CHAPTER 6

Leading at Scale

第六章 規模優先

In Chapter 5, we talked about what it means to go from being an “individual contributor” to being an explicit leader of a team. It’s a natural progression to go from leading one team to leading a set of related teams, and this chapter talks about how to be effective as you continue along the path of engineering leadership.

在第五章，我們討論了從"個人貢獻者 "到一個團隊的領導意味著什麼。從領導一個團隊轉變到一系列的相關團隊是一個很自然的過程，這章我們將討論如何在管理工程團隊中持續保持高效。

As your role evolves, all the best practices still apply. You’re still a “servant leader”; you’re just serving a larger group. That said, the scope of problems you’re solving becomes larger and more abstract. You’re gradually forced to become “higher level.” That is, you’re less and less able to get into the technical or engineering details of things, and you’re being pushed to go “broad” rather than “deep.” At every step, this process is frustrating: you mourn the loss of these details, and you come to realize that your prior engineering expertise is becoming less and less relevant to your job. Instead, your effectiveness depends more than ever on your general technical intuition and ability to galvanize engineers to move in good directions.

隨著你角色的變化，之前的最佳實踐仍適用。你仍然是一個服務型領導；你只不過是開始服務於更大的團隊了。這就是說，需要你解決的問題領域更大，更抽象了。你被迫上升到了"更高層次"。你越來越不太能接觸到具體的技術上的或工程上的細節，被迫地，你需要知道的更“廣泛”，而不是更“深入”。這個過程的每一步都令人沮喪：你喪失了對技術細節的掌控，然後漸漸意識到之前的工程經驗與你現在的工作的關聯性越來越少。你的工作效率變得更加依賴對通用的技術領域的直覺和引導工程師找對正確的前進方向上。

The process is often demoralizing—until one day you notice that you’re actually having much more impact as a leader than you ever had as an individual contributor. It’s a satisfying but bittersweet realization.

這個過程通常是令人沮喪的 -- 直到有一天你意識到作為一個領導者你比作為個人貢獻者有多得多的影響力。這是個令人滿意的，但是也是喜憂參半的領悟。

So, assuming that we understand the basics of leadership, what it does it take to scale yourself into a really good leader? That’s what we talk about here, using what we call “the three Always of leadership”: Always Be Deciding, Always Be Leaving, Always Be Scaling.

至此，假設我們已經知道了領導的本質，那麼到底什麼才能讓你提升為一個真正優秀的管理者呢？這就是我們這裡想要討論的，我們稱之為“管理上的三個總是”：始終保持決斷力，始終保持離開，始終保持擴張。

Always Be Deciding 始終保持決斷力

Managing a team of teams means making ever more decisions at ever-higher levels. Your job becomes more about high-level strategy rather than how to solve any specific engineering task. At this level, most of the decisions you'll make are about finding the correct set of trade-offs.

管理團隊組成的團隊意味著在更高層面上做決定。你的工作從解決具體的工程任務變成制定更高的策略。在這個層面上，你將做的決策大多數是關於更好地權衡。

The Parable of the Airplane 關於機場的寓言故事

Lindsay Jones is a friend of ours who is a professional theatrical sound designer and composer. He spends his life flying around the United States, hopping from production to production, and he's full of crazy (and true) stories about air travel. Here's one of our favorite stories:

Lindsay Jones 是我們的一個專業的戲劇聲音設計師和編曲朋友。他一生大部分時間都在美國各地飛來飛去，在不同的作品之間切換。他有很多瘋狂（而且真實）的關於航空旅行的故事。下面是我們最喜歡的一個故事：

> It's 6 a.m., we're all boarded on the plane and ready to go. The captain comes on the PA system and explains to us that, somehow, someone has overfilled the fuel tank by 10,000 gallons. Now, I've flown on planes for a long time, and I didn't know that such a thing was possible. I mean, if I overfill my car by a gallon, I'm gonna have gas all over my shoes, right?
>
> 現在是早上 6 點，我們都登機了，飛機馬上就要起飛。機長在廣播中跟我們解釋說，不知怎麼的有人給飛機多加了 10,000 加侖汽油。我已經在飛過很久了，從沒聽說過怎麼可能發生這種事。我是說，如果我給汽車多加 1 加侖的汽油，很可能我的鞋裡都會灌滿汽油，是吧？
>
> Well, so anyway, the captain then says that we have two options: we can either wait for the truck to come suck the fuel back out of the plane, which is going to take over an hour, or twenty people have to get off the plane right now to even out the weight.
> No one moves.
>
> 好吧，無論如何，機長說我們有兩個選擇：要麼我們等 1 個多小時等卡車把多餘的汽油吸走，要麼請 12 名乘客下飛機來減輕飛機的重量。
> 沒有人下飛機。
>
> Now, there's this guy across the aisle from me in first class, and he is absolutely livid. He reminds me of Frank Burns on *MAS*H*; he's just super indignant and sputtering everywhere, demanding to know who's responsible. It's an amazing showcase, it's like he's Margaret Dumont in the Marx Brothers movies.
>
> 現在，在我頭等艙的過道對面有一個人，是真的很生氣了。他讓給我想起了**MASH**裡的 Frank Burns；他現在是真的很生氣，在到處嚷嚷，想知道到底誰該為此負責。這是個非常精彩的案例，簡直就像電影 Marx Brothers 中的 Margaret Dumont 一樣。

So, he grabs his wallet and pulls out this massive wad of cash! And he’s like “I cannot be late for this meeting!! I will give $40 to any person who gets off this plane right now!”

然後，他從錢包裡拿出一大沓鈔票！他說“我這個會不能遲到！現在誰下飛機我就給誰 40 美金！”

Sure enough, people take him up on it. He gives out $40 to 20 people (which is $800 in cash, by the way!) and they all leave.

當然人們很買他的帳，他給了 20 人每人 40 美金（總共 800！），然後這些人都下飛機了。

So, now we’re all set and we head out to the runway, and the captain comes back on the PA again. The plane’s computer has stopped working. No one knows why. Now we gotta get towed back to the gate.

然後我們都坐下了，飛機出發準備到跑道了，結果機長又在廣播上開始說話了。飛機的電腦宕機了。沒人知道為什麼。現在我們必須被拖回登機口。

Frank Burns is apoplectic. I mean, seriously, I thought he was gonna have a stroke. He’s cursing and screaming. Everyone else is just looking at each other.

Frank Burns 現在真的暴跳如雷了。真的，我感覺他快要氣抽過去了。他開始咒罵和尖叫。其他人都開始面面相覷。

We get back to the gate and this guy is demanding another flight. They offer to book him on the 9:30, which is too late. He’s like, “Isn’t there another flight before 9:30?”

我們回到登機口，這個人要求另一個航班。他們提議給他訂 9:30 的航班，但這已經太晚了。他說，“9 點半前沒有其他的航班嗎？”

The gate agent is like, “Well, there was another flight at 8, but it’s all full now. They’re closing the doors now.”

登機口工作人員說："嗯，8 點還有一個航班，但現在都滿了。他們現在要關門了。"

And he’s like, “Full?! Whaddya mean it’s full? There’s not one open seat on that plane?!?!?!”

他就說，"滿了？你說滿了是什麼意思？那架飛機上沒有一個空位？

The gate agent is like, “No sir, that plane was wide open until 20 passengers showed up out of nowhere and took all the seats. They were the happiest passengers I’ve ever seen, they were laughing all the way down the jet bridge.”

空乘說，“不，這趟航班本來是有空餘座位的，直到不知從哪冒出了 20 名乘客坐滿了所有位置。他們是我見過的最高興的乘客，他們一路有說有笑走下了廊橋。”

It was a very quiet ride on the 9:30 flight.

後來 9 點半的這趟航班一路上都很安靜。

This story is, of course, about trade-offs. Although most of this book focuses on various technical trade-offs in engineering systems, it turns out that trade-offs also apply to human behaviors. As a leader, you need to make decisions about what your teams should do each week. Sometimes the trade-offs are obvious (“if we work on this project, it delays that other one...”); sometimes the trade-offs have unforeseeable consequences that can come back to bite you, as in the preceding story.

這個故事是關於權衡(Trade-Offs)的。儘管這本書的大部分內容都聚焦在講工程系統中的技術權衡，但是事實證明在人類行為方面同樣適用。作為一個領導，你需要決定你的團隊每週都做什麼。有時權衡很明顯（“如果我們做這個專案，那另一個專案可能會延期”）；還有的時候這些權衡會有不可預料的結果，回過頭來會反咬你一口，就像前面的故事。

At the highest level, your job as a leader—either of a single team or a larger organization—is to guide people toward solving difficult, ambiguous problems. By ambiguous, we mean that the problem has no obvious solution and might even be unsolvable. Either way, the problem needs to be explored, navigated, and (hopefully) wrestled into a state in which it’s under control. If writing code is analogous to chopping down trees, your job as a leader is to “see the forest through the trees” and find a workable path through that forest, directing engineers toward the important trees. There are three main steps to this process. First, you need to identify the blinders; next, you need to identify the trade-offs; and then you need to decide and iterate on a solution.

在最高層，你的工作是作為一個領導——一個小團隊或一個更大的組織--引導人們解決棘手的、模糊的問題。模糊意味著這個問題沒有顯而易見的解法，甚至可能沒有解法。另一方面，這個問題需要被探索、指引、摸爬滾打到一個可控的狀態下。如果把寫程式碼比作砍樹的話，你作為一個領導的工作就是“撥開樹木見森林”，找到穿越森林的路徑，指引工程師找到最重要的樹。首先，你需要找到專家；然後識別權衡；然後在解決方案上個反覆地決定並迭代。

Identify the Blinders 找到盲點

When you first approach a problem, you’ll often discover that a group of people has already been wrestling with it for years. These folks have been steeped in the problem for so long that they’re wearing “blinders”—that is, they’re no longer able to see the forest. They make a bunch of assumptions about the problem (or solution) without realizing it. “This is how we’ve always done it,” they’ll say, having lost the ability to consider the status quo critically. Sometimes, you’ll discover bizarre coping mechanisms or rationalizations that have evolved to justify the status quo. This is where you —with fresh eyes—have a great advantage. You can see these blinders, ask questions, and then consider new strategies. (Of course, being unfamiliar with the problem isn’t a requirement for good leadership, but it’s often an advantage.)

當你初次接觸一個問題時，通常你會發現有很多人已經在這個領域摸爬滾打很多年了。這些傢伙在這個領域呆了很久，以至於他們好像戴著*眼罩*——他們無法“撥開樹木見森林”。對於這個問題（或解決方案），他們會做一系列的假設，但從不會去重新認識這個問題本身。他們會說，“我們一直是這樣做的”，他們已經失去了思考問題現狀的能力。有時，你會發現一些奇怪的應對機制或合理化建議，這些都是為了證明現狀的合理性而演變的。這就是你的優勢所在——你有一雙新的眼睛。你可以看到這些盲點，提出問題，然後考慮新的策略。（當然，對問題不熟悉並不是好領導的要求，但它往往是一種優勢。）

Identify the Key Trade-Offs 確定關鍵的權衡要素

By definition, important and ambiguous problems do not have magic “silver bullet” solutions. There’s no answer that works forever in all situations. There is only the best answer for the moment, and it almost certainly involves making trade-offs in one direction or another. It’s your job to call out the trade-offs, explain them to everyone, and then help decide how to balance them.

根據定義，重要的和模糊的問題沒有所謂的神奇的“銀彈”解決方案。沒有一勞永逸的解決方案。只有當下的最佳答案，而且幾乎肯定涉及到在某個方向上的權衡。你的工作是指出這些權衡，向大家解釋，然後幫助決定如何取捨。

Decide, Then Iterate 做決定，然後反覆迭代

After you understand the trade-offs and how they work, you’re empowered. You can use this information to make the best decision for this particular month. Next month, you might need to reevaluate and rebalance the trade-offs again; it’s an iterative process. This is what we mean when we say Always Be Deciding.

在你瞭解了這些權衡以及它們如何運作之後，你就被賦能了。這個月，你能利用這個資訊來做最佳的決定，然後下個月你可能需要重新評估和權衡——這是一個反覆迭代的過程。這就是我們所說的“始終保持決斷力”。

There’s a risk here. If you don’t frame your process as continuous rebalancing of trade-offs, your teams are likely to fall into the trap of searching for the perfect solution, which can then lead to what some call “analysis paralysis.” You need to make your teams comfortable with iteration. One way of doing this is to lower the stakes and calm nerves by explaining: “We’re going to try this decision and see how it goes. Next month, we can undo the change or make a different decision.” This keeps folks flexible and in a state of learning from their choices.

不過這裡也有風險。如果你不給你的分析過程框定一個邊界的話，你的團隊容易陷到尋找“最完美的解決方法”的漩渦中，這樣使團隊進入所謂的“分析癱瘓”的狀態中。你需要使你的團隊習慣於這個迭代過程。一種方法是每次把這種變更和風險降低，然後嘗試給團隊成員解釋“我們將嘗試這個方案看看效果，然後下個月我們可能撤銷這個變更或做出不同的決定”，來使他們冷靜下來。這將使團隊成員變得敏捷，並能從他們的選擇中得到成長並進步。

Case Study: Addressing the “Latency” of Web Search 案例學習: 定位網頁搜尋的“延遲”

In managing a team of teams, there’s a natural tendency to move away from a single product and to instead own a whole “class” of products, or perhaps a broader problem that crosses products. A good example of this at Google has to do with our oldest product, Web Search.

在管理一個團隊的過程中，有一個很自然的趨勢，那就是走出單一的產品，轉而擁有多元產品的 "類別"，或者也許是一個跨越產品的更廣泛的問題。這方面一個很好的例子就是 Google 歷史最有悠久的產品，網頁搜尋。

For years, thousands of Google engineers have worked on the general problem of making search results better—improving the “quality” of the results page. But it turns out that this quest for quality has a side effect: it gradually makes the product slower. Once upon a time, Google’s search results were not much more than a page of 10 blue links, each representing a relevant website. Over the past decade, however, thousands of tiny changes to improve “quality” have resulted in ever-richer results: images, videos, boxes with Wikipedia facts, even interactive UI elements. This means the servers need to do much more work to generate information: more bytes are being sent over the wire; the client (usually a phone) is being asked to render ever-more-complex HTML and data. Even though the speed of networks and computers have markedly increased over a decade, the speed of the search page has become slower and slower: its latency has increased. This might not seem like a big deal, but the latency of a product has a direct effect (in aggregate) on users' engagement and how often they use it. Even increases in rendering time as small as 10 ms matter. Latency creeps up slowly. This is not the fault of a specific engineering team, but rather represents a long, collective poisoning of the commons. At some point, the overall latency of Web Search grows until its effect begins to cancel out the improvements in user engagement that came from the improvements to the "quality" of the results.

多年以來，數以千計的工程師為提升搜尋結果頁的"品質"做了很多最佳化。結果發現對內容品質的追求也有兩面性：它逐漸使產品變得緩慢。很久以前 Google 的搜尋頁每頁只有不到 10 個藍色的連結，每個連結代表一個相關網頁。再過去的十年間，上千個關於"品質"的微小的變化導致搜尋結果變成了一個前所未有的複雜的結果：圖片、影片、有維基百科結果的文字框、甚至還有可互動的 UI 元件。這意味著伺服器需要做更多的工作來產生這些資訊：在網路上傳輸更多的資料；客戶端（通常是手機）需要渲染更複雜的 HTML 和內容。儘管網路和計算機在近十年都有了飛速的提升，搜尋結果頁速度還是越來越慢了：延遲增加了。這看上去沒什麼大不了，但這直接影響著使用者粘性。即使將網頁渲染時間增加 10ms 都有影響。延遲總是一點點地增加。這並不是某一個工程團隊的問題，而是一個跨團隊的，長期的慢性毒藥。從某些方面看，網頁的總延遲將會一直增加，直到它帶來的副作用能夠抵消掉品質最佳化為使用者粘性帶來的收益。

A number of leaders struggled with this issue over the years but failed to address the problem systematically. The blinders everyone wore assumed that the only way to deal with latency was to declare a latency "code yellow"[1] every two or three years, during which everyone dropped everything to optimize code and speed up the product. Although this strategy would work temporarily, the latency would begin creeping up again just a month or two later, and soon return to its prior levels.

多年裡，多位領導都嘗試過系統性地解決這個問題然而最終都以失敗告終。每個專家都是隻有一個解法，那就是制定一個延遲的"黃線"，每一到兩年就檢查一次，如果延遲到達了"黃色代號"，每個人都停下來最高優先順序最佳化程式碼來給產品提速。儘管這個策略會短時間的生效，但是僅僅一兩個月後，延遲就又會慢慢增加，然後很快就又回到之前的水平。

So what changed? At some point, we took a step back, identified the blinders, and did a full reevaluation of the trade-offs. It turns out that the pursuit of "quality" has not one, but two different costs. The first cost is to the user: more quality usually means more data being sent out, which means more latency. The second cost is to Google: more quality means doing more work to generate the data, which costs more CPU time in our servers—what we call "serving capacity." Although leadership had often trodden carefully around the trade-off between quality and capacity, it had never treated latency as a full citizen in the calculus. As the old joke goes, "Good, Fast, Cheap—pick two." A simple way to depict the trade-offs is to draw a triangle of tension between Good (Quality), Fast (Latency), and Cheap (Capacity), as illustrated in Figure 6-1.

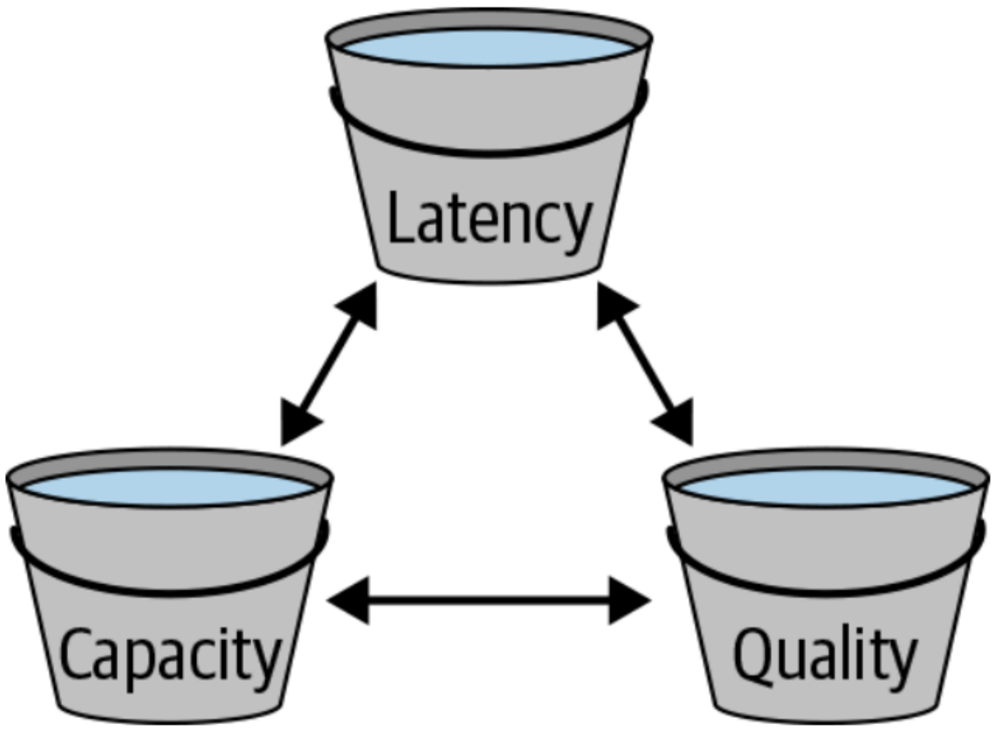

Figure 6-1. Trade-offs within Web Search; pick two! 圖 6-1. 網路搜尋中的權衡；選擇兩個!

那麼什麼變了呢？在某種程度上，我們退後一步，確定了盲點，並對權衡做了全面的重新評估。結果證明對於“品質”的追求，有不是一個，而是兩方面的開銷。第一個開銷是對使用者的：更好的品質通常意味著需要傳輸更多的資料，也就意味著更多的延遲。第二方面的開銷在 Google 本身：更好的品質意味著需要更多的工作來產生這些資料，這將消耗我們更多的伺服器 CPU 時間，也就是我們說的 “服務容量”。儘管管理者曾經仔細地在品質和容量之間來回踱步，“延遲”從未被當做一等公民對待。就像老話說的“好、快、便宜只能選兩個”（魚和熊掌不可兼得？）。描繪這其中的權衡的最好的方式就是畫一個這三者之間的三角形：好（品質），快（延遲）和便宜（容量），如下圖 6-1 所示。

That’s exactly what was happening here. It’s easy to improve any one of these traits by deliberately harming at least one of the other two. For example, you can improve quality by putting more data on the search results page—but doing so will hurt capacity and latency. You can also do a direct trade-off between latency and capacity by changing the traffic load on your serving cluster. If you send more queries to the cluster, you get increased capacity in the sense that you get better utilization of the CPUs—more bang for your hardware buck. But higher load increases resource contention within a computer, making the average latency of a query worse. If you deliberately decrease a cluster’s traffic (run it “cooler”), you have less serving capacity overall, but each query becomes faster.

這正是這裡在發生的情況。透過損壞另外一個或兩個特性，改善其中的一個特性很容易，比如說，你可以透過在搜尋結果頁展示更多的內容來提升搜尋品質，但這將會損害容量和延遲。透過改變系統負載，你還可以在延遲和容量之間做一次權衡。如果你向叢集傳送更多的查詢，你就會得到更多的容量，也就是說，你可以更好地利用 CPU--為你的硬體付出更多。但是，更高的負載增加了計算機內的資源爭奪，使查詢的平均延遲變高。如果你故意減少叢集的流量（"冷卻 "執行），你的整體服務能力就會減少，但每個查詢都會變得更快。

The main point here is that this insight—a better understanding of all the trade-offs—allowed us to start experimenting with new ways of balancing. Instead of treating latency as an unavoidable and accidental side effect, we could now treat it as a first-class goal along with our other goals. This led to new strategies for us. For example, our data scientists were able to measure exactly how much latency hurt user engagement. This allowed them to construct a metric that pitted quality-driven improvements to short-term user engagement against latency-driven damage to long-term user engagement. This approach allows us to make more data-driven decisions about product changes. For example, if a small change improves quality but also hurts latency, we can quantitatively decide whether the change is worth launching or not. We are always deciding whether our quality, latency, and capacity changes are in balance, and iterating on our decisions every month.

這裡的核心點是下面的這個洞察力--對所有權衡的更好理解--使我們能夠開始嘗試新的平衡方式。與其說將延遲作為一個不可避免的意外副作用，我們現在可以將它與其他目標一樣，看做一等目標。這將引領我們採用新的策略。例如我們的資料科學家能夠準確地測量出延遲對使用者參與度的損害程度。這使他們能夠為延遲建構一個指標放入品質驅動的指標體系中，標識其有助於提升短期使用者粘性但是對長期提升使用者粘性有害。例如，如果一個小的改動能夠提升品質但同時影響延遲，我們可能需要客觀地從數值上判斷這個改動是否值得釋出。在對改動做評估時，我們將一直追求在品質、延遲、容量保持平衡，並且每個月都會對我們的決定進行迭代。

> 1 "Code yellow"是 Google 的術語，指的是 "緊急駭客馬拉松，以修復一個關鍵問題"。受影響的團隊被要求暫停所有工作，並將 100%的注意力集中在這個問題上，直到緊急狀態被宣佈結束。

Always Be Leaving 始終保持離開

At face value, Always Be Leaving sounds like terrible advice. Why would a good leader be trying to leave? In fact, this is a famous quote from Bharat Mediratta, a former Google engineering director. What he meant was that it's not just your job to solve an ambiguous problem, but to get your organization to solve it by itself, without you present. If you can do that, it frees you up to move to a new problem (or new organization), leaving a trail of self-sufficient success in your wake.

從字面意思上看，"始終保持離開" 聽上去是一個可怕的建議。為什麼一個好的領導者要嘗試離開他的團隊呢？事實上這是參考的前 Google 工程主管 Bharat Mediratta 的話。他的意思是你的任務不僅僅是解決邊界不清晰的問題，而是還要引導你的組織在沒有你在場的情況下自己解決問題。如果你能做到這點，將釋放你一部分精力去解決新的問題（或去管理新的組織），在你身後留下一個個能自給自足的團隊。

The antipattern here, of course, is a situation in which you've set yourself up to be a single point of failure (SPOF). As we noted earlier in this book, Googlers have a term for that, the bus factor: the number of people that need to get hit by a bus before your project is completely doomed.

這裡的反模式是，把你自己置為單點故障的情況。就像我們在本書前面說的那樣，Google 員工有一個個專用短語來說明這個情況——“巴士因子”：在你的專案完全失敗之前，需要有多少人被公共汽車撞倒。

Of course, the “bus” here is just a metaphor. People become sick; they switch teams or companies; they move away. As a litmus test, think about a difficult problem that your team is making good progress on. Now imagine that you, the leader, disappear. Does your team keep going? Does it continue to be successful? Here’s an even simpler test: think about the last vacation you took that was at least a week long. Did you keep checking your work email? (Most leaders do.) Ask yourself why. Will things fall apart if you don’t pay attention? If so, you have very likely made yourself an SPOF. You need to fix that.

當然，這裡的“巴士”是一個隱喻。人們會生病，會換團隊或公司，會離職。作為一個試金石，可以想想一個你團隊正在嘗試解決並取得了良好進展的難題。然後想象作為領導的你消失了。你的團隊還能夠繼續前進嗎？它還能繼續成功嗎？還有一個更簡單的測試：想想上一次你休超過一週的假期的時候。你是在不停地檢視郵件嗎？（大多數管理者會）然後問問你自己為什麼。如果你不注意，事情會一團糟嗎。如果是，你很可能把你自己置於“單點故障”的境地。你需要解決這個問題。

Your Mission: Build a “Self-Driving” Team 你的使命：打造一個“自我驅動”的團隊

Coming back to Bharat’s quote: being a successful leader means building an organization that is able to solve the difficult problem by itself. That organization needs to have a strong set of leaders, healthy engineering processes, and a positive, self-perpetuating culture that persists over time. Yes, this is difficult; but it gets back to the fact that leading a team of teams is often more about organizing people rather than being a technical wizard. Again, there are three main parts to constructing this sort of self-sufficient group: dividing the problem space, delegating subproblems, and iterating as needed.

讓我們回到參考的 Bharat 的話：做一個成功的管理者意味著建構一個能夠獨自解決問題的組織。這個組織需要有一套強有力的領導，健康的工程流程，一個積極的，能夠自我延續，經時間沉澱的文化。是的，這很難；但是這回歸到了事情的本質，領導團隊的通常更多地意味著管理人，而不是作為一個技術嚮導。再強調一次，這種自給自足的團隊有三個主要的組成部分：劃分問題域，委託子任務，以及對於不足的地方反覆迭代。

Dividing the Problem Space 劃分問題域

Challenging problems are usually composed of difficult subproblems. If you’re leading a team of teams, an obvious choice is to put a team in charge of each subproblem. The risk, however, is that the subproblems can change over time, and rigid team boundaries won’t be able to notice or adapt to this fact. If you’re able, consider an organizational structure that is looser—one in which subteams can change size, individuals can migrate between subteams, and the problems assigned to subteams can morph over time. This involves walking a fine line between “too rigid” and “too vague.” On the one hand, you want your subteams to have a clear sense of problem, purpose, and steady accomplishment; on the other hand, people need the freedom to change direction and try new things in response to a changing environment.

有挑戰的問題通常由許多有困難的子問題組成。如果你在管理一個由團隊組成的團隊，一個顯而易見的做法是讓每個團隊負責一個子問題。然而這樣做的風險是問題會隨著時間而改變，一個死板的團隊邊界可能不能夠察覺或適應這種情況。如果你能夠決定，可以嘗試建構一個組織結構鬆散的團隊，它能夠動態地調整團隊規模，員工能夠在子團隊直接切換，而且每個團隊處理的子問題能夠切換。這意味著要在太死板和太鬆散的邊緣遊走。一方面，你想要你的團隊能夠對問題和目標有一個清晰的認知，能夠有較高的完成度；另一方面，人們需要能自由的切換方向來嘗試新鮮事物，來應對不斷變化的環境。

Example: Subdividing the “latency problem” of Google Search 例子： 細分 Google 搜尋的“延遲問題”

When approaching the problem of Search latency, we realized that the problem could, at a minimum, be subdivided into two general spaces: work that addressed the symptoms of latency, and different work that addressed the causes of latency. It was obvious that we needed to staff many projects to optimize our codebase for speed, but focusing only on speed wouldn’t be enough. There were still thousands of engineers increasing the complexity and “quality” of search results, undoing the speed improvements as quickly as they landed, so we also needed people to focus on a parallel problem space of preventing latency in the first place. We discovered gaps in our metrics, in our latency analysis tools, and in our developer education and documentation. By assigning different teams to work on latency causes and symptoms at the same time, we were able to systematically control latency over the long term. (Also, notice how these teams owned the problems, not specific solutions!)

當開始接觸搜尋延遲的問題時，我們意識到在最小粒度下我們可以將問題大體劃分為兩個方面：一方面是定位延遲的現象，另一方面是挖掘延遲的根本原因。很顯然我們需要為很多團隊配備人員來最佳化專案的程式碼庫的效能問題，但僅僅關注效能是不夠的。與此同時仍有數千名工程師增加系統的複雜度和搜尋結果的“品質”，使對系統延遲的最佳化剛上線就被抵消掉了。因此我們還需要人關注一個平行的問題域，從一開始就防止增加延遲的變更。我們發現在我們的監控指標、延遲分析工具以及員工培訓和文件中都存在有差距。透過在同一時間將延遲定位和原因分析分配給不同團隊，我們能夠長期系統性地控制延遲問題。（同時，這裡需要關注的是這些團隊是如何管理這些問題的，而不是關注具體的解決方案！）

Delegating subproblems to leaders 將子問題授權給子團隊領導

It’s essentially a cliché for management books to talk about “delegation,” but there’s a reason for that: delegation is really difficult to learn. It goes against all our instincts for efficiency and achievement. That difficulty is the reason for the adage, “If you want something done right, do it yourself.”

對於管理類書籍來說，“授權”有點陳詞濫調了，但是這背後其實有一個原因：授權太難學了。對於效率或是成功，它幾乎是反直覺的。這個困難的原因就如諺語說的那樣“如果你想做對某件事，那就自己去做”。

That said, if you agree that your mission is to build a self-driving organization, the main mechanism of teaching is through delegation. You must build a set of self-sufficient leaders, and delegation is absolutely the most effective way to train them. You give them an assignment, let them fail, and then try again and try again. Silicon Valley has well-known mantras about “failing fast and iterating.” That philosophy doesn’t just apply to engineering design, but to human learning as well.

也就是說，如果你的使命是建構一個自我驅動型的組織，那主要的方法就是透過授權。你必須培養出一系列的能夠自我成長、自給自足的領導，允許他們失敗，然後一遍又一遍的嘗試。矽谷有一個有名的咒語“快速失敗，然後反覆迭代。”這個理論不僅適用於工程設計方面，同樣也適用於人類學習。

As a leader, your plate is constantly filling up with important tasks that need to be done. Most of these tasks are things that are fairly easy for you do. Suppose that you’re working diligently through your inbox, responding to problems, and then you decide to put 20 minutes aside to fix a longstanding and nagging issue. But before you carry out the task, be mindful and stop yourself. Ask this critical question: Am I really the only one who can do this work?

作為一個領導，你的桌面上經常擺滿了各種重要的問題需要被解決。大多數對於你來說都很簡單。假設你正在努力地處理來自收件箱的問題，然後決定花 20 分鐘處理一個長期困擾的問題。但是在你開始著手這件任務前，先停下來想想。問自己這個關鍵的問題：你是唯一一個能解決這個問題的人嗎？

Sure, it might be most efficient for you to do it, but then you're failing to train your leaders. You're not building a self-sufficient organization. Unless the task is truly time sensitive and on fire, bite the bullet and assign the work to someone else—presumably someone who you know can do it but will probably take much longer to finish. Coach them on the work if need be. You need to create opportunities for your leaders to grow; they need to learn to "level up" and do this work themselves so that you're no longer in the critical path.

當然，你做這個工作可能效率最高，但是這意味著在培訓領導這件事上你正在失敗。你不是在建構一個自給自足型的組織。除非這件事真的是迫在眉睫，你要硬著頭皮把這件工作指派給一個你覺得能夠完成，但可能會花費更多時間的人，並且你需要給與適當的指導。你需要創造機會來讓你的團隊成長；他們需要學著"升級"，然後自己解決這個問題，這樣你就不在關鍵路徑上了。

The corollary here is that you need to be mindful of your own purpose as a leader of leaders. If you find yourself deep in the weeds, you're doing a disservice to your organization. When you get to work each day, ask yourself a different critical question: What can I do that nobody else on my team can do?

這裡的推論是，在做領導的領導這件事上，你可能需要格外留心。如果你發現你陷在細節裡陷得太深，你可能正在破壞你的團隊。每天工作時，你要問自己另外一個重要的問題：有什麼事情是除了我以外團隊裡的其他人做不了的？

There are a number of good answers. For example, you can protect your teams from organizational politics; you can give them encouragement; you can make sure everyone is treating one another well, creating a culture of humility, trust, and respect. It's also important to "manage up," making sure your management chain understands what your group is doing and staying connected to the company at large. But often the most common and important answer to this question is: "I can see the forest through the trees." In other words, you can define a high-level strategy. Your strategy needs to cover not just overall technical direction, but an organizational strategy as well. You're building a blueprint for how the ambiguous problem is solved and how your organization can manage the problem over time. You're continuously mapping out the forest, and then assigning the tree-cutting to others.

這個問題又很多很好的答案。你可以避免你的團隊搞辦公室政治；你可以給他們鼓勵；你可以確保每個人都能善待彼此，創造一種謙遜、信任和尊重的文化。同時"向上管理"也很重要，確保你的領導能夠知道你的團隊做的怎麼樣，以及確保在公司規模較大時團隊不與公司脫節。但通常最常見也是最重要的答案是："透過樹木見森林。"換句話說，你能夠指定一個高層次的戰略。你的戰略應該不僅包含技術戰略，還應該包含組織戰略。你在建構一個關於如何解決邊界不清晰的問題，以及如何讓團隊能夠長久地解決問題的藍圖。你持續性地在森林中規劃砍樹的路線，而把砍樹的具體問題交給別人。

Adjusting and iterating 調整與迭代

Let's assume that you've now reached the point at which you've built a self-sustaining machine. You're no longer an SPOF. Congratulations! What do you do now?

讓我們假設現在你已經建構了一個能夠自給自足的團隊，你不再是團隊的單點瓶頸。恭喜！那麼接下來你做什麼呢？

Before answering, note that you have actually liberated yourself—you now have the freedom to "Always Be Leaving." This could be the freedom to tackle a new, adjacent problem, or perhaps you could even move yourself to a whole new department and problem space, making room for the careers of the leaders you've trained. This is a great way of avoiding personal burnout.

在回答這個問題之前，請注意實際上你已經解放了你自己--現在你有"始終保持離開"的自由了。你可以自由地選擇去解決一個新的問題，甚至你可以去到一個全新的部門解決新的問題，來為你培養的其他領導者騰出一些上升空間。這是避免職業生涯倦怠的很好的方法。

The simple answer to “what now?” is to direct this machine and keep it healthy. But unless there’s a crisis, you should use a gentle touch. The book Debugging Teams[2] has a parable about making mindful adjustments:

“現在怎麼辦？”這個問題的一個簡單的回答是引導你的團隊然後讓它持續保持健康。但是除非有很難解決的危機，你就不應該過多地去插手管理團隊了。《進化:從孤膽極客到高效團隊》這本書對於如何做有意義的調整有一個比較好的隱喻：

> There’s a story about a Master of all things mechanical who had long since retired. His former company was having a problem that no one could fix, so they called in the Master to see if he could help find the problem. The Master examined the machine, listened to it, and eventually pulled out a worn piece of chalk and made a small X on the side of the machine. He informed the technician that there was a loose wire that needed repair at that very spot. The technician opened the machine and tightened the loose wire, thus fixing the problem. When the Master’s invoice arrived for $10,000, the irate CEO wrote back demanding a breakdown for this ridiculously high charge for a simple chalk mark! The Master responded with another invoice, showing a $1 cost for the chalk to make the mark, and $9,999 for knowing where to put it.
>
> 有一個關於一位早已退休的機械大師的故事。他的前公司遇到了一個沒人能解決的問題，所以他們請了這個大師來看看能否幫助解決這個問題。大師仔細檢查了機器，並貼近聽了聽。最終他掏出一截粉筆然後在機器側面畫了一個小小的叉。他告訴技術員開啟機器，然後在他打叉的地方有一根電線鬆了需要綁緊。技術員打開了機器然後綁緊了那根電線，然後機器就修好了！當公司收到這位大師的 10,000 美金的賬單後，CEO 大怒並向大師索要賬單明細。然後大師又寄了一張有明細的賬單，上面寫著：做標記用的粉筆 1 美元，知道在哪裡做標記 9,999 美元。
>
> To us, this is a story about wisdom: that a single, carefully considered adjustment can have gigantic effects. We use this technique when managing people. We imagine our team as flying around in a great blimp, headed slowly and surely in a certain direction. Instead of micromanaging and trying to make continuous course corrections, we spend most of the week carefully watching and listening. At the end of the week we make a small chalk mark in a precise location on the blimp, then give a small but critical “tap” to adjust the course.
>
> 對我們來說，這是一個關於智慧的故事：一個經過深思熟慮的細微的調整能夠產生巨大的作用。在管理人員時，我們也使用這個技巧。我們想象我們的團隊在一個巨大的飛艇上飛行，朝著一個特定的方向緩慢而確定地前進。我們不是透過微操作來不斷地修正航向，而是花數週的時間仔細地觀察和傾聽。最終，我們在飛艇上的某一個精確的位置畫一個很小，確至關重要的記號，輕輕一擊，來修正航線。

This is what good management is about: 95% observation and listening, and 5% making critical adjustments in just the right place. Listen to your leaders and skip-reports. Talk to your customers, and remember that often (especially if your team builds engineering infrastructure), your “customers” are not end users out in the world, but your coworkers. Customers’ happiness requires just as much intense listening as your reports’ happiness. What’s working and what isn’t? Is this self-driving blimp headed in the proper direction? Your direction should be iterative, but thoughtful and minimal, making the minimum adjustments necessary to correct course. If you regress into micromanagement, you risk becoming an SPOF again! “Always Be Leaving” is a call to macromanagement.

這個故事說明了好的管理的意義：95% 是觀察和傾聽，5% 是在適當的位置做關鍵的調整。傾聽你的團隊和跳過彙報。和你的客戶聊聊，而且這些客戶經常並不是外部的終端客戶（尤其是當你的團隊是在建構工程化的基礎設施時），而是你的同事。要想讓客戶滿意，就得像認真看報告那樣，認真傾聽你的客戶。什麼有效，什麼無效呢？這個自驅的飛艇的航線正確嗎？你的指引需要是反覆迭代的，但是需要是經過深思熟慮地，透過最小的調整來修正航線。如果你退行到了去過度細節，你需要警惕你可能又成為了單點瓶頸！“始終保持離開”是說要進行宏觀管理。

> 2 Brian W. Fitzpatrick 和 Ben Collins-Sussman，《進化:從孤膽極客到高效團隊》(Boston: O'Reilly, 2016)。

Take care in anchoring a team’s identity 謹慎地確定團隊的定位

A common mistake is to put a team in charge of a specific product rather than a general problem. A product is a solution to a problem. The life expectancy of solutions can be short, and products can be replaced by better solutions. However, a problem — if chosen well—can be evergreen. Anchoring a team identity to a specific solution (“We are the team that manages the Git repositories”) can lead to all sorts of angst over time. What if a large percentage of your engineers want to switch to a new version control system? The team is likely to “dig in,” defend its solution, and resist change, even if this is not the best path for the organization. The team clings to its blinders, because the solution has become part of the team’s identity and self-worth. If the team instead owns the problem (e.g., “We are the team that provides version control to the company”), it is freed up to experiment with different solutions over time.

一個常見的錯誤是讓一個團隊負責一個特定的產品而不是負責解決一類問題。一個產品是一個問題的一種解決方案。一個解決方案的生命週期可能很短，一個產品可能會被更好的方案替代。然而，一個問題（如果這個問題的定位比較合理）卻可以是經久不衰的。將一個團隊定位為一個特定的解決方案（“我們是負責 Git 儲存庫的團隊”）隨著時間的推移將會帶來各種各樣的麻煩。假如很大一部分工程師想切換到一個新的版本控制系統怎麼辦？這個團隊很可能會“鑽牛角尖”，堅持它原有的解決方案，拒絕改變，即使它並不是最適合整個組織的方案。這個團隊依賴它的“觀點”，因為解決方案已經成為團隊身份和自我價值的一部分。如果團隊改為是負責解決這個問題（比方說“我們是為這個公司提供版本管理的團隊”），那麼隨著時間的推移,這個團隊將不再被束縛去做實驗嘗試不同的解決方案。

Always Be Scaling 始終保持擴張

A lot of leadership books talk about “scaling” in the context of learning to “maximize your impact”—strategies to grow your team and influence. We’re not going to discuss those things here beyond what we’ve already mentioned. It’s probably obvious that building a self-driving organization with strong leaders is already a great recipe for growth and success.

很多領導力書籍會在講“最大化你的影響力”的上下文中講“擴張規模”——透過這種策略來增長你的團隊和影響力。由於我們已經提過的原因，我們不會在這裡討論這些問題。建構一個自驅的、擁有強大領導力的組織已經是增長和成功的一個顯而易見的方法。

Instead, we’re going to discuss team scaling from a defensive and personal point of view rather than an offensive one. As a leader, your most precious resource is your limited pool of time, attention, and energy. If you aggressively build out your teams’ responsibilities and power without learning to protect your personal sanity in the process, the scaling is doomed to fail. And so we’re going to talk about how to effectively scale yourself through this process.

相反，我們將討論從保守和從個人觀點出發而不是進攻的視角來討論擴張團隊。作為領導者，你最寶貴的資源是你有限的時間、精力和能量。如果你在沒有學會保護維持自己的精力正常的情況下，就激進地增加團隊的職責和權力，你的擴張將註定失敗。於是我們接下來將討論如何在擴張的過程中有效地提升自己。

The Cycle of Success 成功的迴圈

When a team tackles a difficult problem, there’s a standard pattern that emerges, a particular cycle. It looks like this:

- *Analysis*
  First, you receive the problem and start to wrestle with it. You identify the blinders, find all the trade-offs, and build consensus about how to manage them.
- *Struggle*
  You start moving on the work, whether or not your team thinks it’s ready. You prepare for failures, retries, and iteration. At this point, your job is mostly about herding cats. Encourage your leaders and experts on the ground to form opinions and then listen carefully and devise an overall strategy, even if you have to “fake it” at first.[3]
- *Traction*
  Eventually your team begins to figure things out. You’re making smarter decisions, and real progress is made. Morale improves. You’re iterating on trade-offs, and the organization is beginning to drive itself around the problem. Nice job!
- *Reward*
  Something unexpected happens. Your manager takes you aside and congratulates you on your success. You discover your reward isn’t just a pat on the back, but a whole new problem to tackle. That’s right: the reward for success is more work... and more responsibility! Often, it’s a problem that is similar or adjacent to the first one, but equally difficult.

當一個團隊遇到困難的問題， 往往會顯現出一個標準的解決模型–一個特殊的迴圈，正如下面這樣：

- *分析*
  首先，你收到一個問題，然後開始嘗試解決它。你找出了相關的盲點，找到所有權衡點，然後為如何解決他們在團隊內達成共識。
- *掙扎*
  你開始著手工作，無論你的團隊是否已經準備好。你準備好了迎接失敗、重試和反覆迭代。從這點上來講，你的工作就像養貓。鼓勵你手下的團隊和專家們坐下來整理觀點，然後仔細傾聽，制定全域戰略，哪怕最開始你不得不“編造一個戰略”。
- *前進*
  終於你的團隊開始把事情搞清楚了，你做的決策越來越明智，問題也有了實質性的進展。士氣得到了鼓舞。你開始反覆迭代權衡，組織開始自驅地解決這個問題。幹得漂亮！

- *獎勵*
   一些意料之外的事情發生了。你的上級把你叫到一旁然後祝賀你的成功。你發現獎勵不僅僅是領導在你後背輕輕拍了一下祝賀你，而是給了你一個新的問題去解決。是的：對於成功的獎勵往往是更多的工作--和更多的責任！通常是一個類別似的，或者關聯的問題，但是同樣困難。

So now you’re in a pickle. You’ve been given a new problem, but (usually) not more people. Somehow you need to solve both problems now, which likely means that the original problem still needs to be managed with half as many people in half the time. You need the other half of your people to tackle the new work! We refer to this final step as the compression stage: you’re taking everything you’ve been doing and compressing it down to half the size.

所以現在你陷入了困境。你被分配給了一個新的問題，但通常並沒有更多的人力。莫名其妙的你需要同時解決這兩個問題，這意味著原來的問題仍需被解決，但只有一半的人力和一半的時間。你需要另一半的人力來解決新的問題！我們把這最後一步稱為壓縮階段：你需要處理所有你正在處理的事情，而且需要把它的規模壓縮到原來的一半。

So really, the cycle of success is more of a spiral (see Figure 6-2). Over months and years, your organization is scaling by tackling new problems and then figuring out how to compress them so that it can take on new, parallel struggles. If you’re lucky, you’re allowed to hire more people as you go. More often than not, though, your hiring doesn’t keep pace with the scaling. Larry Page, one of Google’s founders, would probably refer to this spiral as “uncomfortably exciting.”

所以，成功的迴圈更像是一個螺旋（參見圖 6-2）。長年累月以來，你的組織透過解決新問題來擴張，然後壓縮所需的人力來能夠接受新的、並行的問題。如果你足夠幸運，你才能被允許招聘更多的人。然而更常見的情況是你招聘的速度趕不上你團隊規模擴張的速度。Larry Page，Google 的創始人之一，喜歡把這個螺旋比作“令人不適的刺激”。

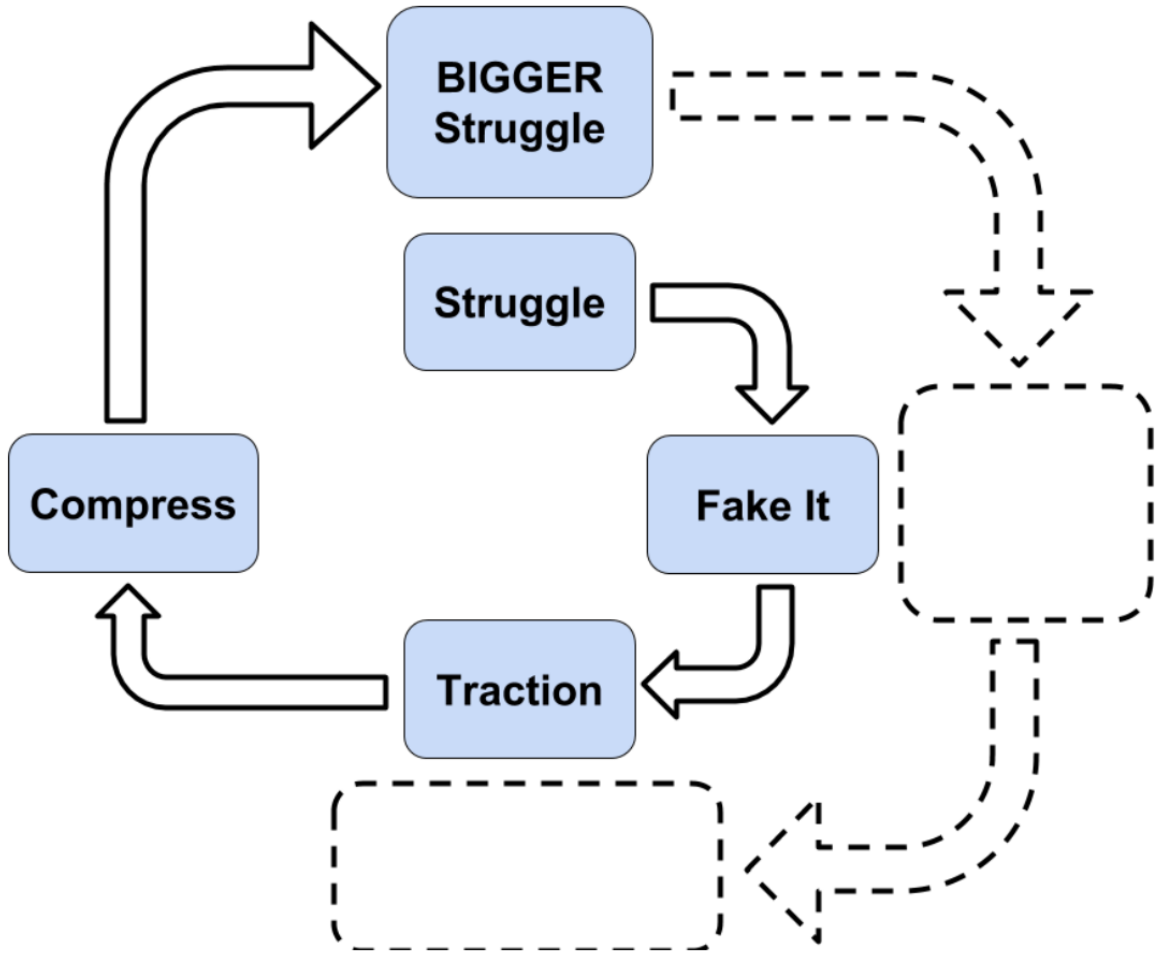

Figure 6-2. The spiral of success 圖 6-2. 成功的螺旋

The spiral of success is a conundrum—it's something that's difficult to manage, and yet it's the main paradigm for scaling a team of teams. The act of compressing a problem isn't just about figuring out how to maximize your team's efficiency, but also about learning to scale your own time and attention to match the new breadth of responsibility.

成功的螺旋確實是個難題--這是難以管理的，而且這是擴充團隊的團隊的核心正規化。壓縮問題的行為不只是關於找出使團隊效率最大化的方法，而且是關於如何擴充你自己的時間和注意力來應對新的責任。

> 3 在這一點上，冒名頂替綜合症很容易發作。克服這種感覺的一種技巧是，你不知道自己在做什麼，只需假裝某位專家確切知道該做什麼，他們只是在度假，而你只是暫時代替他們。這是一個很好的方法，可以消除個人利害關係，允許自己失敗和學習。

Important Versus Urgent 重要和緊急

Think back to a time when you weren't yet a leader, but still a carefree individual contributor. If you used to be a programmer, your life was likely calmer and more panicfree. You had a list of work to do, and each day you'd methodically work down your list, writing code and debugging problems. Prioritizing, planning, and executing your work was straightforward.

回想你還不是領導的時候，那時候你還是個無憂無慮的個人貢獻者(IC)。如果你曾經是個程式設計師，你的生活很可能要比現在平靜，沒有很多讓人恐慌的狀況需要處理。你有完整的工作清單，上面的每個條目你都能有條不紊的解決，寫程式碼或是除錯問題。排優先順序、定計劃，然後執行你的工作，還是很簡單的。

As you moved into leadership, though, you might have noticed that your main mode of work became less predictable and more about firefighting. That is, your job became less proactive and more reactive. The higher up in leadership you go, the more escalations you receive. You are the “finally” clause in a long list of code blocks! All of your means of communication—email, chat rooms, meetings—begin to feel like a Denial-of-Service attack against your time and attention. In fact, if you’re not mindful, you end up spending 100% of your time in reactive mode. People are throwing balls at you, and you’re frantically jumping from one ball to the next, trying not to let any of them hit the ground.

當你在管理崗工作後，你可能慢慢察覺到了，你的工作模式變得不可預測，天天都在救火。你的工作變得被動，更多的是響應別人的訴求。你的崗位越高，這個現象越明顯。你成了一堆代號的最終負責人。你的所有通訊手段--郵件、聊天室、會議開始讓你感覺像是你時間和精力的“拒絕服務型攻擊”。事實上，如果你不留心，最終你將花費你的全部時間在被動響應別人的需求上。人們開始像你扔球，你必須拼命地從一個球跳向另一個球，儘量避免讓任何一個掉到地上。

A lot of books have discussed this problem. The management author Stephen Covey is famous for talking about the idea of distinguishing between things that are important versus things that are urgent. In fact, it was US President Dwight D. Eisenhower who popularized this idea in a famous 1954 quote:

```
I have two kinds of problems, the urgent and the important. The urgent are not important, and the
important are never urgent.
```

很多書都討論過這個問題。管理學作者 Stephen Covey 因討論如何區分重要的事情和緊急的事情的想法而出名。事實上，是美國總統埃森豪威爾在 1954 年一次演進中參考而讓其出名的：

```
我有兩類問題，緊急的問題和重要的問題。緊急的並不重要，重要的也從不緊急。
```

This tension is one of the biggest dangers to your effectiveness as a leader. If you let yourself slip into pure reactive mode (which happens almost automatically), you spend every moment of your life on urgent things, but almost none of those things are important in the big picture. Remember that your job as a leader is to do things that only you can do, like mapping a path through the forest. Building that meta- strategy is incredibly important, but almost never urgent. It’s always easier to respond to that next urgent email.

這兩者直接的關係是威脅你作為領導的工作效率的最大的危險之一。如果你放入自己變成純被動響應式的工作模式（這往往會自然而然的發生），你將會花費你全部的時間和精力解決緊急的事，但是這些東西在宏觀層面幾乎都是不重要的。一定要記住你作為一個領導的工作是要做那些必須由你來做的事，比如規劃穿越森林的路線。建構這些“元策略”是非常重要的，但幾乎從不緊急。相比起來，回覆下一封緊急的郵件永遠更簡單。

So how can you force yourself to work mostly on important things, rather than urgent things? Here are a few key techniques:

- *Delegate*
  Many of the urgent things you see can be delegated back to other leaders in your organization. You might feel guilty if it's a trivial task; or you might worry that handing off an issue is inefficient because it might take those other leaders longer to fix. But it's good training for them, and it frees up your time to work on important things that only you can do.
- *Schedule dedicated time*
  Regularly block out two hours or more to sit quietly and work only on important- but-not-urgent things—things like team strategy, career paths for your leaders, or how you plan to collaborate with neighboring teams.
- *Find a tracking system that works*
  There are dozens of systems for tracking and prioritizing work. Some are software based (e.g., specific "to-do" tools), some are pen-and-paper based (the "Bullet Journal" method), and some systems are agnostic to implementation. In this last category, David Allen's book, Getting Things Done, is quite popular among engineering managers; it's an abstract algorithm for working through tasks and maintaining a prized "inbox zero." The point here is to try these different systems and determine what works for you. Some of them will click with you and some will not, but you definitely need to find something more effective than tiny Post- It notes decorating your computer screen.

那麼，怎麼才能強迫你自己花更多精力在重要的事情上，而不是緊急的事情上呢？下面列舉了幾個關鍵技巧：

- *委託*
  許多緊急的事件實際上可以委託給你組織裡的其他領導者。如果是比較瑣碎的任務你可能會感到有一點點罪惡；或者你可能會擔心有點低效，如果其他的領導者將花較長時間來解決。但這對他們來說是很好的鍛鍊的機會，而且能夠為你騰出時間來去解決重要的事情。
- *安排專注時間*
  定期安排佔據 2 個小時或更長時間的整段時間來專注處理重要但不緊急的事，比如團隊策略，團隊中管理者的職業生涯規劃，或者制定如何與其他團隊協作的計劃。
- *找到一個有效的進度追蹤系統*
  市面上有很多關於進度追蹤和排優先順序的系統。一些是有基於現成的軟體的（比如"待辦"管理工具），一些是基於紙筆的（"Bullet Journal"方法），以及另一些沒有指明具體實現方法的系統。在這最後一類中，David Allen 的書《搞定 I : 無壓工作的藝術》在工程師管理者們之間很流行；它是一套關於工完成任務和將收件箱清零的抽象的方法論。這裡的關鍵點是要去嘗試不同的系統，然後選擇一個對你來說最有效的系統。他們其中一些會很合適，一些並不合適，但你絕對需要找到比在電腦上貼便籤更有效率的方法--它更多是在裝點你的電腦螢幕。

Learn to Drop Balls 學會丟球

There's one more key technique for managing your time, and on the surface it sounds radical. For many, it contradicts years of engineering instinct. As an engineer, you pay attention to detail; you make lists, you check things off lists, you're precise, and you finish what you start. That's why it feels so good to close bugs in a bug tracker, or whittle your email down to inbox zero. But as a leader of leaders, your time and attention are under constant attack. No matter how much you try to avoid it, you end up dropping balls on the floor—there are just too many of them being thrown at you. It's overwhelming, and you probably feel guilty about this all the time.

還有一個管理時間的重要方法，但是它表面上看起來有些激進。對於大多數人來說，這違反多年以來養成的工程師的直覺。作為一個工程師，你關注細節，你製作清單，然後將完成的事一件件劃掉，認真仔細，保證每件事有始有終。所以在錯誤追蹤系統中關閉 bug 或清空收件箱中的待辦時，你感覺很良好。但是作為領導者的領導，你的時間和精力處在持續的壓力下。哪怕你全力嘗試，但總無法避免會有球接不住掉到地上--同一時間內總是有太多球扔向你了。它是難以避免的，而且你可能一直會為此感到內疚。

So, at this point, let’s step back and take a frank look at the situation. If dropping some number of balls is inevitable, isn’t it better to drop certain balls deliberately rather than accidentally? At least then you have some semblance of control.

所以，在這一點上，讓我們後退一步，然後仔細地審視下當前的局勢。如果漏掉一些球是不可避免的，那麼主動地去選擇丟掉某些球不是比意外的丟掉更好嗎？至少看起來這樣更可控一些。

Here’s a great way to do that.

這裡有一個很好的方法來達成這個目的。

Marie Kondo is an organizational consultant and the author of the extremely popular book The Life-Changing Magic of Tidying Up. Her philosophy is about effectively decluttering all of the junk from your house, but it works for abstract clutter as well.

Marie Kondo 是一名組織顧問，同時也是暢銷書《怦然心動的人生整理魔法術》(The Life-Changing Magic of Tidying Up) 的作者。她的理論是關於如何高效地清理家中的雜物，但對於如何清理抽象的雜物也同樣適用。

Think of your physical possessions as living in three piles. About 20% of your things are just useless—things that you literally never touch anymore, and all very easy to throw away. About 60% of your things are somewhat interesting; they vary in importance to you, and you sometimes use them, sometimes not. And then about 20% of your possessions are exceedingly important: these are the things you use all the time, that have deep emotional meaning, or, in Ms. Kondo’s words, spark deep “joy” just holding them. The thesis of her book is that most people declutter their lives incorrectly: they spend time tossing the bottom 20% in the garbage, but the remaining 80% still feels too cluttered. She argues that the true work of decluttering is about identifying the top 20%, not the bottom 20%. If you can identify only the critical things, you should then toss out the other 80%. It sounds extreme, but it’s quite effective. It is greatly freeing to declutter so radically.

假設你的物質財產有成三堆.這些東西中有大約 20%是沒用的--你永遠都不會去觸碰它們，很好識別出來，並扔掉。大約 60%的東西很有趣；對於你來說，它們的重要程度不一，有時會使用它們，有時不會。另外 20%對你來說極其重要：你會一直使用，這有很強的主觀情感傾向，或者用 Kondo 女士的話來說，就是“只要抱著它們，就感覺充滿快樂”。她的書中的主要論點是，大多數人錯誤地進行斷舍離：它們來回折騰那最沒有用的 20% 的物品，然而剩下的 80% 的物品看上去仍然很亂。她指出斷舍離的關鍵是分辨出那 20%最重要的東西，而不是最不重要的那 20%。如果你能分辨出那些最關鍵的東西，那麼剩餘的 80%都是你可以扔掉的。這聽上去有些極端，但卻是十分高效的。能夠如此徹底的斷舍離，一定會感覺很棒。

It turns out that you can also apply this philosophy to your inbox or task list—the barrage of balls being thrown at you. Divide your pile of balls into three groups: the bottom 20% are probably neither urgent nor important and very easy to delete or ignore. There’s a middle 60%, which might contain some bits of urgency or importance, but it’s a mixed bag. At the top, there’s 20% of things that are absolutely, critically important.

事實上你也可以將這個理論應用到你的收件箱或任務清單中--那些像彈幕一樣像你飛來的球。將你的球分成三堆：最底下的 20% 是那些從來不重要，也不緊急的，很容易刪除或忽略。中間的 60% 有一些是緊急的，有一些是重要的，但是混在一起很難分清。在最上層，是那些至關重要的事。

And so now, as you work through your tasks, do not try to tackle the top 80%—you'll still end up overwhelmed and mostly working on urgent-but-not-important tasks. Instead, mindfully identify the balls that strictly fall in the top 20%—critical things that only you can do—and focus strictly on them. Give yourself explicit permission to drop the other 80%.

現在，在你工作時，不要嘗試解決上面 80%的問題--否則最終你仍會被問題淹沒，然後花大部分精力解決那些緊急但不重要的問題。相反，你應該識別出頭部最重要的 20%--那些只能由你來完成的最重要的事，然後專注於他們。給你自己明確的許可來丟棄剩餘的 80%。

It might feel terrible to do so at first, but as you deliberately drop so many balls, you'll discover two amazing things. First, even if you don't delegate that middle 60% of tasks, your subleaders often notice and pick them up automatically. Second, if something in that middle bucket is truly critical, it ends up coming back to you anyway, eventually migrating up into the top 20%. You simply need to trust that things below your top-20% threshold will either be taken care of or evolve appropriately. Meanwhile, because you're focusing only on the critically important things, you're able to scale your time and attention to cover your group's ever-growing responsibilities.

最開始，這麼做可能會感覺很可怕，但隨著你故意丟掉這麼多球，你將會發現兩件令人驚奇的事。第一，即使你沒有託管中間 60%的事，你的下屬領導者們通常會意識到並主動接住它們。第二，如果中間這堆球中有真正重要的事，它最終無論如何都會回到你這裡，然後轉換到頂部 20%那堆球裡。你只需相信在 20%閾值下的事情最終都會被有人接管，或是在適當時候知會給適當的人。與此同時，因為你只關注最重要的事情，你可以花更多時間和注意力在承擔你的團隊不斷增長的責任上。

Protecting Your Energy 保護你的精力

We've talked about protecting your time and attention—but your personal energy is the other piece of the equation. All of this scaling is simply exhausting. In an environment like this, how do you stay charged and optimistic?

我們討論過了如何包含你的時間和精力--但是你的個人精力又是另一個等式。單單是這些擴張就足以讓你精疲力盡。在這樣的環境中，你如何保持持續充電和樂觀呢？

Part of the answer is that over time, as you grow older, your overall stamina builds up. Early in your career, working eight hours a day in an office can feel like a shock; you come home tired and dazed. But just like training for a marathon, your brain and body build up larger reserves of stamina over time.

這個答案的一部分是，隨著時間的推移，你年齡增長，你的耐力會隨著增長。在你職業生涯的早期，在辦公室連續工作 8 個小時就會讓你感到震驚；回到家後你會感覺疲勞和空虛。但是就像馬拉松訓練一樣，你的大腦和身體會能夠儲備越來越多的耐力。

The other key part of the answer is that leaders gradually learn to manage their energy more intelligently. It's something they learn to pay constant attention to. Typically, this means being aware of how much energy you have at any given moment,and making deliberate choices to "recharge" yourself at specific moments, in specific ways. Here are some great examples of mindful energy management:

- *Take real vacations*
 A weekend is not a vacation. It takes at least three days to “forget” about your work; it takes at least a week to actually feel refreshed. But if you check your work email or chats, you ruin the recharge. A flood of worry comes back into your mind, and all of the benefit of psychological distancing dissipates. The vacation recharges only if you are truly disciplined about disconnecting.[4] And, of course, this is possible only if you’ve built a self-driving organization.
- *Make it trivial to disconnect*
 When you disconnect, leave your work laptop at the office. If you have work communications on your phone, remove them. For example, if your company uses G Suite (Gmail, Google Calendar, etc.), a great trick is to install these apps in a “work profile” on your phone. This causes a second set of work-badged apps to appear on your phone. For example, you’ll now have two Gmail apps: one for personal email, one for work email. On an Android phone, you can then press a single button to disable the entire work profile at once. All the work apps gray out, as if they were uninstalled, and you can’t “accidentally” check work messages until you re-enable the work profile.
- *Take real weekends, too*
 A weekend isn’t as effective as a vacation, but it still has some rejuvenating power. Again, this recharge works only if you disconnect from work communications. Try truly signing out on Friday night, spend the weekend doing things you love, and then sign in again on Monday morning when you’re back in the office.
- *Take breaks during the day*
 Your brain operates in natural 90-minute cycles.[5] Use the opportunity to get up and walk around the office, or spend 10 minutes walking outside. Tiny breaks like this are only tiny recharges, but they can make a tremendous difference in your stress levels and how you feel over the next two hours of work.
- *Give yourself permission to take a mental health day*
 Sometimes, for no reason, you just have a bad day. You might have slept well, eaten well, exercised—and yet you are still in a terrible mood anyway. If you’re a leader, this is an awful thing. Your bad mood sets the tone for everyone around you, and it can lead to terrible decisions (emails you shouldn’t have sent, overly harsh judgements, etc.). If you find yourself in this situation, just turn around and go home, declaring a sick day. Better to get nothing done that day than to do active damage.

這個答案的另一部分是領導者漸漸學會更智慧的管理他們的精力。這是他們持續學習投入的結果。通常，這意味著他們能夠意識到自己還剩多少精力，然後決定在某個特定的時刻透過自己的方式給自己“充能”。以下是一些很好的細心管理能量方式：

- *給自己真正放個假*
 一個週末並不算一個真正的假期。你需要至少三天來“忘記”你的工作；至少需要一週來讓你重新感覺充滿能量。但是如果你檢查你的郵箱或工作聊天，你就破壞了這個充電過程。洪水般的焦慮充滿你的腦袋，物理上遠離工作的好處消散殆盡。只有在你真的斷開與工作的連線時，你的假期才能使你真正重新充能。當然，這一切建立在你已經建立了一個自我驅型組織的前提下。
- *讓失聯的代價微不足道(主要指訊息模式切換)*
 當你失聯時，將你的工作筆記本留在辦公室。如果你有工作的通訊工具留在你的手機上，將它們移除掉。比如，如果你的公司用 Google 的 G 套件(Gmail, Google Calendar 等)，一個很方便的技巧是在手機上安裝一個叫“工作資料” 的軟體。這將花費你幾秒鐘的時間來把軟體標記為是否是工作軟體。例如，你將有兩個 Gmail 應用：一個為個人郵件，一個為工作郵件。在安卓手機上，你能夠一鍵切換工作模式。所有工作應用軟體將變灰，就像它們未安裝。你也不可能“不小心”查看了工作資訊直到你重新啟用工作資料。

- *也要享受真正的週末*
  一個週末並不像假期一樣有效，但它仍有讓你振奮起來的能力。再強調一次，這樣的充能只有在你斷開工作聯絡的時候才有用。試著在週五晚上徹底退出工作狀態，把週末的時間花在你喜歡的事情上，然後在週一早晨回到辦公室時再重新進入工作狀態。
- *在一天之中偶爾小休一下*
  人的大腦每 90 分鐘會有一個自然的迴圈。利用這個機會站起來在辦公室走一走，或者花 10 分鐘出去走一走。像這種微小的休息只能獲得很小的充能，但是這能給你的緊張度和下一個小時的工作上帶來巨大的影響。
- *給自己一個心理健康日的許可*
  有時候，沒有任何理由，你度過了糟糕的一天。你睡的很好，吃的很好，也進行了運動，但還是在很糟糕的情緒裡。如果你是個領導，那這將是很悲催的一件事。你的壞情緒影響了你周圍所有人的情緒，而且這將會導致很糟糕的決定（發出去不該發的郵件，給別人下達了過於殘酷的評價等）。如果你發現你在這個狀態下，你應該請個病假，轉身回家。什麼都不幹也比干破壞性的事強。

In the end, managing your energy is just as important as managing your time. If you learn to master these things, you'll be ready to tackle the broader cycle of scaling responsibility and building a self-sufficient team.

最後，管理你的精力和管理你的時間一樣重要。如果你學會掌握這些東西，你就會準備好應對擴大責任範圍和建立一個自給自足的團隊這一更廣泛的迴圈。

> 4 你需要提前計劃，並在假設你的工作在休假期間根本無法完成的情況下進行建設。在休假前後努力工作（或聰明地工作）可以緩解這一問題。
>
> 5 你可以在 https://en.wikipedia.org/wiki/Basic_rest-activity_cycle，瞭解更多關於BRAC的資訊。

Conclusion 總結

Successful leaders naturally take on more responsibility as they progress (and that's a good and natural thing). Unless they effectively come up with techniques to properly make decisions quickly, delegate when needed, and manage their increased responsibility, they might end up feeling overwhelmed. Being an effective leader doesn't mean that you need to make perfect decisions, do everything yourself, or work twice as hard. Instead, strive to always be deciding, always be leaving, and always be scaling.

成功的領導者在管理的過程中很自然地會承擔更多的責任（這是件好事，也是很自然的事）。除非他們能夠高效的想出技術來快速地做決策，按需時授權，管理他們日益增長的責任，否則他們很快會感覺不知所措。作為一個高效的領導者並不意味著你要做完美的決策，所有事都親力親為，或者付出雙倍的努力。而是需要努力地始終保持決斷力，始終保持離開，始終保持擴張。

TL;DRs 內容提要

- Always Be Deciding: Ambiguous problems have no magic answer; they're all about finding the right trade-offs of the moment, and iterating.
- Always Be Leaving: Your job, as a leader, is to build an organization that automatically solves a class of ambiguous problems—over time—without you needing to be present.
- Always Be Scaling: Success generates more responsibility over time, and you must proactively manage the scaling of this work in order to protect your scarce resources of personal time, attention, and energy.
- 始終保持決斷力：模糊的問題沒有靈丹妙藥；他們都是關於找到當下最佳的權衡，然後反覆迭代。
- 始終保持離開：你作為一個領導者的工作是建構一個能夠自主解決一類模糊問題的組織--並且隨著時間的推移，漸漸不需要你出面。
- 始終保持擴張：隨著時間推移，成功會產生更多責任，你必須主動管理規模擴大的工作，來保護你最稀缺的資源：時間，注意力和精力。

---

1. "Code yellow" is Google's term for "emergency hackathon to fix a critical problem." Affected teams are expected to suspend all work and focus 100% attention on the problem until the state of emergency is declared over. ↩

2. Brian W. Fitzpatrick and Ben Collins-Sussman, Debugging Teams: Better Productivity through Collaboration(Boston: O'Reilly, 2016). ↩

3. It's easy for imposter syndrome to kick in at this point. One technique for fighting the feeling that you don't know what you're doing is to simply pretend that some expert out there knows exactly what to do, and that they're simply on vacation and you're temporarily subbing in for them. It's a great way to remove the personal stakes and give yourself permission to fail and learn. ↩

4. You need to plan ahead and build around the assumption that your work simply won't get done during vacation. Working hard (or smart) just before and after your vacation mitigates this issue. ↩

5. You can read more about BRAC at https://en.wikipedia.org/wiki/Basic_rest-activity_cycle. ↩