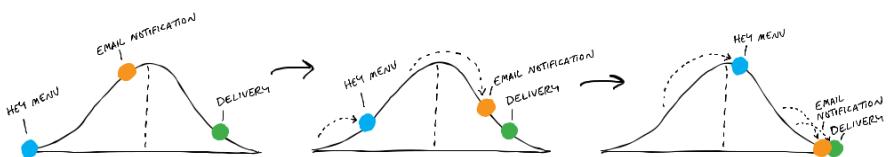
现在，研究小组可以移动每个点来准确地显示工作的位置。



138  
138

The benefit comes at the second order. With the scopes separated out, they can move independently over time. Now the team can show more progress more frequently than before.

好处来自于第二个阶段。随着范围的分离，它们可以随着时间的推移独立移动。现在，团队可以比以前更频繁地显示更多的进展。



## Build your way uphill 建设你的方式上坡

Some teams struggle with backsliding when they first try the hill chart. They consider a scope solved, move it to the top of the hill, and later have to slide it back when they uncover an unexpected unknown.

一些团队在第一次尝试山形图的时候，就在与倒退做斗争。他们认为一个范围解决了，把它移到山顶，然后当他们发现一个意想不到的未知时，不得不把它滑回来。

When this happens, it's often because somebody did the uphill work with their head instead of their hands. Coming up with an approach in your head is just the first step uphill. We often have a theory of how we'll solve something—"I'll just use that API"—and then the reality turns out to be more complicated. It's good to think of the first third uphill as "I've thought about this," the second third as "I've validated my approach," and the final third to the top as "I'm far enough with what I've built that I don't believe there are other unknowns."

当这种情况发生时，通常是因为有人在上坡时用头而不是用手。在头脑中想出一个方法只是上坡的第一步。我们经常有一个关于如何解决问题的理论——“我就用那个 API”——然后事实就变得更

加复杂了。第一个三分之一的上坡是“我考虑过这个问题”，第二个三分之一是“我验证了我的方法”，最后一个三分之一是“我已经做得足够远了，我不相信还有其他的未知数。”

### Solve in the right sequence

#### 按正确的顺序解决问题

In addition to seeing where the work stands, we can use the hill chart to sequence the work—which problems to solve in which order.

除了了解工作的进展情况，我们还可以使用山形图对工作进行排序——哪些问题需要按照哪种顺序解决。

Some scopes are riskier than others. Imagine two scopes: One involves geocoding data—something the team has never done before. The other is designing and implementing an email notification. Both have unknowns. Both start at the bottom of the hill. This is where the team asks themselves: If we were out of time at the end of the cycle, which of these could we easily whip together—despite the unknowns—and which might prove to be harder than we think?

有些范围的风险比其他的更大。想象一下两个范围:一个涉及地理编码数据——团队以前从未做过的事情。另一个是设计和实现电子邮件通知。两者都有未知数。都是从山脚下开始的。这就是团队问自己的问题:如果我们在周期结束时没有时间了,哪些问题可以很容易地解决(尽管存在未知因素),哪些问题可能比我们想象的更难解决?

That motivates the team to push the scariest work uphill first. Once they get uphill, they'll leave it there and look for anything critically important before finishing the downhill work to completion. It's better to get a few critical scopes over the top early in the project and leave the screw-tightening for later.

这促使团队首先把最可怕的工作推上山。一旦他们上了坡,他们就会把工作留在那里,在完成下坡工作之前寻找任何至关重要的东西。最好在项目的早期就把一些关键的范围扩大到顶部,把螺丝拧紧的工作留到以后再做。

Work expands to fill the time available. If the team starts with the email template first, they could easily spend weeks iterating on copy or creating the ultimate best-ever email design. But they don't *need* to do that. There's some version of an email template that could be worked out in a day during the final week and it would be sufficient. The geocoder, on the other hand, might present novel problems that the team could struggle with for weeks. They don't want that surprise to come at the end of the cycle.

工作扩展到填满可用的时间。如果团队先从电子邮件模板开始，他们可以很容易地花费数周的时间迭代复制或创建最终最好的电子邮件设计。但是他们不需要这样做。有一些版本的电子邮件模板，可以在最后一周的一天内完成，这就足够了。另一方面，地理编码器可能会提出一些新问题，团队可能会为此纠结数周。他们不希望在周期结束时出现这样的意外。

Journalists have a concept called the “inverted pyramid.” The idea is their articles start with the most essential information at the top, then they add details and background information in decreasing order of importance. This allows print newspaper designers to get the crucial part of the story on the front page and cut the end as needed without losing anything essential.

记者们有一个叫做“倒金字塔结构”的概念这个概念是他们的文章从最基本的信息开始，然后他们按照重要性的递减顺序添加细节和背景信息。这样印刷报纸的设计者就可以在头版上找到文章的关键部分，并根据需要在不丢失任何重要内容的情况下把结尾部分剪掉。

Effective teams sequence their problem solving in the same way.

They choose the most important problems first with the most

有效的团队以同样的方式排列他们解决问题的顺序。他们首先

选择最重要的问题

unknowns, get them to the top of the hill, and leave the things that are the most routine or least worrisome for last.

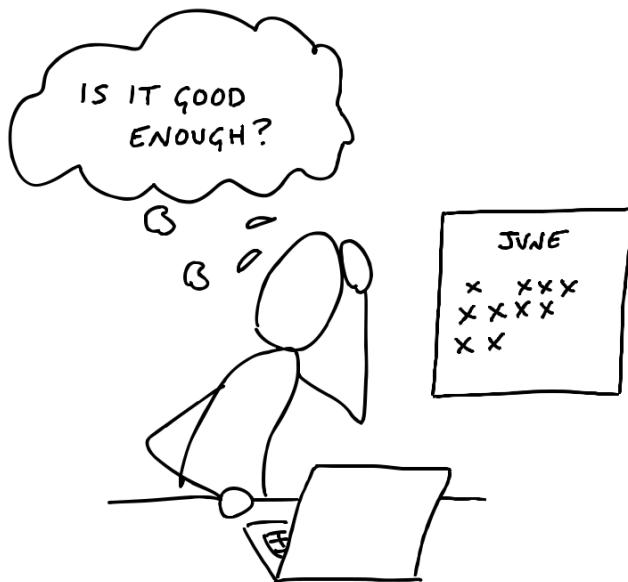
不知道的人，把他们带到山顶上，把那些最平常或者最不让人担心的事情放在最后。

As the end of the cycle approaches, teams should have finished the important things and left a variety of “nice to haves” and “maybes” lingering around. That brings us to the next chapter, on deciding when to stop.

随着周期接近尾声，团队应该已经完成了重要的事情，并留下各种各样的“好拥有”和“也许”徘徊在周围。这就把我们带到了下一章，决定什么时候停止。

CHAPTER 13 - Show Progress  
Chapter13-showpres141

141



## Decide When to Stop 决定何时停止

When the end of the cycle approaches, the techniques we covered so far will put the team in a good position to finish and ship. The shaped work gave them guard rails to prevent them from wandering. They integrated one scope at a time so there isn't half-finished work lying around. And all the most important problems have been solved because they prioritized those unknowns first when they sequenced the work.

当这个周期接近尾声的时候，我们到目前为止所涉及的技术将使团队处于完成和发布的有利位置。成形的工作给他们提供了护栏，以防止他们走神。他们一次集成一个望远镜，这样就不会有半成品在周围闲置。而且所有最重要的问题都已经解决了，因为他们对工作进行排序时，会优先考虑那些未知因素。

Still, there's always more work than time. Shipping on time means shipping something imperfect. There's always some queasiness in the

stomach as you look at your work and ask yourself: Is it good enough?

Is this ready to release?

然而，工作总是比时间多。准时运送意味着运送不完美的东西。

当你看着自己的工作并问自己：它够好吗？准备好释放了吗？

## Compare to baseline 与基线比较

Designers and programmers always want to do their best work. It doesn't matter if the button is on the center of the landing page or two pages down a settings screen, the designer will give it their best attention. And the best programmers want the code base to feel like a cohesive whole, completely logically consistent with every edge case covered.

设计师和程序员总是想做到最好。无论按钮是在着陆页面的中心还是设置页面下面的两个页面，设计师都会给予它最好的关注。最好的程序员希望代码库看起来像一个有凝聚力的整体，完全符合每个边界情况的逻辑。

Pride in the work is important for quality and morale, but we need to direct it at the right target. If we aim for an ideal perfect design, we'll never get there. At the same time, we don't want to lower our standards. How do we make the call to say what we have is good enough and move on?

工作中的自豪感对质量和士气很重要，但是我们需要把它引向正确的目标。如果我们的目标是一个理想的完美的设计，我们永远不会实现。与此同时，我们不想降低我们的标准。我们如何做出决定，说我们所拥有的已经足够好，然后继续前进？

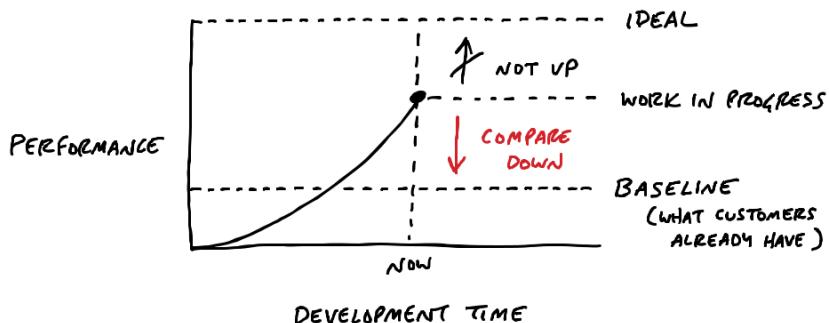
It helps to shift the point of comparison. Instead of comparing up against the ideal, compare down to baseline—the current reality for customers. How do customers solve this problem today, without this feature? What's the frustrating workaround that this feature eliminates? How much longer should customers put up with some-thing that doesn't work or wait for a solution because we aren't sure if design A might be better than design B?

这有助于改变比较的观点。与其与理想相比较，不如将比较降低到基线——客户当前的现实。如果没有这个功能，客户今天如何解决

这个问题？这个特性消除了哪些令人沮丧的解决方案？因为我们不确定设计 a 是否比设计 b 更好，客户还要忍受一些不起作用的东西或者等待一个解决方案多久？

Seeing that our work so far is better than the current alternatives makes us feel better about the progress we've made. This motivates us to make calls on the things that are slowing us down. It's less about us and more about value for the customer. It's the difference between "never good enough" and "better than what they have now." We can say "Okay, this isn't perfect, but it definitely works and customers will feel like this is a big improvement for them."

看到我们的工作到目前为止比当前的替代方案更好，我们对我们已经取得的进步感觉更好。这激励我们去做那些拖慢我们的事情。与其说是关于我们，不如说是关于顾客的价值。这是“永远不够好”和“比他们现在拥有的更好”之间的区别我们可以说：“好吧，这不是完美的，但它确实有效，客户会觉得这对他们来说是一个巨大的进步。”



*Make scope cuts by comparing down to baseline*

通过比较下降到基线来缩小范围

*instead of up to some perfect ideal*

而不是达到某种完美的理想

## Limits motivate trade-offs

### 限制激发权衡

Recall that the six-week bet has a circuit breaker—if the work doesn’t get done, the project doesn’t happen.

回想一下，为期六周的赌注有一个断路器——如果工作没有完成，项目就不会发生。

This forces the team to make trade-offs. When somebody says “wouldn’t it be better if...” or finds another edge case, they should first ask themselves: Is there time for this? Without a deadline, they could easily delay the project for changes that don’t actually deserve the extra time.

这迫使团队做出取舍。当有人说“如果……会不会更好?”或者发现另一个边缘情况时，他们应该首先问问自己：还有时间做这个吗？如果没有最后期限，他们很容易因为一些不值得额外时间的改变而推迟项目。

We expect our teams to actively make trade-offs and question the scope instead of cramming and pushing to finish tasks. We create our own work for ourselves. We should question any new work that comes up before we accept it as necessary.

我们希望我们的团队能够积极地做出权衡，质疑范围，而不是死记硬背，强迫自己完成任务。我们为自己创造工作。在我们接受任何必要的工作之前，我们应该质疑任何出现的新工作。

## Scope grows like grass 范围像草一样生长

Scope grows naturally. Scope creep isn't the fault of bad clients, bad managers, or bad programmers. Projects are opaque at the macro scale. You can't see all the little micro-details of a project until you get down into the work. Then you discover not only 范围自然扩大。Scope 蠕变不是糟糕的客户、糟糕的管理者或糟糕的程序员的错。项目在宏观尺度上是不透明的。在你开始工作之前，你不可能看到一个项目的所有微观细节。然后你不僅会发现

complexities you didn't anticipate, but all kinds of things that could be fixed or made better than they are.

你没有预料到的复杂性，但是所有的事情都可以被修复或者做得比现在更好。

Every project is full of scope we don't need. Every part of a product doesn't need to be equally prominent, equally fast, and equally polished. Every use case isn't equally common, equally critical, or equally aligned with the market we're trying to sell to.

每个项目都充满了我们不需要的范围。一个产品的每个部分不需要同样突出，同样迅速，同样精致。不是每个用例都同样普遍，同样重要，或者同样符合我们想要销售的市场。

This is how it is. Rather than trying to stop scope from growing, give teams the tools, authority, and responsibility to constantly cut it down.  
事情就是这样。与其试图阻止范围的扩大，不如给团队提供工具、权力和责任来不断削减它。

## Cutting scope isn't lowering quality 缩小范围并不会降低质量

Picking and choosing which things to execute and how far to execute on them doesn't leave holes in the product. Making choices makes the product better. It makes the product better *at some things* instead of others. Being picky about scope *differentiates* the product.

Differentiating what is core from what is peripheral moves us in competitive space, making us more alike or more different than other products that made different choices.

挑选和选择要执行的内容以及在这些内容上执行的距离并不会在产品中留下漏洞。做出选择会让产品更好。它使产品在某些方面比其他方面做得更好。对范围的挑剔使产品与众不同。区分哪些

是核心产品，哪些是外围产品，会让我们在竞争空间中更具竞争力，使我们比其他做出不同选择的产品更加相似或者更加不同。

Variable scope is not about sacrificing quality. We are extremely picky about the quality of our code, our visual design, the copy in our interfaces, and the performance of our interactions. The trick is asking ourselves which things actually matter, which things move the needle, and which things make a difference for the core use cases we're trying to solve.

可变范围并不意味着牺牲质量。我们对代码的质量、视觉设计、界面的复制以及交互的性能都非常挑剔。诀窍在于问问我们自己哪些事情真正重要，哪些事情起了作用，哪些事情对我们试图解决的核心用例产生了影响。

## Scope hammering 瞄准镜敲击声

People often talk about “cutting” scope. We use an even stronger word—hammering—to reflect the power and force it takes to repeatedly bang the scope so it fits in the time box.

人们经常谈论“削减”范围。我们使用一个更强烈的词——锤击——来反映重复敲击范围所需要的力量和力量，这样它就能适应时间盒。

As we come up with things to fix, add, improve, or redesign during a project, we ask ourselves:

当我们在一个项目中想出要修复、增加、改进或重新设计的东西时，我们会问自己：

- Is this a “must-have” for the new feature?  
这是新功能的“必备品”吗？

- Could we ship without this?  
没有这个我们能发布吗？

- What happens if we don’t do this?  
如果我们不这么做会怎么样？

- Is this a new problem or a pre-existing one  
that customers already live with?

这是一个新的问题还是一个已经存在的问题？

- How likely is this case or condition to occur?  
这种情况发生的可能性有多大？

- When this case occurs, which customers see it? Is it  
core—used by everyone—or more of an edge case?  
当这种情况发生时，哪些客户会看到它？它是核心—  
—每个人都使用——还是更多的边缘情况？

- What’s the actual impact of this case or con-  
dition in the event it does happen?

如果这种情况发生，实际的影响或条件是什  
么？

- When something doesn’t work well for a particular use case,  
how aligned is that use case with our intended audience?

当某个特定的用例出现问题时，该用例与我们预期的用户之  
间的关系如何？

The fixed deadline motivates us to ask these questions. Variable scope enables us to act on them. By chiseling and hammering the scope down, we stay focused on just the things we need to do to ship something effective that we can be proud of at the end of the time box. 固定的最后期限促使我们提出这些问题。可变范围使我们能够对它们采取行动。通过逐步缩小范围，我们将注意力集中在我们需要做的事情上，以便在时间框的尽头运送一些我们可以引以为傲的有效的东西。

Throughout the cycle, you'll hear our teams talking about must-haves and nice-to-haves as they discover work. The must-haves are captured as tasks on the scope. The scope isn't considered "done" until those tasks are finished. Nice-to-haves can linger on a scope after it's considered done. They're marked with a tilde (~) in front. Those tasks are things to do if the team has extra time at the end

在整个周期中，你会听到我们的团队在发现工作时谈论必须拥有的东西和美好的东西。必需品被捕获作为范围内的任务。在这些任务完成之前，作用域不被认为是“已完成”的。Nice-to-have 可以在作用域被认为已经完成之后在其上逗留。它们前面有一个波浪标记(~)。这些任务是团队在最后有额外时间时要做的事情

and things to cut if they don't. Usually they never get built. The act of marking them as a nice-to-have is the scope hammering.

以及如果他们没有削减的东西。通常它们永远也建不起来。把它们标记为一个值得拥有的好东西的行为就是大范围的锤击。

3/4 completed

## Campfire visibility

- Reconsider the "Change" link
- Tweak campfire visibility to propagate visibility to all children like all other types Pratik N.
- Figure out how to deal with client access for chat
- Decide on visibility path

A *finished scope with one nice-to-have (marked with a “~”)*  
一个完成的范围，有一个很好的已有(标有“~”)

*that was never completed*  
从来没有完成

## QA is for the edges

质量保证是针对边缘的

At Basecamp's current size (millions of users and about a dozen people on the product team), we have one QA person. They come in toward the end of the cycle and hunt for edge cases outside the core functionality.

以 Basecamp 目前的规模(数百万用户和产品团队中的十几个人)，  
我们只有一个 QA 人员。他们在周期快结束的时候进来，寻找核心功能之外的边缘情况。

QA can limit their attention to edge cases because the designers and programmers take responsibility for the basic quality of their work. Programmers write their own tests, and the team works together to ensure the project does what it should according to what was planned. This follows from giving the team responsibility for the whole project

instead of assigning them individual tasks (see Chapter 9, Hand Over Responsibility).

QA 可以限制他们对边缘案例的注意力，因为设计师和程序员对他们工作的基本质量负责。程序员编写他们自己的测试，团队一起工作，以确保项目根据形状做应该做的事情。这是因为要求团队对整个项目负责，而不是分配给他们单独的任务(见第 9 章，移交责任)。

For years we didn't have a QA role. Then after our user base grew to a certain size, we saw that small edge cases began to impact hundreds or thousands of users in absolute numbers. Adding the

多年来，我们一直没有 QA 的角色。然后当我们的用户基础增长到一定规模，我们看到小的边缘案例开始影响成百上千的绝对数量的用户。添加

extra QA step helped us improve the experience for those users and reduce the disproportional burden they would create for support.

额外的 QA 步骤帮助我们改善了这些用户的体验，减少了他们为支持造成的不成比例的负担。

Therefore we think of QA as a level-up, not a gate or a check-point that all work must go through. We're much better off with QA than without it. But we don't depend on QA to ship quality features that work as they should.

因此，我们认为质量保证是一个升级，而不是一个门或检查点，所有的工作必须通过。我们有 QA 比没有 QA 要好得多。但是我们并不依赖 QA 来提供质量特性，这些特性应该能够正常工作。

QA generates discovered tasks that are all nice-to-haves by default. The designer-programmer team triages them and, depending on severity and available time, elevates some of them to must-haves. The most rigorous way to do this is to collect incoming QA issues on a separate to-do list. Then, if the team decides an issue is a must-have, they drag it to the list for the relevant scope it affects. This helps the team see that the scope isn't done until the issue is addressed.

QA 生成的发现任务都是默认情况下可以拥有的。设计师-程序员团队对这些任务进行分类，根据严重程度和可用时间，将其中一些任务提升为必须完成的任务。做到这一点最严格的方法就在一个单独的待办事项清单上收集即将到来的 QA 问题。然后，如果团队认为某个问题是必须要解决的，他们就会把它拖到清单上，以确定它所影响的范围。这样可以帮助团队看到，在问题得到解决之前，范围还没有完成。

We treat code review the same way. The team can ship without waiting for a code review. There's no formal check-point. But code review makes things better, so if there's time and it makes sense, someone senior may look at the code and give feedback. It's more

about taking advantage of a teaching opportunity than creating a step in our process that must happen every time.

我们以同样的方式对待代码审查。团队可以不用等待代码审查就可以发布。没有正式的检查点。但是代码审查使事情变得更好，所以如果有时间并且有意义的话，一些高级人员可能会看看代码并给出反馈。它更多的是利用一个教学机会，而不是在我们的过程中创造一个每次都必须发生的步骤。

## When to extend a project

### 何时扩展项目

In very rare cases, we'll extend a project that runs past its deadline by a couple weeks. How do we decide when to extend a project and when to let the circuit breaker do its thing?

在极少数情况下，我们会将一个超过最后期限的项目延长几周。

我们如何决定什么时候延长一个项目，什么时候让断路器做它的事情？

First, the outstanding tasks must be true must-haves that withstood every attempt to scope hammer them.

首先，突出的任务必须是真正的必需品，经得起任何尝试。

Second, the outstanding work must be all downhill. No unsolved

第二，出色的工作必须全部是下坡路，没有悬而未决的

problems; no open questions. Any uphill work at the end of the cycle points to an oversight in the shaping or a hole in the concept.

Unknowns are too risky to bet on. If the work is uphill, it's better to do something else in the next cycle and put the troubled project back in the shaping phase. If you find a viable way to patch the hole, then you can consider betting more time on it again in the future.

问题：没有公开的问题。周期结束时的任何艰难工作都指向塑造上的疏忽或概念上的漏洞。未知因素太冒险，不能下注。如果工作是上坡路，最好在下一个周期做点别的事情，把麻烦重重的项目放回到形成阶段。如果你找到了一个可行的方法来弥补漏洞，那么你可以考虑在未来再次投入更多的时间。

Even if the conditions are met to consider extending the project, we still prefer to be disciplined and enforce the appetite for most projects. The two-week cool-down usually provides enough slack for a team with a few too many must-haves to ship before the next cycle starts. But this shouldn't become a habit. Running into cool-down either points back to a problem in the shaping process or a performance problem with the team.

即使考虑扩展项目的条件得到满足，我们仍然倾向于遵守纪律，强化大多数项目的胃口。两周的冷静通常会给团队提供足够的休息时间，让他们在下一个周期开始之前有太多必须完成的任务。但是这不应该成为一种习惯。进入冷却阶段要么指向塑造过程中 的问题，要么指向团队中的性能问题。

CHAPTER 14 - Decide When to Stop

149

Chapter14-deceeeeeeeeeeeeeeeeeeeeeeeeeeeee  
ee  
eee

# Move On

## 继续

Let the storm pass  
让暴风雨过去吧

Shipping can actually generate new work if you're not careful. Feature releases beget feature requests. Customers say "Okay, that's great, but what about that other thing we've been asking for?" Bugs pop up.

Suggestions for improvements come in. Everyone is focused on the new thing and reacting to it.

如果你不小心，运输实际上可以产生新的工作。功能发布会产生功能请求。客户说：“好的，这很好，但是我们一直在要求的其他东西呢？”Bug 弹了出来。提出了改进的建议。每个人都专注于新事物并对其做出反应。

The feedback can be especially intense if the feature you shipped changes existing workflows. Even purely visual changes sometimes spur intense pushback. A small minority of customers might over-react and say things like "You ruined it! Change it back!"

如果您提供的特性改变了现有的工作流程，那么反馈可能会特别强烈。即使是纯粹的视觉变化有时也会引起强烈的反弹。一小部分顾客可能会反应过度，说“你毁了它！把它换回来！”

It's important to stay cool and avoid knee-jerk reactions. Give it a few days and allow it to die down. Be firm and remember why you made the change in the first place and who the change is helping.

保持冷静，避免下意识的反应是很重要的。给它几天时间，让它慢慢平息下来。要坚定，记住你一开始为什么要做出改变，改变对谁有帮助。

Stay debt-free

## 还清债务

It can be tempting to commit to making changes in response to feedback, but then you no longer have a clean slate for the next cycle. Remember: these are just raw ideas coming in. The way to handle them is with a gentle “no.” Saying “no” doesn’t prevent you from continuing to contemplate them and maybe shape them up into future projects. Saying “yes,” on the other hand, takes away your freedom in the future. It’s like taking on debt.

为了回应反馈而做出改变是很诱人的，但是这样你就不再有下一个周期的清白记录了。记住：这些只是一些原始的想法。处理这些问题的方法是温和地说“不”说“不”并不能阻止你继续思考它们，也许还能把它们塑造成未来的项目。另一方面，说“是”会剥夺你未来的自由。这就像背负债务。

Remember, the thing you just shipped was a six-week bet. If this part of the product needs more time, then it requires a new bet. Let the requests or bugs that just came up compete with everything else at the next betting table to be sure they’re strategically important.

记住，你刚运来的东西是六周的赌注。如果这部分产品需要更多的时间，那么它需要一个新的赌注。让刚刚出现的请求或 bug 与下一个赌桌上的其他所有东西竞争，以确保它们在战略上的重要性。

## Feedback needs to be shaped 需要塑造反馈

Here we come full circle. The raw ideas that just came in from customer feedback aren't actionable yet. They need to be shaped. They are the raw inputs that we talked about in step one of the shaping process: Set Boundaries.

我们又回到原点了。刚刚从客户反馈中得到的原始想法还没有付诸实施。他们需要被塑造。它们是我们在塑造过程的第一步中讨论过的原始输入: 设置边界。

If a request is truly important, you can make it your top priority on the shaping track of the next cycle. Bet on something else for the teams to build and use that time to properly shape the new idea. Then, when the six weeks are over, you can make the case at the betting table and schedule the shaped version of the project for the greatest chance of success.

如果一个请求真的很重要，你可以把它作为下一个周期的重中之重。把赌注押在其他的东西上，让团队去构建和利用这些时间来适当地形成新的想法。然后，当六个星期结束时，你可以在赌桌上注，安排项目的成形版本，以获得最大的成功机会。

CHAPTER 15 - Move On

151

Chapter 15-move on 151

# Conclusion

## 结论

### Key concepts

#### 关键概念

The Shape Up method presented in this book is tightly interwoven. It may take some thought and experimentation to pull out the right pieces and adapt them to your team.

这本书中介绍的塑形方法是紧密交织在一起的。它可能需要一些思考和实验来找出正确的部分并使它们适合你的团队。

Whether your team can adopt the method at once or not, I hope that the language and concepts in this book gave you some things to take home immediately:

无论你的团队能否立即采用这种方法，我希望这本书中的语言和概念能够给你带回家一些东西：

- Shaped versus unshaped work  
有形与无形的作品
- Setting appetites instead of estimates  
设置欲望而不是估计
- Designing at the right level of abstraction  
在正确的抽象层次上进行设计
- Concepting with breadboards and fat marker sketches  
用面包板和粗糙的草图来构思
- Making bets with a capped downside (the circuit breaker)  
and honoring them with uninterrupted time  
用有上限的下限(断路器)下注，并用不间断的时间来兑现  
它们
- Choosing the right cycle length (six weeks)

选择正确的周期长度(六周)

- A cool-down period between cycles  
周期之间的冷却期

- Breaking projects apart into scopes  
把项目分解成不同的范围

- Downhill versus uphill work and com-  
municating about unknowns

下坡与上坡的工作和沟通的未知

- Scope hammering to separate must-haves from nice-to-haves  
范围: 把必须拥有的东西和好的东西分开

## Get in touch 保持联系

We'd love to hear what you think so we can make the Shape Up method easier to adopt. What did we miss? What still isn't clear? What do you wish we had talked about that we didn't? We'd also love to hear about your successes and challenges as you try to apply it to your teams and projects.

我们很乐意听听你的想法，这样我们就可以使 Shape Up 方法更容易采用。我们错过了什么？还有什么不清楚的？你希望我们谈过哪些没有谈过的事情？我们也希望听到你的成功和挑战，因为你试图将它应用到你的团队和项目中。

Send us an email at [shapeup@basecamp.com](mailto:shapeup@basecamp.com).  
给我们发邮件到 [shapeup@basecamp.com](mailto:shapeup@basecamp.com)。

**C H A P T E R 1 6 - C o n c l u s i o n      1 5 3**

Chapter 16 conclusion 153



# Appendices

# 附录

# How to Implement Shape Up in Basecamp

## 如何在 Basecamp 中实现 Shape Up

We built Basecamp to implement the Shape Up method. Instead of scattering our work across multiple tools, Basecamp centralizes all project communication, task management, and documentation in one place. Here's how we use it.

我们构建了 Basecamp 来实现 Shape Up 方法。Basecamp 没有将我们的工作分散在多个工具之间，而是将所有的项目沟通、任务管理和文档集中在一个地方。下面是我们如何使用它。

### A Basecamp Team for shaping 一个用于塑形的大本营团队

1. Create a Basecamp Team for shaping. We call ours “Product Strategy.”

创建一个用于塑造的 Basecamp 团队。我们称之为“产品策略”

2. Add the people doing the shaping, any trusted people who give feedback on pitches, and the people who bet at the betting table. Keep this group small and announce the bets more widely elsewhere (we use Basecamp's HQ for that), when it's time to kick off a cycle.

再加上负责塑形的人，任何对球场进行反馈的值得信赖的人，以及在赌桌上下注的人。让这个团队保持小规模，并在其他地方更广泛地宣布赌注(我们使用大本营的总部)，当是时候开始一个周期的时候。

3. Post pitches as Messages on the Message Board.

We created a Message Category called “Pitch”  
with the light bulb emoji for the icon.

在留言板上以消息的形式发布。我们创建了一个  
名为“Pitch”的消息类别，其图标用的是灯泡表  
情符号。

4. Use the Campfire chat room to bounce ideas and co-  
ordinate the betting table between cycles. We conduct the  
actual betting table meeting over video chat.

使用 Campfire 聊天室来交流想法，协调周期之间的赌  
桌。我们通过视频聊天进行实际的赌桌会议。

The screenshot shows the 'Message Board' section of the Basecamp interface. At the top, there's a navigation bar with links for Home, Pings, Hey!, Activity, My Stuff, and Find. Below that is a header for 'Product Strategy'. A green button labeled '+ New message' is on the left, and a link 'All messages' is on the right. The main area is titled 'Message Board' and contains a list of six pitches, each with a small profile picture, the title, a brief description, and a blue circular badge with a number indicating the count of notifications:

- Group notifications**: Pitch by Ryan Singer - Feb 18 — Last November I wrote a pitch for adding Groups ala BCX to BC3: Groups! - Product Strategy. After looking at the pitch, 1 notification.
- Untangle the people spreadsheet**: Pitch by Ryan Singer - Oct 11, 2018 — Software gets complex when one thing does two things. Today the people spreadsheet is one interface that does at least 5 notifications.
- Sort Messages Setting (revised)**: Pitch by Ryan Singer - Sep 21, 2018 — Here's a revised second take on the Sort Messages A-Z pitch. After talking to support and a couple customers via phone, 6 notifications.
- Follow-up: Client-visible recordings**: Report by Ryan Singer - Sep 13, 2018 — In April we launched the new client-visible toggle to replace the Clientside. Do people use the new client toggle? How 2 notifications.
- Creation on the iPhone**: Report by Ryan Singer - Sep 7, 2018 — While chatting w/ Conor today some questions came up about what gets created on the iPhone. Here's a rundown from 2 notifications.

*Pitches on the Message Board of the Product Strategy team in Basecamp  
在 Basecamp 产品战略团队的留言板上进行推销*

The screenshot shows a hand-drawn sketch on an iPad. At the top, there's a navigation bar with links for Home, Pings, Hey!, Activity, My Stuff, and Find. Below that is a header for 'about it'. The main content area contains a note: 'With that in mind, I looked for broad design solutions for those two use cases only.' Below this is a section titled 'Solution for choosing Message subscribers'. It includes a note: 'We could use a BCX-style approach to solve the first case. We have a single modal today for setting Subscribers app-wide, whether you are posting a new Message or changing who gets notified about some commentable. That "Who should be notified?" modal could offer Groups at the top. Clicking them checks members of the Groups below.' To the right of this text is a hand-drawn sketch of a modal window. The title of the modal is 'Who should be notified?'. Inside, there are four blue circles representing individual users. To the right of the circles is a red arrow pointing to the word 'GROUPS' above a list of four user icons with checkboxes next to them. The checkboxes for the last three icons are checked.

User Icon	Checkboxes
①	unchecked
②	checked
③	checked
④	checked

*A sketch drawn on an iPad in the middle of a pitch  
一个在 iPad 上绘制的草图*

Chaptr17hooooooooooooooooooooooo  
oooooooooooooooooooooooooooo  
oooo

The screenshot shows a Basecamp HQ interface. At the top, there are navigation links: Home, Pings, Hey!, Activity, My Stuff, and Find. Below that, it says "Basecamp HQ · Message Board". The main content area has a title "Announcing Cycle 4". Below the title is a post by Jason Fried from November 17, 2017, which says "It's next cycle time!". The post continues: "First, this is a short cycle. We've got a few major holidays this cycle, plus the end of the year. People are busy with life, travel, snow (!), and all the other stuff that goes with ramping down one year and starting another." Another section discusses "Merge Latest Activity and Reports", explaining how reports will move into the Latest Activity screen. It also mentions running non-traditional projects at the end of the year.

*Jason announces the bets for the next cycle in the HQ*  
杰森在总部宣布下一轮的赌注

*—a Team in Basecamp that includes the whole company*  
——一个包括整个公司在内的基本营团队

## Basecamp Projects for the cycle projects Basecamp 项目的周期性项目

1. Create a Basecamp Project for each project in the six-week cycle. We usually prepend a name or number for the cycle like this: “Cycle 4: Autopay.”

在六周的周期内为每个项目创建一个 Basecamp 项目。

我们通常会在这个周期的前面加上一个名字或者数字: “Cycle 4: Autopay”

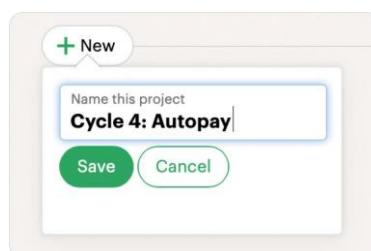
2. Add the designer and program-

添加设计器和程序

mers who are working on this project to the Basecamp Project.

在这个项目上工作的人们, 到

Basecamp 项目。



3. Post a kick-off message to the

Message Board with the pitch

在留言板上发布一条开始信息

or a restatement of the shaped

或者是对形状的重新描述

work for the team's reference.

为团队工作提供参考。

*Creating the project*  
创建项目

Cycle 4: Autopay

## People on the project

+ Add people...

1. Who do you want to add? (X)

Name: Nicole Katz	(X)
Email: nicole@honchodesign.com	
Title: Programmer	
Company: Honcho Design	

Name: Jared Davis	(X)
Email: jared@honchodesign.com	
Title: UI Designer	
Company: Honcho Design	

Add another person or Pick people from a company... ▼

*Adding the designer and programmer*  
添加设计器和程序员

Home Pings Hey! Activity My Stuff Find

BC3: Hill Charts > Message Board

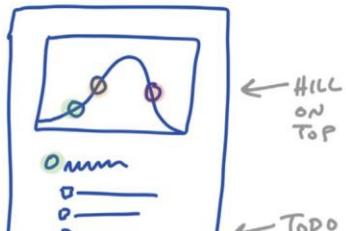
## Hill Charts concept

Announcement by Ryan Singer  
Nov 17, 2017 · No one was notified.

Here's the starting concept for the Hill Chart feature in Basecamp.

### How they relate to to-dos

The basic idea is there's a hill chart at the top of the todoset. Hill **charts** map one-to-one with todosets, and the **dots** on the hill map one-to-one with todolists.



*The first thing on the project is a kick-off message with the shaped concept*  
这个项目的第一件事就是一个带有成形概念的开始信息

Chaptr17hooooooooooooooooooooooo  
oooooooooooooooooooooooooooo  
oooo

The screenshot shows a Basecamp chat interface. At the top, there are navigation links: Home, Pings, Hey!, Activity, My Stuff, and Find. Below that, a breadcrumb trail says "BC3: Hill Charts". The main area is titled "Campfire". A message from "Scott Upton" at 3:46pm says: "We'll need to stub in the Hill Chart Editor, too (i.e. where you can annotate and update positions for each list)". A message from "Pratik Nalik" at 4:07pm responds: "Scott I can add an empty controller for the editor. But I think it'd help to have some mocks first.". Scott replies at 4:15pm: "Cool, OK. I'll work something up tomorrow AM.". Pratik responds at 4:15pm: "Sounds good!". In the middle of the conversation, there's a note: "Thinking a little bit more about it, I think we may want to put the hill chart attributes in a new table and add one-one mapping with recordings. Similar to how the Positioning model works." Another note follows: "But we'll cross that bridge when we get there". At the bottom, a message says: "For now, those attrs will live in the todolists table".

*The team uses the chat room in the Basecamp project  
团队使用 Basecamp 项目中的聊天室*

*to communicate as they get started  
在他们开始交流的时候*

## To-Do Lists for scopes 作用域的待办事项清单

1. After the team gets oriented they start spiking, discover tasks, and map them into scopes.

在团队确定方向后，他们开始达到峰值，发现任务，并将它们映射到范围内。

2. In the Basecamp Project, the team creates a To-Do List for each scope, like “Start Autopay” or “ACH Option.” Sometimes we use the description field on the To-Do List to summarize the scope.

在 Basecamp 项目中，团队为每个范围创建一个待办事项列表，比如“Start Autopay”或“ACH Option”有时我们使用 To-Do List 上的描述字段来总结范围。

3. Add design and programming tasks to each scope as To-Do Items. For example, “Start Autopay” has one task for design the

UI and another task for wiring it to the existing recur-ring  
billing API. They can use the discussion thread on each To-Do  
Item to post updates or ask each other questions.

将设计和编程任务添加到每个作用域中作为待办事项。例如，  
“Start Autopay”有一个任务用于设计 UI，另一个任务用于  
将其连接到现有的循环计费 API。他们可以使用每个待办  
事项的讨论线程来发布更新或相互提问。

4. Repeat as the team discovers new scopes and tasks.  
当团队发现新的范围和任务时重复。

The screenshot shows a 'To-dos' section with three items:

- Start Autopay**: 0/3 completed. Tasks: Design button on the Invoice to start (Jared D.), UI for the Start Autopay screen (Jared D.), Hook into FI recurring billing API (Nicole K.).
- Owner Shutoff**: 0/4 completed. Tasks: UI for the owner to turn off autopay for a customer (Jared D.), Implement turning off (Nicole K.), UI: What do they see after turning off? (Jared D.), Decide: Can they turn on again?
- ACH Option**: 0/0 completed. Task: Add an ACH option to the Autopay form.

*To-Do Lists for each scope with designer and programmer tasks under each.*  
每个作用域的待办事项清单，每个作用域下有设计师和程序员的任务。

*Note: these are just the tasks discovered so far.*

注意：这些只是到目前为止发现的任务。

## Track scopes on the Hill Chart 在山图上追踪范围

1. Navigate to the To-Do List page for each scope and click the options menu (••) in the upper right. Click “Track this on the Hill Chart.” That will display a Hill Chart at the top of the overall To-Dos section of the Project, with a dot corresponding to that scope (To-Do List).

导航到每个作用域的 To-Do List 页面，然后单击右上角的选项菜单(••)。点击“在山形图上跟踪它”它将在项目的整个 To-Dos 部分的顶部显示一个 Hill 图表，其中一个点对应于该范围(To-Do List)。

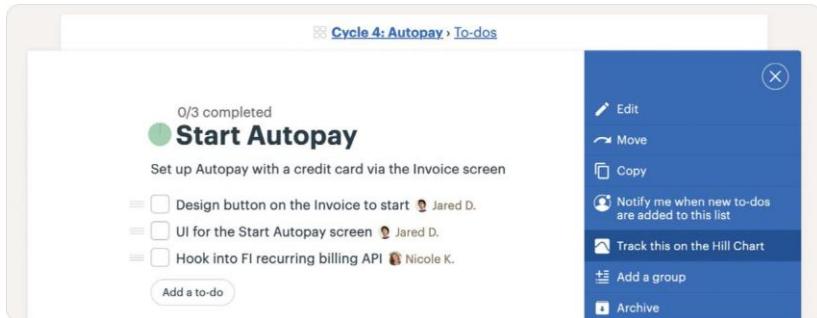
2. Repeat for each scope (To-Do List).  
对每个范围重复(To-Do List)。

3. Click “Update” on the Hill Chart and drag the dots to show progress from “unknown” to “known” to “done.” Use the annotation feature to add commentary when necessary.

点击山图上的“更新”并拖动点来显示从“未知”到“已知”到“完成”的进度必要时使用注释功能添加注释。

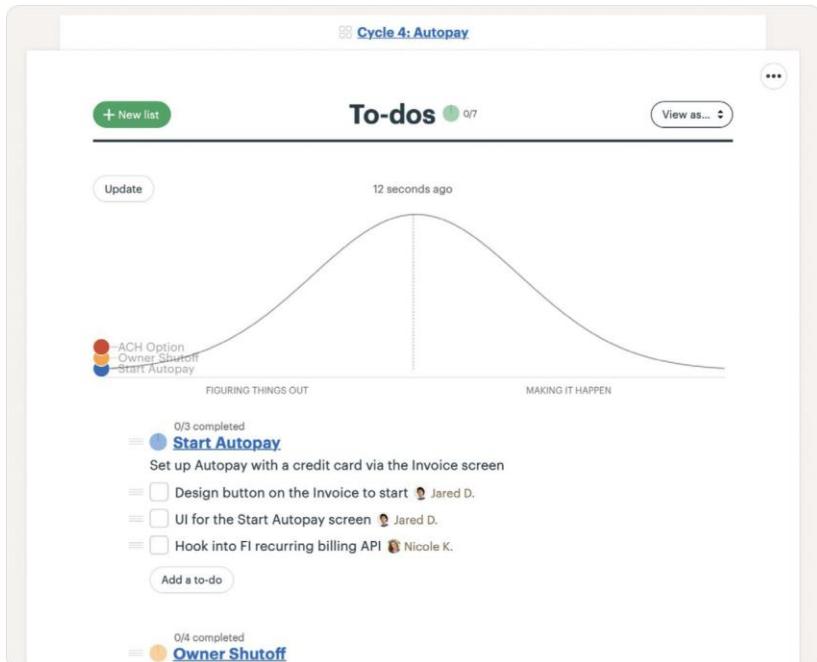
4. To see the history of updates to the Hill Chart, click the time-stamp above the Hill Chart where it says “last update.”

要查看 Hill Chart 的更新历史, 请单击 Hill Chart 上方显示“最后更新”的时间戳



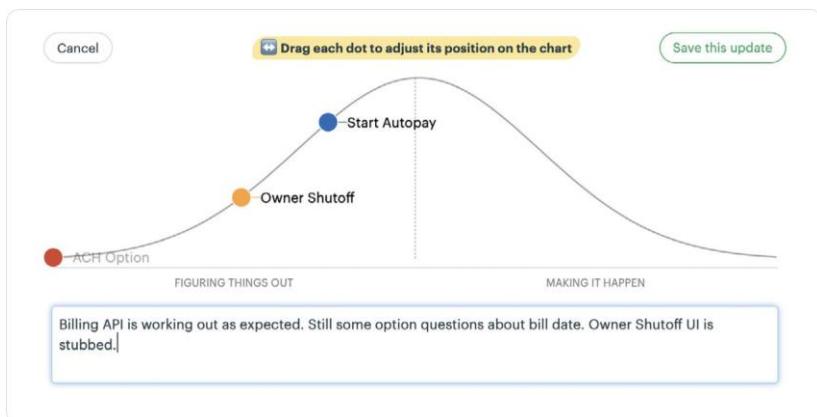
*Tracking a scope (To-Do List) on the Hill Chart via the options menu  
通过选项菜单跟踪 Hill Chart 上的一个范围(To-Do List)*

*on the To-Do List's page  
在待办事项清单的页面上*



*After enabling the Hill Chart on each To-Do List, a dot appears for each scope  
启用每个待办事项清单上的 Hill Chart 后, 每个作用域上都会出现一个点*

162  
162



*Drag the dots to update the Hill Chart*  
 拖动点来更新山形图

+ New list

To-dos

View as... ⚙

Update

Last updated 2 minutes ago

Start Autopay

Owner Shutoff

ACH Option

FIGURING THINGS OUT

MAKING IT HAPPEN

2/5 completed

**Start Autopay**

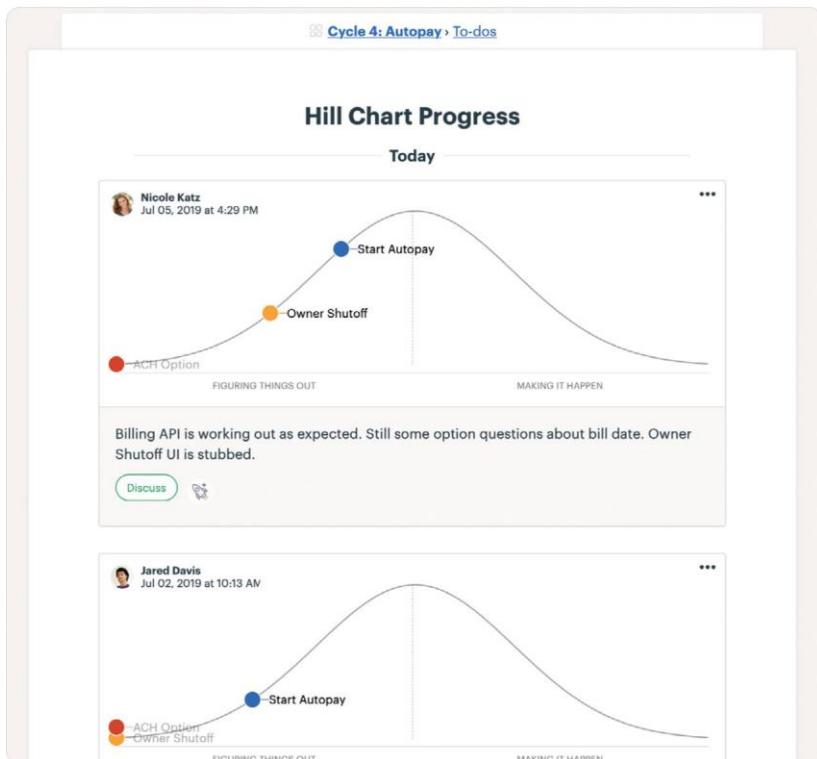
Set up Autopay with a credit card via the Invoice screen

- Hook into FI recurring billing API 🚧 Nicole K.
- Set correct bill date 🚧 Nicole K.
- Move button to action menu on mobile 🚧 Jared D.
- UI for the Start Autopay screen 🚧 Jared D.
- Design button on the Invoice to start 🚧 Jared D.

Add a to-do

*The updated Hill Chart appears at the top of the To-Dos page*  
 更新后的山形图显示在 To-Dos 页面的顶部

Chaptr17hooooooooooooooooooooooo  
oooooooooooooooooooooooooooo  
oooo



*Clicking the timestamp at the top of the Hill Chart shows a history of updates.  
单击 Hill Chart 顶部的时间戳显示更新的历史记录。*

*The most recent update is at the top.  
最近的更新在顶部。*

Other tools make it hard to see everything about the project in one place. Basecamp combines chat, messages, to-dos, and documents in one interface that's powerful enough for technical people and friendly enough for everyone else to use. Programmers, design-ers, QA and managers feel equally at home and can work together seamlessly on projects. Visit [basecamp.com](http://basecamp.com) to try Basecamp free for 30 days.

其他工具使得很难在一个地方查看项目的所有信息。Basecamp 将聊天、消息、待办事项和文档集成在一个界面中，这个界面对技术人员来说足够强大，对其他人来说也足够友好。程序员、设计师、质量保证人员和管理者都感觉很自在，可以在项目上无缝合作。访问 [Basecamp.com](http://basecamp.com) 免费试用 30 天。

**164**

# Adjust to Your Size

## 调整你的尺寸

Basic truths vs. specific practices  
基本真理 vs 具体实践

To apply Shape Up to your company, it helps to separate out the basic truths from the specific practices.

将塑形应用到你的公司，它有助于从具体的实践中分离出基本的真理。

Work has to come from somewhere, and it takes work to figure out what the right work is. This is shaping. Shaping the work sets clearer boundaries and expectations for whoever does the work— whether that's a separate team or just your future self. If we don't make trade-offs up front by shaping, the universe will force us to make trade-offs later in a mad rush when we're confronted by dead-lines, technical limitations, or resource constraints.

工作必须从某个地方开始，并且需要努力找出正确的工作是什么。这就是塑造。塑造工作设置更清晰的界限和期望谁做的工作-无论是一个单独的团队或只是你未来的自己。如果我们不提前做出权衡，宇宙将迫使我们做出权衡，当我们面临死胡同，技术限制，或资源约束时，我们以后疯狂地匆忙做出权衡。

The same is true with betting. Six weeks might not be the exact time frame for your team. But the consequences of making unclear or open-ended commitments are the same for everyone. Regardless of the specific time frame we bet on, we should be deliberate about what we bet on and cap our downside with a circuit breaker.

赌博也是如此。对于你的球队来说，六周可能不是精确的时间框架。但是做出不明确或无限期的承诺的后果对每个人都是一样的。

不管我们押注的具体时间框架是什么，我们都应该仔细考虑我们押注的东西，并用断路器来限制我们的负面影响。

In the building phase, there will be unknowns to deal with whether you track them on a hill chart or not. We need to distinguish the knowns from the unknowns so we can sequence the work in the right order and reserve capacity for the unknowns.

在建设阶段，将有未知数来处理是否你跟踪他们在一个山图或没有。我们需要将已知和未知区分开来，这样我们才能按照正确的顺序排列工作，并为未知预留容量。

These truths apply regardless of the size of your organization. The specific practices, on the other hand, are scale-dependent. Let's have a look at what it means to implement Shape Up at a very small start-up and an organization that's grown big enough for specialized roles and more structure.

这些真理适用于任何规模的组织。另一方面，具体的实践是依赖于规模的。让我们来看看在一个非常小的初创企业和一个已经发展到足以承担特殊角色和更多结构的组织中实现 Shape Up 意味着什么。

## Small enough to wing it 小到可以即兴发挥

When your team is just two or three people, everybody does a bit of everything. Since a few people are wearing many hats and performing many roles, it's difficult to commit long chunks of uninterrupted time to specific projects. The person doing the programming might also be answering customer requests and dealing with an infrastructure issue all at the same time.

当你的团队只有两三个人的时候，每个人都会做一些事情。由于少数人身兼多职，扮演多种角色，因此很难将大量不受干扰的时间投入到特定的项目中。编程人员也可能同时回答客户的请求和处理基础设施问题。

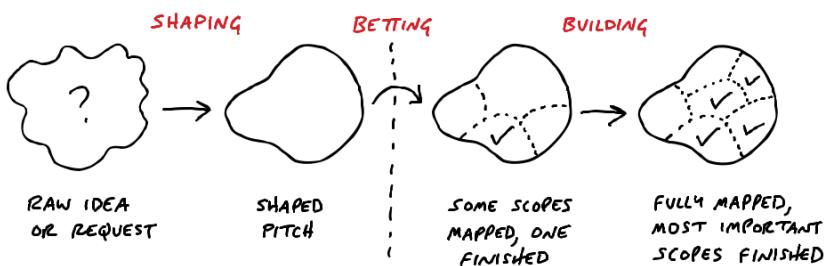
It's also easier to communicate and change course when you're small. You can drop something in the group chat or talk about it in person and everyone's immediately on the same page.

当你还小的时候，交流和改变方向也更容易。你可以在群聊中谈论一些事情，或者当面谈论这些事情，这样每个人都能立刻达成共识。

For these reasons, a tiny team can throw out most of the structure. You don't need to work six weeks at a time. You don't need a cool-down period, formal pitches or a betting table. Instead of parallel tracks with dedicated shapers and builders, the same people can alternate back and forth. Be deliberate about which hat you're wearing and what phase you're in. Set an appetite, shape what to do next, build it, then shape the next thing. Your bets might be different sizes each time: maybe two weeks here, three weeks there. You're still shaping, betting, and building, but you're doing it more fluidly without the rigid structure of cycles and cool-downs.

由于这些原因，一个小团队可以抛弃大部分的结构。你不需要一次工作六个星期。你不需要一段时间的冷静，不需要正式的投球

或者一张赌桌。而不是平行的轨道与专门的塑造者和建设者，相同的人可以来回交替。仔细考虑你戴的帽子和你所处的阶段。设定好胃口，想好下一步该做什么，做好它，然后想好下一步该做什么。你的赌注可能每次都不同：这里可能两周，那里可能三周。你仍然在形成、下注和建立，但是你在没有周期和冷却的僵化结构的情况下做得更流畅。



*The phases of the work still hold true even if you don't work in cycles*  
即使你不循环工作，工作的各个阶段仍然是正确的

*or have dedicated people to do the shaping and building*  
或者有专门的人来进行塑造和建造

## Big enough to specialize 大到可以专攻

After you hire more people, all of this fluidity flips from an asset to a liability. Winging it with ad-hoc meetings and chat room discussions doesn't work anymore. Coordination starts to eat up more of your time and things begin to slip through the cracks.

在你雇佣更多的人之后，所有这些流动性就从资产变成了负债。

即兴的特别会议和聊天室讨论不再起作用。协调开始占用你更多的时间，事情开始从缝隙中溜走。

This is when it makes sense to take on the structure of six-week cycles, cool-downs, and a formal betting table. With more people available to build, someone needs to carve out more time to do the work of figuring out *what* to build. This could mean a founder spends more time shaping than building, or it could mean elevating an employee from doing in-cycle design work to more out-of-cycle shaping work.

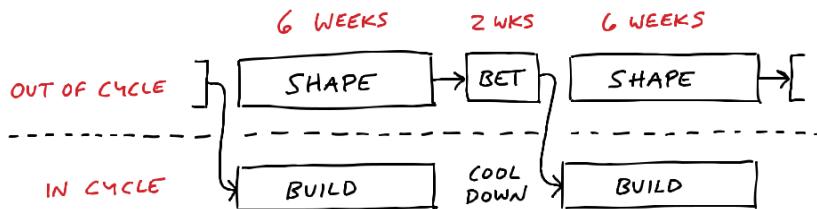
这时，采用六周周期、冷静和正式赌桌的结构是有意义的。有更多的人可以建造，有人需要挤出更多的时间来做工作，弄清楚建造什么。这可能意味着创始人花在塑造上的时间比建造上的时间要多，也可能意味着将员工从周期内的设计工作提升到周期外的塑造工作。

At Basecamp's current size (about 50 people in the whole company, roughly a dozen in the product team) we've been able to specialize roles so teams of designers and programmers can work without any interruption in the cycles. A dedicated team called SIP (Security, Infrastructure, and Performance) handles technical work that's lower in the stack and more structural. Our Ops team keeps the lights on. We have technical people on the Support team who can investigate problems raised by customers. All this means that we don't need to interrupt the designers and programmers on our Core Product team who work on shaped projects within the cycles.

以 Basecamp 目前的规模(整个公司大约有 50 人，产品团队大约有 12 人)，我们已经能够专门设计角色，这样设计师和程序员团队就可以在周期中不受任何干扰地工作。一个叫做 SIP (Security, infrastructure, and Performance) 的专门团队负责处理技术工作，这些工作在堆栈中的位置较低，而在结构上的位置较多。我们的操作团队一直在工作。我们的支持团队中有技术人员，他们可以调查客户提出的问题。所有这些都意味着我们不需要打断我们核心产品团队的设计师和程序员，他们在周期内处理有形的项目。

With dedicated shapers and builders, the picture is more structured. Shapers work on an “out of cycle” track. Cool-down between cycles gives everyone room to fix bugs and address loose ends that pop up. The betting table is held during cool-down and then bets are placed for the next cycle.

与专门的塑造者和建设者，图片是更有条理的。Shapers 工作在一个“不循环”的轨道上。周期之间的冷却让每个人都有空间去修正错误和解决突然出现的问题。在冷却期间，赌桌被放置，然后下一个周期的赌注被放置。



*With more people, shaping and building happen on separate tracks*  
随着人数的增加，塑造和建造在不同的轨道上进行

*and bets are made to fill six-week cycles*  
并下注以填补六周的周期

**168**  
**168**

# How to Begin to Shape Up

## 如何开始塑造自己

### Option A: One six-week experiment

#### 选项 a: 一个为期六周的实验

You don't need to change everything all at once. If the whole product team isn't ready to make a big change, just start off with a single six-week experiment. These are the steps to take:

你不需要一下子改变所有事情。如果整个产品团队还没有准备好做出重大改变，那么就从一个为期六周的实验开始吧。以下是下面的步骤：

1. Shape one significant project that can be comfortably finished within six weeks. Be conservative and allow extra time on your first run.

塑造一个可以在六周内轻松完成的重要项目。保

守一点，在你的第一次跑步中留出额外的时间。

2. Carve out one designer and two programmers' time for the entire six weeks. Guarantee that nobody will interrupt them for the length of the experiment.

在整整六个星期里，挤出一个设计师和两个程序员的时

间。保证在整个实验过程中没有人会打扰他们。

3. Instead of a proper betting table, simply plan for the team to do the work you shaped for this one experiment.

不是一个合适的赌桌，而是简单的计划让团队做你为这个实验设计的工作。

4. Kick off by presenting your shaped work to the team, with all the ingredients of a pitch. Set the expectation that they will discover and track their own tasks. (Hand Over Responsibility)

首先，向团队展示你的成形作品，包括所有的推销要素。期望他们能发现并跟踪他们自己的任务。(移交责任)

5. Dedicate a physical space or a chat room to the cross-functional team so they can work closely together.

为跨职能团队提供一个物理空间或聊天室，这样他们可以密切合作。

6. Encourage them to Get One Piece Done by wiring UI and code together early in the project.

鼓励他们通过在项目早期将 UI 和代码连接在一起来完成一件事情。

You don't need to worry about Mapping the Scopes or Showing Progress right away. You should see a big leap in progress just by dedicating uninterrupted time, shaping the work in advance, and letting the team work out the details.

你不需要担心映射 scope 或立即显示进度。你应该看到一个巨大的进步，只要投入不间断的时间，提前制定工作计划，让团队制定出细节。

Once the team gets used to Getting One Piece Done, the stage will be set for properly mapping scopes down the road. It's the same idea, just repeated. Later still, when they are good at defining scopes, you can use the hill chart to Show Progress on those scopes.

一旦团队习惯了完成一部分工作，这个阶段就可以正确地绘制未来的工作范围。这是同样的想法，只是重复。稍后，当他们擅长定义范围时，你可以用山形图来显示这些范围的进展。

This approach lets you demonstrate success with one team and a single six-week commitment. With credibility gained from a good outcome, it'll be easier to lobby for a bigger change and convert the wider team to working this way.

这种方法可以让您通过一个团队和一个为期六周的承诺来证明成功。有了良好结果带来的可信度，你就更容易游说更大的改变，让更广泛的团队采用这种方式工作。

## Option B: Start with shaping 备选方案 b: 从塑造开始

Sometimes it's not possible to get a team together to work for six weeks because somebody else, a CTO perhaps, controls the pro-grammers' time. In that case, you can start by shaping a compelling project with clearer boundaries than past projects. Present the project and put it through your company's existing scheduling process (even if it's a paper shredder). Better-shaped work can shine a light on the engineering team and help them open up to things like longer cycles or a more deliberate betting process.

有时不可能让一个团队一起工作六个星期，因为其他人，也许是首席技术官，控制了程序员的时间。在这种情况下，你可以开始塑造一个引人注目的项目，比过去的项目有更清晰的界限。展示这个项目，并把它放到你公司现有的日程安排流程中(即使它是一个碎纸机)。形状更好的工作可以给工程团队带来光明，帮助他们接受更长的周期或更深思熟虑的投注过程。

## Option C: Start with cycles

### 选项 c: 从循环开始

Another approach is to start by working in six week cycles. For teams that formerly used two-week sprints, this removes the over-head of constant planning meetings and gives programmers more time to build momentum and hit their stride. Once the team has more room to breathe, it'll be natural to think more about how to shape the work to take advantage of this new capacity.

另一种方法是以六周为周期开始工作。对于以前使用两周冲刺的团队来说，这消除了持续计划会议的开销，并且给程序员更多的时间来建立动力和迈出他们的步伐。一旦团队有了更多的喘息空间，就会自然而然地思考如何利用这种新能力来完成工作。

## Fix shipping first 先解决运输问题

Build your shipping muscles before you worry too much about improving your research or discovery process. You can have the best customer insight in the world, but if you can't turn it into a project and ship, it won't matter. First get the team into a rhythm of finishing things. Once you have the capability to ship, then you can start improving the inputs to your shaping process.

在你过于担心改进你的研究或发现过程之前，先锻炼你的航运肌肉。您可以拥有世界上最好的 customer insight，但是如果您不能将其转化为一个项目并发布出去，那就没有关系了。首先让团队进入完成工作的节奏。一旦你有能力发货，那么你就可以开始改进你的成型过程的输入。

## Focus on the end result 专注于最终结果

Sometimes it can be scary to give the teams more free rein to set their own tasks and schedule. You might wonder: What if they don't use up all the time we dedicate for the cycle? What if one of the programmers or designers sits idle at some point in the cycle?

有时候给团队更多的自由去设定他们自己的任务和时间表是很可怕的。你可能想知道：如果他们没有用完我们在这个周期中投入的所有时间怎么办？如果其中一个程序员或者设计师在某个时候无所事事怎么办？

To overcome these worries, shift the mindset from the micro to the macro. Ask yourself: How will we feel if we ship this project after six weeks? Will we feel good about what we accomplished? When projects ship on time and everyone feels they made progress, that's the success. It doesn't matter what exactly happened down at the scale of hours or days along the way. It's the outcome that matters.

要克服这些忧虑，就要把心态从微观转向宏观。问问你自己：如果我们在六周后发布这个项目，我们会有什么感觉？我们会对我们完成的事情感觉良好吗？当项目按时完成，每个人都觉得自己取得了进步，这就是成功。在这个过程中，在小时或者天的范围内究竟发生了什么并不重要。重要的是结果。

# Glossary

## 词汇表

### *Appetite*

#### 食欲

The amount of time we want to spend on a project, as opposed to an estimate.

我们想在一个项目上花费的时间，而不是估计的时间。

### *Baseline*

#### 基线

What customers are doing without the thing we're currently building.

如果没有我们现在正在建造的东西，客户会怎么做。

### *Bet*

#### 打赌

The decision to commit a team to a project for one cycle with no interruptions and an expectation to finish.

决定让一个团队参与一个项目一个周期，没有中断，期望完成。

### *Betting table*

#### *Bet table* 投注表

A meeting during cool-down when stakeholders decide which pitches to bet on in the next cycle.

在冷静阶段召开的会议，当利益相关者决定在下一个周期中押注哪个项目时。

### *Big batch*

#### 大批量

One project that occupies a team for a whole cycle and ships at the end.

一个项目占用了一个团队的整个周期，并在最后发布。

### ***Breadboard***

**面包板**

A UI concept that defines affordances and their connections

without visual styling.

一个用户界面概念，它定义了无视觉样式的提供及其连接。

### ***Circuit breaker***

**断路器**

A risk management technique: Cancel projects that don't ship in one

cycle by default instead of extending them by default.

一种风险管理技术：取消那些默认情况下不会在一个周期内发

布的项目，而不是默认情况下扩展它们。

### ***Cleanup mode***

**Cleanup 模式**

The last phase of building a new product, where we don't shape or bet

on any particular projects but instead allocate unstructured time to fix

whatever is needed before launch.

构建一个新产品的最后阶段，我们不会对任何特定的项目进行定型或押注，而是在发布之前分配非结构化的时间来修复任何需要的东西。

### *Cool-down*

冷静一下

A two-week break between cycles to do ad-hoc tasks, fix bugs, and hold a betting table.

在两个周期之间休息两周，做一些特别的任务，修复 bug，并且举行一个赌桌。

### *Cycle*

循环

A six week period of time where teams work uninterruptedly on shaped projects.

一个为期六周的时间段，团队不间断地完成有形的项目。

### *De-risk*

降低风险

Improve the odds of shipping within one cycle by shaping and removing rabbit holes.

通过塑形和移除兔子洞来提高一个周期内运输的可能性。

### *Discovered tasks*

已发现的任务

Tasks the team discovers they need to do after they start getting involved in the real work.

团队在开始参与真正的工作后发现他们需要做的任务。

### *Downhill*

下坡

The phase of a task, scope or project where all unknowns are solved and only execution is left.

任务、范围或项目的阶段，所有未知的问题都得到解决，只剩下执行。

### *Fat marker sketch*

脂肪标记素描

A sketch of a UI concept at very low fidelity drawn with a thick line.

一个用户界面概念的草图，在非常低的保真度下用粗线绘制。

### ***Hill chart***

**山形图**

A diagram showing the status of work on a spectrum from unknown to known to done.

一个图表显示了从未知到已知完成的工作状态。

### ***Iceberg***

**冰山**

A scope of work where the back-end work is much more complex than the UI or vice versa.

后端工作比用户界面复杂得多的工作范围，反之亦然。

### ***Imagined tasks***

**想象中的任务**

Work the teams decide they need to do after just thinking about the project. See discovered tasks.

工作团队在考虑完项目之后决定他们需要做的工作。

## ***Layer cake*** **分层蛋糕**

A scope of work you can estimate by looking at the surface area of the UI.

您可以通过查看 UI 的表面区域来估计工作范围。

## ***Level of abstraction*** **抽象层次**

The amount of detail we leave in or out when describing a problem or solution.

在描述一个问题或解决方案时，我们忽略或省略的细节数量。

## ***Must-haves*** **必备品**

Tasks that must be completed for a scope to be considered done.  
必须完成的任务，才能认为已完成的范围。

## ***Nice-to-haves*** ***Nice-to-have 美好的拥有***

Task left for the end of the cycle. If there isn't time to do them, they get cut. Marked with a ‘~’ at the beginning.

留给周期结束的任务。如果没有时间做，他们就会被淘汰。开头用“~”标记。

## ***Pitch*** **音高**

A document that presents a shaped project idea for consideration at the betting table.

一个文件，提出了一个成形的项目想法，以供在投注表上考虑。

## ***Production mode*** **生产模式**

A phase of building a new product where the core architecture is settled and we apply the standard Shape Up process.

构建新产品的一个阶段，其中核心架构已经确定，我们应用标准的 Shape Up 过程。

### ***Rabbit hole***

**兔子洞**

Part of a project that is too unknown, complex, or open-ended to bet on.

项目的一部分，太不为人所知，太复杂，或者无法确定。

### ***R&D mode***

**R & d 模式**

A phase of building a new product where a senior team spikes the core features to define the core architecture.

构建新产品的一个阶段，其中高级团队突出核心特性来定义核心体系结构。

### ***Raw ideas***

**原始想法**

Requests or feature ideas that are expressed in words and haven't been shaped.

用语言表达的、尚未形成的请求或特征思想。

### *Scopes*

#### **瞄准镜**

Parts of a project that can be built, integrated, and finished independently of the rest of the project.

独立于项目其余部分的可建造、集成和完成的项目部分。

### *Scope hammering*

#### **瞄准镜敲击声**

Forcefully questioning a design, implementation, or use case to cut scope and finish inside the fixed time box.

强烈质疑设计、实现或用例，以缩小范围并在固定的时间框内完成。

### *Shape*

#### **形状**

Make an abstract project idea more concrete by defining key elements of the solution before betting on it.

在下注之前，先定义解决方案的关键要素，使抽象的项目想法更加具体。

### *Six weeks*

#### **六个星期**

The length of our cycles. Six weeks is long enough to finish something meaningful and short enough to feel the deadline from the beginning.

我们周期的长度。六周的时间足以完成一些有意义的事情，而且从一开始就能感觉到最后期限的到来。

### *Small batch*

#### **小批量**

A set of 1-2 week projects that a single team ships by the end of a six week cycle.

一组为期 1-2 周的项目，由一个单独的团队在六周的周期结束时发货。

***Time horizon***

**时间界限**

The longest period of time where we can feel a deadline pushing on us from the beginning. Six weeks.

最长的一段时间，我们可以感觉到最后期限从一开始就迫在眉睫。

六周。

***Uphill***

**上坡**

The phase of a task, scope or project where there are still unknowns or unsolved problems. See *downhill*.

任务、范围或项目的阶段，仍然有未知或未解决的问题。参见下坡。

# About the Author

## 关于作者

Ryan Singer has worked on all levels of the software stack, from UI design to back-end programming to strategy.

从 UI 设计到后端编程再到策略，Ryan Singer 在软件堆栈的各个层面都有工作经验。

Through over 17 years at Basecamp, he's designed features used by millions and invented processes the teams use to design, develop, and ship the right things.

在 Basecamp 超过 17 年的时间里，他设计了数百万人使用的特性，发明了团队用来设计、开发和运送正确东西的流程。

These days he's focused on product strategy: understanding what Basecamp's customers are trying to do and how to make the product fit them better.

这些天他专注于产品策略：了解 Basecamp 的客户想要做什么，以及如何使产品更适合他们。

Thanks for reading. I'd love to hear about your questions and experiences as you try to apply the Shape Up method. Reach out to me on Twitter at @rjs or send an email to [shapeup@basecamp.com](mailto:shapeup@basecamp.com).

感谢阅读。当你尝试使用塑形方法时，我很想听听你的问题和经验。通过 Twitter@rjs 联系我，或者给 [shapeup@basecamp.com](mailto:shapeup@basecamp.com) 发邮件。

176  
176年