[Ship While You Sleep Cover]

Ship While You Sleep, version 2

The Power of Code As Cards

By Amol Sarva

Nov 4 2018

Based on Version 1 by Amol Sarva and Walker Donohue

About Amol

This book on hiring and building is based on some experience. In 1995, I worked as an intern learning Java at one of legendary investor Fred Wilson's early venture deals (I was a hobbyist hacker and philosophy major at Columbia in New York, and he now runs Union Square Ventures which is famous for their bets on Twitter and blockchain). I started my own little startup that put NYC menus online in 1996. In 2000, while I was finishing a cognitive science Ph.D. at Stanford, I helped start Virgin Mobile USA—a huge business that raised $500 million in venture, hired a team of hundreds of engineers and went public worth more than $1 billion.

Since then, I've started 10 more companies and invested in 25. Some of the better efforts include Blue Mobile, where we raised $20 million in backing to make another mobile phone service in the U.S. with Walmart and Verizon as partners (less cool than Virgin) while putting 100 engineers to work; and Peek, where we designed a Blackberry-type smartphone that cost 95% less to make. That was a real feat that won us some awards like Wired's #1 Gadget and *Business Week*'s Gadget of the Year and became a software business whose U.S. unit was acquired by Twitter with the global platform acquired by Bharti SoftBank.

Since 2012, I've been part of building a litany of products with large startups and new startups: a streaming music player called Gramofon, a wearable neurostimulation startup called Halo Neuroscience, a new kind of digital team workspace called Knotable, a headquarters-as-a-service company called Knotel. I've also been on a number of 1–2 person teams building mobile apps, with public companies deploying our products into $100 million revenue customers.

Then there are the hundred companies where I'm an investor or advisor: Ouya (the open source hackable Android game microconsole acquired by Razer), Jump (the self-organizing bikeshare network for cities, acquired by Uber in 2012), KeyMe (the key-copying robot and smartphone app), Plethora (the manufacturing robot), WorkMarket (the platform for managing freelance workers, acquired by ADP), MarleySpoon (the dinner recipe in a box delivery service which went public in Australia in 2018), and more.

The way we build and ship software products is changing a lot. The next step is already visible. As computer scientist Alan Kay said, "The future is already here. It's just unevenly distributed." In this book, I lay out a way to work and ship that ties together a dozen trends we are seeing in how cutting-edge organizations make things. It's a recipe. But if you look at the ingredients, you see what the future of work looks like.

Chapter One: Beyond Code as Craft

On their blog Code as Craft, the team of engineers at Etsy espouse a method of software development they compare to the way a cathedral might have been constructed in the Middle Ages:

> *"Each took thousands of person-years of effort, spread over many decades. Lessons learned were passed down to the next set of builders, who advanced the state of structural engineering with their accomplishments. But the carpenters, stonecutters, carvers, and glass workers were all craftspeople, interpreting the engineering requirements to produce a whole that transcended the purely mechanical side of the construction. It was their belief in their individual contributions that sustained the projects: We who cut mere stones must always be envisioning cathedrals."*

It's an amazing visual. In many ways, the uber-hackers of today closely resemble the master craftsmen of the 1490s. Long beards, skilled hands, makers—artisanal rather than commercial. The cultural cousins of these people are making chocolate by hand in Brooklyn or writing novels in longhand. They believe in the supreme, divine power of genius and wisdom. They make masterpieces of code.

There is a small irony in that passage about cathedral-craft, and it concerns a famous 1990s paper by Linux and open source theorist Eric Raymond. Back in that era, the enemy was Microsoft and their ponderous operating system Windows; Linux was thought to be the wave of the future. The monopolists in Redmond made an operating system so good and so carefully designed to work on so many computer makers' machines that many corporate and consumer technology buyers believed nobody could ever catch up.

It was at this time that the open source movement was gathering steam around an open source operating system that nobody would own or direct: Linux. Led by a loose band of hackers all over the world, the Linux codebase was the Wikipedia-style global project of its day. Nobody in particular entirely in charge, no top-down architectural edicts, no dedicated full-time army of engineers laboring away.

Back in the 1990s, the idea that a Linux operating system would one day conquer the world was silly (it was also correct; more computers now run Linux-based OSes than Windows by a large margin). And the guy who most famously argued for why this might work out used a simple metaphor. In a paper called "The Cathedral and the Bazaar," Raymond more or less advocates a marketplace of ideas, the specialization of different tasks and the continuous improvement of an iterative approach. Quite the opposite of the grand cathedrals.

So it's a clue: Something in the current engineering culture has pulled people way off the path of a powerful, crowd-sourced, open-source, Bazaar ethos to a new form of guild-making craft.

That's all well and good. But say you have an idea for a web app—for example, a site that will let people essentially start their own businesses by uploading their wares and selling them to whoever is interested. And you need to develop this idea, and you hold the philosophy above dear to your heart. How do you motivate engineers to see themselves as master craftsmen?

How do you hire these master craftsmen in the first place—away from their well-appointed guilds and long-term artistic crusades?

According to a thousand profiles of tech companies (e.g. this gem from *Bloomberg Businessweek*), you give new employees stipends to decorate their offices, hire a chef to prepare meals with locally-sourced ingredients and let dogs roam around (how cute!).
Or look at Rap Genius, mentioned in the same article. Every new employee there gets $1,000 a day, can take unlimited vacations, has access to unlimited free Seamless and Fresh Direct, as well as a shower, a gym and a laundry room. At bigger tech firms, the luxury is famously lavish too: Michelin-star chefs prepare meals at Google's cafeterias around the world.

From the business side of things, very, very few people with ideas for apps have the kind of infrastructure and money required to support this kind of work: the coddling, luxury style that so many hip new startups seem almost obliged to provide if they're going to attract talented workers.

Same for established companies with a little less go-go money to burn. Maybe at your company you have a project you think should happen. But you don't have the budget to go headhunting for hotshot CTOs or perhaps even internal engineering staff. How do you get it going? Many of the problems a startup founder faces translate directly to those of a larger organization.

Starting at ground zero, what's the first step to turning your mock-up for an app into a real, shippable product? For many in the startup world, it's finding a cofounder, someone who understands the scope of your project and can do the work of coding it. Which raises the question: how do I find that person?

Wolfgang Bremer, German designer, entrepreneur and cofounder of an online service for (the irony is impeccable) finding cofounders, outlines the conundrum: "I'm constantly going to these startup events, meetups and tech talks... however, I often notice that always the same people seem to go there. And they're going there to find a cofounder for their own idea. So very rarely there's actually a potential cofounder available. And that sucks."

It sure does. It's really easy to see this firsthand, especially if you live in New York City. You can even go to parts of Brooklyn that were basically warzones in the 1970s, and today, the sidewalks outside bars are full of people on Friday and Saturday nights scoping out potential partners for their new idea. Why bars in particular? Three reasons. One, because that's where like-minded people congregate and where you might be more likely to find your perfect fellow

traveler. Two, because startups need to be cool, right? And three, because people in bars are tipsy and susceptible.

Or even consider how teams you have already assembled manage to work together. Engineers start working, coding away, headphones on, eyes on screen. “Meetings suck,” they say. They roll their eyes at management types. They argue with business leaders about what the product should do and how—or even who the customers are.

Partly, engineers have embraced a solo hacker culture where people collaborate at their own discretion. No hierarchy. Only grudgingly taking directions.

And partly, engineering culture today has gotten deep into the supply-demand imbalance of the global tech market. Engineers are scarce. They are prima donnas. They do what they want. CEOs fear them.

It’s an expensive worldview to adopt. If you are starting a company and you need to find this hacker cofounder, it’s not simple if you’re searching for a Code as Craft master. If you are launching an internal project, you need to plead with the CTO for the precious time and interest of their All-Star Team—a group of people who normally wouldn’t bother to say hello to you in the cafeteria. And if you are running a large engineering organization, reasonable efforts to be flexible and productive leave you constantly skirmishing with an irritated
Mob of Villagers who hate “suits.”

Why is this happening? At the heart of the issue is simple supply and demand. In rich places like the U.S. and Europe, there is huge appetite for software engineers and limited supply. Immigration laws are constraining, and so where venture capital, startups and technology projects are plentiful, business managers just cannot get enough people. So the competition for people takes all forms: high compensation, risk-averse engineers, bad attitudes among teams, high churn among staff. Like any system with a scarce few stars, you start seeing poor behavior by the overpaid, irreplaceable franchise players.

It’s such an imbalance that many early-stage companies cannot raise money if they don’t already have a compelling engineering organization. Nobody believes you will get one when you have a million dollars in the bank. Engineering cofounders are something money can’t buy.

Ask any technology manager or startup founder these days and you will hear some mix of these complaints about the “typical” engineering team (maybe not their own—it’s too dangerous to badmouth your people). Engineers:

- Hate meetings.
- Hate task management.
- Hate schedules.
- Hate email.

- Communicate poorly in general.
- Argue about what end-users actually want.
- Continually demand more compensation, or else leave.
- Expect perks or frequently compare to others' perks, e.g., free lunches.
- Don't collaborate with each other.
- Like the technology more than the customer problem.

A scene in HBO's *Silicon Valley* perfectly illustrates the importance of culture. Two engineers, roommates working on a sound compression app, bet on whether one could identify an object the other touched using only his sense of smell. While one is exploring the room sniffing everything, it becomes apparent that the two had accidentally built the same DRM feature. Each had thought he was responsible for its development. As a result, their CEO gives them their first taste of project management.

What the boys had before the popular new method of "Scrum" was not culture. It was waste. It's wasteful to run silly pranks like the one they do while they should be working, yes.

But more importantly, it's wasteful for two engineers to both be working on the same project independently because there was no communication protocol in place.

A friend who works for one of the big New York banks was half-complaining to me recently. There's a new CTO at the bank, and he wants to make the bank more agile—"more like a startup." He's tearing out the offices, pushing everyone into big open rows of workstations, changing the hierarchical meeting culture, eliminating the "phase-gate" step-by-step design and build and test and test and revise and release process.

In a way it resembles the statement NYC Mayor Michael Bloomberg made when he walked into City Hall in 2001 and got rid of the traditional "West Wing office" setup that most government follows. He took a desk in the center of a huge room and arrayed 100 top managers all around him. Wide open workplace. Just like at the giant company he founded—Bloomberg LP, the most successful New York tech startup ever. Listening to my friend talk about his bank CTO's push to make Wall Street more like Silicon Valley, I reflected on whether his company setup looked like mine—or any of the startups I've been helping.

We don't have that big bullpen at Knote. We have something very, very different:four people in New York who work in an environment that perhaps looks like that setup. But we have 56 people in 15 other countries, no other offices, perhaps 16 time zones when we are working.

We are seeing setups like this at other early stage companies—teams of one, two or three engineers shipping interesting projects for very little money. Stirplate.io, Halo Neuroscience, Myndset and many more.

There is this other way to do things. In this five-part series, we're going to draw it out in detail: how to build an app, how to get your idea off the ground and out the door, without any free massages or unlimited Seamless or 20% time.

It's called Code as Cards.

Chapter Two: Hiring Through Firing

In Chapter One, we discussed the predominant philosophy of Code as Craft, espoused by elite startups and engineering outfits around the world. We also covered the problems inherent to it if 1) you have an idea but lack the technical knowledge to pull it off yourself, or 2) have an idea but lack the resources to attract and to hire the people to pull it off. Talent, the common parlance goes, is not cheap. But that's not necessarily true.

Thomas Friedman, in his book *The World is Flat*, makes the case that globalization has produced, especially in the technological industry, a level playing field. The rate at which technical knowledge has spread throughout the world is incredible and Friedman argues that this is a positive development insofar as it puts pressure on companies in the Western world to adapt to that change.

Friedman's argument got huge global attention and it would be safe to call it the way most people think about globalization. We see the evidence everywhere. Knowledge-powered growth rates in China and India have made rich countries look pathetic over the last decade, and many of the key growth industries are driven by knowhow in manufacturing, engineering and especially software. Many of the largest new businesses created in India the last decade are software and digital knowledge outsourcers: Tata Consultancy Services, Wipro, Infosys and many more.

So where is the evidence for this flat world in startup land?

Some people think they know why some engineering is still so hard to find. John Larson, an online programmer personality, wrote a popular blog post about this very topic entitled "Why I Will Never Feel Threatened by Programmers in India." In it, he defends the exceptional American tech talent pool and recounts stories of Indian programmers, who were paid $14 an hour, wrote code that needed to be fixed by an American for a significantly higher rate.

Which obviously plays into a common cultural touchstone about outsourcing labor: You get what you pay for. Spaghetti code, poor architecture, hacks, shortcuts. The experiences Larson talks about have been widely felt among the Code as Craft engineers that run and build most impressive products today. Their institutional memory is large and uniform and produces a kneejerk "no way!" to outsourced networks of engineers.

Fortunately, the received wisdom is oversimplified and wrong, because there is a way to make outsourcing work. It just requires a very particular approach: You have to think deeply about who and how you decide to hire, how much you pay and how to vet your hires.

We call the people approach a "network" in Code as Cards.

You need to build a freelancer network—which takes work and maintenance. But this network—which can be assembled quickly—can build quality code fast and cheaply.. When you have a network of freelance engineers, you have a flexible and interconnected team that can dial up and down to your budget and produce high-quality output.

Let's talk about how.

• • •

Unless you've been living under a rock for the last several years, you've heard about the gig economy.

Work is changing. Professionals, and younger ones in particular, are increasingly opting to take on short-term jobs or freelance work (i.e., "gigs") instead of signing on for full-time employment.

This is a win-win for employers and workers alike:

- **Employers.** Instead of investing a ton of resources to source, interview and hire full-time workers who might not pan out—and paying salaries, benefits, equipment, and other expenses—employers that tap into the gig economy get access to more affordable workers they can use on an as-needed basis. Scale up during busy season, scale down when the waters are calmer. Hire the right freelancers, and you won't have to invest many resources training them—if you have to invest any at all.

- **Workers.** Work a full-time job and the scope of what you do tends to be relatively narrow. Become a member of the gig economy and there's no shortage to the number of companies you can work with. Not only do gig economy workers get exposed to a wider array of industries, technologies, and knowledge, there's also no limit to the money they can make. Through the course of their travels, gig workers meet a lot of people—even if it's just digitally. These relationships can often lead to additional opportunities. Sure, gig workers have to pay self-employment tax and cover their own healthcare costs. But for many, the tradeoff is more than worth it.

It's estimated that 34% of workers today are part of the gig economy. That number is expected to grow to 43% by 2020. That's not because employers are cheap or gig workers aren't talented enough to land full-time jobs. It's just that the nature of work is changing, and forward-thinking organizations and professionals both understand this. Employers want to find the best freelancers before they're too busy and gig workers want to establish themselves in the new economy before their competitors do.

Finding skilled programmers has probably never been easier than it is today. With freelance hiring sites like Elance and Odesk and Freelancer.com, as a few examples, you can literally source entire world for programmers capable of doing the exact kind of work you need to have

done. And you can even check out their reviews and ratings to see which ones are officially up to snuff.

These are markets that look very different than local job listings. If your startup puts up a post on a tech job board somewhere, you are likely to get exactly zero candidates clicking into your job in the first hour or first day. If you do, you have to be waving huge compensation, full-time roles, equity, an exciting startup story with funding and so forth. Post on a freelance hiring site, and you'll get some replies no matter what you post. A great start.

Now the next problem: You have a lot of people to evaluate.

Maybe you want to quickly pick the top talent—the highly rated folks. Five-star programmers on a site like Elance, though, will charge an accompanying premium, and that's just not sustainable for a cash-strapped operation. The top talent is actually good at working the system. They don't just charge a high hourly rate, they also bill projects on fixed-cost budgets. So they quote you something like $1,000 or $5,000 to do "the whole project." It's a bigger bit size. These operations are also busily surfing all the other offers coming into the platforms, so they usually have dedicated sales and marketing staff, e.g., someone to write those friendly emails to you. This pads their cost as well. And, more dangerously, sometimes these guys are just the salesperson who then tries to cobble together some engineers behind the scenes.

Yikes. That shortcut won't work.

So there's a high ratio of noise to information with these folks. The natural solution? Scrutinize them. Dive into their portfolio, get references, stay up late one night and interview a bunch of them via Skype. Maybe even have them do some detailed spec work in advance of awarding the project.

It's a solution to the Larson problem. Right? You want to hire coders from low-compensation countries like Ukraine, who are willing to work freelance on sketchily described projects. But since you will weed through dozens of resumes and pick the stars yourself, you'll avoid the pitfall of spaghetti code.

This isn't a shortcut, it's a longcut. But it won't work either. First and foremost, it suffers from the world's oldest problem with hiring: Interviewing sucks. There is a longstanding body of research on how useless interviewing is, like this piece by the cognitive psychologist Daniel Willingham from University of Virginia in the *Washington Post*:

> *"You do end feeling as though you have a richer impression of the person than that gleaned from the stark facts on a resume. But there's no evidence that interviews prompt better decisions."*

That was the longcut—interviewing people, for 20 minutes, by Skype, in the middle of the night, in broken English. Of course, if it doesn't work for universities and large American companies locally, as Willingham's research suggests, then it probably won't work for you.

They do say the resumes are useful. But unfortunately, the world of freelancers has very mingled and confusing profiles. You won't recognize anybody's schools or grades or concrete accomplishments from browsing through their profile. Many might not even have examples of their work or the companies they've worked with readily available. In fact, close scrutiny to profiles walks you into the second dangerous aspect of this longcut: your time.

Your time is precious. The ideas are yours. You have stuff you need to get done. If you dive deep into the land of evaluation and hiring, you are doing that. But the more time you sink into a first hire, the more your investment increases and the harder it becomes when we get to step two of the system (how to manage the work)—meaning the more disappointed you'll be when your first experiment fails.

We've worked with folks who tried Code as Cards and crashed and burned on this step. They "overhired" and got stuck with someone they couldn't move on from. Don't do it.

And anyway, the whole point of Code as Cards is to get you up and rolling fast.

There is a way.

The answer is so simple that it almost hurts: Fire people as quickly as you hire them.

Code as Cards relies much more on firing than hiring.

So here's how you hire: Post a job with very brief statement about what the project is, select a low compensation rate like $15/hour as the maximum and specify a rule banning any "teams" or "companies," preferring solos, and specifying a skill or two. Remember, $15/hour might not sound like much to someone based in Manhattan. But to a programmer in India or Thailand or Kenya, it's a lot of money. And, even if more would be better, it's a way to grow your GitHub profile or bolster your resume. We usually post something like, "Hiring for a web app. Skills needed: Javascript. First 10-hour paid trial increasing up to 40 hours per week. Maximum $15/hr. Send your GitHub and Trello usernames. No teams."

Then we just hire five or six people who reply to this without scrutinizing them at all. We just say, "Hey, please send your usernames. We meant that!" and then, "You're hired. Here is a link to read."

With project management tools like Trello and GitHub, which will be covered further in the next chapter, the vetting process has never been easier. Say you hire five engineers (wherever they are—foreign, domestic, skilled, not skilled) on a freelance basis on a Monday. You then invite

each one to a password-protected blog post that outlines the project and assign them all a “card” (i.e., a small task) and tell them that they will be paid a certain amount for a couple hours of work on it.

It may sound silly to pay out $100 to a bunch of random, unvetted coders, but what you’re looking for is reliable, quality workers, and in the end it’s a small price to pay. Much cheaper than committing $5,000 to a team on their proposal or spending weeks interviewing tons of people. **Spend $100 per person to find out if they are good.**

Cap that initial exposure. You give each of the cards a short timeframe to instill in your potential engineers the fact that this project requires having a quick turnaround on small, compartmentalized tasks. Assess the results the next day. If what’s been produced doesn’t meet your specifications exactly, fire them. “Thanks, but we don’t need any more work.” You may only end up with one viable engineer, but that is just fine: You’ve just hired that person for far less than you would have otherwise. And after 40 weeks, you can have twenty people working for you for just $300 a week—people you know can do the work.

This matches our experience and we’ve seen it in multiple companies. For every five you hire, you end up keeping maybe one. So keep doing it and you start having reliable people. And for every ten that you keep, you find that one is a star—a really extraordinarily motivated and high-impact person. And, depending how it goes, that person can stay with you forever as you bounce from project to project and company to company.

At the core, we are filtering on two things when we hire people: Do you have the will to immediately start contributing and do you have the skill to immediately contribute? If it’s a match, there will be progress right out of the gate and we reward you with more work.

There are some small tweaks that need to be done to our ordinary idea of hiring, of course, if this is going to work. One key element is using strong frameworks to de-skill the engineering. You want to make sure the cards are tiny tasks that still *produce visible work*. One major pitfall of remote working is failing to make sure that the work is actually happening. And finally, you want a high-visibility, live environment for those results. You want to lower the latency on feedback, but you also want your freelancers to be motivated by peer pressure.

In the next chapter, we’ll really get into this part of it: the work. How to build a project, how to assign tasks and the tools to use to get it all done.

Chapter Three: Visible Work

So far, we've talked about how to hire and fire, how to pay, and overall, how to assemble the kind of workforce needed to make your idea for an app work on a tight budget. Now we need to address the toolset you need to make this system produce results.

Making an app from scratch ("as craft") is difficult and time-consuming when you're working in the same room as people that you know. When you start delegating duties to engineers all around the world, the problems multiply. That is, unless you have the right tools at your disposal.

These tools aren't that well known. But used correctly, they make the philosophy of Code as Cards possible. In the last chapter, we touched on attitudes that experienced engineers have today about turning to remote, freelance engineers. Those attitudes come from past experience and past tools. But things are different now; different enough that it can work. And it's very concrete. We'll explore a few tools in great detail.

But first a big idea.

Let's begin with the concept of "visible work." You're not hiring "team members" or "colleagues" —not really. You're not hiring someone who you feel like you might want to chat up at the water cooler. You're hiring someone who can do the work. That's obvious enough. But Code as Cards focuses intensely on the work. The hiring process completely ignores the "who." It just says "start working."

Visibility is the crucial added aspect. Engineers can give some pretty compelling reasons for why they have to do "invisible work"—work that you can't quantify, see, measure or evaluate. These are smart people. Research, planning, rebuilding, refactoring, cleaning up, organizing, simplifying, editing, improving—these are some keywords to think about.

They aren't exclusively engineering concepts. And "invisible" doesn't mean they are bogus.

But a lot of them are in fact invisible. Let's do a non-technical example. You are reading this book. Maybe you are reading it to get better at managing product development projects. Worthwhile! But to any observer, there isn't a measurable difference on your desk or in your work output. . .yet. You are doing invisible work.

If you had a contract worker for your think tank working in a far off country, you would understandably be concerned that this contractor was goofing off or perhaps not reading closely or somehow doing something useless. What's the solution to this risk of outsourcing the job of reading this book?

Option 1. Do it yourself. Or maybe hire some local person and have them sit in your office. Watch them work. See that they are doing it. You see the parallel to the way tech companies work here, no doubt.

Option 2. Ask the researcher to send a daily email with a synopsis including at least five main points from each chapter. After the first email you will find out a) “is this person reading this stuff in a timely way or goofing off?”, and b) “is this person turning the reading into something usable by me?” Here, we have taken invisible work and made it into visible work. If the researcher read a chapter, there should have been a short email showing what they did. Simple cause and effect.

As Peter Drucker, the management guru, has written, if “you can’t measure it, you can’t improve it.” And this is the operative concept for making work done out of sight into work you can easily observe, measure and improve.

So, visible work. That’s what you want. How do we make this relevant for software? After all, writing code is in fact visible. The engineers write some code. You can count lines of code they write. But good luck doing that. It’s like saying you can measure how many hours someone is at their desk to see if they are contributing at the office. It’s visible, but it might not be work.

You could also just read the code and see if it’s good. Good luck with that one, too. You need nearly as much expertise and time as the engineer in the first place. You replace the problem of coding with the problem of measuring.

Visible work has to be easily observed so you aren’t spending tons of time figuring out what happened and when and how. And it has to be meaningful, too, so it moves your product agenda forward.

• • •

Now for the how-to.

First you need a place for the code to go. When someone writes some code and hits save, it needs to be somewhere you can see it.

GitHub, the premier platform for collaborative code-writing, does a lot to assist in the process. Not only does it give the entire group of engineers access to the same code, it also opens up an avenue for peer review—which is essential, especially when it comes to working with engineers from all across the world. You can eventually make quality assurance a task in itself with GitHub: You can have coders all checking each other, making concerns about the quality of these freelance networks basically moot. And, with save-to-deploy, a coder in China can hit “save” on a card and you’ll see the results immediately on your iPhone in New York City. Talk about visible.

The change-control features in GitHub (and in other similar tools, like Bitbucket) help you manage other risks. You can see changes and revert back if something isn't good. You can see who changed what. You and all the engineers have access to all of the code, so they go find places where some interdependency is causing problems. Essentially, it gives you a safety net for the incredible risk you take in giving near-strangers access to your entire project. What if some dummy deletes everything or inserts a virus? Just click one button.

GitHub is also your training ground and onboarding for new developers. The way the project code is organized and documented in this environment should include some "readme" files that say how to use it. You have your first engineers do this documentation from the outset. You ask someone to write a "New to this project? Here's how to get all the tools you need and begin" document, then you assign a new coder to actually do those steps. If they have problems, they complain and you ask them to edit the document for clarity. Iteratively, the team improves the readme material. And, at the end, you no longer have to train new people. You just pay them to spend a little time reading the self-help documents.

There is just one thing you need to track in GitHub. Don't get lost reading people's code. Just track "commits." A commit is when someone saves their changes into the overall repository. Like a Facebook or Twitter feed, you can see who committed what and when, and you see a short line where they explain what they committed. That's all you have to track and understand most of the time. Because this visible work is going to be tracking closely to the next layer of the stack: Trello.

Trello is a project management app that lets you plot out whatever you're working on as cards. It borrows from the Japanese system of *kanban* (in Japanese, *kan* = "visual" and *ban* = "card"). Trello is based on the board, the list and the card. Boards house your projects or products: the metaphorical whiteboard. Then there are lists, which give you columns with which to organize your projects into lists so you can understand them visually in terms of tasks. And lest you protest that projects don't always flow linearly, that's the beauty of Trello: It lets you embrace complexity and move things around, position them as parallel tasks or ongoing ones or whatever else you need.

Lists, finally, are made up of cards. Cards are Trello's building blocks. With Code as Cards, each card that you create should be, to wit, the smallest observable increment of work you could possibly observe. This is difficult, and requires some discipline. But in the end, it will ensure that you're initiating productive work that can be quickly incorporated.

And it will ensure that the people you're working with turn in quality work, too. Science tells us that we're most productive and effective when we work in 52-minute increments (followed by 17-minute breaks). Assigning small tasks to engineers, then, not only makes it easier to track visible work, it also increases the chances that the work that's ultimately turned in is up to the caliber you're looking for.

Instead of Trello, you might choose a different project management tool. Asana is one and Pivotal Tracker is another. There are many. Here are the keys, though, to making this a system for cards: a) it should be a group view where lots of people can see what tasks are posted and b) the tasks should be constrained and simple so they fit on little visible objects like cards.

By making a group view, you save a lot of the updates, summaries and “what’s happening?” project information dissemination. You just don’t need to do it. Someone expert on the project can see all the things people are working on, the things that are to do next and the things that folks say they completed. Someone entirely new to the project can eyeball the overall picture, then pick one card and say “I’ll try this one.”

A manager can glance at the lists and see if cards are moving from left to right fast enough and hassle folks that are sitting on cards.

So, just as you add someone to the GitHub repo when you first hire them, you add them to one of your project boards. Maybe you start with just one but, over time, have separate boards for separate themes of work. Instantly, rookies can see what’s happening and get started. You assign them a card yourself or maybe they pick one to try.

Now here is where visible work comes back. Taking a card assignment is visible—their name goes on it. Then it needs to move visibly. From “to do” to “doing.” And, after a little while, maybe a day or two, to “done.” When it’s in “done,” the engineer is saying it’s complete. You should see a commit in GitHub that corresponds to this; they should even name their commit with the name of the card they were assigned. You have a really clear way to see what someone is up to.

Here is where the second key ingredient of the project tool comes in. The cards have to be short and simple. A card is going to say something like:

> *Login button should be square-corners and turn blue when clicked.*

Now that’s a really simple little ask. It’s easy to understand and it should be quick to do. If you assign someone the task of tweaking the appearance of a button a little bit and they can’t move it from “doing” to “done” pretty fast, well, you have a problem on your hands.

Small, simple cards are ideal for visible work because the cards have to move. If the card demands an output—is the button blue? —then there has to be a commit associated to. The code changes. (Cards that don’t have an output are therefore dangerous. Don’t create those.)

Let’s dig in on simple a bit. Everyone comes to Code as Cards doubting whether everything is simple. Everyone thinks the blue login button is nice, but how do you explain the entirety of My Awesome App in those terms? No need to fear. Let it out. Vent a little. It’s a difficult step.

The principle at work is that every complicated idea can be disaggregated into a collection of small, visible tasks. We will explore this more in the next chapter. Writing cards effectively is really important. But, for the time being, we only need to believe that if you write short cards, the visible work system hums along.

If you write long cards, it dies. Engineers start saying they are “30% complete with a card” or “need another few days” or “it works on my machine but I need to refactor it to merge with the major codebase”. When they say this you start losing the ability to evaluate skill and will. Is this person lazy or have they gotten distracted after doing lots of good stuff? Are they stuck on a hard area and unable to realize this? Even good people can “change” in this way—something comes up or they get assigned something too hard. If you can’t track visible work at all times and just trust people at their word—people you barely know and never see working—then you are going on very little information. Don’t do it.

Not only can this be a difficult road, you won’t have everyone’s support. Your engineers themselves might hate it— especially if your organization already has some clever, talented local engineers. But, even among the freelance network you are building, you will meet resistance—from the “model builders.” Some people want to see the entire puzzle at once, and when confronted with a small task may claim that it just can’t be done, that you haven’t thought through the project in its entirety. They’ll debate you and they are smart. You may lose your nerve and back down “just this once.”

To avoid this “antibodies-against-a-new-approach” response, you have to select your team carefully. The hiring approach in Chapter 2 is designed to weed out the “big thinkers.” It’s made for the hackers-at-heart, people who are willing to say, “I don’t care about theories. I made the fix. Check it out please. Pay me.” Pragmatic folks love this approach.

To resist the antibodies-against-a-new-approach response, in the case that you already have engineers on board, you are going to struggle with the following feeling: “But I don’t have the knowledge required to make this work! I’m not an engineer!” There’s something you can tell them: “You probably do know what you need to know. Just start with something small and visible and keep moving one step by one step.” Folks can’t argue with one tiny step of progress, they can only predict you will eventually fail. Carve off a small project somewhere to try Code as Cards, away from the antibodies.

Here’s another problem you may have noticed by this point. Your developers have now committed some code and moved a card. You see movement. But what have they done really? Is it any good? Does it work? This is another step where people in the past got stymied. If you have to actually read their code, you will spend too much time evaluating. And, in fact, part of how we got here is that you aren’t super technical yourself. If you could write the code yourself you would not be reading this manual for shortcuts. So, is there a better way to evaluate the work? The next level of the toolstack answers this question. Let’s go there.

So far we have a code repo and a project board. The next place we will go is the app framework.

These days, the tools and environments people build with make it really easy to go from code to something you can use. It used to be a long process of “releasing a test build.” Now it’s really simple. You can ask your team to glue together a few things so that the following happens:

1. Engineer works on some new code, testing it locally on their own computer. Then they hit “save” and commit it to the repo.
2. Once in the repo, a continuous integration tool notices the commit and packages the entire software project to deploy it to a test server. This can also be done as a quick command or two that the developer is asked to execute whenever they commit.
3. A virtual server—like one from Amazon Web Services or Google App Engine—gets this package and runs it.
4. You can visit this web server on your browser—or on your smartphone even—and see the live code.

In our projects, we have relied heavily on the new web framework called Meteor, which is a bundle of frontend and backend Javascript code that can do pretty much all the things various web applications and servers are designed to do. It has many packages and add-ons that are easy to activate. It’s free and open source—ideal for getting your project going. They’ll probably have some business model that you can consider once your product is really big and humming.

Meteor comes with a hosting environment, so you don’t really need a separate virtual server. Your coders can type a command after they commit code and send that code to TestA.meteor.com, for instance. Then when they update their project cards, you can visit that server and see if the changes are there.

Meteor’s credo:

> *“... [the] new way should be radically simple. It should make it possible to build a prototype in a day or two, and a real production app in a few weeks. It should make everyday things easy, even when those everyday things involve hundreds of servers, millions of users, and integration with dozens of other systems. It should be built on collaboration, specialization, and division of labor, and it should be accessible to the maximum number of people.”*

Frameworks like Meteor basically make it so you can build an app without having to think at all about architecture. Architecture is one of those scary “invisible work” topics that can make Code as Cards difficult. But here inside the Meteor framework, you don’t have to analyze and debate the merits of MySQL vs. MongoDB; those choices are made for you already by the framework. A bunch of stuff comes with Meteor. You use that and just focus on your own unique needs.

There are other frameworks like Meteor for the web, so use what you want. But the main point is that the new generation of developer frameworks free you from agonizing decisions or expensive specialist greybeards who tell you such-and-such operating system plus database cluster design is needed. (This is true for the hardware side too. Once upon a time you had to pick your servers and their setups. Now you rent them from someone, with their expertise on design and efficiency already layered in.)

Another critical benefit of using a strong framework like Meteor: It's idiot-proof. As they loudly advertise on their marketing materials, Meteor is "all Javascript"—which happens to be the most widely known and used programming language. Any idiot can code Javascript. This means your hiring pool is massive. Sometimes techies get interested in trendy elite languages like Erlang or Go. They probably have their advantages. But the advantage of Javascript is that anyone can do it and so the cost for recruiting and maintaining is low.

There is a reason that so many people learn Javascript. It's not just that it's popular. It's that it is easy. Javascript handles lots of problems that other languages made coders think hard about. Architecture, memory management, syntax. Javascript is really permissive on this stuff. It fixes problems for you. You can get away with worse code. Now the purist might say "hey, this is a cheap shortcut". For Code as Cards, this is just a plain old shortcut. Technology is doing the work. Think of it as a spellchecker or the autosave feature for Microsoft Word.

Meteor is a web framework. But a lot of these truths apply in mobile platforms for iOS and Android. The developer pool for Objective-C (the iOS language) and for Java (on Android) is smaller, admittedly. But they have done a lot to make coding faster and easier. The easy-to-test facts are still true. Any time a coder hits save, they can publish a build of the software to you via a testing app like TestFlight or HockeyApp. You can get multiple versions of the app per day.

Using Trello, GitHub and Meteor in the ways outlined above basically ensures that even a novice product manager can put out a clean, easy-to-use product with minimal expenditure of time and money.

One Code as Cards-based startup put out an iPhone app in 2.5 months with out-of-pocket costs that surprised a roomful of practitioners. The startup founder managed the entire project on her own from start to finish. The app hit the App Store and has been getting initial feedback and traction. She had received quotes from ultra-lean, New York-based "minimum viable product" development shops for the project before she started. Pricetag: $150,000 and three months. Another venture capital-backed CEO heard the story and said, "Shipping that product for $50,000 would have been a steal." Her actual cost: $5,000.

The fourth layer of the stack is the highest up, and its job is to help you coordinate the overall work. You can dive in to the first few layers and produce results solo as we have laid out. But as you start managing more than 10 engineers, you will need some help and coordination: product management help. Roughly one manager is needed for every ten engineers. Questions come

up, feedback is needed, cards need to be tested and refined. So working that pipeline of commits and cards and testing requires work and attention. And, of course, you have to feed the pipeline. With this big team working away, someone needs to think up all the tiny fixes and improvements and new “cards” that need to be written. (More on the work of product management in the next section.)

In this section though, comes the question of visible work and coordination for the product management team, and the tool we have built and use for this at Knote is. . .Knote. Here is where we manage the discussion, prioritization and decisions of the product managers who are working on big bundles of product features or tasks.

A “knotepad” is a shared board or notepad where a few people can see, post, and edit “knotes.” The knotes are messages or lists or notes like “these five things are my top priorities this week.” Since everyone can edit them, they can be kept up to date. They work seamlessly with email—you get an update when there’s a change and can reply back with your comment—so it stays integrated into people’s inboxes.

The knotepads let you comment and annotate, so you can have a more complicated discussion than in the linear thread structure of email. And you can use more than words—post task lists, polls for making decisions, deadlines, files—so it has some simple project management aspects that make it more functional than just writing a weekly email. In fact, using Knote in this context largely replaces the “weekly meeting” or “can we meet and discuss this change we want to make?”

Yep, you read that right. Using knotepads can liberate you from the routine of weekly meetings and meetings about meetings—which have become longer and more frequent over the last several decades. According to the *Harvard Business Review*, executives spent less than 10 hours each week in meetings in the 1960s. Today, they spend upwards of 23 hours a week.

This is great stuff. Everybody wants fewer meetings. In fact, nearly half of employees—who spend an average of 31 hours in unproductive meetings each month—would rather do just about anything else than sit through a regular work meeting; 17% of them would rather watch paint dry! Entrepreneurs and executives want fewer meetings, too. Collectively, U.S. businesses waste $37 billion in salary expenses each year on meetings.

Why else do we hate meetings? There are endless reasons. Some of the classics:

- They’re often unnecessary, i.e., “meetings for the sake of meetings”
- They veer off topic
- People repeat each other over and over again
- People don’t pay attention
- There are technical issues (e.g., “I’m having trouble with PowerPoint. Please bear with me!”)

- One person dominates the conversation
- The great ideas the team just brainstormed never get implemented

OK, enough about meetings. We can agree nobody likes them. So how do knotepads help eliminate meetings anyway?

While engineers are making commits that reflect cards moving and can be tested in your phones web browser, the product managers are updating their knotepads to show where the progress is and even working through decisions through back-and-forth discussions with you.

As with the engineers, you will meet business people and product managers who “just want to get in a room and discuss” or who repeatedly turn back to email threads. But the benefits of the toolset are important.

Canceling all meetings from the workflow means you never have to be in the same place at the same time. You and your people are freer. You can work together perfectly asynchronously. You can get the best people and get the best parts of people (maybe they only want to work evenings?).

Canceling all meetings also means that everything you discuss and decide is visible work. There is a pad with a task list or a chain of comments analyzing a decision. You can bring someone into the “discussion” after it has happened and get their full weigh in.

You can do something you might have had a meeting about “slowly”—brainstorming ideas for names over the course of time or collecting research on a new feature. You can let people get into “flow,” so they have big blocks of time to get into detailed analysis and planning of the next key feature or test complicated scenarios in their products.

We all know meetings suck. These are just of a few of the reasons why. And with knotepads, you can avoid them. And on the other side you can avoid the other universal office pest: email. Emails get fragmented, lost, go out of date by the time you read them, fail to move the conversation forward and all the rest. A way to focus work without email is what you get with a knotepad.

Now you have the last layer of the stack—a way to make knowledge work like planning and designing features into visible work, and to extend the effectiveness of Code as Cards’ global networks of freelance engineers to the product management function itself.

In the next chapter, we’ll go over what comes next. You’ve hired engineers, you’ve gotten the toolstack together and you are about to start assigning some cards.

It’s time to figure out what those cards and pads should say and what your product managers should be writing.

Chapter Four: Moving the Cards

So you've come this far. You've got an idea and you've gone through the necessary channels to find the right people for the job. You've hired programmers through Elance, set up GitHub and Trello, and gotten started on the project.

Homestretch, right?

Well, somewhat. Software development, even using these techniques, is not something that runs on autopilot. Even with frameworks like Meteor that let novice product managers coordinate an app's development, there is a (relatively small) amount of active participation required on your part—or on the part of the manager you've hired!

In the last chapter, we talked about visible work. That's what you're hiring these engineers for. But even with visible work, there's some maintenance required on the part of management to make sure that that work stays visible. What do I mean by that? It's easiest to think of visibility in four layers.

The first layer is the engineers and contributors who have signed on to your project. The second is what happens on the level of apps like GitHub; like the "Track Changes" feature on word processors, GitHub records each instance of an engineer taking a piece of code and changing it. You don't quite know exactly what kind of a change was made, but you know something's happening, you know that an engineer has made a commit. The third layer is project management apps like Trello, where you can contextualize those changes in terms of the larger project. And the fourth is the live working output of the actual code itself. Working in a framework like Meteor is critical here. With this fourth layer, the first three come together into something tangible, something that you can actually look at and measure to see if it's successful: live, committed code.

In Code as Cards, project management is all about making sure this evolution, from the first to the fourth layer, goes smoothly. The developers at Tint published a blog post recently explaining their approach to management. They found that they had some issues using the kanban system, previously discussed in Chapter 3, because members of the team were at odds when it came to knowing who was responsible for moving cards around on their Trello board.

The Tint developers optimized their use of Trello by ensuring that each card represented a small action item. This is essential: Cards cannot represent anything more significant than a single developer can finish in a small amount of time. If too many items are incorporated into a single card, you open up a space for your engineers to perform invisible work. Keeping things small and simple ensures that tasks get done one after another after another.

Think about it this way: If you set up a checklist for everything you need to do in the morning before you go to work, how are you going to set it up? You could tell yourself to make breakfast,

get dressed, clean up and get ready for work. Or you could tell yourself to eat a cup of Greek yogurt, put on your jacket and pants, shower for fifteen minutes and pack your suitcase. One of these methods introduces ambiguity; one of them cuts down on it as much as possible.

But this is to go back to Tint's first point about the kanban system being ambiguous for them. This is a problem of project management that can be solved quite easily: Just give one single person the authority and responsibility to check in on each phase of the project and move cards as they need to. Simple as that! Cut down. Of course, keeping cards small is a challenge in itself. This is because the idea of Trello inherently involves compartmentalizing your project into many small pieces. What ends up requiring discipline is enforcing a mentality of moving just the cards—which means that your cards need to, in their totality, be the project.

Now, things do get a little complicated once you get past ten or twelve engineers working under you. This is where what I said earlier about hiring product managers comes into play. Ten or twelve is really the maximum number of engineers that can be handled by a single manager. Beyond that, you need to hire people to oversee the visible workflow, people to follow up on cards and make sure they move, people to write good cards as things come up and people who can delegate duties.

For proof of this, one only need look at a comparison of retail giants Staples and Amazon, as discussed in a recent issue of the *Harvard Business Review*. Staples, the way Andy Singleton puts it, has large teams build software enhancements that get rolled out every six weeks, which are then sent to another team to be tested for three weeks, in what IT professionals might call a "best practice."

Amazon, on the other hand, has split up their entire infrastructure into thousands of compartmentalized services. Teams handle only a handful of services at a time, and once they're done, they release them. A change is released out of Amazon about once every eleven seconds, which according to Singleton means that Amazon makes 300,000 changes every time Staples rolls out a new release.

This is the epitome of Agile development—and it is the way that a software entrepreneur needs to think in order to successfully develop a new app.

A brief refresher: The Agile software development movement emerged in 2001 with the publication of *The Agile Manifesto*, a list of twelve principles that guide most leading engineering teams today. The Agile approach focuses on cross-functional team collaboration, gathering feedback from end users regularly and using that data to continuously roll out software updates.

The Agile movement has been accompanied by the rise of DevOps teams, which bring software developers and people from operations together in attempt to accelerate development lifecycles and push out updates more frequently. DevOps teams move quickly while focusing on all aspects of software, including testing, integration, releasing, monitoring and infrastructure.

According to a recent survey, when compared to their slower-moving peers, DevOps teams roll out 46x more software updates, recover from failures 96x faster and enjoy 440x faster lead time for changes.

Succeeding in the incredibly competitive world of software is much easier when you're able to build quickly and fix mistakes fast.

Forming large, clunky teams and passing massive projects around produces "slow and steady" results whereas the Amazon approach (i.e., Agile/DevOps) produces a lot of changes very quickly. If you're wondering which approach works better, remember which company is Amazon and which one is Staples.

This all points to one of the essential truths about Code as Cards: a lot of this ideology is already at play in some of the major players involved in developing software. Another *Harvard Business Review* article heralds the arrival of a new theory of product development most prominently seen at Google: "To manage new software releases at their huge scale,

Google has replaced traditional testing systems that depend on people with a testing machine, known as a "continuous integration" system... Continuous integration and automated testing is important for all modern, large-scale software development." We would add only that it's as if not more important for small-scale software development.

The beauty of frameworks like Meteor and apps like GitHub and Trello is that they allow for this kind of rapid production to happen very easily. Like we've said before, the fact that a coder in China can hit "Save" and you can see the results immediately on an iPhone in New York means that you can continuously assess the visible work your engineers are putting out, that you can constantly test and that you don't have to waste time with bulletproof schedules and lengthy periods of gestation.

In Chapter 3, we talked about the kinds of cards to assign: a log-in needing to be blue with square corners, for instance. This is product development with small, disaggregated cards. Imagine a storyboard for a mobile app.

Here's the key: Assign one task. The first one that would produce valuable results. And then assign nine more. What you need, more than anything at this stage, is momentum. Make a homepage with a logo. Then make it blue. Put a sign-up button. Create a form that follows. Have an email sent when that form is filled out. Build a database to save sign-ups. Add a login button on the homepage. Virtually every app is going to need something like this, so start there.

We brought up Google a moment ago, so let's look at them for a second.

The beginning stages of Google, obviously, would be pretty straightforward—at least from a user-experience perspective. But one of the primary factors in Google's success has always

been kind of an abstract thing and that's what happens in between you clicking the "Google Search" button and the page of search results you get afterwards: speed. How can you assign "speed" as a task to your freelance network? Quite simply, actually.

It goes back to our Peter Drucker quote from Chapter 3: measure and improve. Start off by being specific. What's too slow about the app? What screen? All of them or just the first one? The login page? Here's how you fix it: tell a developer to put an onscreen alert in as soon as the server is hit. Then install a timestamp. Then have the developer add another timestamp once the page fully loads and show the full time elapsed. Now you can assign a rather straightforward task: lower that time. That's just an example of how to turn an idea like "speed" into a step-by-step process.

This is also an example of where your crafty engineers may throw up their hands in exasperation. "No, this won't work unless it's done exactly this way," they may say. It's somewhat similar to an old argument about wings and evolution. What good is half a wing, the creationist says. Evolution could never produce a wing straightaway, so it must be God that created them! But it's not quite like that: Wings didn't start as flying implements. They began as tiny growths, eventually becoming delicate protrusions of feathers that helped keep the animals that had them at a stable temperature. And further down the line, there were so many feathers that animals started jumping a bit better, even gliding a little bit, to catch flies, for instance. And eventually you get to fully-winged animals—all in a step-by-step process. This is product strategy. Get momentum. Build something interesting and useful right away. Get your freelance network working and showing that they are useful. And start producing output that you can show other people. Get daily updates to the app that you can try out on your own. This is how products progress.

Chapter Five: Doing Everything Through Cards

In the last four chapters, we've gone over almost everything you need to know about building an app from scratch in an affordable and efficient way—in short, a way that has a much better chance of working than your conventional method. No more attending lame meetups trying to find a hacker genius CTO or reviewing proposals from "friend prices!" consulting firms that quote 50k or 100k for something you can draw on one side of a napkin. No more massages or free food or gym service for a crew of 100+ engineers who play ping pong two hours every day. No more waste and laziness.

Work in general

Some other bonuses you may have noticed: Your people are now all over the place, not in a local office, so you don't need one. The time zones are all varied, so you never have meetings to review things. You essentially have no schedule on a given day that requires you being in synchronous meetings with someone. You do it when it works for you, they do the same and it all gets done.

Visible progress is not in the doing—meetings or calls—but in cards and commits. As long as the cards are moving and the code is going live, you know that work is happening.

But all we have changed is the software engineering department in this discussion of Code so far.

An exciting prospect you may have already been thinking about: Cards can work higher and higher up in the "work stack" —up to the levels of thinking, planning and coordinating.
Of executing strategy, marketing and operations.

Let's move one critical layer up to see how. Up to the layer at which you and your peers—the product managers, marketing planners, sales leads—are going to think through and dictate the flow of work. In part we are thinking like the user or consumer or customer here and in part we are turning around and saying "build this." The power of Code as Cards is that this role, which is basically product management, which can be played by someone who is close to the person intended to use the product.

As your customer-facing group grows, you will be a team of product managers. If you do everything solo for a small Code as Cards experiment, you may try to stay organized inside your own head. But in a bigger effort, you may have folks you are consulting for advice, assistants you are enlisting to drive families of cards and test them and thought partners with whom you are dividing up the work. Whether you are one or many, managing lots of Cards effectively requires one to take notes, organize tasks and discuss important choices.

What's our recipe for this?

Before we see the recipe, let's see what the meal should look like. In Code as Cards, we have been managing to achieve visible work and have defined that as code commits and moving cards. But your team of product managers won't commit code. The work produced by product managers is information exchange, discussion and thinking and eventually decisions.

It's easy to come up with and refine ideas such as "Make the button blue" or "We should add a pop-over on this screen to make it easier to understand." The problem, of course, is that the work associated with coming up and refining such ideas is too difficult to see with your eyes. How do you know that it is happening if you are not present to witness it taking place?

The one missing piece of our toolbox is the one which aids in this high-level communication, planning and decision-making that any large-scale project needs.

You will occasionally need to communicate vital details to your engineers, your managers or other people. There will be times when GitHub and Trello will not be enough when it comes to actually explaining what you need done or having a frank discussion or brainstorm about some feature of the project.

How higher-level thinking work can change

Conversations and decisions regarding the messaging or positioning of your project don't belong in columns as Trello cards because it is in their nature to move across the board toward a finish line. Sometimes you need something to live in one space and evolve over time into its final form.

Chances are you are using two stalwart office regulars: email and meetings.

Email doesn't really lend itself too well to this task. There are lots of companies and teams that primarily use email to communicate with their employees and those employees end up using email to communicate with other employees, too. This creates an overwhelming amount of information to wade through; McKinsey says the average worker spends 13 hours each week sifting through their inbox. You've probably experienced the feeling of not checking your email for one day and then being completely lost on whatever's going on at work.

Meetings work better. But they have their own drawbacks that the majority of office cultures are familiar with. It's simple physics. You have to be in the same place or at the very least available at the same time. In our digital age, it has become much easier to hold a meeting with people who are not nearby but are still sharing the same metaphorical space: a telephonic or video conference room. It isn't possible for them to participate on their own time. The meeting requires people to be together in at least these two dimensions. It is also difficult to bring people to the

same space and time reliably. People will have the wrong dial-in, or they'll be late or they'll miss the meeting entirely—especially when multiple time zones are involved.

Both email and meetings struggle with one last thing: scalability. With email, as you add people to your project and more and more discussions begin, you will find that you have too much information to wade through. With meetings, as you add people to your project and more discussions begin, you have too much wasted time. The people tied up in an unproductive meeting could all have been doing something valuable.

Ultimately, everything comes down to this: Your team's time is valuable. Every time you are tempted to call a meeting, consider its real cost. Let's say you pay your team members $50/hour and you invite 10 of them into a 2- hour meeting. That's a $1,000 meeting. Someone once suggested that before you call a meeting, it's worth considering whether you would throw a computer out the window to get what you want out of a meeting. Because that's the real cost of the meeting—even though we don't often think about it that way.

Here's our tool for high level work

Knote was built as a better way to get people on the same page. A tool that's more organized and more helpful than email—and less time-consuming, and more flexible, than meetings or calls. Knote compartmentalizes conversations and organizes the work being done by the people you've hired to manage your projects. And importantly, it allows you to stay on top of everything on your own time—that is, without dragging your whole team to a grinding halt.

The spirit of this book could be summed up as this: finding the right tool for the job. And, for something where a team needs to be constantly informed and be constantly on the same page, able to share new information but easily refer to the old, Knote shines.

Knote augments both email and meetings in order to address them with the goals of fixing and improving them—but also with the goal of ultimately replacing them as much as possible.

Knote is a platform that lets you move email threads into shared spaces where you can do way more than what you could do with email alone. You can share information across spaces, merge threads and split them, add deadlines and elections and checklists and other things. When it comes to building a full-scale app, whether it's for the web or a mobile device, you're going to have a lot of things going on at once. Knote makes handling many parallel assignments a breeze. If you've ever used email, you understand that it's not the most graceful task manager in the world.

OK, pause for a second. What's all the Knote talk? Well, we are talking about Knote because it's a tool we made. We made it to address exactly the types of problems and opportunities about collaboration we have been discussing all through this book. Some things already existed—like freelancer networks, global talent, shared software repositories and collaborative task

management tools. But one thing didn't exist: a place to put your thinking so teams could compare notes and move their agendas in a way that used to be mostly talking, meetings, emails and whiteboards. We made Knote so you could think about stuff collaboratively.

Which is why it's worth noting that the entire system of this book—Code as Cards—and the toolkit we are reviewing right now is a system that goes way beyond just software development.

Think about the production of a television show or a movie. Even a low-budget reality television show will involve a litany of emails, not only to the entire crew but to well-defined subsets within the crew: production, camera, audio, etc. Add to that the fact that there are emails that need to be sent out every single day, such as schedules for the next day of shooting. This kind of information becomes neat and easy to navigate as soon as you start updating it daily on a centralized board rather than sending out new iterations every single day. You have knotepads for different teams to consult: the latest info, agenda, some Q&A, some files to download. No more daily email blasts. Instead, a page you can go to (or app screen to open) that has the latest information.

Or imagine the team at an architecture studio working to design a new building. Knote means no more overwhelming writing and rewriting of product specs and requirements, no more flow charts and drawings. One person talks to the clients and gets a bunch of replies posted onto a pad. Then they remove the client from that knotepad and bring in the internal team. All the info is there. The pad goes from info gathering to project management in one pivot. Knote eases it all into one space and allows the product to take on multiple threads as proves necessary. Say you have one thread for something like your initial thoughts on the space where the building will go up. You can build a thread for this on Knote, go by the space and take a bunch of photographs, then upload them to the thread along with diagrams or blueprints of what's to come. You can write a bit, and then later, organize what you've written into a cohesive text. Others who you copy on the thread can then respond via email, and yet everything will be maintained on this one repository that you created and therefore control.

Members of your team who insist on email can still use it, which makes Knote's application totally seamless. It supports "grumpy mode" for people who don't want to sign up and prefer to sit in email. If you're a part of a Knote thread, then you can receive updates via email and your email responses will be incorporated into that thread. But those who do choose to use it will find that it makes working on a project with a lot of moving parts a whole lot easier.

And you'll definitely find that it makes owning your project easier as well. Think of all the excruciating meetings you've sat through that would have been completely unnecessary if members of your team were simply empowered to write up an update about what they were doing. Or think about the many time zones in between New York City and Delhi. All of the annoyances involved with Agile development—these are out the window. No more daily or weekly "sync-up" or "retro" meetings to update each other on what tasks everyone has been working on. No more frantic email sessions trying to deliver the right update to the right person

in time. Why bother, when your team members have a place to blog about their process and priorities in such a way that you can come in and modify, edit and expand their work —that is, communicate without the hassles of old methods.

OK— let's end the Knote section. Now for a recap of past, present and future in the way people work:

The old, now, and new of human talent, capital, workflow and management

	Old	Now	New
Keeping in sync	Sit near each other, chatter	Emails, team calls, sit near each other	Streams like Yammer or Slack
Making Decisions	Get in a room together and voice opinions	Email chain then a call	Discuss on a knotepad or a card
Keeping up to Date	Sit in a meeting while someone talks	Get emails with many cc's	Follow an internal blog or knotepad or project page
Learning new stuff	Go to classes, internal meetings	Search the web, teach self	Start doing it and get the tutorial in real-time

Managing a project	Weekly meeting, all give updates, track status to an overall plan	Project is many micro tasks in a project app, weekly meetings to review	No projects. Just tasks. Software creates a timeline and forecast automatically
Collecting work products	Thumb drive	Email attachment, post a Dropbox Folder	Post to the relevant task e.g., into a repo or project item or knotepad

What else can you transform with Cards?

We have used examples in a wide range of fields—software development, architecture, films. What can't profit from the Code as Cards approach?

Put differently, what are some of the really exciting areas that you can tackle using this approach? What aren't you doing 10x that you could be doing 10x faster, better, cheaper and in your spare pockets of time? Some examples from our work.

Publishing. We noticed that big publications like *Huffington Post* use tons of freelance writers. We rebuilt it ourselves using Code as Cards. Editor, writers and the product team were all given assignments using this approach. We hired 50 writers using some Craigslist posts ("We'll pay you $100 for your first post about the topics we list on this link <here>. If we like your work we'll pay you $75 per post."). We ended up only spending $5,000 to hire 10 writers who wrote 300 posts over three months—some of which got so popular our blog crossed 2 million unique visitors in the third month after launching. That's huge. You can hire, manage and push forward writing/editing using Code as Cards. Have a look at science.knote.com to see what we made.

Design. A lot of the design in our apps is done by freelancers who look and behave very similarly to the engineers we work with. The recruiting pools are a little different (Behance, Dribbble) but the process is quite similar.

Chapter Six: Content Through Cards

You used code to understand cards.

What if your job is a marketer or journalist or writer? What if you're an executive who needs to put together a massive S-1 IPO filing? Can you apply the principles we've learned in the first five chapters to a field that is entirely different?

But of course.

1. Marketing through cards

Whether you're launching an app, selling a new service or trying to promote your landscaping company online, you're going to need content.

Content marketing—blogs, infographics, ebooks, social media, case studies, etc.—is not only cheaper than traditional marketing, it's also more effective. According to one study, content marketing can help you generate three times as many leads as traditional methods while costing 62% less. With the right approach, it's a win-win. It turns out 70% of consumers prefer learning about products through content than other mediums.

But creating content takes time. HubSpot says the average marketer spends 1–3 hours writing a single blog post. That might not sound so terrible. But HubSpot also says that companies that publish *at least* 16 blog posts per month (i.e., new posts on roughly 80% of business days each month) get 3.5x more traffic than companies that publish four or fewer blogs over the same period.

As an entrepreneur, you barely have time to write one blog post every month. And that's assuming you have superior writing skills, which isn't always the case. (You can't just hammer away at your keyboard and call it good content, after all.)

All of a sudden, you have a content problem. To drive installs of your app, you need to pump out a steady stream of engaging content.

But, if you followed the principles covered in this book, you don't have the time. It's your number one resource, after all. You are already busy enough as it is. At this stage in the game, money's probably tight, too. And, depending on how much Joyce and Faulkner you read in college, you may not have the writing skills to begin with. That's okay.

If the Code as Cards system can work for engineering, why can't it also work for marketing? I didn't see a reason why it couldn't so I ran an experiment. I thought I'd be able to delegate furiously to create a wealth of content while avoiding putting pen to paper myself. To do that, I'd involve other great minds and set them loose. But how?

In October 2014, I decided to bring a *Huffington Post*-style approach to our marketing efforts at Knote. We were building a productivity app, so we wanted to create an enormous repository of content highlighting all kinds of productivity tips and tricks. The site would also contain interesting stories about influential figures from the world of productivity—-like W. Edwards Deming and Vilfredo Pareto. *HuffPo* is built on top of an enormous network of freelance writers, and we figured we should try something similar using the Code as Cards philosophy while building out our site for Knote.

To build content, we needed writers. And not just any writers. Good ones.

First came the inaugural Craigslist post: “We’ll pay you $100 for your first post about any of the topics we list <here>. If we like your work, we’ll pay you $75/post moving forward.”

We got about 50 applicants in no time at all. To be frank, some of the trial posts were awful. Others were so-so. But there were also several posts that were *awesome*—exactly the kind of material we were looking for.

Recall our lesson from Chapter 2: Spending $100 to find out whether someone is good, i.e., whether someone is worth spending *another* $100 on, is a small price to pay to gauge someone’s merits. After we paid the lackluster candidates, we quickly told them it wasn’t the right fit and moved on to the people we wanted to keep working with.

When the dust settled, we ended up spending only $5,000 to hire 10 writers who wrote upwards of 300 posts in three short months—some of which got so popular our blog crossed 2 million unique visitors in the third month after launching. That’s huge. (Head over to science.knote.com to see what we built.)

Freelance writers are an interesting bunch. Those 10 writers we hired? Over time, some of them dropped out. A few others let their work go after a while.

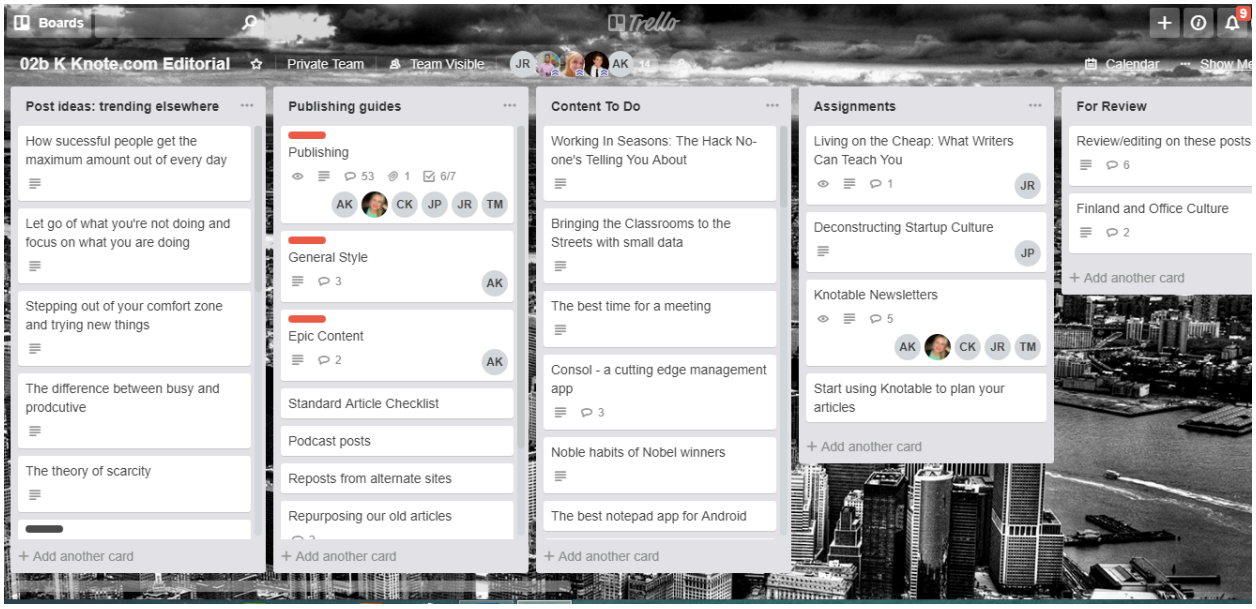
Ultimately, we ended up with a stable of five rockstar writers who were great at their craft, reliable and easy to work with. These folks ended up helping us build the Knote site over the course of several years.

We had a marketing team. What we didn’t have was payroll taxes, equipment and furniture expenses, office politics, health insurance costs, or any kind of formal long-term commitment.

As our site began to grow, we offered one of the writers an opportunity to serve as managing editor of the blog. She obliged. This was more of a coordination role—making sure we had enough content ideas coming down the pike and enough people writing them. Writers, who were given bylines (i.e., skin in the game), were all responsible for editing each other’s pieces.

True to Code as Cards philosophy, we used Trello to manage our blog. Here's how our process worked:

- Our internal team would scour the web for trending topics that were relevant to productivity and we figured might generate significant interest on our site.
- We'd assign writers basic content aggregation tasks to test headlines and see which topics were most appealing to our audience.
- Once our audience indicated their interest, topics would then be moved to a "Content To Do" column.
- Next, topics would be assigned to writers with specific due dates; cards were then moved to a column called "Assignments".
- The writer would spend time researching the topic and then create original content around it. Once the piece was written and optimized for SEO, the writer would move the assignment to the "For Review" column.
- Next, another writer would review the piece, making minor tweaks here and there. When that person was done, the card would be moved to the "Title Selection" column.
- Once a title was picked by the managing editor, the piece was published, and the card was moved to the "Published" column.



Note the "Publishing Guides" column in the screenshot above. That's where we'd stick publishing tips and cadences, style guides, checklists, and other helpful resources. In the event a writer had a question, they'd know where to go to find an answer. Nobody's time was wasted.

This whole system doesn't seem too hard, does it?

And trust me, it's even easier when you hire people you can trust. During our heyday, science.knote.com was an incredibly well-oiled machine. I had good people taking care of things for me. I knew that the content would be there and I knew that the content would be good. And I knew that, after a while, I didn't even have to so much as lift a finger to make it happen (until it came time to approve payments, naturally).

In other words, our Knote content site was growing. And it was hardly occupying *any* of my time.

Which leads me to the next hidden bonus from this approach.

Whether you're hiring engineers, writers, graphic designers, or whoever else for a specific project, when you find the right people, you may never have to repeat the process again.

Case in point? The writers we hired for Knote worked with us for several years. But some of them have followed me around: from Knote to Halo Neuroscience to Knotel, with various projects stitched in between. Same with some of the engineers I've hired over the years.

So, not only is this approach to hiring quicker and more cost-effective than traditional methods, odds are you'll end up with at least one or two people you could see yourself working with for the foreseeable future. Spend a few hours building up a stable of writers, engineers, and product managers, and you'll know exactly who to ask for help when you need it on a future project.

2. Journalism through cards

Cards work well for marketing needs. They can also support news production.

Let's say you're an editor or producer who's managing the next several weeks of news coverage for a regional newspaper. You oversee a team of 20 reporters spread out across several states, many of whom are collaborating with one another on stories.

You can use cards to coordinate all this coverage. Create cards, populate them with story ideas, and assign them to relevant reporters. Tell your team to update the cards as they move their stories forward, writing notes about completed interviews, potential new angles, follow-up ideas, and the like.

You'll want to create several columns, too, so you can easily track where things stand: Story Ideas, In Progress, Ready to Edit, Needs Revisions, Needs Images, Ready to Publish, and Published.

All of a sudden, your inbox is decluttered and you can manage an entire distributed team, all of whom are working on several stories at once, from a single interface.

3. Writing projects through cards

Writing a novel? Helping your grandmother finish her memoirs?

Cards can help here, too.

As we've learned earlier in this book, Code as Cards enables us to move software development projects forward quickly and affordably by breaking down large ideas into the smallest tasks possible. This philosophy can be used to bring your writing projects to life.

For example, let's say you have an idea for the next great American novel. From the outset, you're not sure how many chapters the book will end up with. But you do have a pretty good idea about what the introduction, body, and conclusion will look like. You know what the main conflict will be and you're also familiar with several of the characters you intend to explore and what makes them tick.

Before you even write your first sentence, you can begin organizing your story on cards. Write little synopses about each character—information about their background, interesting events in their lives, what they do for a living, what they're like when they're just hanging out, what have you. You can keep adding to these cards as you get to know your characters better.

You can also write little vignettes to describe the beginning, middle, and end of your story. As you continue moving your novel forward, you can break those three sections down into smaller individual chapters—applying the cards mindset to your writing.

While this approach may mean you have to spend more time writing your book, it will result in a more organized and cohesive narrative.

Use a similar approach to help your grandma get the words to her memoir down on paper. Since cards help you visualize your progress, you'll increase the chances the book is eventually completed.

4. Work and academic projects through cards

Coordinating a team effort to produce a major document like a company annual report? Putting together a massive research paper for a college course?

It's much easier with cards—particularly when you're collaborating with other folks.

Let's say you and your executive team are putting together an annual report. Use cards to break the report down into several sections: Chairman's Letter, Business Profile, Analysis, Financial Statements. Depending on how big your company is and how thorough of a report you want to create, you may also want to break those sections down even further. For example, Financial

Statements can be split up into Income Statement, Balance Sheet, Cash Flow Statement, and Statement of Retained Earnings.

Once you've broken the project into small pieces, you can begin assigning cards to the relevant people and move them from column to column as their efforts evolve from drafts into finished copy.

Remember, your time is finite. Team work = less work for you.

Finally, if you're a student, you can use a similar approach when putting together and end-of-the-semester paper or dissertation. Use cards to get all of your research organized.

The more you're able to break down a large project into smaller sections, the easier it'll be to complete.

We've spent the last several chapters exploring a new way to work and how you can use the approach to not only build transformative products, but also power other departments. Now, it's time to turn our attention to the mindset needed to pull all of this off.

Chapter Seven: Mindset

Not everyone is innately wired to manage an operation under the Code as Cards system.

The people driving much of your success, after all, can be based anywhere in the world. You're unlikely to ever meet many of these folks in the flesh—if you ever meet *any* of them to begin with. The nature of remote work means you can forget about looking over anyone's shoulder as they're plugging away on various tasks.

Thriving under this system requires a specific approach. Micromanagers and worrywarts need not apply.

Recall our discussion from Chapter 3. In the Code as Cards system, you're hiring people who can do the actual work—and do it well. Maybe you'd be more at ease if you were working out of the same room as the people you're paying. But when you hire the right talent, you should be able to tell them what to do and expect them to perform to your specifications without much involvement on your end.

For many people, this managerial approach is much more hands-off than what might be taught in business school—or in the school of life. But it's an approach that works so long as you're willing to give it a try.

If you're feeling a bit uneasy about the prospect of lightly managing a self-motivated group of folks from around the world, don't sweat it. Never forget you have the power to change the way you look at things.

Dr. Carol Dweck, Ph.D., pioneered the concept of mindset, i.e., a set of assumptions and ways of thinking that governs our outlooks, actions and behaviors.

Dweck argues that, at a very basic level, there are two kinds of mindsets:

- A **fixed mindset** tells us that our talents and skills are limited. We're dealt a hand of cards when we're born and we're forced to play that hand for the rest of our lives. Managers with fixed mindsets may look at their employees as cogs in a machine that can take care of certain responsibilities—but their abilities stop somewhere.

- A **growth mindset** says that we can continue developing our talents and skills as much as we want through education, training and hard work. We're dealt a hand of cards when we're born, but that's just a starting point. Managers with growth mindsets may look at their employees as continuous works-in-progress who are always becoming better at what they do—and therefore are able to take on increasingly complex or important tasks.

Thriving in the Code as Cards world requires a growth mindset. Per Dweck's research, that mindset can spread across the company and be adopted by the people on your team—leading to happier employees, more innovation and more risk-taking.

Think you're governed by a fixed mindset? Good news: You don't have to be. You can switch to a growth mindset by simply changing the way you think about the world.

To some extent, that might be easier said than done. But it's still possible.

Beyond having a growth mindset, you also need to have what we'll naturally call a Cards mindset.

Working with a Cards mindset is a lot like treating your project like a startup. Writing a new book? Opening a new restaurant? Planting a new garden? Building a new app? A project is a project and all projects start the same way: as ideas you can scribble on the back of a napkin over cocktails.

If projects are like startups, founders can teach us a lot that we can apply to the Cards mindset.

I taught a class on entrepreneurship at Columbia University during the Spring 2017 semester. In one lecture, we explored Han Zhao and Scott Seibert's 2006 research that illustrates how the five personality dimensions relate to entrepreneurialism.

While everyone is different, Zhao and Seibert uncovered several similarities successful entrepreneurs tend to have in common:

- **Openness.** Successful entrepreneurs are open to experiences. They're driven by the growth mindset and they understand that new experiences help them learn new things, increasing their intelligence and ability. Entrepreneurs, who aim to transform the world, are curious and creative by nature. They're good at using their imaginations to envision a future that doesn't yet exist. In addition to experience, entrepreneurs are also open to feedback. They might be confident in their abilities—cocky to a degree even—but they are aware of their limitations (or at least should be!). So they take feedback and use it to iterate and improve upon their ideas further. The Cards mindset requires a similar degree of openness. You first need to willing to experience the system. You need to get creative, thinking about how you can split enormous ideas and projects into tiny tasks. As time goes on, you need to listen to your people and take what they have to say into consideration as you refine your system further.

- **Conscientiousness.** Studies have shown that conscientiousness is linked to higher salaries, increased job satisfaction and more job security. This makes sense: People who are conscientious are usually more organized, which means they waste less time looking for things and spend more time getting things done. Conscientious people are

practical. When they don't achieve a high goal they've set for themselves, they don't give up. They might aim for something a bit lower or something a bit different (or even drastically different). But that beats throwing a tantrum and bringing the folks around you down with you. You need to be conscientious to thrive with a Cards mindset, too. It'll be impossible to pull off Code as Cards if you're disorganized. How can you manage other people if you can't even manage yourself? Every project probably won't work out exactly as you hope. Instead of admitting defeat, the conscientious builder rethinks and refines their approach—or, after enough attempts, moves on to the next project altogether.

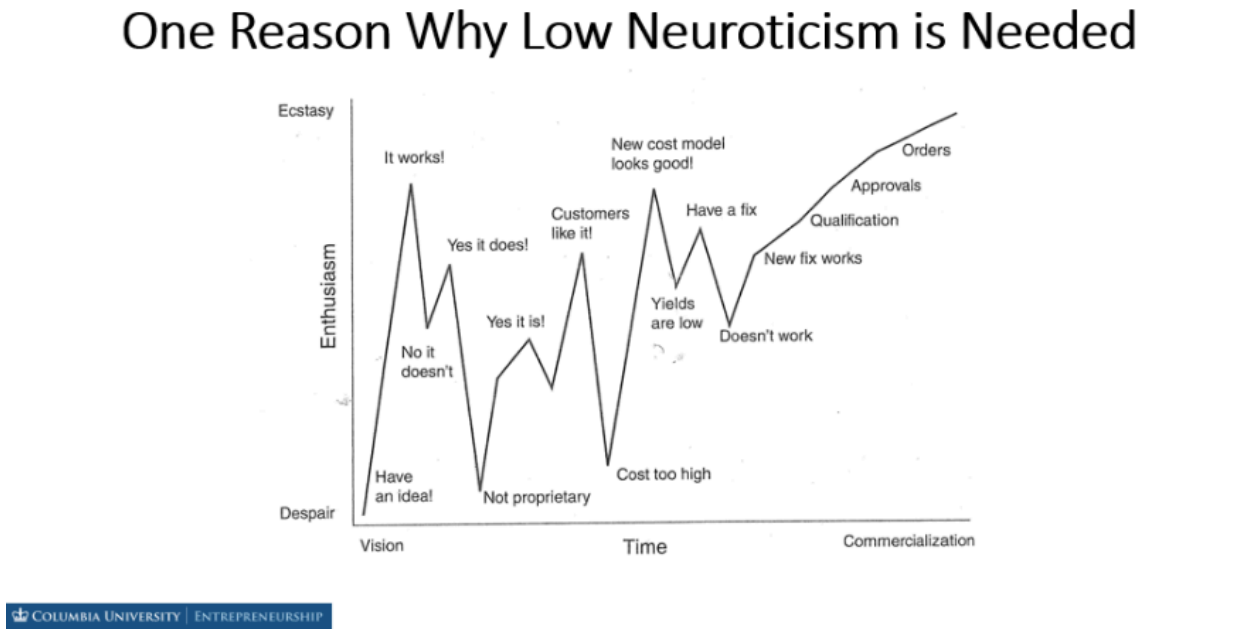
- **Extroversion.** Extroverts like being around people. They're outgoing and talkative. You'll find extroverts at bars, networking events and other social functions while the introverted among us prefer curling up with a good book or talking with a best friend. Because entrepreneurs need to talk to a *lot* of people—investors, peers, media, employees, what have you—and attend a *lot* of events, it's not much of a surprise that many of them tend to be extroverted. This is not to say that quieter folk can never make successful entrepreneurs. Rumor has it that people like Mark Zuckerberg, Larry Page and Bill Gates are introverts (or *were* introverts in their past lives before their fame and success forced them into more extroverted lives). Since the Cards system generally involves managing remote workers (or, in a case of say, party planning, friends you "manage" through the computer), you might not have to be as extroverted as the average entrepreneur to pull it off. But extroversion will not hurt when it comes time to raise money, market your idea or hire your first employees. It won't hurt when you tackle collaborative projects with other people, either (e.g., your content from the last chapter).

- **Agreeableness.** People don't change the world by agreeing with everyone they come across. Just ask Galileo, who was famously convicted of heresy during the Spanish Inquisition for arguing against popular opinion that we lived in a heliocentric universe. Time, of course, would vindicate the Italian genius. Unsurprisingly, entrepreneurs are *not* agreeable. (Social entrepreneurs, who focus on solving problems to enact positive social change, tend to be more agreeable than other entrepreneurs, however.) While you need to be open and accepting of feedback to adopt the Cards mindset, you also need to be stubborn—at least to an extent. You have an idea of what you want to create and how you want to go about building it. You need to drive that idea forward while relying on a distributed group of people you've never met. Doing that requires being less trusting and caring than you might prefer to be in your personal life. In the event you're working on a different kind of project with Cards—say, remodeling your house—you will need a similarly narrow focus if you want to wrap it up in a reasonable amount of time.

- **Neuroticism.** People who are neurotic tend to get anxious often. They're impulsive and vulnerable and can become depressed—or even hostile—when things don't work out their way. Suffice it to say it's not particularly easy to build a brand-new business with a founder who has high levels of neuroticism. The entrepreneurial path is cacophonic. Massive highs are followed by massive lows. Uncertainty is the name of the game. Even

the most calm, cool and collected founder can get flummoxed—or way worse. Successful entrepreneurs tend to score low on the neuroticism scale. They're able to roll with the punches and adapt as the situation changes, as it does every day. Pulling off the Cards mindset requires the same kind of stability and levelheadedness. Once again, your projects will rarely if ever work out the precise way you hope they do. You need to be able to shift gears immediately as needs change. For example, let's say you've put together a solid team of four engineers based all around the world. All of a sudden, one of them you've relied on to do the heavy lifting announces they've landed a full-time gig somewhere and can no longer help on your project. Ah! But things were going so well. Bollocks! Don't let these setbacks thwart your progress. Adapt to the situation at hand, find a replacement and keep moving forward.

Kaplan and Warren (2006)



Possessing the optimal amount of the preceding traits should make it easier for you to adopt the Card mindset successfully.

But beyond that, founders usually share several other attributes that you'll need to adopt the Cards mindset. Here are six of them.

1. Vision

Turning your ideas into successful business endeavors starts with a vision.

In the world of entrepreneurialism, vision is what drives companies forward as they grow from one-person operations to organizations powered by 100 people.

Having a clearly defined vision and being able to articulate it to others should make it easier to persuade talented folks to join your cause. As to *who* should be invited to join your cause? As Dr. Dweck advises, "focusing on pedigree…is not as effective as looking for people who love challenges, who want to grow, and who want to collaborate."

Don't underestimate the importance of selling your vision to the people you work with. Here's what author and motivational speaker Simon Sinek has to say on the subject:

> *If you hire people just because they can do a job, they'll work for your money. But if you hire people who believe what you believe, they'll work for you with blood and sweat and tears.*

If you could pick an employee working for a check or an employee working for you with blood, sweat and tears, which would it be?

Bottom line: Get people on board who believe in your vision and they'll do great work—and you'll barely have to manage them.

2. Commitment

If building a business was easy, everyone would be an entrepreneur. But it's not, which is why 70% of businesses fail within 10 years.

Making the Code as Cards system work requires commitment. You need to be willing to work harder than everyone in the room and you need to be willing to work around the clock. If you're just popping by the office an hour a day or you're perpetually incommunicado, odds are you won't inspire the folks you're working with.

Success also requires ambition. Demonstrate your commitment by viewing failures as learning opportunities instead of drawbacks (which keeps in line with your newly adopted growth mindset).

3. Trust

Trust is an integral component of any successful company. Employees who work in high-trust environments are happier, more productive and more collaborative, and they're less likely to be on the lookout for the next gig.

But how exactly can you establish trust at your organization?

It turns out that we have a tendency to immediately trust the folks we work with—remotely, in particular—often because we have no choice. So, right off the bat, once you hire people, a modicum of trust will accompany their onboarding.

You can fortify that trust further by learning more about the people you work with. For example, you might find out that an engineer you hired in India doesn't enjoy constant email interruptions. If you're in the habit of emailing your team several times over the course of a day, you may want to see if you can hold back just a bit. Try to send a single email a day—or better yet use Knote for your communications and reduce the strain on your team's inboxes. Modifying your behavior to accommodate employee preferences should help strengthen trust.

Trust your employees and they'll trust you. Don't be afraid to give up some control and let other people own things. Build even more trust by sharing privileged information (within reason) and investing in your employees' development.

4. Fearlessness

Your plans won't always come to fruition. You'll hit speed bumps along the way. Unpredicted disasters and misfortunes will rear their ugly heads from time to time. Every brilliant idea or initiative you come up with won't look so great when they're finally enacted.

Successful entrepreneurs are fearless. They understand failure is part of the territory. They embrace adversity and treat every shortcoming as an opportunity to improve. They aren't afraid to speak their minds, even when what they say might be unpopular. And they never give up, no matter how challenging the situation may be.

Embrace fearlessness. You'll inspire your people and become a formidable foe to your competitors.

5. Responsiveness

As an entrepreneur, your time is limited. You can't be in your inbox all day. And you can't respond to every Slack message in real time.

You also can't ignore your team for extended periods of time. That's a surefire way to demotivate them, at least to some extent.

The trick is finding the balance between the two extremes.

Create a culture filled with self-motivated, resourceful folks who can generally track down the information they need on their own.

At the same time, encourage your team to reach out to you when they have legitimate questions. The faster you're able to respond to these inquiries, the more agile your organization will become.

Again, this is where Knote can really come in handy. Questions and answers can be posted publicly on an easily searchable platform, so you won't have to repeat yourself should two team members have the same question.

6. Gratitude

It's your idea. Your money's probably on the line, too. You've spent countless hours building a company, hiring people, and making it all work. At the end of the day, your neck is on the line.

But you can't do it all on your own. Any success you realize would not be possible with your people.

To this end, be thankful. Express gratitude to your workers on a regular basis. Congratulate them on a job well done.

Not only does expressing gratitude make us feel better, it also helps encourage your team while increasing trust.

You don't have to do anything crazy. Just saying *thank you* or writing a quick email should do the trick. While you're at it, create a dedicated knotepad for gratitude and get things rolling by complimenting employees there. Encourage your team to sing each other's praises, too, when it's warranted.

Learning from the sages

This whole cards approach is a synthesis of many teachings I've had presented to me. It didn't come from nowhere. As you roll around in these ideas, you may be curious to go deeper. Here are some of the greats.

Peter Drucker's The Effective Executive is a 50-year-old bible for the leader in the age of knowledge workers. The management guru with a business school named after him was not only a best-seller but tremendously foresighted on many of the core issues of modern work. Drucker wrote way back then about how long your meetings should be -- 90 minutes, not 60 minutes, because that's how long it takes to really get into a topic. But how often should those meetings be? Infrequent. Since you also need personal time -- big, four-hour blocks of it, to do detailed individual work like reading and thinking and planning and writing. You can't do all your thinking by brief dictation. If you can create an effective system around you, where others are free and responsible to do good things, then you also can contribute seriously on the topics where you are an expert. Much more in there. A great.

Andy Grove's High Output Management: his effort to teach his peers and followers what made the chip juggernaut so great in its heyday, and indeed its heyday seems still to be today. A lasting company, long after Grove moved on. He co-founded the company with the wrong original business but in the right area, then made the gutsy move to drop memory chips and making computing chips. Boom. A near monopoly in the 1990s, the company got that way under his leadership. Grove tells us how to run a company and the tools you have: delegation is high on the list (cards…), and so is output (visible work…) as well as the rest of his core methods including meetings, decisions, planning. His ideas have been shaped and re-shaped in the decades since, and the latest incarnation is the trendy OKR for managing and focusing people's performance (Operational Key Results -- tangible stuff that you can get done now, not abstract visions and missions).

Taiichi Ohno's Workplace Management, written by the creator of the earth-shaking Toyota Production System, a system that took Toyota from literally zero in car production after the Second World War to global number 1. Ohno encourages managers to question culture and tradition, and be persistent in asking for show-me results. He says anyone should be able to and indeed is obliged to stop the entire factory line if they see something wrong -- radical delegation. So much good stuff in Ohno, including: "Wasted Motion is Not Work" and "Go See What Failed with Your Own Eyes" and "The Misconception that Mass Production is Cheaper".

And of course Jeff Sutherland's Scrum outlining the co-founder of Agile Development's approach is valuable. As is Ronald Heifetz's Leadership without Easy Answers, the manual from the Harvard Kennedy School legend for leaders facing problems that can't be outsourced.

Now you know how you need to think and act to build a successful company under the Code as Cards system. Up next, we'll explore how you need to treat your people in this new way of working to keep them happy, engaged and productive.

Chapter Eight: People

In Chapter 2, we explored the concept of hiring through firing.

When looking for people to take care of specific tasks for you on an ongoing basis, the Code as Cards system encourages you to spend money on trial tasks to weed out the less than desirable candidates and identify the folks you want to work with.

It's one thing to find great people to get started on your awesome project. It's another thing to get those people on board and keep them happy and engaged.

Chapter 2 also demonstrated how work is changing thanks to the rise of the gig economy.

Code as Cards teaches you how to run a company with a patchwork system of freelancers (and maybe a few employees) based anywhere in the world. It's important to remember that just because the freelancers you're working with aren't bona fide employees doesn't mean you no longer have an obligation to treat these folks with professional courtesy and respect.

Treat your freelancers like faceless cogs in a machine and odds are you'll have to start looking for their replacements before you know it.

The gig economy promises several benefits for companies: lower expenses, on-demand help and wider talent pool, among other things. As for the freelancers themselves? More income potential, a wider array of projects to work on and more flexibility in general.

But, from the worker's perspective, the gig economy is not without its downsides. You're not an employee, so no one is contributing to your healthcare costs. You don't have an office to go to, so you can forget about catered lunches, afternoon happy hours and nap pods. You have to pay for all of your equipment and your internet. And you have to pay twice as many payroll taxes as a regular employee.

Still, the gig economy shows no signs of slowing down—quite the contrary. It's just the way work is moving.

How do you make it work?

It starts with not being a dick. Follow the silver rule: *Do not do unto others as you would not have them do unto you.*

Freelancers forego the comfort of a steady paycheck to have more control over their schedules and more variety in their work. But these people, like everyone else, have bills to pay every month.

Whenever possible, keep their work as steady as you can. If you anticipate the spigot of cash will shut off in four weeks, tell your freelancers that. Otherwise, they're out of a gig overnight and will not have had the opportunity to find new work to replace it. And, if you don't give them a heads-up about scaling down, you probably won't be able to rehire them in the future if you need their talents again.

Beyond these things, you can keep freelancers engaged by giving them opportunities to do meaningful work.

The power of meaningful work

In a recent TED Talk, Dan Ariely, a professor of psychology and behavioral economics at Duke University, shares an anecdote that highlights the power of meaningful work.

One of his former students had dropped by campus to visit. The student explained that, for the last two weeks, he'd been working relentlessly on a PowerPoint presentation in preparation for a merger the big bank he worked for was planning. Ultimately, the former student's boss told him that the merger had been called off. All of his work—every day of coming in early and staying late—was for naught. The student, who was happy the whole time he was plugging away on the presentation, was suddenly depressed.

This makes sense: What's the point in busting your tail if no one will ever see the fruits of your labor?

On the flipside, when work is meaningful to us, we don't mind going above and beyond.

Research suggests people who are able to do meaningful work are happier, healthier, more collaborative and more engaged. They're also more likely to view failures as learning opportunities. (There's that growth mindset again.)

So how do we make work meaningful? Three conditions need to be met:

1. **The work needs to make sense.** We need to be able to accomplish the tasks assigned to us.
2. **The work must have a point.** The tasks we're doing need to tie into something larger (e.g., how a steering wheel is a vital component of a car).
3. **The work must benefit the greater good.** We need the reassurance that what we're doing makes at least some people's lives better.

Sounds like your project, right?

So, freelancers don't get any perks. But it appears they're willing to trade off perks for the opportunity to do meaningful work.

Patty McCord served as Netflix's chief talent officer from 1998 to 2012. She's largely responsible for shaping Netflix's culture and was the brains behind the company's then-innovative no-formal-limit vacation policy, which has now become widespread in the world of startups. In a recent *Harvard Business Review* piece, McCord explores the tendency of today's startups to shower their employees with over-the-top perks. For McCord, we're simply in a transient era of extravagant perks that will disappear, sooner or later, as VC dollars dry up. McCord understands the allure of perks but believes that meaningful work beats incentives every time:

> *The startups that I'd bet on to succeed (or I'd choose to work at if I were just starting my career) are not the ones where people are lying in hammocks drinking the craft beer du jour. They're the ones where people are going to work because they get to collaborate with great colleagues on important products. Perks are nice, but meaningful work is better.*

How do you get people to find meaning in the work they're doing?

- **Question them.** Be curious to get to know the people you're working with. Ask questions to pick their brains and give them feedback based on what they're telling you. If you're just assigning cards to people you'll never meet and never offer them feedback when they turn in tasks, odds are they'll feel a bit disengaged from the whole process. Make work meaningful by being actively involved in what they're doing (but from a distance).

- **Trust them.** Nobody who's committed to working the freelance lifestyle and has been hired by a genius like yourself wants you blowing up their inbox every few minutes to check on the status of their work. This is where Knote can save the day. Have your people track the progress of the various tasks they're working on a knotepad. Check in when you need to but otherwise leave them alone to do the work you hired them to do.

- **Challenge them.** Sell freelancers on the opportunity to not only do challenging work but also learn new things. For example, you might hire someone to write code for you, which you'll be managing through GitHub and Trello. That person might have next-level GitHub familiarity while never even hearing of Trello. Boom. Your freelancer is getting paid to write code for a project that matters. They're also getting paid to learn a new platform like Trello, something to pad their resume with. You might hire a content writer to post articles in WordPress, which they're familiar with, and ask them to manage email marketing through Mailchimp, which they've never used. It's the opposite of college: getting paid to learn new things.

A recent study highlighted six additional factors freelancers believe makes their work meaningful:

1. **Advancement.** Should the opportunity arise, freelancers enjoy taking on additional responsibilities and climbing the ladder.
2. **Affiliation.** They might work from home every day, but freelancers like becoming part of the group.
3. **Autonomy.** Unsurprisingly, freelancers are keen on independence.
4. **Balance.** They also enjoy being able to work when they want to work (within reason).
5. **Service.** Freelancers are happy to see their efforts improve the world around them.
6. **Variety.** The ability to learn new things and develop new skills is motivating, too.

A final takeaway: You've mastered Code as Cards and built an incredible product. Your idea is now a company and you're looking to hire some full-time people.

Don't forget the folks who got you to where you are. If full-time positions open up in the future and it's a fit, consider whether it makes sense offering a job to one of the freelancers you've been working with. You should already have a good idea as to whether they have what it takes.

By now you've got a solid understanding of how the Code as Cards system can be used to build amazing things. But like anything else, the system has its limits. We shine a light on those limits next.

Loving Kindness

Teams full of smart people -- different teams but full of similar people -- can vary wildly in how well they perform. In the old days, folks thought this was a matter of the leader. Similar team, different leader, different outcome. Must be the leader. The military has come to learn this idea and the SEAL team gurus like Jocko Willink who wrote Extreme Ownership advocate this.

There is a newer, better idea that actually explains the mechanics. Team IQ. Team Intelligence Quotient. The first author to mention this idea in our research is Sandy Pentland and colleagues from MIT, where they published a series of work about "Collective Intelligence" or "c". If 20th Century psychology discovered a mystical statistical concept called "g" or general intelligence or IQ, the factor that seems to measure how some people are more effective at certain types of cognitive tasks than others, a concept we might call intelligence, then maybe teams has a mystical quality like this too.

They found something. The Pentland lab conducted a series of research observing the differences between high performing teams and lower performing teams, and they controlled for the members in those teams. They studied big groups of teams and compared like to like. Yet some did better. Which ones?

The ones with "c", or Team IQ as opposed to just individual-level IQ, had more different types of people in them, and had each person participate in their discussions more equally, and weren't hierarchical. They collaborated with equal energy, face to face, and had personal connections

with each other, not only “all business” relationships. Groups with women did better than groups without, perhaps because women tend to prefer some of the collaborative styles anyway.

The Pentland lab went into some situations and changed the group behaviors. Voila, the group performance changed. Same people, same leaders, different rules of engagement, different outcomes.

A future version of Ship While You Sleep will do a lot more detail on this topic, as we have a large experiment underway at Knotel in exploring the effectiveness of ideas from this vein.

Chapter 9: Limits

The Code as Cards system teaches us a new way of working, how to hire the right people and fire the wrong people quickly, how a distributed operation can succeed, how the new way of working can be applied to several traditional work functions, the mindset needed to pull it all off, and how to manage people effectively under the system.

Every system has its limitations.

The CEO who wants to practice radical transparency needs to keep their mouth shut about *some things*—personnel problems, potential mergers or acquisitions that have not yet come to fruition, certain financial metrics, what have you. Then there's the lean startup philosophy, which has gained more and more traction in recent years. Ted Ladd describes the method's upside and limitations in a recent *Harvard Business Review* piece:

> *First, the good news: In general, the lean startup method works. We measured success by looking at how teams performed in a pitch competition in front of a panel of industry experts at the end of the accelerator program (a proxy, albeit an imperfect one, for long-term financial performance). Teams that elucidated and then tested hypotheses about their venture performed almost three times better in the pitch competition than teams that did not test any hypotheses.*
>
> *Now, the bad news: There was no linear relationship between the number of validated hypotheses and a team's subsequent success. In short, more validation is not better. I also found that teams that conducted both open-ended conversations and more formalized experiments with customers actually performed worse in the competition than teams that conducted either one or the other during the early stages of venture design.*

Despite these limitations, we can assume that transparency and the lean ideology aren't going anywhere anytime soon.

The Code as Cards system is no different. While those who follow it can transform an idea into an app into a company, there comes a point when you probably can't use the system to manage every aspect of your operation.

There's a reason why companies like Google, Facebook, and Amazon have thousands of in-house engineers, after all. When you have millions of users who expect high-quality experiences every time, you probably can't rely on your patchwork team of hustlers to meet those requirements.

So what do you do?

Evolve the system. Create a balanced team of bona fide employees *and* freelancers, with your in-house people managing those working remotely.

Here are some ideas as to what that might look like.

Product

Without a product, you don't have a company.

When you become too busy to assume the responsibilities of the product manager—i.e., when you simply don't have enough time to manage the Code as Cards system yourself—it's time to hire your first product manager to assume that role.

This individual will have what it takes to take over the day-to-day management of the product and manage a patchwork team of freelancers efficiently and cost-effectively.

Marketing

In Chapter 6, we learned the concept of content through cards—the idea that you can manage much of your marketing efforts with a project management tool like Trello and a handful of reliable freelance content producers spread out across the world.

While this approach will work as you're just getting started, assuming your idea takes off, you may have to revise your approach.

But it's a simple fix: Hire one or two in-house marketing managers to oversee your team of freelancers, which can keep growing in parallel to your content needs.

HR

When you're just starting out and hiring full-time people, you will probably be able to handle the bulk of the HR responsibilities by yourself—or at least pass off the duties to someone you can trust. When you're still a tiny operation, you can even run HR as Cards, creating cards for each employee and shuffling them from column to column (e.g., Accepted Offer -> Paperwork Complete -> Fully Onboarded).

You're not an HR expert, though. So how can you pull it off?

Managing HR has gotten exceptionally easier in recent years thanks to the emergence of a wealth of HR automation platforms, like Gusto, Zenefits, and Namely, which take care of things like payroll, benefits, and onboarding. There are also tools like Lattice and BambooHR which can be used to increase employee engagement and satisfaction.

Despite these tools, as your company grows, there will come a point in time where you'll need to hire a dedicated HR manager—and, over time, as your business grows even more, a dedicated HR team.

When is the right time to do that? It will really depend on your company, your temperament (i.e., how much HR work you can handle), and your people (let's hope they're not a rambunctious bunch).

On one end of the spectrum, founders suggest that you need to start focusing on people operations after you've hired 10 or 15 people. On the other end, there are startups that have hundreds of employees that lack dedicated HR teams.

How much chaos can you tolerate?

Finance

Have to make payments to a few freelancers and suppliers? You can manage your accounts payable in a Bills as Cards system. Fill a Trello board with columns named "To Be Paid," "Due Soon," and "Paid," for example. Easy enough.

As you grow, finances get much more difficult for folks from the non-financial persuasion to handle.

Hired a ton of people? Thinking about pitching to VCs for funding and need to get your financial statements in order? Trying to chart your business' long-term financial prospects?

You probably need a dedicated finance person (or team) for these kinds of tasks.

Miscellanea

The Code as Cards system promises to eliminate meetings, distribute work efficiently, accelerate innovation, and reduce overall reliance on email, among other things.

That said, you probably won't be able to (or, rather, won't want to) eliminate all four of those things completely. Yes, you definitely want to reduce them as much as you possibly can. But running a company on cards completely isn't the right approach, either.

Here are some examples of where the Code as Cards system falls shorts in these four areas:

- **Meetings.** Daily scrum meetings. Weekly one-on-ones. Routine team meetings. These are the kinds of meetings that you can get rid of with a knotepad. But you'll still need to meet to discuss long-term strategy, major company developments, and other similar game-changing announcements.

- **Vital projects.** In many instances, the Code as Cards system can accelerate timelines, helping you wrap up projects earlier. There are only so many hours in a day, after all. Your most talented engineer can't do everything themselves when they're facing a tight deadline. That's where breaking everything down into smaller tasks and assigning them to your freelancers works

wonders. That said, there are probably certain situations when it makes more sense for your top developer to tackle a major project or feature on their own (or mostly on their own, at least).

- **Innovation.** In our technology-driven world, many work tasks that previously had to be in person can now be taken care of digitally. People work from home. You don't have to walk into your boss' office to ask a question—just ping them on Slack. The list goes on. Still, we do our best work when we are in each other's presence. (This is the major reason Knotel has grown so quickly, but that's another story.) The Code as Cards system lets you get away with a lot fewer in-person interactions. But if you want to innovate and come up with the best ideas, you'll still need to make time for brainstorming sessions and the like.

- **Emails.** Ah yes. Emails. Code as Cards will help you drastically reduce your reliance on email thanks to Trello, GitHub, and Knote. But it's not like you can get rid of your inbox altogether. Email still serves several purposes. You'll need to use email to talk with folks outside your organization (e.g., VCs, business partners, media, etc.). You'll need email to talk about confidential matters. You'll need it to schedule meetings. You'll need it a lot less. But you'll still need it.

The takeaway from all this? Every company is different. So it's impossible to know, with certainty, exactly *when* the right time to adjust the Code as Cards system will be and exactly what your unique approach to it will look like.

You know your outfit better than anyone else. Only you will know when you need to make modifications of your own.

OK. Nine chapters is enough about work.

Up next, we'll talk about how you can apply the Code as Cards system to other aspects of your life. Dig in!

Chapter 10: Life as Cards

We've spent nine chapters talking about a new way of working and how you can use a network of employees and freelancers to build great products and ship while you sleep.

While the Code as Cards system can undoubtedly help you become a more effective executive, it can also be applied to other areas outside of your professional life to achieve similar effects.

What follows are five use cases to give you an idea of how you can use cards to organize your life and accomplish more.

1. Communal living situations

In the United States, the cost of shelter is rapidly increasing. The same holds true in Europe. It's becoming harder and harder for the average person to figure out how to pay for a roof over their head in major cities around the world.

To counter insanely high rent prices, we're collectively reverting to how our ancestors lived tens of thousands of years ago: communally.

In recent years, several startups—like node., Common and Quarters—have emerged to offer affordable communal living spaces to young professionals looking for shelter.

Anyone who's been a part of a communal living situation before knows exactly how it goes. You might be the messiest person in the world with an incredibly filthy bedroom. But God forbid your roommate leaves a dish in the sink.

Making communal living situations work requires an even and fair distribution of chores and other household tasks. But it's one thing to pay lip service to a chore—*Yeah, I'll totally clean the bathroom by the end of the week*—it's another thing to have a system in place, visible to all roommates.

Let's say five people live together in one house. Each week, the kitchen needs to be cleaned, the bathrooms need to be cleaned, the house needs to be vacuumed, the garbages need to be taken out and the living room needs to be tidied up. There are other equally important tasks—like emptying the refrigerator and thoroughly scrubbing it down, doing yard work and dusting everything—that also need to be taken care of intermittently.

How do you divvy everything up so that a) every task gets done and b) nobody is unfairly taking advantage of their roommates by essentially doing nothing?

Easy: Create a card for each chore. Create columns: To Do This Week, To Do This Month and Done. Each week, say on a Sunday, the "administrator" of this board can assign tasks to each

roommate, giving them deadlines. The roommates can then move the cards into the appropriate columns as they tackle their responsibilities. Everyone will be able to see who's responsible for what and whether they're being slackers.

At the end of each week, reassign the recurring chores to make sure that everyone has to clean the bathroom in even intervals.

Think your laziest roommate will ignore the new system? You're wrong. Science says we act better when we believe we are being observed by other people.

Use cards for chores and say goodbye to bitching about a dirty house every few days.

2. The bucket list

Ah, the bucket list. There's so much we want to do, but there's so little time.

Do you really think that writing tasks or experiences down on a piece of paper will actually help you conquer all of them?

Probably not. Bucket lists are passive. They're full of things you plan to do *one* day—not *today*. You might get excited while you write your bucket list down. But as time goes on and your list starts collecting dust, you'll find yourself wondering whether you'll be able to check all of the items off one day—or even half of them.

The easy hack: Make your bucket list active with cards. Not only will such an approach make you more engaged with meeting your goals, it'll also make it *easier* to achieve them.

Let's say one of the items on your bucket list is summiting Mount Everest. You're in your 40s, and you're not exactly in the best shape of your life.

If you really want to climb Everest, you'll have to do more than write down "1. Summit Mount Everest" on your bucket list. You'll have to set aside something like $35,000. You'll have to get into tip-top shape, and then you'll have to train. You'll have to procure all the right gear. You'll have to plan your trip, take time off of work, figure out who you're going to go with. On and on and on.

In other words, every item that will appear on your bucket list can likely be broken into a series of prerequisite tasks. Failure to tackle these smaller tasks, in many cases, means you'll never be able to check the main item off your bucket list for good.

What would a bucket list as cards look like? You might be best off creating a dedicated board for major items like Everest. Create columns: To Do, In Progress, Done. Move cards as you take care of the prerequisite tasks and enjoy your active approach to the bucket list concept.

Less ambitious items on your bucket list—reconnecting with an old friend, going to see your favorite baseball team for the first time or trying a Michelin-rated restaurant—can probably share the same board.

3. Learning how to cook

Cooking for yourself makes you happier and healthier. Cooking for other people has similar positive benefits. Not to mention all the money you'll save by becoming your favorite chef.

Whether you're a complete rookie, you can cook up a handful of tasty dishes or you're a borderline professional chef, you can always add a few recipes to your arsenal.

How many times have you told yourself *I want to learn how to cook a new dish*? How many times have you actually followed through on those sentiments?

Neuroscience teaches us that we're more likely to accomplish our goals when we write them down, according to Mark Murphy, the author of *Hiring for Attitude*:

> *Vividly describing your goals in written form is strongly associated with goal success, and people who very vividly describe or picture their goals are anywhere from 1.2 to 1.4 times more likely to successfully accomplish their goals than people who don't.*

Use cards to learn how to cook to begin with—or to learn how to cook several new dishes.

Create a card with each dish you want to learn and fill them with recipes, images of the meal and other relevant information (e.g., whether you need any special cooking accoutrement to whip something up). Columns can include Recipes Ideas, Take 1, Take 2, Take 3, Blech and Yum. Move cards over from Recipe Ideas to Take 1 after your first attempt, Take 2 after your second attempt and so on (you probably won't be an expert on a certain meal the first time you cook it). After several attempts, you can decide whether you should avoid a dish in the future (Blech) or you've mastered it (Yum).

4. Planning an event

You can also use cards to plan a vacation, a house party or any other kind of event—whether you're coordinating the entire shindig yourself or you're collaborating with several other people.

Let's say you and your significant other are going on a two-week drive across the United States. You're renting a car outside of New York City and you plan to drive to California before flying back from SFO to LGA. You both have places you want to see. But America's an enormous country, road trips encourage spontaneity and you don't have enough time to see *everything*.

Use cards to plan your trip. Organize your board into 14 columns, one for each day you'll be on the road. Start populating each column with cards that represent time you'll spend driving, places you want to see, restaurants or breweries you want to try, parks you want to visit, etc. Assign yourself and your significant other to each card you come up with to make sure that there's an even "split" as to who chooses which activities (if that's your thing).

You can also use cards to plan parties. Set up columns like Invited Guests, RSVP'd Guests, Declined Guests, Food, Drinks, Entertainment and Party Supplies. Move folks from Invited Guests to the RSVP'd Guests or Declined Guests as they let you know whether they can make it or not. Once you've got a handle on who's coming, assign RSVP'd guests to certain Food, Drinks, Entertainment and Party Supplies tasks to make sure everyone brings the necessary ingredients for an awesome party.

You get the gist. You can use cards to plan all sorts of events, making sure nothing slips through the cracks.

5. Personal goals

Recall how we're much likelier to achieve our objectives when we write them out and visualize them.

You can use cards to accomplish your personal goals, too. It's an easy way to keep track of progress (or lack thereof) and see which areas you need the most help in.

For example, you can use cards to make progress on your:

- **Financial goals.** Eyeing certain stocks? Have your sights set on diversifying your crypto portfolio? Want to make some real estate investments? Is it time to buy some bonds? Do you have enough money in your savings account in case of an emergency? Drowning in credit card debt? Paid off your student loans yet? Have enough saved up to cover your recurring monthly expenses? Create a board to keep track of your investments and your overall money situation. The better understanding you have of your finances, the easier it'll be to reach your goals.

- **Career goals.** You created your personal financial board. That's great! Then you figured out you don't make enough money to meet your financial goals anytime soon. That's not so great! In the past, employees would work at one company for their entire career. In exchange for their loyalty, they'd get solid pay and benefits—and the increasingly disappearing private-sector pension. Things are different these days. Some studies suggest people who stay at the same company more than two years get paid 50% less than folks who hop from job to job. Of course, this doesn't mean that *every* company pays its loyal workers paltry wages. But it's fodder to think about. Create a board to chart your career goals. Do you want to climb the ladder? Is it more important to get paid

more? Do you want to work for an early-stage startup and get equity on the off chance the company hits it big? Again: Visualize your career goals using cards to increase the chances you wind up where you want to be.

- **Health goals.** You can't enjoy your financial situation or your career if you're not healthy. Whether you aim to lose weight, exercise more, eat better or kick bad habits, visualization can also help you achieve your health goals. Use cards to track your progress and get the results you're looking for.

By now you've got a good idea of how versatile cards can be. They practically have unlimited use cases.

The onus is now on you. Get creative and figure out how you can apply the Code as Cards system to your unique professional and personal lives. That's the ticket to reaching your goals and unlocking your full potential.