

DEVELOPER HEGEMONY

THE FUTURE OF LABOR



ERIK DIETRICH

Developer Hegemony

The Future of Labor

Erik Dietrich

This book is for sale at <http://leanpub.com/developerhegemony>.

This version was published on 2017-04-12



* * * * *

This is a [Leanpub](#) book. Leanpub empowers authors and publishers with the Lean Publishing process. [Lean Publishing](#) is the act of publishing an in-progress ebook using lightweight tools and many iterations to get reader feedback, pivot until you have the right book and build traction once you do.

* * * * *

© 2015 - 2017 Erik Dietrich

Table of Contents

[Introductory Notes](#)
[More info after reading](#)
[Other books by me](#)

Part 1: A Fictional Not-So-Distant Future

[Chapter 1: Wednesday Morning](#)
[Chapter 2: Wednesday Afternoon](#)
[Chapter 3: Wednesday Night](#)
[Chapter 4: Thursday Morning](#)
[Chapter 5: Thursday Night](#)

Part 2: The Nature of the Game

[Chapter 6: You Are Here](#)
[Chapter 7: The Corporate Cave](#)
[Chapter 8: Growing Up](#)
[Chapter 9: Cynical Theories of Management](#)
[Chapter 10: Defining the Hierarchy \(With Less Cynicism\)](#)
[Chapter 11: Interviews, Induction, and Nonsense](#)
[Chapter 12: The Bad Economics of Pragmatism](#)
[Chapter 13: The Worse Economics of Idealism](#)
[Chapter 14: The Lonely Profit of Opportunism](#)
[Chapter 15: Faux Ownership—Managers and Owners](#)
[Chapter 16: The Cult of Hours](#)
[Chapter 17: Performance Reviews and Advancement Theater](#)
[Chapter 18: Your Company Doesn't Care About You](#)

Part 3: The History of the Game

[Chapter 19: Less Than the Sum of Its Parts](#)
[Chapter 20: Legacy—Ancient Corporations](#)

[Chapter 21: Influence—Medieval Corporations](#)
[Chapter 22: Gestalt—Mercantilism](#)
[Chapter 23: Barriers to Entry—Industrial Revolution](#)
[Chapter 24: Layered Organizations—Taylorism](#)
[Chapter 25: Ubiquity—Organized Labor](#)
[Chapter 26: Anachronism—Rise of Knowledge Work](#)

[Part 4: How to Win the Game](#)

[Chapter 27: Succeeding in the Corporate World, Such as It Is](#)
[Chapter 28: Pragmatists Succeed with Alternate Narratives](#)
[Chapter 29: Idealists Succeed with Merged Identity](#)
[Chapter 30: Opportunists Become Other](#)
[Chapter 31: The Realpolitik Tragedy of Corporate Scrum](#)
[Chapter 32: The Programmer's Escape Plan](#)
[Chapter 33: Avoiding the Delivery Trap](#)
[Chapter 34: Partnership and Transcending the Realpolitik Glass Ceiling](#)
[Chapter 35: Avoiding Carnival Cash](#)
[Chapter 36: The Art of No](#)
[Chapter 37: Advancement](#)
[Chapter 38: The Madness of it All](#)

[Part 5: How to Stop Playing Games](#)

[Chapter 39: Where We Go from Here](#)
[Chapter 40: The Coming Crunch](#)
[Chapter 41: Studies in Success—Those Who've Already Won](#)
[Chapter 42: Building a Composite—Developer Übermensch](#)
[Chapter 43: The Developer Übermensch is a Businessperson](#)
[Chapter 44: Developers Becoming Efficiencyers to Take Control](#)

[Chapter 45: Turning App Dev Firms into Efficiency Firms](#)

[Chapter 46: Anatomy of the Efficiency Firm](#)

[Chapter 47: Becoming an Efficiency and Joining an Efficiency Firm](#)

[Chapter 48: But How Do We Fix Non-Efficiency Corporations?](#)

[Chapter 49: Concrete, Immediate Steps You Can Take to Become an Efficiency](#)

[Chapter 50: Full Circle—The Fate of Pragmatists, Idealists, and Opportunists](#)

[Chapter 51: What This Looks Like, Long Term What Now?](#)

[Appendix A](#)

Introductory Notes

Who should read this? Is it only for software developers?

Let me start out by answering this question unequivocally. Not, it is not only for software developers. Anyone can (and hopefully will) find this interesting.

As far as readers who will have skin in the game, though, there will be a hierarchy. Assuming you're not just in it for my clever turns of phrase and charming wit, this will feel most relevant to you if you earn your living as a knowledge worker.

A knowledge worker earns a living mainly via non-routine problem solving. Doctors, lawyers, engineers, entrepreneurs, management, scientists, professors—and, yes, software developers—all count. If you earn a living this way, this book will resonate more with you.

You'll find even more of interest in this book if you work in a standard corporate workplace, and more still if you work in or around software development. So many work with software in some capacity these days, so I imagine this applies broadly. This book covers things like software development project management techniques, so the subject matter will be more familiar to those with direct experience.

And finally, this book should hit home the most for those who work or have worked as software developers. We are the stars of the show here, and the book is really about our path forward and our fate. While these topics will probably interest just about anyone, they directly and personally affect us in the development world.

In this book, I talk in pretty frank and cynical terms about the corporate world and office politics. My central thesis is that the pyramid-shaped corporation is fundamentally flawed as a way to

manage knowledge work. If you like comics like Dilbert and movies like *Office Space*, I promise you will relate.

But if you're a software developer, there will be both empathy and calls to action in this book. This is *our* story and thus *our* book.

A note about names and demographics of developers

For the names in this book, I used the site [Fake Name Generator](#) with the name set American. This is in part to outsource the activity of coming up with names, but it's also to absolve myself from the perception that I'm baking in any statements about programmer demographics.

The only exception to true randomness was that I specifically chose a female name for the protagonist of Part 1 of the book. I did this not to pander but because I think that the suggested future I propose will naturally lead to more equal-opportunity entry into the field of software development. Things like earnings and value delivered will not be obfuscated and vague the way they are now, hidden by a corporation whose self-interest competes with that of its employees.

More info after reading

What if you want more information about my proposed future of developer hegemony after reading the book?

Well, I've created a landing page on my site called, ["What Now?"](#). I intend for it to be a living follow-up to the book—one that evolves with time. It's here that I've started to create content aimed at helping people realize developer hegemony. The page will serve as a "start here" guide.

Other books by me

If you wind up enjoying this book, you can check out some other ones that I've written.

[*The Expert Beginner*](#)

[*Starting to Unit Test: Not as Hard as You Think*](#)

[*How To Keep Your Best Programmers*](#)

PART 1: A FICTIONAL NOT-SO- DISTANT FUTURE



Chapter 1: Wednesday Morning

There was noise...voices. Loud and impossibly cheerful voices. And buzzing. Rhythmic and insistent. It made no sense.

For some reason she couldn't understand, Emma was afraid to open her eyes. Comprehension of her situation was oddly slow and elusive. It was light out. She was at home. No hangover. But grogginess. So much grogginess.

The alarm was going off. The cheerful voices she was hearing were a couple of idiots on whatever AM radio channel had been on last. The buzzing might be her phone. There were too many unknowns for the snooze button. With a herculean effort, she opened her eyes and looked around. That seemed to open the comprehension floodgates.

Mid-morning light made her bedroom a lot brighter than it normally was when she woke up. The alarm clock blaring its AM talk show read 10:00, which normally would have meant a long, luxurious night of sleep. But the night before had not been a normal night. She'd stayed up until about 5:00 AM, determined not to go to bed until she had successfully completed the surgery she had started on her team's build earlier that night. She'd gone to bed happy, exhausted, and successful, and set her alarm for as late as she dared.

Turning the alarm off, Emma sat up and picked up her phone, which had been the source of the previously mysterious buzzing. Two missed calls and a handful of texts. That made sense, since she'd normally have started working an hour and a half ago or so. She glanced at the texts.

Craig: You not coming to the powwow this morning?

Emma chuckled, thinking of the origin of the term. She, Craig, and the others were mildly disillusioned children of the Scrum revolution and had “grown up,” so to speak, having stand-ups and retros. They had all done tours in companies where Scrum was implemented by technical executives, resulting in the weird dance of managers trying to manage teams into being self-managing. Or something.

Since she'd switched to the freelance team model, she'd found that a lot of what they'd done before as a matter of course wasn't necessary any longer. Being on a team that was literally self-managed, she'd become aware of how much of her former software development processes had been designed to create the illusion of autonomy. If she and her teammates failed to have a daily stand-up to self-manage, they were chided by management. Scrum and other methodologies like it had been devised by teams of extremely effective software developers. The point that countless managers and execs had missed in their haste to replicate exactly the success of these pioneers of agile was that if you hired smart, creative people, they would find the most effective ways for themselves to work.

Emma's team, for instance, had found that they were getting diminishing returns out of daily stand-ups. What they'd taken to doing instead is scheduling a powwow whenever the team felt they ought to catch up a bit and do a bit of knowledge sharing. It usually worked out to be a quick chat every two or three days. And she'd missed the one they'd scheduled for 9:30 that morning. Oh well.

Craig: Oh, nvm. Saw a commit from you at 4:30 this morning. Wow, nice work on the build! We all owe you some beers.

Emma felt a surge of validation around her night's labors. Damn right they owed her some beers. The build had been rotting for some time, taking at first five, then ten, then twenty minutes at a time. By the time Emma had become fed up late yesterday evening, it was routinely taking more than thirty minutes to check code in and have the build and tests run. They all knew it was ridiculous, amateur-hour

trade-craft, and yet it never seemed to be the right moment to sink six hours into it, cleaning things up.

Since her team billed by features, sinking extra hours into thumb-twiddling while builds happened meant working more hours for the same pay. Sure, they tended to get other things done while waiting, but that was only so beneficial, and it still tied them to the keyboard for that many more minutes per day. Her work meant a very real quality of life boost for all of them. And when team members had quality of life boosted, they tended to buy beer for the booster.

Emma: Will take you up on those beers when I get back from my trip.

Shaking off the last remaining grogginess and swinging her feet over the side of the bed, Emma checked her email before starting to pack an overnight bag. Her cab would be there in an hour to take her to the airport, leaving her time only to deal with urgent emails before the last-minute prep. Most of the emails in her inbox were build notifications and low priority informational messages, but one stood out as something she needed to address.

From: JamesIAbbott@rhyta.com [James Abbott]

Subject: For This Afternoon's Meeting

Hey Emma,

Looking forward to seeing you this afternoon. I have a favor to ask, and I apologize in advance for this, but Linda Manders really wants full blown project documentation. We're really happy with the work that you're doing, and I know this isn't how you operate, but I was hoping maybe you could at least bring a Gantt chart so she'll stop bugging me for a couple of weeks. Again, I'm sorry about all this.

Best,

Jim

Emma fought the temptation to bang her phone repeatedly against her nightstand. It seemed as though, no matter how far in her rearview mirror she put this old bluster-laden, command-and-control approach to writing software, she could never truly be free of it. She didn't have *time* for this. Furiously, she thumbed through her contacts, and dialed Michael Hurst, the team's mostly dedicated administrative assistant, hired through an agency. He was the one that handled all of the responsibilities that traditional orgs handed to the project manager.

"Hey, Emma, what can I do for you?" Michael's voice was a mixture of calm and upbeat that should be illegal when talking to someone on five hours of sleep.

Emma schooled her attitude to patience, since Michael had no share of the blame for the thorn in her side named Linda Manders. "Hey Michael, can you make a Gantt chart for our work on Rhyta over the past quarter? Nothing too crazy—just show dependent features and time. Tell anyone on the team that gives you resistance I need it for my 4:00 today at their office."

"Man, Emma, I'd love to help you out, but there's some kind of problem with the Jira account and so I'm kind of in crisis mode here, doing triage and handling all of the tracking manually. That's taking up all of the hours budget that I have, so anything additional would be extra time, and...you know, I'm not exactly a kid anymore. I'm trying to keep it to forty or less these days, you know?"

"I know, and I totally understand. Any chance you could suffer through a ten- or twelve-hour day today for us and just take a day off once Jira's back online? Seriously, we'll do double time today of hours over eight, so when you do take the comp time off, you can take a full day basically for free. I'm sorry even to ask like this, but I really need it."

Michael sniffed a little and let out a breath.

“Sure, Emma. I’ll do that today. I know it’s been a tough week for everyone.”

Emma thanked him and hit “end.”

Jackpot, Emma thought. She knew Michael would come through, so she turned her attention to getting ready for her trip.

Chapter 2: Wednesday Afternoon

Emma sat, fighting her tiredness and watching sunlight stream through the enormous, spotless windows in the lobby of Rhyta's building. She'd been in this building a few times before and knew that the impressive lobby was not representative of the rest of the building. It devolved into a stuffy cube farm in pretty short order, once through the door behind the receptionist's desk. She mused on the appropriateness of this bait-and-switch as a microcosm for the corporate world in general but then supposed she was being overly cynical. "Focus on staying positive," she thought.

Jim Abbot shuffled through the doorway and raised his hand in a friendly wave. Stout, with wisps of white hair around a shiny pate, Jim was the kind of man she imagined was a kindly uncle or perhaps grandfather in his spare time. It's not that she couldn't picture him having children so much as that she couldn't picture him *disciplining* them. Jim was, to put it succinctly, a sweetheart.

"Come on back, Emma! We're in the Newton conference room, so there's chips and soda if you'd like." Rhyta named their conference rooms after scientists. Emma supposed it was to create an atmosphere of innovation by trite osmosis. "Stay positive," she chided herself in her mind.

"Thanks, Jim, but I'm good," she replied, waving a bottle of water she'd picked up at the airport. The flight had been predictably late, but she'd also baked some slack time into her plans. The result was that she'd had no time to check into her hotel, but there was plenty of time to pick up the rental car, drive here, and go through emails in the parking lot for twenty minutes before making her way to the lobby.

She and Jim exchanged small talk on the way to pay homage to Isaac Newton with Gantt charts and slide decks. No doubt the father

of calculus would be impressed by the many arrows in the project management quiver these days, Emma thought. Arriving in the conference room, she set her laptop bag down on the table, removed her computer, and took a seat.

“You remember Linda, of course,” Jim said, nodding toward an alert-looking woman with sharp features. Emma nodded in Linda’s direction and received a symmetric nod in return. There was no outright hostility between the two of them, but neither particularly cared for what the other represented.

Before Jim could resume speaking, a man with a distracted, sour expression on his face entered the room, walking quickly. He took a seat. “Good to see you again, Emma,” he said, a forced grin adding to Emma’s impression that it was not, in fact, good to see her again. But then, from the few other occasions she’d met Rhyta’s CTO, this just seemed to be Shane’s demeanor. She hadn’t realized that he’d be attending this meeting. “Nice to see you, as well, Shane.”

This meeting was essentially a formality. Rhyta had, for some years, been in the habit of meeting three times per year to plan out the next four months worth of software and infrastructure work. It was a typical shop that awkwardly straddled the waterfall and agile worlds. From their interest in agile ceremony, it was apparent they understood that massive batch planning of software work wasn’t possible. And yet they engaged in a Sisyphean struggle to try to do it anyway. In that struggle, Jim, the manager of internal software development, was Sisyphus, Emma mused to herself. Linda was the boulder. Shane was just an impatient man that wanted some rocks moved.

Emma had not witnessed this planning firsthand, but she did know that Jim, Linda, and perhaps others polled Rhyta’s internal developers and any vendors that they were using for status prior to planning upcoming work. It was for this polling that she was here right now, as it had been four months ago, and eight months ago before that.

“Why don’t we get started?” Jim said, pulling an HDMI plug from the electronics at the center of the meeting table. “I understand that Emma has an overview of the functionality delivered over the last few months.”

Taking the cue, Emma plugged in her laptop and pulled up the Gantt chart that Michael had forwarded, suppressing a sigh. She mechanically recounted the various features delivered and their dependencies upon one another, reading barely concealed boredom from her audience as she did so.

Toward the end of her ten-minute spiel, Linda began to look smug. At the first possible moment where it wouldn’t be an interruption, Linda proclaimed that this didn’t include features under development, nor did it include features intended for the next four months. None of this was news to any of them. But at that point, a nasty curveball metaphorically made its way from Linda’s hand out over the plate.

“You know, you’re the only vendor that refuses to forecast out your estimates, and we are open to considering others at this point.”

“Linda—” Shane started, irritably.

“I don’t mean it as a threat, but you do make our planning much harder than it needs to be. If we weren’t winding this effort down over the next few months, we’d be seriously weighing our options.” Jim looked sheepish and Shane snorted quietly, but Linda seemed oblivious to her coworkers. “Why is it so hard just to estimate out what you’re going to have and when? I mean, I know you have this principled agile thing or whatever, but seriously, you can’t just take a guess at it? It’s not like we’re going to sue you if it’s wrong. We know it’s an estimate.”

Emma knew that it was a somewhat rude move, but she ignored Linda, turned to Shane, and asked the leading question.

“Have you been satisfied with our work in terms of cost, timeliness, and quality?”

Shane had tried to sneak a glance at his watch but quickly looked away when he realized the conversation had been steered toward him. “I absolutely have. It’d be nice if you could help Linda out with her planning, but I’m not going to press the point as the project winds down. I’ll leave that to you, Linda, and Jim to sort out, since it’s the two of them that provide estimates on your behalf.”

Historically, vendors had offered Rhyta exactly the kinds of estimates Linda was pouty about not having. Since Emma and her team didn’t operate that way, Linda and Jim had done it for them. Contrary to what Linda accused her of, the reason for not providing estimates wasn’t some kind of principled stand but rather an avoidance of wasting time. Things happened, priorities changed, and long-tail estimates were always wrong.

Linda sighed angrily. “That just means that it will be up to me to do them.” Then she tilted her head back, looking at the ceiling, and softened her tone a bit. “You guys do good work, but it’s just a hassle to cover all of this stuff for you. I don’t see why you don’t have a project manager we can work with. It would make things easier for everyone.”

Emma couldn’t help herself. “Most of the typical project management role can be handled by an admin. So we just *have* our admin do it. Our team does anything that he can’t handle.” Jim was looking at his hands, and she thought she heard Shane give a faint chuckle. Linda was staring daggers at her, but said nothing.

Shane stood up, stretching, and said, “I’m satisfied with where things are. We’ll have you finish out the remaining features over the next month or so, Emma. I’ll leave you to go over them in more detail with Jim and Linda so that you can take them back to your folks and get started next week. I’ll send someone in with the revised MSA and first feature contract for you to sign.”

Emma smiled inwardly. She wasn’t looking forward to Linda trying neurotically to plan two months’ worth of functionality down to the minute, but any possibility of danger was now past. It was her team’s

preference to have a general master services agreement for their work with a client and then to sign a contract for the features to be delivered in a given sprint. If the relationship prospered, they'd switch over to somewhat of a more binding model, but there was something very honest and reliable about a software development outfit agreeing to deliver a bunch of things over a two-week period. It allowed neither for slack time nor for things to drift too far off course. Her team was almost always right about what they could reasonably deliver, and it was easy for them to agree with clients on the value of that delivery, since they priced features as opposed to hours.

Rhyta had more or less agreed to that, but they insisted on having MSAs that lasted the same duration as their planning period. Every four months for the last year, Emma had showed up, participated as little as possible in the planning dance, and signed a new MSA with Rhyta. This project was coming to a successful close soon, so it was the last time she'd have to play this role here. It had gone well enough for her team to continue to have work for another month or two. The rest was just details.

Chapter 3: Wednesday Night

Emma smiled at the incongruity of pulling up to the Hilton and having her rental Hyundai Accent valeted. She and her partners had agreed on an expense policy for travel at the outset of working together. It meant staying at relatively cheap hotels, renting low end cars, and doing meals at places like Chipotle or Chili's. The reasoning had been that, since they all split profits and made good money, it'd be fairer to set the bar low. That way, the onus would be on those with more expensive tastes to pay out of pocket rather than putting the onus on those with cheaper taste to subsidize their partners' high end choices. So here Emma was at a Hilton with a tiny car because she liked to stay in nice places and didn't care too much about the car that got her there.

Not for the first time, she thought about telling Will Tierney, the team's young and clever accountant, to bill them for a couple of hours this month and put together a slightly more generous expense policy. Each of them on the team could requisition up to \$500 per month in services from a given vendor at his or her discretion, so Emma would be perfectly within her purview to use Will's time this way. But, as always, she decided against it. Spending a little extra money on a hotel room wasn't the end of the world.

Lack of sleep and airport travel tend to turn any day into a long one, so it was with heavy eyelids that Emma handed over her Visa for incidentals and checked into the hotel. She asked the desk attendant for a pizza delivery menu, not intending to leave her room until the morning. Not only did she have to prep a bit for her 9:00 AM meeting tomorrow, but she was exhausted and needed to make sure to get a good night's rest.

While she waited on her pizza to arrive, Emma lounged in the hotel bed and popped her laptop open. Unable to resist, she remoted in

and accessed the build machine's log for builds that day. The team's builds had been averaging less than five minutes for the day, she noted with satisfaction. They would probably wrap up the current sprint a day early, so they *definitely* owed her beers.

An hour later, a pizza had arrived and promptly disappeared as Emma, who hadn't eaten all day, realized that she'd been famished. She felt like she'd just celebrated her own mini-Thanksgiving and was struggling not to doze. There was a very un-Thanksgiving-like amount of work to be done, unfortunately. She started by rereading the research she'd put together on Payless Cashways, their prospective client with whom she was meeting in the morning before heading back to the airport.

Payless had a business matchmaking buyers and sellers of products, using some kind of logistics and ship-on-demand scheme. Emma was fuzzy on the details but didn't view that as a problem. If they landed the contract, they'd sort that out when building out the product backlog and tentative feature roadmap. She believed in filling her head with details only at the moment those details became necessary. And right now, the only necessary details to understand were the high level goals of Payless and, more specifically, of Elaine Graham, an IT director in the e-commerce department.

Emma grimaced a little, thinking about the description of what was needed. They wanted to build a "portal" for their suppliers to be able to manage their merchandise more in real time, even though there could conceivably be network outage. Of course they did. Non-technical corporate types loved calling everything a "portal" so much that the term seemed to have no meaning. Truth be told, Emma amused herself by picturing some kind of sci-fi stargate every time people used this term. Nevertheless, what they were clamoring for seemed like a pretty clear-cut request for a single page app. And several of the team members wanted to take this project so they'd have a chance to work with eventually consistent client-side technologies, which had shown no signs of cooling off.

Emma's preference, on the other hand, was to partner with a local company for their next gig. Why make things more complicated than they needed to be? They were only taking this meeting in the first place because she could piggyback the meet-and-greet onto the Rhyta meeting. But she had to admit, the Payless work did seem fun.

The last impediment to Emma's rest was prepping a letter of intent, in case she knocked the socks off everyone at Payless tomorrow morning. She'd had Frank, their lawyer, draft a copy with Payless and Elaine filled in where appropriate. She headed down to the hotel's little business center, printed out a few copies of the letter, stuffed them into a manila folder in her laptop bag, and ordered a 7:30 wake-up call on her way past the front desk. She trusted that when she received that call, she'd be much better rested than she'd been today.

Chapter 4: Thursday Morning

“You must be Emma!”

Blinking, Emma looked up from her study of a copy of *TIME Magazine* and took in a tall, heavysset, matronly figure offering her a warm smile. After a wonderful night of sleep, she'd beaten the wake up call, checked out early, had time for a nice little breakfast, and still been early to this appointment at Payless. And, most importantly, her tolerance for dealing with manically cheerful people was dramatically improved from yesterday.

“I'm Elaine Graham,” said the woman who was apparently Elaine Graham. “It's wonderful to see you in person after exchanging so many emails. I always like meeting people in person better, you know?”

Before Emma could politely confirm that she knew, Elaine prattled on, asking how she took her coffee, how her flight had been, where she was staying, and so on, only occasionally waiting for answers to her questions. She did all of this as she bustled her way down a series of hallways, motioning absently for Emma to follow. Emma found herself smiling, liking this woman in spite of the fact that she herself wasn't much for idle chit-chat. Perhaps it was simply a matter of appreciating a person who could be relied upon to fill awkward silences, almost as if specifically hired to do so.

As they walked, Emma took in the spartan hallways, drab cubicles, and lack of any apparent windows. Unlike Rhyta's, Payless's building had no facade that falsely promised fashionable architectural splendor within. It was pretty much just a concrete bunker, whether you looked at it from the outside or the inside. Emma wondered who Payless thought might start shelling them during business hours. Perhaps they resold cannons to customers that were quick to anger.

A few minutes later, she found herself seated in a conference room with Elaine and a handful of men in business-casual dress. They were introduced to her, but names and roles didn't really register except for Bruce McConnell, whose name she had seen CCed on a few emails and who she knew to be Elaine's boss. The rest of them remained an indistinguishable mix of architects, project managers, and miscellaneous interested parties.

After pleasantries had been exchanged and coffee furnished, Elaine plugged in a laptop that must have been waiting in the conference room for her. She then launched briskly into a well-polished presentation of what Payless was looking for and why. Emma almost raised her eyebrows in her surprised reappraisal of Elaine. Chatterbox or no, this was a sharp woman.

Once she'd finished, it was Emma's turn to talk about her team and their approach. "Before I show you the slide deck I have, I'd like to explain a bit up front so that we're all on the same page here. Elaine and Bruce, this is a bit of a recap of some of what we've discussed over email, but I think I should touch on it for the sake of everyone else here. We were referred here by a vendor of yours with whom we did some work a couple of years ago, so if you've talked to them, you probably know that we do things differently than a lot of so-called software 'consulting' companies."

"You mean because you do agile?" asked one of the project-architect-program-manager guys.

"No, not because of that. Everyone these days 'does agile,' or at least claims to. I'm talking about how we interact with our clients. What you're probably used to is issuing an RFP to a handful of vendors, who scurry off to break the work into releases displayed on Gantt charts with the explanatory caveat that this is all pending a 'discovery phase.' Each phase has a ballpark cost associated with it, and that rolls up into what they call an estimate: they'll work for six months and charge you \$600,000 to complete the project. But they also tell you that actually billing will be 'time and materials,' which,

they explain, means that you'll actually be charged based on the number of hours that it turns out to take."

Elaine was smiling quizzically, Bruce looked interested, and the management hydra was mostly nodding without realizing it. So far, so good. Time to see how much honesty they had an appetite for.

"So, they give you this estimate but make sure you can't really hold them to it. Then they try slyly to get you to tell them what their competitors are estimating."

Bingo. She saw a couple of sets of eyes go wide in her audience, meaning they thought she was some kind of soothsayer about software consultancies. These firms had clearly done just that.

"The reason they do that is because they're just making things up out of thin air. They have literally no idea how much the project will cost nor how long it will take, even rounded to the nearest quarter million—or quarter year. What they're trying to do is find the magic number where they're cheaper than their competitors, but not too cheap as not to seem credible. It's not an estimate. It's a guess at the figure you want to hear. But they can't just go to the bottom, because they might not seem credible, and they know that you'll line up all of the so-called estimates and pick the second cheapest one. So their real goal is to try to be second cheapest, sort of like a weird version of 'The Price is Right.'"

Looking around the room, she saw even wider eyes and a few people looking self-consciously at their hands. Clearly a few of them had just recently been lobbying for the second cheapest bid. But no one was interrupting, which was a good sign.

"We don't do that. I'll freely admit that I have no idea what your project is going to cost when it's done. No one does. And one of the first things that we do differently from those other firms is that we don't make up a figure for the sake of earning your business."

She pressed on before anyone could interject.

“I realize it’s a tough sell. You have budgets and people looking for forecasts, and ambiguity doesn’t sit well in that world. The next way that we do things differently is that we invite our clients to consider software development as a service instead of considering software as a commodity. You aren’t buying a web application the way that you’d buy a building or a tractor or something. You’re partnering with us to expand your organization’s capabilities. A helpful comparison that I’ve used in meetings like this is to imagine that you’re contracting for snow removal during the winter. If a bunch of companies come in and tell you that they know how much snow there will be and how much it would cost you, you wouldn’t really take them seriously. What you’d want is an agreement where they provide you with the service of hauling snow away when snow appears and needs to be hauled away. We’re like that. When you and your users discover that your site needs a better admin component or a simple login mechanism, you let us know, and we do it.”

Perplexed silence.

“It’s truly a different way to think about software. The commodity mindset has a lot of inertia behind it, because it’s how the world has treated software since people first started incorrectly comparing it to building construction. Someone mentioned agile a few minutes ago, and agile is, at its core, a response to try to correct this. Weekly sprints or iterations, working against a backlog—that’s a service-based approach. But even when software development shops advertise themselves as agile, they still get lured back into playing the long-tail, wild-guess estimate game. But not us. We just don’t operate that way.”

There was a pause, and Bruce laughed a bit. “Wow, okay. I’m not sure if that’s the best sales pitch I’ve heard in a long time or the worst,” he said. That broke the tension in the room and people began to shift and stretch a bit. “So how do you advise your clients when it comes to project budgets?”

Emma smiled a little mischievously and said, “Frankly, we try to stay out of that and leave it to accountants,” which drew some chuckles. “But really, we suggest that they start to think in terms of a monthly software budget. It’s a question of what you want departmental spend to be on growing your software’s capabilities versus maintaining existing software. And where we come in is on the capability growth end. When companies partner with us, it’s with the understanding that, for some period of time, they’re going to be spending more than usual on software growth. And, if we’re doing our job well, when we’ve been helping you for a while, we’ve either minimized your bottom line or bumped your top line enough that the cost to have someone sustain what we’ve done should be easily offset by the improved profitability.”

Both Elaine and Bruce were now looking at her thoughtfully, even though some members of the peanut gallery looked a bit skeptical. “And that brings me to another thing that we do differently. We strongly prefer to work with you to assess how what we’re doing is going to affect your profitability. Most firms will just take your specs and then take your money, but we don’t want to work on projects that we don’t think are going to be profitable for our clients. It’s just bad business. If what we’re doing isn’t making it easier for the company to find money to divert to software initiatives, then we don’t want to be doing it.”

“Now, that said, I wouldn’t be here if it didn’t appear to us that adding capability for merchandise management wasn’t going to help you. It seems pretty clear cut that making merchandise management more efficient will mean more suppliers adding merchandise, more purchasers purchasing it, and more profits for you and for us. And I also think that we can work with you to build a backlog where we implement things that allow us to test that assumption as quickly and inexpensively as possible, minimizing your risk. So, if no one objects, I’d like to go through my slide deck now.”

No one objected. Emma went through her presentation, noting that the folks in the room seemed receptive and, at times, impressed.

This angle was always a risk because talking frankly about strategy had a tendency to encroach on the territory of managers and business folks. But it was a good, “fail fast” risk, because her team didn’t want to work with the sorts of organizations that didn’t think the developers should be part of the strategy. And, here, it probably wasn’t a terrible risk, since this was a referral. Payless clearly already had some idea of how they operated.

At the end, there were a lot of earnest questions. Emma considered this to be a good sign. One of the hydra heads asked her how soon they could start, assuming they were awarded the contract, and Emma told him that it’d probably be a couple of months. “Oh, well, once we select someone, we’ll want to get started pretty much immediately,” he countered.

“That’s fair, and we understand that. However, we’re engaged with another client at the moment, and we won’t be wrapping that up for a little while yet. If you like the model we’re offering, though, there are other similar firms to whom we can refer you.”

Bruce looked a little puzzled. “Why don’t you just hire more bodies while your developers finish out this contract? Frankly, you’re impressive enough that I’m sure you can land more work for the first team before they hit the bench, or whatever it’s called.”

Careful here, Emma thought to herself.

“Well, two things. First off, I’m not actually a manager; I’m a developer. And secondly, we actually don’t have any interest in expanding. We’re happy keeping our team intact, working on one project at a time.”

Now Bruce looked truly puzzled. “Well, take it as a compliment that I thought you were the manager, since you’re obviously pretty savvy. But why didn’t your manager make this trip along with you?”

“We don’t have a manager or any kind of role like that,” Emma offered. “We’re a partnership, like a law firm.”

Into the semi-awkward silence that ensued, Emma cracked, “You know managers. They tend to like to get paid more than the people they manage. So if we hired one, we’d have to bump our ballpark rate from 80K per month to 100K per month. We prefer to manage ourselves and pass the savings on to you.”

Bruce gave her a look as if to say *well played*. Then he asked, almost philosophically, “So if you don’t have anyone managing you and you avoid growth, do you have no desire to scale up in any way? You’re passing on extra income left and right, I’d think. Why not grow the practice and retire young or something?”

Emma felt a touch uncomfortable, but saw no reason not to be honest. “We have a network of groups with skills and beliefs similar to ours. New players are constantly getting added. Whenever we turn down an engagement, we refer one of these groups, and we take a cut. We currently have a pretty good stream of passive revenue from referral business, and that seems to be growing. It turns out that clients tend to be pretty happy with this model.”

By this point, everyone in the room looked pretty thoughtful, and the conversation had hit a natural lull. Elaine wrapped things up neatly by resuming her bustling air and saying, “Well, you’ve certainly given us a lot to think about, regardless of what we opt to do for this project—or, excuse me, ‘extension of capabilities,’ to use your term, Emma.”

Taking it as a good sign that the mental model had stuck with her, Emma smiled mischievously again and said, “So, I do have a letter of intent in my bag, if you’d like to get started sooner rather than later...”

That got a group chuckle. Bruce replied, “You were really good, but not quite *that* good. We have a few other firms yet to talk to, and we have to think about what we want to do about you not being available for a couple of months.”

Emma didn't need Elaine's conspiratorial reassurances on the way out to know it was a good sign that Bruce was already thinking about when her team could start.

Chapter 5: Thursday Night

Emma reclined in a chair in her den, sipped a beer, and let the mildly unpleasant residue of airport travel evaporate off of her. It had been as successful a trip as she could have wished. Rhyta was on board for a successful ramp-down and, almost certainly, a good reference. Payless seemed intrigued and likely to offer them the contract. And she'd seen an email from Jim Abbot when she'd toggled her phone off of flight mode.

From: JamesIAbbott@rhyta.com [James Abbott]

Subject: Broader Question

Hi Emma,

I was intrigued by our conversation yesterday, in particular the part about staffing software teams without needing project managers. We've been looking to staff a PM role, but maybe we can get away without adding that resource? Anyway, nothing formal. Just off the record, wanting to see if I could pick your brain for a few minutes.

Thanks!

Jim

She knew it was a little petty, but Emma couldn't help delighting in a wonderful irony to which Jim was probably oblivious. After spending the vast majority of her career being called "resource" by self-important project managers, it was satisfying to note who was now expendable.

PART 2: THE NATURE OF THE GAME



Chapter 6: You Are Here

Be forewarned. It's going to get worse before it gets better. The pages you're about to read may seem deeply cynical at times, but it's necessary framing for what follows. After all, to convince you that we can and should take rather dramatic steps to change our concept of careers, I first need to convince you that the status quo isn't so great.

I can easily recall my reverence for the corporate world when I graduated college. College, after all, was dress rehearsal for adulthood, while a corporate job was the real thing. No student was buying himself a brand new car with a salary paid to him by some college. That, my friend, was only for VIPs who had been ushered past the velvet rope by a hiring authority somewhere.

I went to school at Carnegie Mellon University from 1998 to 2001, and during my time there I watched people graduate from the filthy dorms and fast food of college to the rich, yuppie world of twenty-two-year-old kids making seventy thousand a year or more. It was right in the middle of the dot-com boom, and what a time to be an upcoming computer science graduate. The job market was so hot and frenzied that the dean of computer science at our school called a special assembly of my class in the year 2000, pleading with us not to drop out to take jobs with desperate companies who couldn't be bothered to wait for us to graduate. It was a feeding frenzy. I watched my older peers graduate, take high-paying jobs, buy fancy cars, and rent swanky apartments. I couldn't wait for my turn.

But, in a development that experts would knowingly predict after the fact, the dot-com bubble, well...popped. I was at the head of the line, having freshly graduated, and was taking interviews. They would go well until, one by one, I'd be told things like, "We're implementing a temporary hiring freeze," and, "Oh, we've just revised our policy and

we're no longer hiring entry-level candidates." I was frozen out. Offers turned into "Oops, sorry," and second interview invitations turned into "We'll call you when things settle down."

It was actually a lot worse for those a little older than me. I hadn't signed any leases or bought any cars when the downturn hit. But it was, in some ways, more discouraging. The club had closed when I'd been the last person standing in line, waiting for admittance past the velvet rope, and I'd never gotten so much as a glimpse at the swanky wonders within. This combination of expectation and then elusiveness made the corporate world seem even more formidable to me. Not only was it the home for money, power, and influence but it was also a fickle, cruel master that could shrug me right out into the cold. Oh, how I wanted in.

When I reflect back through a veil of experience and the accompanying cynicism, I realize I never had much of a chance with this early outlook. My parents are both college grads with professional master's degrees, and both of them spent accomplished careers working their way up to impressive director and C-level positions. During the protracted job search and discouraging retail jobs that followed my ill-timed graduation, they kept me from giving up, reassuring me that I would eventually get my foot in that door. They told me I'd even do better for the lessons I'd learned, dealing with early adversity. They were right, as I'd learn much later.

And so it was that the corporate world remained in my mind some kind of promised land, populated by people that I simultaneously envied and wanted to emulate. While I was slinging cell phones at Radio Shack, corporate people were out doing important, wonderful things, like wearing slacks, having meetings, and zeroing their Outlook inboxes. It got so bad that, at one point, I showed up to a meeting with a guy who I'd met as a customer in Radio Shack. He said that he had an e-commerce business venture and wanted to talk with me about it. I thought it was a job opportunity. When I got there, I found myself in an auditorium full of gleefully shifty people

cheering at a flip-chart drawing of a pyramid. They chanted along in unison with the emcee of this affair when he demanded to know why so many people were so unhappy: “because they’re sick of having jobs!” You see, the allure of the Amway—excuse me, e-commerce—promise is that the wonders of multi-level marketing allow you to get rich passively while the suckers of the world go to work.

It was a low point for me. As I sneaked out of there, dodging the huckster that had tricked me into meeting him, I could almost have wept with bitterness. Those lazy people, frothing with avarice and glee at the prospect of suckering others out of their money. *You horrible human beings*, I thought. *I would give anything to have a job. You’re the suckers—not the people who are collecting money to do honest work.*

Almost fifteen years have passed since that day, and their take on the nature of nine-to-five work no longer offends my sensibilities in the slightest. Don’t get me wrong. The enemy of my enemy is not my friend. I find the value-destroying proposition of Amway to be reprehensible, no matter how many stadiums and lobbyists they buy as they dance narrowly within the realm of legality. But I also find that many of the trappings of the corporate world are empty, arbitrary, and life-wasting.

There’s an irony to a room full of get-rich-quick schemers criticizing the citizens of the corporate world for being suckers. You have to work extraordinarily hard to con enough people into a multi-level marketing scam to make any real money at it—harder, in fact, than people work in most nine-to-five gigs. It’s a lot easier to get roped into an Amway meeting and be parted with your dollars to sign up than it is to convince dozens of people to do the same and then inspire them to convince dozens more each. But all of you reading know people at the office that sit and browse Facebook every day in lieu of doing any actual work. You all know people that contribute exactly nothing to your group’s effort. You all know programmers that, literally, do not know how to program computer software. This

means that we all know people who earn a comfortable wage adding absolutely zero value to anything.

With this in mind, is it really amazing that there's a group of people out there that feel spurned because they couldn't hack it in the corporate world, so they fork over \$200 (or whatever the buy-in winds up being) to go to a series of meetings that could aptly be titled "Catharsis for Scumbags"? And how ironic is it that they've opted out of a life of loafing for a comfortable wage in favor of pursuing a high-risk, grueling, "entrepreneurial" career of moving endlessly from one small con to the next? And how weird is it that, in spite of creating such an unsavory personality cocktail of greed, laziness, and stupidity, they've basically got the modern corporate world pretty well pegged?

Does that sound harsh? Well, let me prepare you. This may be a section of bleakness, but things will get better by the end of the book. I promise. To understand where we *can* go, where we *should* go, and how we might get there, it is essential to be completely frank and forthright about where we are.

Chapter 7: The Corporate Cave

I got a job, you know. That Amway debacle was almost certainly the nadir of my post-college job search, and I'd like to treat you to the kind of storybook redemption that had me land a job the next day. That didn't happen, though. It would be months before I received my coveted invitation to the corporate dance. But it did come, and I knew vindication. I can still remember mouthing my new title with a sense of reverence: "software engineer." (Well, initially, "software quality engineer," but I won't tell if you don't.)

The job treated me well and provided me with some of my most stable professional years. I lasted longer at that company by far than I have at any since. I attribute this to how new I was to professional software development and to an excellent manager that provided me with plenty of challenges, autonomy, and air cover from organizational politics (while still allowing me to observe them). I became a sponge. A new chapter began in my own education, and I dual majored in corporate politics and software development in the real world.

As a fresh college grad, I had viewed the corporate world as a place of remarkable constancy. This remained true through the years I spent at my first job. Every three or four years growing up, I had switched schools, but in the corporate world at that time, prevailing wisdom held that you switched jobs every five to fifteen years. For my parents' generation, this number would have been even larger; corporate jobs used to be more akin to marriages in duration. And I was thus a young newlywed, convinced that the next fifty years held nothing for me but business-y matrimonial bliss. I'd work hard and prove myself, kicking butt on every rung of the corporate ladder all the way to a corner office.

That multi-pronged strategy was partially effective. I worked hard and kicked butt but remained on the exact same rung of the ladder for five years. Oh sure, there were accolades, raises, glowing performance reviews, and leadership responsibilities, but none of this was codified into real career movement. As a matter of fact, I didn't even receive the ego-boost of a fancier title. That galled me by the end and led me, for a time, to value titles to a degree that I now find naively comical. After five years, the reverence with which I had mouthed the title "software engineer" had been replaced by bitterness.

Looking back, my overvaluation of title was just a symptom of a deeper illness. And the deeper illness was the grand delusion that the pyramid-shaped organizational chart is some semblance of a meritocracy. It isn't, and it's not close. But I had some jobs to hop before I'd figure this out.

there at my first job, the great unfairness of it all finally caught up with me. I had sole responsibility for more than one software product, and I was now the lead on other initiatives, directing people with more advanced titles than mine.

I simmered and seethed like an unattended pot of chili left too long on the stove, concluding that your value to the company and your title must have nothing to do with one another. Ironically, I wouldn't learn until well after I quit that this wasn't *entirely* true. But before that, I took stock and realized that, regardless of aptitude or skill, people "earned" their way from software engineer I to software engineer V by waiting for five years and having it handed to them. No matter if I was leading a team of IIs and IIIs and getting more done than all combined. I hadn't put in my five years yet, so a software engineer I is what I would remain. I did some math and had an existential career crisis at the ripe old age of twenty-six. I'd be forty-eight years old before I reached software engineer V, and the cartoon I had in my head of my career had me as a VP or CTO by then. Something had to give.

Eventually (and before I would learn the silliness of my title-focused outlook), I quit and left for greener pastures and more enviable titles—or, at least, so I thought. In quitting, I learned one of the most important lessons of my corporate career, which was that of leverage. On my way out the door, I was offered more money and the coveted software engineer II title. I didn't take it, having been soberly schooled in "the dangers of the counter-offer" by a recruiter taking himself entirely too seriously, but I did note it. For all of their talk about meritocracy, corporations seemed lightning quick to reassess your "merit" when you had other options. Five years of saving troubled projects, producing miracles, and running teams hadn't earned me software engineer II. But offering resignation did. Lesson learned.

Growing up, the world had been simple. We went to class, took the occasional aptitude test, and then went in different directions, depending on our scores. At the risk of sounding cocky, I'll tell you I spent a childhood acing standardized tests and being put in advanced classes. At the time, it seemed overwhelming, but next to the corporate world, it was easy. Show up, get a good score, get "promoted." The criteria were objective and unassailable. You never had to threaten to drop out of school to get into AP physics. That had continued through college as well, this paradigm of taking tests and being judged. And yet, in the corporate world, there was nothing like it. There was no objectivity. In college, if you'd gotten a C and marched into the dean's office to announce that you were quitting, the dean would have laughed at you. In the corporate world, it resulted in them changing your grade to an A and asking you to stay.

After leaving that first job, I started reflecting upon this a good bit. It could be argued that I became mildly obsessed, attacking this problem, mentally, the way that so many of us attack an intractable bug. I ruminated and even brooded about it at night, when I should have been paying attention to my girlfriend or even working on various side projects. I fixated. I needed to understand. How did the strange world of office politics work, and why was it so different than the relatively simple world of software problems?

One of the oldest pieces of philosophy that exists is Plato's allegory of the cave. In it, Plato describes a tribe of people that live in a cave and are bound so tightly and inescapably that they cannot move at all—even to turn their heads. And they are fixed in place to stare at the cave wall. Behind them, people enter the cave periodically, make fires, and put on puppet shows for the cave dwellers. All that the bound cave dwellers can see are the shadows from the puppeteers, who are tricking them. Those shadows are almost the entirety of their lives, and they measure social rank on the basis of who can best predict future movements of the shadows and remember past ones.

Now, imagine that a puppeteer unbinds one of the cave dwellers and shows him the fire and the puppet show. It would irrevocably alter the cave dweller's world and perception thereof. Further, imagine that the puppeteer removes the cave dweller from the cave altogether and shows him the full, vibrant world of the sun beating down on the countryside. The cave dweller would be changed to such a degree that his former life would be inconceivable. Now, imagine trying to put him back there with his former friends, exchanging social status on the basis of how well they remembered moves of the shadows and how well they could predict their next moves. He would predict the next moves easily, take no pleasure in doing so, and find his former friends unrelatable and pathetic.

My first resignation was my initial step away from the cave wall. I wasn't out in the countryside, taking in the full splendor of human existence, but nevertheless, I'd won an important victory. I should mention at this point that it may seem a bit dramatic to compare corporate workers to the cave dwellers. I'd ask you to ignore the *Matrix*-like implication that corporate workers are so sheltered and trapped that their entire lives are shams. Rather, I invoke the allegory of the cave to convey that there are layers to how the corporate world works and how, once you see them, you really can't un-see them. Your perception is changed. This section is about my journey from naivety to relative understanding—relative in that it's allowed me to play the game well enough to get ahead in short order.

Chapter 8: Growing Up

During the course of my aforementioned brooding about the corporate landscape, I came to perceive some common archetypes. At my first company, if you took one of the developers' years in the corporate work force, divided by five, dropping any remainder, and added one, you could predict his or her level of software engineer. So as someone with fewer than five years, I was four-fifths, which rounds down to zero, plus one, equaling software engineer I. Someone with seven years would be software engineer II, while someone with seventeen years would be software engineer IV. And so there was an archetype whose position could be predicted entirely as a mathematical function of "number of years managing not to get fired." Indeed, this was the source of my eventual bitterness at that organization. If I were honest with myself, no small part of that feeling was self-disgust for having tried so hard when it clearly didn't matter.

The next archetype that stood out was the people who had inexplicably cut in line somehow. What I mean is that my boss's boss was the VP of engineering, and his roughly twenty years in the professional work force did not square with $\text{floor}(x/5) + 1$. He should have been a software engineer IV, not my boss's boss. I hadn't worked out the math for "boss's boss," but surely that would have to wait until you were eighty or something. This was even more befuddling when I considered the CEO, who was even a bit younger than his subordinate, my boss's boss. Once LinkedIn became a thing, I vaguely recall looking at their profiles and those of some other similarly inexplicable folks and realizing that some kind of black magic was at work. The black magic seemed to center around switching companies.

Another archetype that I noted was the "line manager as principal engineer emeritus." If you'd given the company something like

twenty-five-plus years, it appeared that you inherited some kind of director/manager position as a sheer product of longevity. It reminded me of the papacy, but without much competition, since the number of people that hung around that long tended to be small enough that it was just a question of waiting for the previous guy to retire. It's like the papacy in the sense that a lifetime of service gets you a crack at the boss role when you're probably not far from retiring either way.

Since I was looking for patterns, I noticed these key archetypes and other comparably minor ones. There were the insufferable (transparent) boss imitators, though these were not normally engineers. There were the "oh noes, don't take my keyboard" reluctant recipients of promotions to project manager. There were the overworking self-flagellators that put in sixty-hour weeks for no reason I could discern and to no benefit. And there were intense loafers whose contributions I struggled to grasp.

At this point, I could continue to walk you through the evolution of my perception of the corporate landscape, but my chronology from here forward isn't especially interesting. Instead, I'll offer a framework of thinking about corporate citizens that makes sense of and encapsulates the archetypes I've described so far. To do that, let's turn back the clock.

Think back to being a kid. You can probably remember a rather dubious rite of passage that occurred when you figured out that you weren't going to be a sports player, lead singer, or Hollywood star. You probably felt sad even as your parents sighed with relief that you'd never be explaining a manual labor gig as your "day job." State lotteries notwithstanding, giving up on improbable dreams is considered by society to be a measure of maturity.

If you think about this, the easy message to hear is "you're not going to be great, so give up." But that's not what's actually being expressed. The real lesson is that the expected value of these vocations is horrendous. For baseball players, actresses, and rock stars, there's a one in a million chance that you'll make ridiculous

sums of money and a 999,999 in a million chance that you'll make \$4,000 per year and have half of it paid to you in beer nuts. The expected value of these "careers" is about a \$4,200 per year salary and a handful of beer nuts. So the message isn't really "give up because you'll never make it" but rather "steer clear because anything short of meteoric success is impoverishing."

The better play, we tell our children, is to head for the corporate world where the salaries range from minimum wage in the mail room to tens of millions per year for CEOs of international conglomerates. Most importantly, you can find every salary in between. If you aim for the heights of CEO and fall short, mid-level manager making \$140K per year isn't a bad consolation prize. And so a funny thing happens. We consider it to be a rite of passage to abandon the delusion that one will be Michael Jordan, but we encourage the delusion that one will be Bill Gates until people are well into middle age.

That's right. "The delusion that you'll be Bill Gates." You won't be him. You won't be a CEO, either, unless you pop for your state's incorporation fee and give yourself that title. You're about as likely to "work your way up" to the CEO's office over the course of your career as any given child is to luck into being the next multi-platinum pop star. So it's a rather strange thing that we tsk-tsk children for indulging pie-in-the-sky fantasies past a certain age while we use nearly identical fantasies as the blueprint for modern industry. Kid wants to be Justin Bieber? Pfft. Thirty-year-old wants to be Mark Zuckerberg? Sure, why not? Keep working hard, kicking butt, and acing those performance reviews, and someday you'll get there!

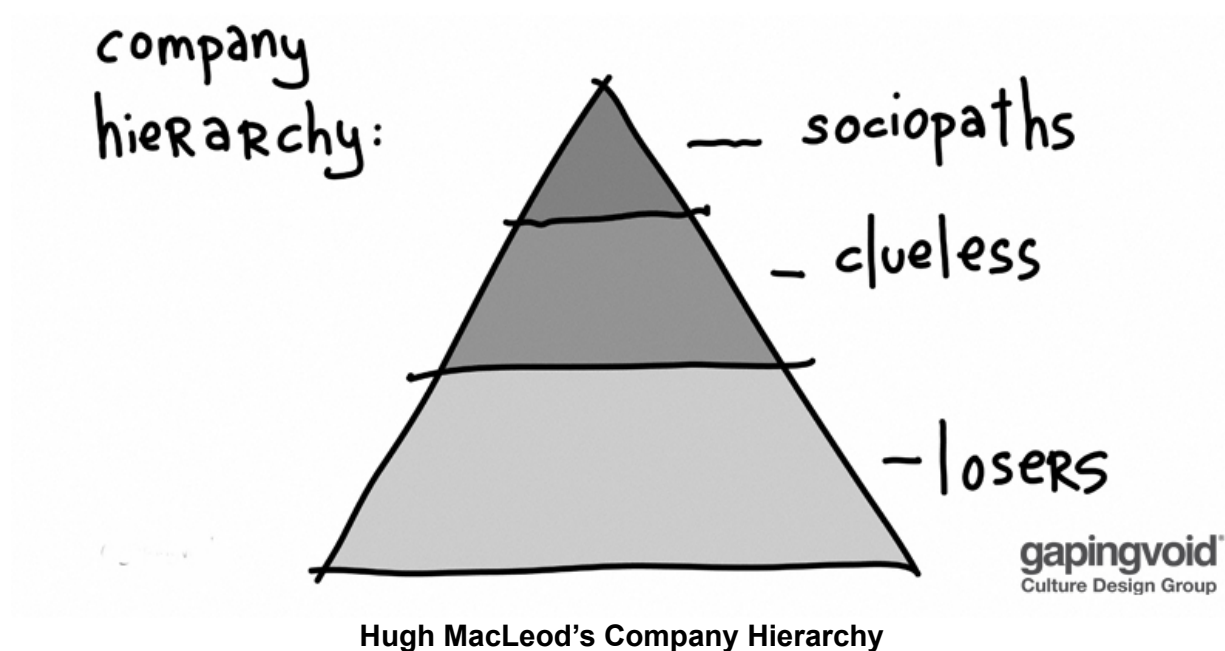
Pfft.

It took me five years in the corporate world to realize that working my way up the ladder to great heights was about as likely as my childhood ambition to play for the Chicago Bears had been. Some may figure this out much sooner, but I suspect that many never actually figure it out. And the cost of not figuring it out isn't particularly high. I was comfortable as I went nowhere fast—I wasn't playing guitar in a bar for beer nuts.

Chapter 9: Cynical Theories of Management

One might consider me to be a something of a cynic. (I submit that I'm actually just an impatient optimist, but that's a story for another time.) I certainly feel cynically toward the corporate structure as it exists today, but before I get too deep into my assessment, I'd like to discuss some theories that I consider to be a good bit more cynical than my own.

There's a drawing by a cartoonist named [Hugh MacLeod \(of gapingvoid.com\)](http://gapingvoid.com) that's ingenious in both simplicity and critique. Drones at the bottom, ruthless manipulators at the top, and a creamy center of buffoons. These are the ones that think they're going to make it to the top. They won't. And they're clueless to that fact. They're the kids that still think they're going to grow up to be Jennifer Lawrence.



Venkatesh Rao took this cartoon, combined it with the television show [The Office \(American version\)](http://TheOffice.com), and created a spellbinding and

beautifully provocative theory of corporate management that cut to the core. [The Gervais principle](#), as he coined it, refined the Dilbert principle, which was itself a refinement of the Peter principle. The two Gervais principle predecessors were cynical to the point of comedy; the Gervais principle is cynical (and accurate) to the point of tragedy.

[The Peter principle](#) holds that people are promoted until they prove incompetent in their roles, and there they remain. Competence is rewarded with promotion and incompetence is rewarded with the status quo. [The Dilbert principle](#), with more of a knowledge-worker focus, rings true to those of us who have seen terrible programmers promoted to project managers. It states that bad employees are promoted into management to prevent them from doing damage with their incompetence. The Gervais principle, with its “sociopaths” (“ruthless manipulators,” in the cartoon), “clueless” (“creamy center of buffoons”), and “losers” (“drones”) gives a lot more credit to those at the very top, which, in my opinion, makes it far more accurate in its reasoning about corporate leadership. This principle says that the sociopaths that run the organization knowingly overpromote dedicated but relatively inept people into middle management.

Why would they do this? In middle management, these clueless folks serve various ends for the sociopaths. But the most important ones are “foil” and “buffer.” As foils, they can be cannon fodder on projects with low chances of success and they can be blamed when things go sour. The buffer play is a bit subtler. The sociopaths that run the company have power, influence, and lives that make most people jealous. The losers at the bottom rungs of the corporate ladder are jaded line-level employees resigned to a relatively powerless lot in life. They put in just enough effort to remain in good standing at work and remain aware that their employment is a pretty bad deal for them and a pretty good deal for the sociopaths at the top. A lot of direct interaction between the executives and the rank and file would quickly lead to resentment. So these high-level sociopaths overpromote a handful of the low-level losers who put forth disproportionate amounts of effort. These former losers enter the clueless ranks of middle management to act as a buffer. The remaining losers can’t really hate the promoted clueless because the

clueless aren't calculatingly taking advantage of them. The clueless believe that they're on track to be CEO while the losers and the sociopaths both know that's absurd. In the *The Office*, Michael Scott represents this archetype—incompetent, fanatically loyal to his company, and clearly not headed for the C-suite, whatever he might think.

If you want to really conceptualize this and have it driven home, consider a hypothetical scenario as follows. The CEO of some large organization with thousands of developers decides to hold a mandatory contest to see who can write the “best” web application, whatever “best” means (left vague intentionally). The contest will be done on the developers' own time, but the winner receives a \$50,000 bonus and a promotion to CTO. Picture what happens next.

A solid majority of the developers roll their eyes, spend an hour implementing some hello world kind of thing because it's required, submit it and forget about the contest. A sizable minority heads in the complete opposite direction and goes absolutely nuts competing, certain that they're going to win that sweet, sweet prize. These developers pour hundreds of man-hours each into it over the course of months, completely losing sight of the fact that they're each individually contributing tens of thousands of dollars in free labor.

Who wins? Well naturally, it's the developer who figured out that his sister is friends with the CEO's favorite nephew, parlayed that relationship into favorable treatment, and then plagiarized some web app from GitHub.

What's the fallout of this? The losers always understood that the contest was pretty hokey and probably too good to be true, so they yawn at the CEO and his nepotism and figure it's business as usual. The clueless are disappointed, but they know that the best, most qualified candidate won. They had their chance but just didn't quite work hard enough. They vow to work even harder next time, and the company sells their free labor for millions, earning fat performance bonuses for the sociopaths at the top. The sociopath who cheated earns a seat in the executive room.

Do the losers resent the C-levels? Sure. Do they mutiny? Not if the clueless are promoted into a role above them. The clueless so genuinely believe in the organization and its wisdom that it's impossible for the losers to hate them. What they feel for them is a mixture of pity, disgust, and occasional gratitude (if they happen to be nice or generally benevolent in application of power), but not hate. The losers satirize them in cartoons with pointy haired bosses and they gossip about them around the water cooler, but because of the clueless buffer, they don't collectively revolt and go out in a blaze of spite.

In effect, the clueless create two different organizations within one organization. There is the organization of losers and clueless, where putting in sixty-hour weeks and being obsequious lets you claw your way up a few of the bottom steps of the pyramid. And then there is the organization of clueless and sociopaths, where putting in sixty-hour weeks and being obsequious keeps you right where you are with that next level of advancement always being oh-so-close-but-better-luck-next-time. Creating the bottom level organization, where tripping over yourself to provide free labor is rewarded with small stakes promotions, allows the top level organization to sustain a model where the losers and clueless get terrible economic deals and keep coming back for more. The clueless have no idea this is occurring. The losers understand it but have no appetite for rebelling against their clueless managers who are answering emails at three in the morning and working sixty-hour weeks. The people they'd *actually* like to rebel against are, quite simply, out of reach.

Chapter 10: Defining the Hierarchy (With Less Cynicism)

Venkatesh's treatment of these archetypes is wonderful, and you should [buy his book](#). In particular, the sections that describe how these archetypes deal with one another are absolute goldmines of strategic understanding of corporations and their players. But I have three main problems with using the archetypes, as described, to elaborate on my own theories of corporate politics: (1) the names themselves, (2) the assertion that overperforming middle managers are generally idiots, (3) and the placing of corporate citizens into one of three buckets on the basis of assigning them serious shortcomings.

In terms of the specific shortcoming buckets, the losers are some mix of lazy and cowardly, having given up on the idea of controlling their own destinies. The clueless are idiots that don't understand the nature of their relationship with the organization. And the sociopaths are ruthless users and manipulators of other people. All three archetypes are mainly defined by their core flaws.

This seems to work well as shorthand and for catharsis. Frequently I feel the need to completely withdraw from the corporate world and recharge a bit, and whenever that happens, I bask in this sort of cynical characterization. The structure and those who inhabit it are flat out ridiculous. But when I take a few deep breaths and calm down, I just can't make myself view anyone who works at a corporation as most aptly identified by what's wrong with them. When every component of a system appears to be functioning poorly, one has to consider that it may be the system, not the components, that isn't working.

I thus prefer not to think of corporate citizens in terms of their shortcomings. Instead, I think of them in terms of what the modern

corporate structure has done to them—broken the losers, tricked the clueless, and forced the sociopaths into ethical conundrums. I don't agree that the corporate structure is optimal or inevitable, and I think its deep flaws show themselves through the human beings that execute its various rituals. Don't think of what's wrong with these folks. Instead, think of what they've lost.

The loser is pretty simple to size up in terms of loss. What's been taken away from the line-level employees who lack organizational faith is their hope. These are people from whom you can expect to hear pithy consolation narratives like, "I don't live to work—I work to live." The loser has forked over any real hope at a dream life in favor of small optimizations designed to make a common grunt's situation more livable.

What's been stolen from the clueless is a bit subtler, but I'll couch it in terms of information. A sense of perspective has been stolen from them by the rat race, resulting in wild overvaluation of perks and honors conferred on them by the organization. Part and parcel with this is the cognitive dissonance of assuming that their ascension in an organization was the result of merit and hard work rather than inevitability and patient waiting.

The most difficult to assess is the sociopath, who has an enviable position at the top of the organization. It's easy enough to think that sociopaths are the ones *taking* things from the other two archetypes and thus are sitting pretty themselves. But in reality, their position is something of a default one. They refuse to cede hope and they refuse to cede perspective, so they acutely understand that the corporate citizenship game is one where the only outcome of playing by the rules is to lose. And so what they give up is the ethical compass they had when they began their corporate journey.

I'll offer a temporary diversion into some terms coined by Karl Marx that roughly parallel the organizational dynamic. Generally, when one thinks of Marx and gets past the modern political notion of communism with the vacuous partisan squabbles it inspires (at least in the USA), the terms that come to mind are "bourgeoisie" and

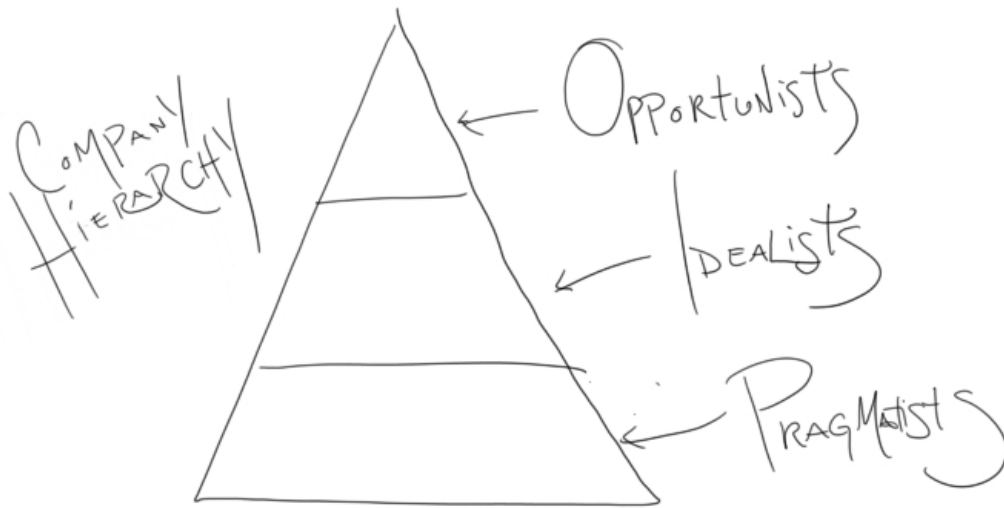
“proletariat.” (The latter was also made famous by George Orwell’s 1984, in which a free peasant class was referred to as the “proles.”) A lesser-known term is “lumpenproletariat,” and it refers to mercenaries and criminals that are products of the system, so to speak. It’s easy to think of the CEOs and jet setters of the corporate world as bourgeois, but resist this temptation. Mid-level managers (clueless) are the corporate bourgeois, while line-level employees (losers) are proles. Sociopaths are lumpenproles. They are the Jean Valjeans of the corporate world, with clueless Javerts as their foils, should those clueless ever come to understand that the sociopaths win by cheating.

Sociopaths are the most quickly disillusioned and repulsed by the corporate rat race, and they recognize that the path to real success and influence is to cheat while everyone else toils away in good faith. They give up guilt-free operation in favor of a feeling of dirtiness as they cut in line to get to the top and make decisions that impact the lives of others. It’s easy to think, particularly given the “sociopath” moniker, that the offices of corporate movers and shakers are filled with remorseless villains, but that’s really not the case. For each true psychopath that gleefully steps on people to get to the top, there are far more that regret various stones along the path.

I’ve spent a lot of time in a lot of different organizations of different sizes and domains, and I just don’t run across cartoonish people like Michael Scott and Dwight Schrute. By and large, the people in these organizations, at all levels, are relatively well-intentioned, reasonably intelligent, and doing the best that they can on the micro, day-to-day level. Corporate structures are, however, substantially less than the sum of their parts, so good faith efforts on a small scale are perverted into rampant dysfunction writ large across the face of industry. Organizations are pathological, as Venkatesh points out, and they are pathological in a way that corrupts their components.

It’s for this reason that I propose that we rename the losers, clueless, and sociopaths to “pragmatists,” “idealists,” and “opportunists.” Their roles and dynamics remain the same, aptly described by Venkatesh. But these suggested new names soften our attitude toward them.

Pragmatists are line-level employees who find value in life outside of work, mainly because the hope of any meaningful advancement and enjoyment of their profession has been taken from them. Idealists believe heartily in the meritocratic company (and organizational superiors) as a benevolent steward of their careers because perspective has been taken from them. Opportunists refuse to yield hope or perspective and recognize that the only way to win the corporate game is to play by their own rules. In this realization, they give up ethical certainty and human connection. Opportunists play a lonely, sad game to get what they get.



My Company Hierarchy

But, as I mentioned, the dynamics are not altered in the least. Pragmatists contribute as little as possible to preserve stability, getting a bad economic deal and recognizing it. Idealists, believing in the company, work even harder and make their economic deal even worse. Don't be fooled by a slightly higher salary and meaningless perks like offices and parking spaces. Working 50 percent more your entire career to eventually get paid fifteen thousand more per year is an abysmal deal, compared not only with opportunists' deals but also with minimum effort, lower-wage pragmatists' deals. And the opportunists feeding the grist to the mill certainly overpromote

idealists because of strategic necessity rather than any kind of merit. Even with different labels and humanized, sympathetic consideration, the show must go on. It just might be easier for you to see what role you have when they aren't all described so negatively.

I should also mention that, in spite of the neatly-drawn layered pyramid, you will find the occasional archetype representative outside of the normal bounds. One will generally become an idealist well before earning an actual promotion to any management role. Opportunists are similarly fired in the forges of lower roles and their opportunism enables their swift ascendancy.

Here's a very simple example to illustrate these dynamics. Let's consider an employee named Alice, laboring away as part of a group of line-level knowledge workers. In the group, there is an official "team lead" position that has no reports but is a leadership role. This role has recently been vacated, and the odds-on favorite to replace the former team lead is a loud-mouthed, long-tenured guy named Bob. Let's further assume that this role is at least passingly desirable to Alice.

Alice the pragmatist (formerly called "loser") looks at this wistfully and shrugs because she knows that, even though Bob is an idiot, his assumption of the role is inevitable. She makes peace with that, feels generally checked out at work, and enjoys her life in other ways.

Alice the idealist (formerly "clueless") looks at this opportunity and resolves to put in sixty-hour weeks to Bob's fifty-hour ones. She also begins to match Bob blow for blow in self-promotion, grandstanding, and managerial posturing. She knows that, for the short term opportunity, she's unlikely to edge Bob out, but over the long haul of several years of sixty-hour weeks, she'll prove that she deserves that role. The economics of working 50 percent more for free to earn an eventual promotion never really occur to her.

Alice the opportunist (formerly "sociopath") looks at the situation and finds common ground with her pragmatist and idealist selves. She realizes that she's no match for Bob the incumbent but she also

knows that trying to prove herself one over the next five years is a sucker's game. Like her idealist self, though, she wants the role. So Alice the opportunist updates her resume to include weasel terms like "thought leadership" and, with plausible deniability, starts interviewing for team lead roles at other companies, eventually landing an offer and either taking it or parlaying it into being placed in the role ahead of Bob.

Throughout the rest of this book, I will use this shorthand liberally, describing corporate citizens as pragmatists, idealists, and opportunists. Before continuing, however, it bears mentioning that I understand the somewhat reductionist nature of this categorization. I promise you that if you look around your office, you will find an example of someone who doesn't fit neatly into three buckets. People can be outside of these categorizations. The purpose of the shorthand here is to make it easier to describe corporate dynamics and speak to general trends.

Chapter 11: Interviews, Induction, and Nonsense

It is overwhelmingly likely that your initiation into the business world was via a job interview. In fact, you probably dealt with this mainstay of employment well before you dealt with Outlook and meetings and whatnot. Fifteen-year-old you went down to the local ice cream parlor and a haggard manager asked you why you wanted to work there and whether you could keep your cool when someone went nuclear over you being out of rocky road. You probably thought it was a little stupid at the time. I know I did as a teenager. But then, we had a lot of growing up to do. It would take a couple of decades and dozens of interviews from both sides of the table before I would finally conclude that the activity is not, in fact, a little stupid. Rather, it is *profoundly* stupid.

Before I describe the history of the job interview and the degree to which anyone has made empirical attempts to measure its effectiveness, let me just describe how it actually works. I get that you know how it works by rote—everyone does—but, please, bear with me. It may be interesting.

Someone somewhere decides that a position is now open, and the game is afoot. The person with the budget writes up a job description in which he lists every skill that anyone could ever possibly use. Then, he passes it on to human resources or recruiting, depending on the size of the company. These folks, or perhaps an outside recruiting agency, munge the job description together with the official marketing on “careers at Acme Inc.” This marketing includes stock photos of people of every imaginable race, creed, religion and background, none of whom have ever worked at Acme Inc. It also includes a description of the corporate environment that makes you wonder if you’re Pinocchio headed for Pleasure Island. They have gaming stations, beanbag chairs, and, apparently—for no reason you can discern—canoes.

In parallel, you're putting together a resume and practicing your interview skills so that you can do largely the same thing writ small. You find yourself wondering, "Would it really be a *lie* if I said I was proficient in Portuguese? I had a pretty good time at that port bar when I went to Lisbon, and the bartender had no trouble understanding me." Once you've massaged your LinkedIn profile, your resume, and various other self-marketing platforms to the very flattering edge of dishonesty, you dispatch this profile into the ether of some job site. Some matchmaker somewhere pairs you with the company.

Next step: the phone interview! Though it's not so much an interview as it is a phone screen. The purpose of this call is for a company representative to determine whether or not you're a lying bag of crap. I don't mean that she's going to try to feel out whether you really know your Portuguese. Rather, she wants to know whether you have even the barest semblance of competence for a role in which you claim to have five years of experience. Since she's short a team member, she's really busy and calls ten minutes late. You make awkward small talk and then dive into the meat of things. You'll both know quickly whether it's going well, but even if it isn't, you'll play out the string. Best case scenario is that you and she wind up making pleasant, if slightly forced, small talk, and you hear who will "be in touch with next steps." In the worst case scenario, the call ends quickly and the company pretends that you're not a person that exists on the planet, never bothering with any feedback at all.

Now, if you pass the phone interview, it's time for the main show. For your part, you put on uncomfortable clothes that you never normally wear because that's what you're supposed to do. You then make sure to arrive early and sit in your car until five minutes before the interview so that you can show that you're perfectly punctual without running the chance of being late. Again, that's what you're supposed to do. If you're nervous, it might not hurt to take a mild tranquilizer. Basically, you want to show them that you're fastidious, impeccable, punctual, calm, cool, and collected, even if you're none of those

things. Meanwhile, they're taking similar if less drastic steps, making sure that the office looks appealing and whatnot.

Once you're there, you meet eight different people over the course of three hours and experience a moment of stomach dropping panic when you realize that you didn't catch that fifth guy's name. But it's all good. You're nailing it. You've practiced folding your hands in front of you so that you don't fidget, and you've practiced projecting and making eye contact. You smile a lot, answer with confidence, and gently pivot when you're asked a question about which you're unsure. Finally, it's over. You walk back to your car, looking good on the outside and sweating profusely into your undergarments, ready to drive quickly home to crack open a beer and unwind from an intense day of pretending to be someone else. Now, the only thing left is for both sides to determine whether an offer would be appropriate and to negotiate over particulars. (In theory, anyway. In reality, the company tends to act like it's playing Roman emperor at the gladiator games, offering thumbs up or down.)

And that's it. In the end, it's about four total hours of two parties grandstanding and putting forward their best faces to determine whether or not they should spend the next bunch of years working together. If that seems reasonable, ask yourself if you'd go out for a night of speed dating with the catch that you had to marry someone by the end. The professional relationship between employer and employee is just that—a relationship. And it's an extremely close and high-contact one, at that. Yet the matchmaking process employed by both sides loosely resembles buying an appliance. You go in, size up a bunch of different ones, make superficial judgments, ask weird questions of the sales associate to convince yourself that you're being deliberate and thoughtful, and then go with your gut either way.

How did this process come to be a standard part of hiring? Has it evolved and been refined over the years into a speed-dating game of chicken? Was it somehow worse than this in the past? Well, mercifully, the answer to that last question is no. It's not worse than it

was in the past. That's because it's not different than it was in the past.

In 1921, tired of hiring college graduates that didn't know as much as he did, Thomas Edison made up a giant trivia questionnaire to administer to inbound applicants. According to [Mental Floss](#), questions included "Who invented logarithms?" and "Why is cast iron called Pig Iron?" If you look at the sorts of questions that modern day tech companies seem to think they're cute for asking, courtesy of [cio.com](#), they include such profundities as "Why is the Earth round?" and "How much do you charge to wash every window in Seattle?" If you mixed Edison's and tech companies' questions together, you'd be hard pressed to tell the difference.

To summarize, almost 100 years ago, an aging, eccentric, and incredibly brilliant inventor decided one day that he didn't like hiring kids that weren't his equals in knowledge. He devised a scheme off the cuff to indulge his preference and we're still doing that exact thing about a century later. But was it at least effective in Edison's day? Evidently not. According to the [Albert Einstein archives](#), Albert Einstein would not have made the cut. So the biggest, trendiest, most forward thinking tech companies are using a scheme that was dreamed up on a whim *and* was dead on arrival in terms of effectiveness.

But surely it's evolved somehow. Right? Well, no, at least not in any meaningful way. In this [piece from Business Insider](#) about the "evolution" of the job interview, we can see that what's actually changed is the media for asking dumb trivia questions. In Edison's day, interviewers had to get cute face to face. Now they can do it over the phone, through a computer screen or even via a mobile app. Who knows what the future will hold for the job interview; they may be able to beam the stupid directly into your cerebral cortex!

So, to recap, the job interview is a process that was dreamed up on a whim about a century ago, never worked in the first place, and hasn't been altered since. Unsurprisingly, they haven't magically started working. According to a [Washington Post article and](#)

[supporting studies](#), the unstructured interview in which the interviewer asks questions of his or her own choosing is actually worse at predicting performance outcomes than simply looking at resumes and not bothering with the activity at all. That's right. A more effective approach to hiring people than conducting these types of interviews is not bothering to conduct them.

The only company that I've heard of actually being empirical about the hiring process is Google, and they've come to a similar conclusion. Laszlo Bock, the senior vice president of people operations at Google, had the following to say in an [interview with *The New York Times*](#).

We looked at tens of thousands of interviews, and everyone who had done the interviews and what they scored the candidate, and how that person ultimately performed in their job. We found zero relationship. It's a complete random mess, except for one guy who was highly predictive because he only interviewed people for a very specialized area, where he happened to be the world's leading expert.

On the hiring side, we found that brainteasers are a complete waste of time. How many golf balls can you fit into an airplane? How many gas stations in Manhattan? A complete waste of time. They don't predict anything. They serve primarily to make the interviewer feel smart.

Bock and Google have been introspective about hiring and sought to improve the way they do things—or, at the very least, be less bad about it. After starting to discover that the traditional interview process conferred no benefit, they ran experiments, such as hiring people that had been narrowly rejected and evaluating their performance. There was no difference between those originally slated for rejection and those who were slated for hiring. Oops. But at least it was corrected.

And that's become a theme with Google. Not only have they moved away from the insipid brain-teaser model, but they've also introduced indirection into the process to prevent natural, in-person biases from creeping in, such as a natural tendency to hire taller or thinner people. They're attempting to improve and to correct injustices, which is laudable. But even after all of the impressive work that they've done, the process is still not especially effective. According to a [recent article written by Bock](#), the *best* current technique they have as part of the hiring process gives candidates a test that simulates work they would actually do, but it only explains 29 percent of a candidate's future performance. This is an improvement from the 14 percent rate of the unstructured interview and from the 7 percent rate of checking their references, but it's hardly a lock. You'd have better odds at any game in a casino.

And yet, hypocrite that I am, I have participated in this process for years and continue to do so to this day. I take some solace in the fact that my mechanism for interviewing candidates for software development jobs is to ask them to write code and then to review it with them, but even this I find to be marginally effective. The reason that I continue to do it is not because I think it's effective but because, if I refused on principle, someone else would do something to hire candidates, and what they did would probably be worse.

It's difficult to speculate as to why such an ineffective approach has remained the unquestioned best practice of hiring for so long, Google's efforts to chip away at it notwithstanding. Most people would probably contend that it has to do with cargo cult approaches wherein we as humans default to unquestioningly follow the status quo without thinking critically about it. And while I imagine there's an element of that, it's rather hard for me to imagine it ensorcelling the entirety of humanity for a century.

My hypothesis is that this is perpetuated as something of a subtle perk for corporate idealists, the only ones who appear to benefit from the process. As Laszlo Bock astutely pointed out, the main purpose of the brain teaser interview question is to make the interviewer feel

superior. I would extrapolate this to the *entire process* and expand the feeling of superiority to encompass an illusory sense of situational control and agency. Idealists have no real power from an organizational perspective, but controlling the interview process does a fairly good job of simulating power.

With the odd exception, the traditional corporate interview process is mainly a game in which corporate idealists create obstacle courses and force supplicant pragmatists to run them. It is, by and large, only pragmatists that are hired this way, and it is also, by and large, only idealists that conduct the process. Opportunists know that sitting on either side of the interview table is a bad deal. As interviewees, it's just a question of sticking to tradition (nice clothes, punctuality, not fidgeting) and hoping for good luck, and as interviewers, it's a fool's game. If they give the thumbs up to an awesome candidate, there's not much benefit, since it will be the hire herself that eventually receives accolades. On the flip side, a thumbs up to a candidate that flames out quickly and spectacularly will stick to the person who did the hiring. And worst of all, if they give too many thumbs downs for whatever reason, they start to be viewed as ineffectual leaders that can't attract and staff talent.

With opportunists avoiding the game altogether, they maneuver idealists into place so that they can act as proxies. There's no real organizational benefit, but the idealists don't see it that way, as they enjoy the superficial power and intrinsic ego-stroking. Opportunists are out searching for the real pathways to influence, while idealists are amusing themselves by forcing people to squirm and answer the same idiotic questions they were once forced to endure. Ah, how the tables have turned!

And the long suffering pragmatists? I'm about to get to them in a whole lot more detail. But broadly speaking, they simply, in the words of The Dude, abide.

Chapter 12: The Bad Economics of Pragmatism

Any pragmatist fortunate enough to make it through the interview process is inducted into the corporate world, as I was all those years ago, grateful to have an honest day's work that didn't involve anything resembling Amway. In a way, these new entry-level inductees are as much organizational stem cells as they are pragmatists since they have yet to choose whether to cede hope, perspective, or ethical high ground. They're too new. But that doesn't alter the fact that they're given a company button-down shirt, a mug, and a seat in the maze of cubicles with the pragmatists. You are the company you keep.

Interestingly enough, when you first start out at a company, everyone you encounter will tend to look like an idealist. After all, you're a new and untrusted commodity and only the most intensely checked-out pragmatists will risk appearing lazy or insubordinate in front of an untrusted commodity. The pragmatists all put on idealist masks for this occasion, and opportunists are always wearing masks anyway. So as a newbie, you come into a world of apparent unbridled optimism about the company.

But on a long enough timeline (and assuming you aren't an idealist), the pragmatists around you drop their guard and start to provide a glimpse into their world of moral victories, labor shortcuts and thinly veiled, familiar disdain for the company. They'll tell you knowingly that the boss tends to leave early on Fridays during the summer and that no one would be any the wiser if you did the same. They'll roll their eyes (as they should) in exaggerated fashion at you during the all-hands meeting when the company values are explained. They'll tell you, "off the record," that you can just submit the same status report each week because no one bothers to read them anyway. Since they cede hope, not perspective, they'll furnish you with exactly the sort of information you need if you want to minimize the

contribution you make to the company without jeopardizing your standing there.

In the movie *Office Space*, Peter got it slightly wrong. He described the pragmatist charter as working just hard enough not to get fired (or possibly hassled), but it's not actually that. The pragmatist shoots for security and predictability, so he works just hard enough to sustain the status quo without risk. The difference is subtle but important. It is indicative of a core characteristic of the pragmatist mentality: risk aversion. Working just hard enough not to get fired is a risky proposition because if you err just a bit, you get fired. If you work just hard enough not to get noticed for slacking, then erring a bit means a reprimand, followed by a relatively easy course correction.

Assuming that you avoid the idealist lunch table in the cafeteria (and you should really try to do that), you'll gain the trust of the pragmatists before too long. The barriers to entry aren't particularly high here. And you'll find yourself surrounded by people who define their lives almost entirely by valuations that are external and usually tangential to the company. They extract meaning from life elsewhere and signal that fact to others around them, sometimes with talismans as trite as mugs that say, "I'd rather be camping."

Think about the different pragmatists in corporate jobs you've held. There's always the "beer thirty" crowd that hits the pub pretty hard after work and commiserates together about hangovers the next morning. These people are signaling that partying is more important than what they do in the office. Then there are the "my family is my life" individuals with lots of photos, finger paint art, and other sentimental keepsakes thrown up in every corner, creating the illusion of a cubicle as a foxhole of family comforts. These folks are signaling that they're only doing the career thing to make that nuclear family bliss just one notch higher on the totem pole of awesome. Still other corporate pragmatists wear their hobbies on their sleeves, be they motorcycle riding, fishing, crochet, or music.

They're signaling that they were just a few bad breaks in life away from an entirely different (and superior) career.

The particulars don't matter, but the point does. Pragmatists have their real thing that they care about, and it isn't the job they're doing. This is an entirely rational means of ego salve, similar to a teenager making a big to-do over how he doesn't "believe in" the prom because of some philosophy or another that he's adopted. Getting dates isn't easy and the attempt may mean embarrassment, so it's a lot safer to create a choice narrative around not trying in the first place. Corporations assemble themselves into pyramid structures where advancement, like prom dates, is a zero sum game. It's a path of far less resistance to be the boozier, the family man, or the woman with the Harley collection than it is to be the aspiring CFO. Only the last item involves competition and the potential for failure.

This isn't to say that pragmatists don't *advance*. They do, sooner or later. But they do so in very limited fashion, being given titles in a predictable cadence and, perhaps eventually a low pressure, token managerial role with a random direct report or two. But that's about as far as you're going to get without doing the thing that corporate pragmatists refuse to do: marrying one's identity to the company. You're not going to wind up in the C-suite if the thing you're known as around the water cooler is "the karaoke guy."

In fact, think about the reputations of people around your office. At the line level of the corporate ocean, where the pragmatists slink about in the mud like catfish, you tend to remember people by their hobbies, interests, preferences—really, everything but their career ambitions. There are some exceptions to this, of course, but I would argue that these exceptions are actually idealists/opportunists-to-be. In other words, if you remember cubicle dwellers by their personalities or their achievements, they are probably idealists and opportunists, respectively, soon to be shunted up to their eventual resting spot within the company. But I'll talk in more detail about those archetypes soon enough.

Now, imagine the portrait of a corporate pragmatist. He's risk averse and aware that he needs a regular job to keep up with the mortgage, cars, and the expense of family. He works hard enough at a corporate gig not to make waves, but he doesn't work hard enough to make other kinds of waves. At office Christmas parties, he's known more for his hobbies, interests, and family than for anything he does at work. All of this is because he's ceded hope. There are people that are in charge and other people that later will be in charge, but the idea of fighting to make himself one of those people is prohibitively daunting. He'll consider it from time to time, sigh, pour himself a beer, and say something like, "You know, I don't live to work—I just work to live!"

This philosophy comes with a price tag. As a matter of fact, it comes with a dreadful price tag.

Knowledge-worker pragmatists tend to be salaried exempt employees, meaning that they work for a salary and not a variable hourly rate. And salaried exempt employment is an atrocious economic deal, especially for programmers. To put an exclamation point on it, let's go through some numbers.

For the sake of easy math, let's consider a pragmatist that is a representative senior software engineer in the United States making \$100,000 per year. If you've ever been a manager (or a contractor), you're probably aware that there are 2,080 work hours in a year. Let's drop those eighty hours off of the end and assume that they count as Christmas, Independence Day, etc., up to ten holidays per year. So that means that the pragmatist works 2,000 hours and receives \$100,000, or \$50 per hour. That's a pretty good wage.

But then, consider the fact that his labor on the freelance market can easily bill out at \$150 per hour. I've seen this pay ratio play out in multiple locations with multiple vendors. When I ran a department, I routinely solicited software services and saw a pretty standard set of rates come across. Bargain basement was \$100 and top-of-the-line specialized rate was \$200. The blended rate would average a senior

developer generating \$150 per hour in revenue for his or her company.

So what gives? Why does this large gap exist? Well, because of all of the expenses that an employer incurs on the worker's behalf, all of the perks of working for a company, and the stability, right? Okay. Fair enough.

But let's do some more math. Let's assume that our pragmatist gets four weeks of paid time off in some form or another—whether sick, vacation, or personal. At \$50 per hour, this is a benefit that's worth \$8,000. Further, let's assume that the employer pops for about \$12,000 in insurance benefits. We're now at a total compensation of \$120,000. Let's further add in \$2,000 for financing a retirement plan and another \$3,000 in miscellaneous perks for a total of \$125,000. Finally, let's add taxes and unemployment insurance on for a generous extra \$10,000 to bring the total to \$135,000 in total compensation. And then let's add another \$15,000 for, oh, say, tuition reimbursement, 401K match, HSA kick, and perhaps other exotic perks. We're now at an even \$150,000 for total comp, for the sake of easy math. And this is [almost certainly a wildly generous figure](#).

So the pragmatist takes home \$50 an hour and costs his employer \$75 an hour, total. His employer charges \$150 per hour, which means that half of the revenue he generates is spent on employing him and the other half goes into the company coffers to pay for expenses, investment, shareholder profits, overhead salary, etc. Of the employer's cut for his time, 25 percent of it goes to the expenses and perks and 75 percent goes to the company. You can think of this 75 percent, or \$75 per hour, as a "stability premium." Every hour that the pragmatist works, he's forking over \$75 for "stability," which means not having to pursue his own leads, handle his own finances, worry about legal representation, and the like.

In case this still sounds good, the economics get even worse.

Is stability worth \$75 per hour? That's a question that I cannot possibly answer for anyone but me since worth is in the eye of the beholder. But what I will say is that with a full year of \$75 per hour (or \$150,000 per year) in his pocket, a former pragmatist could hire a commission-based salesperson and an administrative assistant and still have money left over for incidentals (assuming he had the upfront capital to pay the workers before leads were generated). No doubt about it, though. There is real value and peace of mind in not having to worry about all that stuff. But still, compared to owning an enterprise and having a small staff, working for the man for the same pay is a vastly inferior economic situation.

And even that's not the worst of it. You see, there's another insidious characteristic of the corporate world, which is that forty-hour workweeks make about as much sense as laws that you can't buy alcohol on Sundays after 4:30 PM if you're wearing blue and Mercury is in retrograde. Don't get me wrong. I'm not saying that there's anything wrong with working forty hours in a week. But doesn't it seem odd that everyone everywhere works roughly the same amount of hours? There are strong societal incentives that start to kick in if you go too much over forty (bad reputation for companies as sweatshops) and if you go too much under forty (now you're a part-time employee and don't get substantial medical and other benefits). We're funneled toward the forty-hour mark like cattle being gently prodded into a single-file line.

Now, it's not the forty-hour workweek that's the bad part here. It's the perverse incentives created by the forty-hour workweek. Let's say that Fred, a senior software engineer and pragmatist, vacates his position where he was making \$100,000 per year. The company puts you in as his backfill, for the same salary. Further, let's say that you're way more efficient than Fred was. Within a few months, you're delivering twice the value to the business. So, assuming Fred was paid \$50 an hour and generating \$150 per hour for the company, you're paradropped in and are paid the same \$50 per hour to generate *\$300 per hour* for the company. That's awesome! You rub your hands together excitedly as you prepare to receive your reward.

And you know what that reward is? If it were just, it would probably be to have your pay doubled or at least increased by a modest 50 percent. Otherwise, you should be able to keep the same pay and work the twenty hours per week it takes you to do Fred's job. I mean, that's what's rational, economically.

I won't hold you in suspense any longer. Drum roll please. The reward is...

Well, it's a hearty pat on the back, an "attaboy, keep up the good work," and a 5 percent cost of living adjustment (COLA) instead of a 3 percent one in twelve months, at your annual review. At your \$100,000 salary, that means that you get an extra \$2,000 per year, which totals out to \$1 per hour. And that'll start in a year, minimum, rather than when you start providing the value.

So you make your company an extra \$150 an hour by being awesome, and they toss you a buck. And the next year, they toss you another. Then, maybe in year three as an overachiever, you're "ready" for a promotion. They bump your pay by \$10,000 annually, bringing you up to a total increase, over the course of four years, of \$7 an hour. In your time at this company, you've earned them an extra \$1,200,000. They've responded by letting you keep \$20,000 of it.

Think about what this means. The difference between being an efficiency machine for your employer and for being Fred is \$20,000 spread over four years, which translates to \$2.50 per hour. Now, remember that at this point forty hours a week is a fixed, non-negotiable, sacred figure. You, like any pragmatist, have to be present and looking busy for forty hours per week. So your choices, as an efficiency machine, boil down to "collect \$50 per hour to look busy but coast and duck out early when no one is looking" or "collect \$52.50 per hour to put the pedal to the floor and give your all."

The perverse incentive is that looking busy is far more important to your career than adding value.

At this point, it bears mentioning that your employer isn't screwing you. It's playing by the standard corporate rules. I mean, think about it. What company is going to say, "You know what, let's start paying all of our devs \$250,000 a year?" If they were publicly traded, the shareholders would riot. These are the rules by which individuals and corporations play and pay. It's just that the rules are such that non-ownership employees create gobs and gobs of surplus value they don't get back.

And that brings us back, full circle, to the reason that I call this archetype "pragmatists." While it's true that they've ceded hope to the organization, they haven't ceded their sanity. They probably don't fully grasp just how bad their deal with the company is, but they do understand intuitively that it's a bad deal and that the system heavily favors other players. They also understand the sharply diminishing returns of working harder for a few extra dollars per hour. They're entirely rational to want to put up the minimum effort required not to be noticed since that effort gets them \$50 per hour compared to \$52.50 per hour for a lot more work.

So they'll talk about sports instead of working. They'll miss no opportunity to show photos of their children. They'll come in a little late and hungover when the boss isn't looking. And they'll put on a happy, idealist face when new people are hired. The pragmatists will go along to get along, recognizing that, while their economic arrangement with the company is a horrible deal, it could be worse.

Speaking of worse economic deals, let's talk in depth about the idealist.

Chapter 13: The Worse Economics of Idealism

I've heard it said that if you sit at a poker table and can't spot the sucker within ten minutes, then you *are* the sucker. The same is true of the idealist archetype. Recall that they cede perspective to the company, and this cession comes with a pair of glasses that makes everyone appear to be fellow idealists. But the idealist goggles hide a lot more than just the motivations of other players around the office.

If the company were a church, pragmatists would be the ones there out of obligation, listening to the NFL pre-game on a surreptitious pair of headphones whenever possible. Idealists are the true believers: present, pious, and engaged. To be an idealist is not only to remain enthusiastic about the company but also to believe in its canon, mythology, and cultural norms. And, above it all, it requires believing in the company as one's career salvation.

At any company, you'll find a culture. But don't go looking for it on the "company culture" page of its website. Beer Friday, company paintball outings, and goofy hat day aren't culture—they're a marketing flier made three dimensional and brought to life. Ditto for the company's "values," if your organization has enough bureaucracy that someone's been tasked with defining them. These only answer the question, "What would make us sound good on a quarterly report?"

A company's real culture consists of its pecking order, the stories long-tenured folks tell, the company-specific jargon, and the approach to making money and solving problems. And it's by their investment and belief in this culture that you can identify idealists. Everyone will participate to some degree or another, but idealists will relish participation and judge mutual status by it.

The reason for this is simple. Idealists fuse their identities with the company's identity. I'll discuss this more in a bit, but suffice it to say that this fusion means idealists are competing with one another to be the top guru of the company's culture. This, they believe, is the way to demonstrate merit and to advance. It's also the essence of conceding perspective, since knowing all of the company's lore and using all of the company's jargon contribute not a thing to the company's bottom line nor to the idealist's prospects. It proves loyalty rather than value.

Toward this end, idealists establish a currency of sorts that can be thought of loosely in units of company culture. They put in long hours, answer midnight phone calls, go to company events outside of work, and practice using company jargon with the goal of earning units of this currency and the things that it buys, such as perks like a better parking space or first choice of donuts at the weekly accounts meeting. Part of losing perspective is being willing to delay gratification and make irrational sacrifices to obtain units of this currency, which is completely worthless in the real world. And anyone focused on obtaining this currency tends to assume that everyone else is similarly focused, which makes idealists inclined to assume everyone else is like them. This leads them to make awful strategic and economic decisions as they endlessly chase the seniority dragon.

Let's look at two possible paths through the company: that of the pragmatist knowledge worker and that of the idealist one. There are 2,080 working hours in a year and forty working years in a career for a grand total of 83,200 career hours. Let's assume that the pragmatist knowledge worker averages \$100,000 per year over the course of his career for a total of \$4 million in earnings, while his idealist counterpart averages \$120,000 for a total of \$4.8 million in earnings. Just to recap here, that means that there is an *average* of \$20,000 in difference per year, which, for two people that start at identical comps, is *huge*, given that it will probably be three years before either one of them sees anything more than a COLA. And, on top of that, pragmatists are much more likely to secure better

salaries by job-hopping while idealists stick around for decades, taking whatever obligatory five-year raises the company deigns to dole out. A pragmatist can match an idealist's salary ten years into their careers by jumping jobs twice.

And all of that extra pay later in the career doesn't come cheap for the idealist. It's a life of midnight emails followed by 6:00 AM touch-point meetings. The idealist misses children's first steps because he's on a conference call with the Dubai people that's really critical for moving the company's overseas presence to the next level. He got a fat \$1,000 bonus check when that deal was sealed! Over the course of his career, the idealist pumps in sixty hours per week, whereas his "slacker" counterpart gets by with a mere forty. The idealist assumes that this is a symbol of his dominance over the slacker. The reward is a corner office, fancier perks, and the right to give orders and expect to be obeyed. Of course, that's the least the company can do, since the idealist actually works 124,800 hours to the 83,200 hours for the pragmatist.

The real cost of those middle manager perks comes into true perspective, though, when you consider that the pragmatist earns \$48.08 per hour for his career. Meanwhile the idealist, with the nicer car, better office, and org chart authority earns \$38.46 per hour. So, pragmatists reading this, pity your pointy-haired boss in his Lexus. He makes less than you do in real wages.

Yes. Adjusted for hours, a rationally checked-out pragmatist actually earns a higher wage than a harried, hard-charging idealist working his way into the thick of middle management. Sure, a snapshot later in their careers has the idealist making more, and, by the end, probably even making more after all of the unpaid overtime. But after how many years of taking a ridiculous haircut? He never breaks even.

Why do idealists work such long hours? It isn't because they sit down one day and think, "Gee, if I put in sixty hours per week, I'll eventually get a slightly larger pay increase." In fact, it's precisely because they *don't* think that. It's not really about hours. It's

nominally about proving their dedication to the company's cause. But it's really about proving their prowess within a constrained, artificial environment.

Have you ever gone to a carnival where they sell you nine tickets for \$15, and everything costs four or eight tickets? You literally can't use all of the tickets you buy unless you spend \$60 on thirty-six tickets. What's the benefit of using these tickets instead of actual currency? Absolutely, positively none. It's a complete racket put on by the carnies to sucker you while you're in a festive mood. They introduce and value a currency that's worthless anywhere but at the carnival. There's nothing even remotely impressive about having loads of leftover carnival cash.

Let's switch gears for just a second now and do a thought exercise. What if I offered you a job for \$100,000 per year? But wait, I'm not done. What if I offered you that same job, but I told you that instead of a cubicle, you'd get an office? And what if I told you that I'd add "senior" or "principal" to your title? What if I told you that you'd be on the meeting invite for a lot of meetings involving the CTO and all of the most sought-after people in Outlook? What if I told you that you could sit in on interviews, participate in performance reviews for junior employees, and strut around like a boss? What if I even threw in a gold watch or a company ring after five years of service? Pretty sweet, right? Exactly. And that's why I'm now offering you only \$66,000 per year for it. Sure, it may be a 33 percent pay reduction from the original offer, but did I mention that you get to sit in a room while you're at work? A room with a real, particle board door that you can totally close? That alone has to be worth like forty grand, right?

You'd tell me that I was absolutely, completely insane if I interviewed you and wrote you an offer like this. You'd go onto Glassdoor and post such a scathing review that it would make their servers explode. You would be insulted to the absolute inner core. And yet, if I offered you the job and then made you this same offer, slowly, over the course of a decade, you'd thank me, call me sir, and ask for another. And that, my friend, is because you've been inducted over the

course of that time into the cult of seniority and anointed as a card-carrying idealist.

The modern corporate structure robs idealists of perspective. It introduces a bubble culture and funnels their natural competitiveness into zero sum games for worthless prizes while opportunists quietly brush past, looking for actual items of value. So what's the danger to the entry-level knowledge worker in hitching to a company? It's not that she'll put in extra effort without compensation, which can be a rational, medium-term play. It's that she'll get distracted by the lights, noises, and fun rides at Pleasure Island and begin to hoard carnival cash without realizing it. It's that she'll blink and ten years will have expired. Her market worth will have soared far above her pay while she's collected offices, token titles, meeting invites, and other baubles that have no value outside of the carnival. And, worse still, she'll have minimal leverage with which to go looking for competitive offers, since she's traded a whole ton of value on the market for carnival cash. Being the resident expert on Acme Inc.'s weird internal SAP installation may net a lot of water cooler cred at Acme Inc. But Beta LLC hiring managers are going to raise a skeptical eyebrow and say, "And that helps me how?"

Perhaps the most depressing part of all of this is that the ruling opportunists understand how self-limiting and non-strategic the idealist career path is. When looking for their next CFO, they're not looking for people that get comically over-competitive and dump \$200 into the fast-pitch game in a futile effort to win a \$4 inflatable bat. That's not at all strategic. You can keep that person around, even through frustration, simply by offering progressively bigger inflatable bats. No CEO will take a middle manager that works at a 33 percent discount in exchange for essentially nothing and promote him into a leadership position to negotiate key deals. All he'd know how to do is discount merchandise and services below cost until the opportunist across from him finally took pity on him.

And so the idealist manages to squander an entire career, toiling away for a company that views him working hard to level up as a

sign that he should never level up. He slams up against a glass ceiling that applies only to those who play by the company's rules—a glass ceiling that prevents him from earning like the big boys while forcing him to work way harder than the little guys reporting to him, and for not a whole lot more pay. There may be a moment late in the idealist's career when he doubts that he got a fair shake. But by the time this happens, it's far too little far too late, and the temporary cognitive dissonance is utterly crushed by the opportunity cost of a wasted life. Sure, it was fair. He just didn't work quite hard enough or excel quite well enough to get to the top.

It never occurs to him that breaking the rules set forth by the company was ever an option. And, speaking of rule-breaking, let's talk about the opportunist.

Chapter 14: The Lonely Profit of Opportunism

Pragmatists concede hope and idealists concede perspective. Opportunists concede neither of these things, which creates initial cognitive dissonance. New worker bees destined for pragmatism learn and accept that there's nothing they can do to reach the halls of power. New worker bees destined for idealism drink deep of the company kool-aid and assume that the company will take care of them if they just prove their loyalty through clueless overperformance. The common thread between the pragmatists and idealists is the acceptance that their fate is in someone else's hands.

Upon entering the workforce and being assaulted by a unified front of idealist facade, the natural thing for fresh faced, entry-level folks to assume is that the company marketing material is genuine—Introde is, in fact, a pyramid-shaped meritocracy! Each level of the pyramid hosts a group of people that scored better on some magical “awesomeness test,” administered on an ongoing basis by the company. The assumption is magnified at companies known in the media for being great to work at. You accumulate carnival cash from day one at a company. If you never stop believing in the “awesomeness test” and never stop trying to ace it, you pass some point of no return. You become an idealist. When you figure out that (said in *The Matrix*'s spoon kid voice) “There is no awesomeness test,” the only decision left is whether you wind up an opportunist or a pragmatist.

The opportunist-pragmatist decision is, perhaps, the most important one that can be made for a career. Furthermore, it makes up the core of this book. Consider the beautiful words of Shakespeare through Macbeth.

To-morrow, and to-morrow, and to-morrow,

Creeps in this petty pace from day to day
To the last syllable of recorded time,
And all our yesterdays have lighted fools
The way to dusty death. Out, out, brief candle!
Life's but a walking shadow, a poor player
That struts and frets his hour upon the stage
And then is heard no more: it is a tale
Told by an idiot, full of sound and fury,
Signifying nothing.

This nihilistic soliloquy is Macbeth's reaction to the death of his wife, but it could also be the words uttered by a pragmatist becoming an opportunist and reflecting on his employer. Idealists have the simplest corporate belief structure—they believe in the company as some sort of custodial institution that is more than the sum of its parts. Pragmatists don't believe in the company, per se, but they do believe in the strength of an opportunist for creating corporate order. Whether they think she's a visionary, an incompetent scion, or a tyrant, pragmatists believe that the CEO possesses some kind of intrinsic quality that they lack. It's this quality that allows her to become CEO. Opportunists recognize that neither belief is true.

Becoming an opportunist is to understand that the org chart and what it represents is "but a walking shadow." There's really no order or meaning behind it all. To put it more plainly, an opportunist *becomes* an opportunist the day he figures out that, at the core of things, no one in any position of power or authority at the company really knows what they're doing any better than he does. I'm not saying that an entry-level kid knows finance as well as the CFO—trade or tactical knowledge isn't what I mean by "know what they're doing." Rather, I'm talking about the gestalt of business strategy and decision making. Opportunists realize that there's no playbook and that everyone is just winging it behind a carefully controlled facade.

They recognize that the main fiction of a company is one of fairness and order, neither of which is ever actually present.

Once this realization sinks in, the budding opportunist develops an initial contempt for the implied rules of corporate structure and then, eventually, an indifference toward them. A traditional path of advancement through an organization might be engineer I through engineer V, followed by lead engineer, and eventually engineering manager, VP of engineering, CTO, and CEO. This was the case at my first company.

At five years per engineer numbers I–V, I could expect a lead engineer role by the age of fifty. Then maybe I'd be an engineering manager by sixty, VP by seventy, CTO by eighty, and CEO by ninety. A pragmatist looks at this and says, "I'll never be CEO—you probably have to kiss a lot of backside or be someone's brother to get that job." An idealist looks at it and says, "I bet if I put in sixty hours a week and memorize the company handbook, I'll be engineering manager by forty-five and CEO by sixty-two!" An opportunist looks at this and says, "I'm not going to waste my time getting good at being an engineer, since that's not going to pay off for twenty years or so. Instead, I'll spend my time at the office studying how the current CTO and CEO managed to skip the promotion conveyor belt." This newly minted opportunist is only interested in playing games that she can win.

To understand the mindset of an opportunist, consider an example I once offered in my blog at daedtech.com. I advised software developers to file a "doing business as" (DBA), which essentially means that they create a corporate entity, legally. Instead of doing business as Jane Smith, sole proprietor, Jane can do business as "JaneSoft." I then advised these same developers to spend \$120 per month hiring a virtual assistant (VA) firm to help them perform various tasks around setting up the infrastructure necessary for a software consultancy (a website, bookkeeping, etc.). Finally, I advised them to add "Managing Director, JaneSoft" to their resumes and list management of a global team as part of their day-to-day

responsibilities. Having done this, a software developer has a pretty compelling case to ask her boss for additional responsibilities or a leadership position. If the boss doesn't buy it, well, the developer can start interviewing for management positions elsewhere with a year of management experience at the top of her resume.

If you read this and thought, "You clever bastard—I admire you, but I wouldn't do something like that," you're a pragmatist. If you read it and thought, "CHEATER," you're an idealist. If you read it and started taking notes, you're an opportunist. There is a rich implied set of rules that govern corporate interactions and advancement, and what I've just described is a violation of them. It's *unfair*. And opportunists don't care. They do it anyway. Skirting the rules is the only way to get to the top before you're ninety.

This is smart, and it's effective for advancement. It's also lonely. Group rules, norms, and ethics aren't just about preserving order. Whether we think of them this way or not, they're also about binding us together with a common set of principles and beliefs. Rejecting those beliefs, even covertly, is an inherently isolating activity.

Pragmatists band together in social groups, recognizing that none of them will ever walk the hallways to power. They are relatively unified in their feeling of outside-ness when it comes to the corporate vision. They may not be invited to the meetings and gatherings where important people make important decisions. But they're all left out together, much like a gathering of hippies who couldn't afford the concert and make the best of it by sitting outside together and getting inebriated. And, generally, they're all right with that to boot. Pragmatists aren't looking to make waves via big decisions and responsibility; they're happy to leave that to others.

Idealists compete with one another (and, really, everyone, since idealists believe almost all others to be idealists). But they're steeped in the shared culture of the company. Their competition, while fierce, will often have an air of "may the best competitor win." They're not unlike a high school football team with a deep, abiding respect for their coach. They all want to start and will compete intensely for

individual glory, but, come game time, they'll trust in the coach-enforced, meritocratic decision making, take a knee, and bring it in to the circle, giving a giant cheer for the team. After all, the team is bigger than any individual.

Opportunists aren't hippies, making the best of their situations with "misery loves company" social gatherings. Nor are they competitive jocks willing to put aside ego-driven tiffs for the collective mission. They're lone wolves and iconoclasts, though they may be morally good, bad, or neutral. They step outside the cultural and even ethical norms of the corporations that they inhabit and move about, usually upward, unencumbered.

However, striding toward the tops of organizations (or founding their own) requires a heavy social toll that the other groups don't have to pay. Bands of pragmatists can easily stay in the same place, working side by side for years or decades, forming deep and lasting social relationships. Idealists have each other and they have the company, which serves as their social life. If they're not busy inventing the company culture, they're slamming in mountains of overtime for the nominal promotions and seniority that will allow them to make up the buzzwords and establish the traditions that define the company culture. That kool-aid guzzling and carnival cash acquisition requires a lot of dedication and human interaction.

Opportunists do neither of these things, and they carry attachment loosely. They bank on earning promotions that make their peers become instead their reports—and suddenly. They'll leap to another company if it presents a situational advantage. They'll remain aloof from those around them if it creates the impression that they're a force to be reckoned with. And they'll pretty much always do things that strain the ethos of the company culture, resulting in loneliness and sleepless nights—at least for a time.

Eventually, the opportunists slip their way through invisible cracks in the glass ceiling imposed on idealists and wind up in positions of power. To the pragmatists, it may seem that they're charmed, preternaturally competent, or lucky. To the idealists with their

illusions of the infallibility of the corporate culture, opportunists have, by definition, earned it. To the opportunists themselves, believe it or not, it just seemed inevitable the whole time. Once in power, they bear the burden of pandering to both the pragmatists and idealists, albeit in different ways.

In a moralistic sense, opportunists with a conscience may well make a deeper sacrifice than pragmatists or idealists, counterintuitive as it may sound. On the surface, both of those latter archetypes give up money and power. Beyond that, pragmatists give up hope and the notion that they can truly enjoy their work. Idealists give up perspective and, frankly, a bit of their dignity, though they probably don't see it that way. But opportunists might just give up a bit of their souls. They slice a hole in the social fabric that governs the rest of the players so that they can get to a position of controlling that fabric, and then they hypocritically sew it back up so that the show can go on, full of sound and fury.

It's the opportunists that most truly understand the pathological nature of organizations, and it's likewise the opportunists that perpetuate it. I firmly believe the reason for this is that the nature of the corporate game itself is negative sum. Think of it as Russian roulette. Pragmatists resign themselves to the cruel whims of it. Idealists scream, thump their chests, and embrace it. Opportunists remove the bullets that correspond to their turn from the chamber so that they'll control the game and win. But all of them are the worse for playing.

So as I wrap up the detailed discussion of the players in the corporate hierarchy, I will again ask that you think of them not in terms of their flaws but rather in terms of what is done to them as they pursue, largely in good faith, their goals.

Chapter 15: Faux Ownership—Managers and Owners

The last few chapters have offered a detailed look into the politics that drive garden variety organizations in the corporate world. In fact, you are no doubt picturing a company at which you've worked, categorizing the people that you know in it. Does this company have 1,000 employees? 10,000? 100?

It could be any of those, but I'll also bet that you're picturing an *established* company. After all, I've offered a snapshot in time of a company that's existed long enough to have a "culture" but hasn't existed long enough that the culture is "panic because we're circling the drain." To put a finer point on it, I haven't really talked about the corporate life cycle, offering instead a portrait of the company as an adult near its prime.

To remedy that, let's talk about the birth and evolution of a company. (I won't talk about the end of a company because it's not especially interesting—there's simply a tipping point where the opportunists bail and the company briefly staggers along aimlessly, like a headless chicken in idealist defiance of its imminent death.)

In the beginning, there is an opportunist. Putting aside my archetype jargon for a minute, that statement actually functions well in plain old English. The kind of person that would start a company is an opportunist. But it also applies here in our domain as well.

Pragmatists wouldn't start a company. They've ceded hope, and starting a business is not something that hopeless people do. Idealists wouldn't start a company because they don't have the perspective to realize that they could enjoy success. To them, the path to the top is to hitch on with an existing organization, drink its kool-aid, and prove themselves as understudies.

And so it's left to opportunists to start organizations rather than join existing ones. If you think about it, entrepreneurship is a mild example of cheating against the backdrop of the "steady paycheck" corporate world. You're not supposed to gamble the children's college funds on a business venture, and you're not supposed to hang out in your parents' garage until you're twenty-eight, playing with gadgets in the hopes that investors will come along. It's a bit more acceptable than hacking your way to the top from within a company, but it's still not congruent with "settle down, get a reliable job, and buy a house."

In the early going, the company is a land of opportunists. Each subsequent equity partner or participant they bring on is another opportunist. The reason for this is that exchanging labor for equity is essentially tolerating risk in exchange for explicit power. They're betting on themselves.

When the company hires someone in exchange for salary or contract pay, it has acquired its first pragmatist. There's an interesting implied dialog here, with the owners/partners saying, "we don't trust you enough to offer you power" and with the pragmatist saying, "keep your pie-in-the-sky equity and give me a steady paycheck." This is the essence of the pragmatist condition. He doesn't really believe in what he and the company are doing and he doesn't believe he's the kind of person that could parlay equity into a huge payday. What he believes in is a paycheck and going home at five o'clock.

For a time, the company undergoes an accretion process with the opportunist-pragmatist binary alone. In this phase, it almost exclusively adds pragmatists, though it's not out of the question that it would pick up another partner or two along the way. This is sustainable for a time. But a knee point is coming, at which time the first idealist will be minted.

A company can get by with pragmatists and opportunists for as long as the opportunists can divide the workforce into segments that they can and are willing to manage. The technical co-founder and CTO will manage all the technical pragmatists, the COO co-founder will

manage the operations people, and the CEO co-founder will manage the sales and marketing people. Or whatever. The particulars don't matter—just note that the opportunists are divvying up management.

But then something happens. The organization gets too big for the co-founder oligarchy model to be practical. Or maybe the co-founders don't want to manage people directly. Perhaps they just want to reward an early pragmatist hire for loyalty or for performance. Whatever the catalyst, the owning opportunists pick a line-level pragmatist who, while not considered worthy of partnership, they feel should have some elevated status nonetheless. This is the original company idealist.

This is also the inaugural moment for seniority in the company and, in a very real way, the establishment of what company culture will be. Sure, there's a lot of popular mythos about startup culture and what color hoodie the technical co-founder wears and whatnot, but that's largely superficial. Owners exist mostly outside of corporate cultures. Their fashion choices, hobbies, and personality quirks are only important to aspiring internal power brokers looking to mimic and appease these figures.

But here, in the appointment of the first non-owning manager, the founders have defined what it takes to advance within this nascent company. The most common thing to do is to reward loyalty, like a superstar athlete giving a manager job to a childhood friend. The longest tenured non-owner is similarly promoted to “manager,” and the standard corporate narrative for idealists is reaffirmed. Stick around long enough, and you'll get the leftovers from those above you.

You may occasionally see such a promotion awarded to the person from the group that has done the best work or else based on an assessment of who would be the best leader, but that's not terribly common. In the first place, the entrepreneur-opportunists starting the company are probably not folks with a lot of practice making these sorts of personnel evaluations. Secondly, seniority via loyalty and tenure is the standard corporate narrative, so a founder figuring

things out on the fly is likely to pursue this strategy. And, finally, recall from the Gervais principle that opportunists promote overperforming pragmatists for their own reasons. In the beginning, these “own reasons” may be simple vanity; the opportunists want to publicly reward those who kept the faith in them while others doubted.

But let’s now take a look at what’s happened within this company, its culture having just been defined by the enshrinement of idealist number one. Up until yesterday, owner-opportunists and employee pragmatists toiled away cheek by jowl to grow the emerging company. Today, a third archetype exists within the company. Let’s forget the pragmatist-idealist-opportunist categorization for the time being and redefine it in terms of actual legal power.

Yesterday, we had owners and grunts. Today, we have owners, grunts, and a new thing: managers. Only owners have any legal power as far as the company is concerned, which makes managers and grunts the same thing. The main difference is that, at the pleasure of the owners with the real power, managers can tell grunts what to do.

Consider the word “manager” in non-corporate contexts. Actors and athletes hire managers and delegate non-domain tasks to them. The actors will act, the athletes will play, and the managers will deal with mundane meta-concerns that don’t interest the talent. In your own life, you may actually hire or pay a manager at some point. If you hire an accountant for more than just income taxes, she is acting in a capacity of managing your finances.

Managers take care of details with which their employers cannot be bothered. Owner-opportunists employ people managers when they can no longer be bothered with the grunt-pragmatists.

In a corporate situation, ultimate power lies with the owners (which makes publicly held corporations uniquely interesting in some ways). Non-owner opportunists in the C-suite also possess a tangible form of power in that they can sign contracts and generally act with the full authority of the company behind them. But below that, the only thing

that prevents a low-power, anarcho-capitalist free-for-all is the order imposed by those at the top. And this order is imposed in the form of a pyramid-shaped chain of command reminiscent of a military.

Owners have power. Real power. And the reason I say this is that owners have created situations in which their income is separated from any form of labor, however white collar. If you have enough money, you can buy a restaurant and become the owner. You don't bus tables. You don't cook. You don't even handle personnel management. You hire a *manager* to do the latter and hire other people to do the former. All you do is come in from time to time and bask in how everyone cleans up and looks sharp just for you. And, in spite of that being your only contribution to the enterprise, you make money.

When you're a (successful) owner, your income is on autopilot, which allows you to live the apparently glamorous life for which most pine. You've got money flowing in, so you can travel when you want, get up when you want, eat what you want, and generally do what you want. And whenever you frequent places where your ownership matters, you get the red carpet treatment. You give people instructions, run up lavish bills on a corporate credit card, speak to captive audiences, and carry the boss aura everywhere you go.

If you're an entrepreneur-opportunist, the place where your ownership matters is at your office. So what do you do on the day the enterprise grows too big for a single, flat level hierarchy? You promote someone into a leadership position. You pick the most loyal and hard-working true believer in the company, and you hand him a placard that says "manager." And what do you do to reward him? You're not going to give him an ownership cut, and you're not going to pay him a whole lot more. So you give him what the corporate world has standardized. You give him faux ownership.

You appoint him to a position and tell everyone else in the company that he is your proxy and he speaks with your voice. You grant him power and a mandate. Now he gives people instructions, runs up lavish bills on a corporate credit card, speaks to captive audiences,

and carries the boss aura wherever he goes...as long as you're not there. You grant him the gift of vicarious ownership.

So many of the trappings of working one's way up the corporate ladder are laced with the culture of faux ownership. It's no accident that managers and vice presidents with long tenure are the ones that get the first class plane tickets, corporate credit cards, and the best hotel rooms. Are things like that really necessary for the business to operate efficiently? No, of course not. They're just built-in incentives for idealists to offer their services in perpetuity for the company.

In a lot of contexts, they get to mimic the power and influence of owners. But being a faux owner is not, in and of itself, a terrible deal. First class is nice, caviar is tasty, and the Caribbean is lovely during the board meeting's time of year. Most of the time, a faux owner can temporarily forget that he has access to these things only at the pleasure of a *real* owner.

I'll conclude this chapter by citing one more interesting property of opportunists, as opposed to idealists. Idealists are content as faux owners. Opportunists reject faux ownership as an appealing end and use it as an opportunity to continue a never-ending charge toward real ownership.

Chapter 16: The Cult of Hours

In the last chapter, I drew a distinction between real ownership and faux ownership. If one were to ask, “ownership of what?” the answer seems pretty obvious: ownership of the company. But what exactly does that mean? You might own stock in Target and thus be an owner of Target, but something tells me that they don’t hand you a glass of champagne every time you walk into the local store. Indeed, it’s theoretically possible that your handful of shares of Target would constitute more of an ownership share than that of a CEO they just hired, but, unlike that person, the private jet probably won’t show up at your beck and call.

It may seem straightforward, but ownership is a tricky concept. In the context of a nascent company, ownership and power are uncomplicated and largely the same thing. But context is key because ownership does not exist in a vacuum. Whether your ownership matters hinges on the volume of what you own and the context in which you own it.

For instance, your ownership stake in Target is minuscule and common enough that no one cares. But if you took that same investment that you have in Target stock and used it as seed capital for a venture in which you hired a full time assistant, your ownership stake would be immensely important to that assistant. In that instance, you closely share a context with the assistant, and your ownership is absolute.

It might suffice to say that, with respect to your little company and your assistant, you own the means of production. You pay the pragmatist assistant a wage that doesn’t vary, and any profits from the company go to you. You own all of the company’s assets—and its liabilities, too. And, unpopular as it may be to say to an audience comprised mostly of full-time-wage earners, you own the assistant.

At least for forty (okay, who are we kidding, fifty) hours per week you do.

This assertion might have your hackles up, but can you really say that it doesn't feel true to you? Picture your job and the interplay that you have with folks there for a moment, and ask yourself what you'd do if your boss told you to do something that you didn't want to do—something perfectly legal and ethical, but undesirable. You might bargain or even argue, but if she insisted, you'd probably give in. And your boss is probably just a faux owner—imagine if the actual owner (or shareholding CEO) of the company approached you and told you to do something. I'm guessing you'd kick up even less fuss than you would with the boss.

Now maybe this isn't strictly true of you, and maybe you even believe that it isn't true of many people. But then ask yourself why there are so many laws to protect people that are whistleblowers and victims of harassment or discrimination. If the corporate environment isn't one in which power is so complete as to constitute *de facto* ownership, why is it necessary to have all manner of laws and support groups to prevent people from being coerced into doing things that go against every fiber of their will and being? Is it really voluntary when you're asked to do something if there's a message, always lurking beneath a silk veil, saying, "and if you don't, we can remove your ability to pay your mortgage and medical bills."

Forty-hour-per-week employment is a completely risk-maximized, non-diversified arrangement. When it comes to investing, you're encouraged to spread your assets across a whole host of industries, companies, and countries. But when it comes to earning a wage, you're encouraged to put all of your eggs in the basket of your current employer and loyally do anything that they tell you to do.

They own you. Or, at least, they own you for forty hours a week, until you can surreptitiously land yourself an offer at another company... who will then own you. Why is this the way it works?

Let's take a cartoonish diversion to understand this a bit better. Imagine that it's back in ancient times. I earn my living digging moats and ditches as a sanitary measure. As I get older and sorer, I decide to start hiring help, and some local youths agree to work for me. Each day, they show up and dig, and I pay them in an ancient currency called shells. Except, every now and then, torrential rains turn everything to mud and prevent the work from being done. And when this happens, a couple of the youths show up anyway, protesting that they need the money.

I'm not entirely unsympathetic, but I'm also not a philanthropist. They should be planning for occasional rain days, but apparently they aren't. I'm not going to pay them for doing nothing, so instead I offer to pay them every day, rain or shine: fifteen shells per day. I *had* been paying them per cubic foot of digging, which tended to average twenty shells per day. But they don't seem to mind the pay cut. It works out better for them in the end, anyway, because the only thing that keeps them from blowing all of their money gambling on dinosaur races is the fact that I portion it out this way.

But then something starts to happen. I pay them fifteen shells per day for their labor, but then, on days they don't do anything, I also pay them fifteen shells. I start to feel like a sucker. So I announce that I'm no longer paying people to dig ditches. Rather, I'm paying them to work for me in general. When it's sunny, I pay them fifteen shells for a day of ditch digging. When it's raining, they stay inside and earn their fifteen shells cleaning and maintaining the tools.

I've switched from paying them for the market value of specific labor to paying them what amounts to a retainer for "do whatever I tell you." I now sort of own them. At least, I own them as long as they need the money.

This (obviously simplified) is how the ownership concept develops when it comes to labor. As a worker, you cease to offer your labor as a commodity. You instead offer yourself as a commodity in exchange for a dependable wage. "For fifteen shells per day, I will do whatever you need done."

This arrangement creates a certain opacity to the value of anything that the laborer does. The arrangement is no longer one in which an activity with clear value is completed for clear compensation. Digging ditches may be worth half a shell per cubic foot, but being a laborer of mine is worth fifteen shells a day, whether those days are spent digging ditches, sharpening shovels, fixing handles, or fetching me groceries. For fifteen shells a day, you do any and all of those things, so who really knows what any of them are actually worth? And, with a stable of ditch-diggers-turned-employees, it becomes very difficult for me to determine the value of any one of them to my enterprise.

The arrangement shift itself may seem subtle, but the impact is dramatic. In the cartoonish ancient world, I've gone from owning the ditch-digging contracts to owning *you*. In the real world, I've become an owner and your employer. Your labor doesn't have a value—you do. That value is expressed in tens of thousands of dollars per year, and it's measured mainly by whether or not you show up and whether or not I like you.

Remember, owners aren't paying their employees for completing specific tasks with obvious value. They're paying for laborers to show up and do whatever an owner or faux owner tells them to do. As organizations expand, any semblance of being able to measure the value of labor goes right out the window. Instead, evaluations take on the more nebulous form of the "performance review," which I'll cover shortly. Suffice it to say, this is not a review of the way a human performs a job. It's rather a review of the human himself.

One of the main contributing factors of this evaluation is *presence*. After all, the arrangement in the modern workplace is that you receive a wage for spending forty hours per week working. So it stands to reason that one of the most basic evaluation criteria is whether or not you're present for forty hours. Seems reasonable enough, but it's interestingly devoid of any concept of monetary value, apart from that of salary.

If one employee is paid \$100K per year and another \$50K per year, is it reasonable to assume that one creates twice as much revenue

for the company as the other? Of course not. More likely, one has been with the company a lot longer, logging many more dedicated hours than the other. One is probably a long-tenured, low-level manager-idealist and the other a relatively short-tenured, checked-out pragmatist. One has probably been kicking in sixty hour weeks for the last eighteen years, while the other has been contributing forty hours, if that, for two or three years. Which one makes more money for the company? This is, for all intents and purposes, unknowable. It's also irrelevant in the context of corporations. After all, who cares? Salary has nothing to do with value.

Instead, salary has everything to do with status, which, in turn, has everything to do with culture. And it is in the land of culture that idealists reign supreme. Idealists demonstrate their loyalty to the company, the faux owners, and the owners by reverently taking up the cause and sacrificing their own wants and needs at the altar of being a "team player." A big part of this is an hours-per-week arms race, in which competing idealists show who is more "all in" by being present in the office or generally available for more and more time.

I talked about how idealists, through their loss of perspective, become unproductively overcompetitive. But *why* does this misdirected energy lead so naturally to insane hours in the workplace? Well, it's because that's really the only way that companies know how to quickly measure employee contributions. It's all about presence.

How much of a worker's hourly activity during the day actually accomplishes anything worthwhile? Managers trade war stories about wall-to-wall meetings as badges of honor, in spite of the fact that endless meetings are almost universally derided as pointless. Line level employees spend time at the water cooler, the cafeteria, the bathroom, the smoking area, and just about anywhere else that allows them to snatch back a few minutes of their lives. People waste time, chat, read articles on the internet, instant message each other, and generally find any and all possible ways to do everything

but work when they're at work. But as long as they're in the office, it's still considered work.

The cult of hours is the modern corporate incarnation of the Protestant work ethic, a principle in which hard work and frugality are viewed as the soul's salvation. The general idea of the Protestant work ethic is, "If you're not enjoying yourself, you're doing good." In the corporate world, it translates to, "If you're not enjoying yourself, it must be good. And if you're at work, you're not enjoying yourself, so showing up is good."

In my career, I've shifted from working in offices at salaried jobs to working remotely as a free agent. This shift throws you into an almost existential ethical crisis. When you're at the office, just being there is enough justification for you to receive a wage. If you show up at nine, socialize over donuts for half an hour, go to a pointless two-hour department meeting at which you play phone games, head out to lunch, come back, work for half an hour, stop by Bill's cube for half an hour, go to another long meeting, check your email, and go home, you feel happy having done your corporate duty. If you're a freelance consultant working at home, that eight-hour day you'd cheerfully collect payment for is now half an hour of billable work.

While that's an exaggerated average case, going remote exposes just how little work is involved in the average corporate work day. But that's not the most ethically troubling part. The most ethically troubling part is asking yourself what you should do when, as a remote worker, you do more in two hours than you did as an office worker in eight. Do you bill for two hours only, effectively penalizing yourself for your efficiency? Or do you bill the company, who would be happy to pay it, for eight hours, penalizing them for the extreme waste of valuing presence above all?

There's no easy answer to this question. But in reality, it's a tragedy that it needs to be asked. We've created a corporate structure that separates people so far from the value of their work that the only reliable metric offered for value is, "Did you come to this building?"

And given that this is the corporate world's main guiding metric, is it any wonder that performance reviews are a complete waste of time?

Chapter 17: Performance Reviews and Advancement Theater

If you could read minds and were truly interested in distinguishing ascendant opportunists from idealists in the ranks of management, you'd find no better proving ground than administration of the corporate performance review. As they announced each share of marginal pennies from the COLA pool, the idealists would soberly evaluate who better exemplified the corporate value of "customer focus" while the opportunists would resent their role (and potentially that of the reviewee, as well) in the charade. But let's come back to the idealists and opportunists later. The real star of the performance review show is the person to whom it's done: the pragmatist.

I say this because once you advance to a certain place within a company, the performance review stops being a thing. I can tell you from my time in the C-suite that reviews of your performance no longer happen with MS Word templates and a five-tier grading scale reminiscent of school. They happen instead with subtle cues, bonus dollars, and shifting alliances. There may be companies that technically schedule performance review meetings for C-levels and upper-level managers, but I assure you they look nothing like the awkward exercise you experience as a line-level knowledge worker. Ironically, it's through these informal or even back-channel interactions that true advancement within a company occurs.

The line-level performance review has two ostensible purposes, as far as the reviewee is concerned. One is status adjustment. The other is pay adjustment. Meaningful promotion (e.g., into a role with direct reports) does not arise out of the performance review. Only incremental, titular advancement occurs this way. So in a very real sense, if you're sitting there being reviewed via the MS Word template, you've already failed the real performance review *ipso facto*. At least, you've failed if you're an opportunist. If you're a

pragmatist, victory is a 3 percent COLA instead of a 2 percent one. And if you're a pragmatist being groomed for idealism, victory is marks of "exceeds expectations" (because, while the pragmatist wants to maximize the ratio of dollars earned to hours worked, the budding idealist wants to maximize superior approval and pats on the head).

But let's forget the archetype distinctions and consider everyone being subject to the process as a relatively uniform pragmatist. I want to talk about how performance reviews actually work. To do that, pardon the employment of an admittedly infantilizing parable.

Imagine a cash-strapped father walking home from a long day at the office, wishing he could do more for his two children, Alice and Bob. As he nears his house, he stumbles upon a wad of small bills on the sidewalk, totaling \$20. He picks them up and decides to offer his children a rare treat.

But as he walks in the door, he has a thought. Rather than just dividing up the wad of singles evenly, why not take the opportunity to impart a life lesson to the children? The father asks the kids how their progress went on their homework. It turns out that Alice has gotten hers done promptly while Bob's only about a third of the way through his. So dad gives \$15 to Alice and \$5 to Bob, telling them that good things come to children who do their homework.

The next night, our protagonist walks home but, not surprisingly, finds no money waiting to be plucked from the sidewalk by a lucky passerby. He comes home to find that, having learned their lessons, both Bob and Alice have completed their next three nights' worth of homework. Dad, however, is in the unenviable position of having no way to reward this with \$45 per child, so he gets clever. He picks up each assignment, finds errors in it, and tells the children that there will be no homework bonuses this evening because their homework quality is not up to snuff.

If Dad is an opportunist, he goes to bed feeling guilty. If he's an idealist, he goes to bed feeling as though he's taught the children a

valuable lesson. The pragmatist children go to bed feeling as though it might not actually matter that much what happens with their homework and that money from Dad is just sort of random.

In the corporate world, the determining factor that drives everything is profits and losses. Does Dad come home with money to dole out or not? A company exceeding its expected performance will come home with a surplus to dole out, whereas an underperforming company will not. To anyone who has ever received a “We didn’t do well this past year, so no raises for anybody” memo, this isn’t especially surprising. But what ought to be surprising is that most performance reviews are determined more by the available budget for raises than by your performance, a la Dad with his homework critiques that had more to do with his wallet than with the children being reviewed.

This is not universally true, of course. Performance reviews are often a convenient means of creating a paper trail, mustering cause to terminate problem children. But for those who perform adequately, the company will figure out whether it can afford a raise or not, how much of one, and then create a performance review narrative that supports the financial decision.

If this sounds conspiracy-theory-ish, consider how hard it would be to actually pin this down for what it truly is. After all, if the company obliterates its expected numbers, it becomes pretty easy to support a narrative that everyone is awesome and deserves a promotion. Likewise, if the organization badly underperforms, it’s easy to say that everybody came up short so no advancement makes sense. The problem with this messaging is that it’s predicated upon the idea that company and individual performance are in lockstep, whereas the performance review, by its very existence, is predicated upon the notion that every laborer is a unique snowflake.

So which is it? Is the company’s performance a good proxy for an individual review, or does individual performance stand alone? Recall that the ownership culture of wage employment makes the value of an individual employee extremely difficult to evaluate.

Clearly the outcome of the performance review is not a reasonable expression of individual value. So why the charade about individual performance, and why not give out titular advances in lieu of pay increases?

The latter question is easier to answer. That has to do with HR payment matrices and lawsuit avoidance, which will be under discussion here soon enough. Suffice it to say that relatively insignificant titular distinctions, in office political terms, take on a whole lot of significance when lawyers and discrimination suits enter the fray. The question as to why have the performance charade, however, is more interesting—and the answer more depressing.

I said earlier that opportunists would tend to be resentful of the process while idealists would treat it with reverence. That's because opportunists figure out the real purpose behind the performance review while the idealists take the ostensible one at face value. To them, the purpose is entirely earnest in that it's an opportunity to dole out merits or demerits on behalf of the company. The pay is an incidental detail that people shouldn't worry too much about since the real prize, carnival cash, is dispensed generously at performance reviews to pragmatists headed for idealism. The idealists treat the performance review as the hallowed process by which corporate cultural worthiness is conferred.

The opportunists see reality. They see that budget decisions having little to do with individual, group, or department performance are made and that it's up to them to translate this into a narrative about who is or isn't a good, homework-completing little boy or girl. They understand that it'll be up to them to take, "Meh, we don't really have money to increase payroll for that group beyond COLA" and turn it into, "Gosh, you did some awesome work this year, Alice, but you just need to get a little better at 'business values' and 'corporate integrity' and I'm sure you'll earn that promotion next year!" They resent this even as they understand its necessity. It's necessary because the truth—"We don't really know if your individual performance adds value or not, and either way, it doesn't have much

to do with whether or not you get a raise”—would be demotivating enough to chase you to a company who wouldn’t make the absurd mistake of being honest about this.

Ascendant opportunists understand that line-level performance reviews are a farce, but they put on an idealist face and carry them out because there’s not really a viable alternative. If they’re lucky, they can at least get budget apportioned in a large enough chunk to reward the people in their purview they know to be better performers, even if actual value to the company is unknowable. Alice produces more widgets than Bob, so let’s at least get her a slightly bigger raise than Bob. Their idealist counterparts would base the decision instead on who they thought was more stoked on the corporate culture and on who logged more hours. And they’d feel like they were doing a good job for it.

In neither case is justice done, so to speak. Some opportunists will get creative to keep the people they find most valuable and do things like encourage these folks to technically quit and re-apply. I once had a manager help me secure a promotion he badly wanted to give me, but couldn’t finance, through such a scheme. But these types of shenanigans are politically expensive for an opportunist. The reviewee had better be a huge help to the opportunist in order to justify that.

It is because of this very dynamic—the nihilist reality of performance reviews—that modern knowledge workers such as programmers are better suited to job hop. When people leave the market and nestle into a company, their market value becomes strictly unknowable, creating a situation where advancement requires the stars to align with the company being successful, the company reinvesting in its people, the individual holding her own, and the individual being liked by her manager. If all that happens, she moves up at a pace with the market. If anything goes wrong in that mix, she’s better off re-entering the market, where her valuable is no longer unknowable (and is probably 5–15 percent more than what she’s currently being paid).

Chapter 18: Your Company Doesn't Care About You

There's a Latin phrase that captures well a distrust of authority: *Quis custodiet ipsos custodes?* Translated, it means, "Who will guard the guards themselves?" It goes to show that the desire for checks and balances dates back to antiquity.

I've talked at length about the corporate hierarchy and categorization of its denizens. I've also talked about the birth of companies and the evolution of their culture. But what I've omitted up to this point is a discussion of watchdog, overhead entities that emerge as a corporation grows. These include but are not limited to things such as human resources, legal departments, standards compliance efforts, external consultants, and more. These are the parts of the organization that are overhead, meaning they don't contribute to the bottom line, yet aren't *boss* overhead.

It's these institutions, we think, that prevent organizations from becoming cesspools of political infighting. After all, the legal department mandates that all employees watch videos about graft, bribery, improper behavior, and harassment. Human resources supplies a sympathetic ear to claims of impropriety on the part of a superior. And so, during the course of reading about a pyramid-shaped organization, you have probably been wondering where these sorts of institutions fit in. Are people in these roles idealists, pragmatists, or opportunists? And don't they offer recourse for those not in positions of power?

The answer to that latter question is simply, in practice, "No, not really." The answer to the former question, however, is a much more nuanced. After all, watchdog checks-and-balances roles are necessarily organizational overhead, a concern that generally means lower-level management and thus idealists. And yet the people that

occupy these roles typically have their own pyramid-shaped hierarchies through which they minister to the company. Their structure resembles a separate, smaller pyramid, with goals orthogonal to those of the company. If we were to take the example of a police department, you have the police department itself concerned with crime reduction in the general population and internal affairs departments concerned with crime reduction among the police themselves.

In some ways, the politics of these pure overhead concerns are remarkably similar to the rest of the organization. After all, a minimum-effort pragmatist isn't concerned with whether he's doing mindless data entry in the pursuit of organizational revenue or in the service of filling out and organizing signed performance reviews. Grunt pragmatist work is grunt pragmatist work in whatever form it takes. Similarly, overperformance can be a way to move up and hobnob with organizational idealists whether one is managing a team of software developers or a team of compliance auditors.

But that changes for organizational opportunists, who, in these roles, have a much more bitter pill to swallow than their counterparts in organizational profit centers. The idealists in checks-and-balances roles lack organizational perspective, allowing them to believe wholeheartedly in their cause of making the company a good and just place to work. Pragmatists may take some small satisfaction from it, though they probably don't care. The opportunists, however, understand that they are in a role whose actual purpose is to protect the company from its employees, not the other way around. That's the bitter pill—the one that either drives the formation of callouses or else makes the work particularly sad and lonely.

It might seem strange or improbable to consider the guardian roles in this way. After all, we think of the office of the human resources manager as the place to go in confidence if your manager Sleazy Steve is being sleazy. And this is, in fact, the place to go for recourse. But HR provides this service *to the company* and not to you. If the situation escalates and there is no HR for you to talk to,

your next call will be to an outside attorney, which means a much more expensive problem for the organization. Putting processes in place for internal reporting and policing allows the situation to be handled with the much cheaper internal disciplinary action.

HR is protecting you in this narrative, but if you imagine things from Sleazy Steve's perspective, the company is actually using HR to protect itself from him. And that is the essential, core premise. HR and other internal watchdog concerns exist to stop the actions of employees from being expensive to the company.

This is generally true across the board, even in simpler situations that don't involve internal disputes. Boilerplate safety procedures exist to prevent costly accidents, both in terms of lost labor and legal actions. Preventing you from chopping off your hand is just a pleasant byproduct. All manner of internal standards conformance exists to indemnify the organization against individual incompetence. "Sorry for the violation, standards organization. But as you can see in our audit log, we've trained all of our employees per your protocols, so really, Jones is just incompetent. We've taken steps to ensure this won't happen again."

Make no mistake. This isn't some kind of evil conspiracy. Frequently the interests of the corporate HR and legal departments align with both individual needs and common decency. If Sleazy Steve is being inappropriate and winds up appropriately disciplined, that's a win for you, for the organization, and for humanity all in one shot. It's a good outcome. But the most important beneficiary of the good outcome is the company itself, with anything else simply being collateral good. If Steve could present a counterargument that it was in fact you who was being inappropriate, your knight in shining armor, the HR department, would turn the lance on you without hesitation and run you through. The only damsel in distress that he will unerringly champion is the company itself.

If you start to peel back all logical implications of this onion, you can see it overwhelmingly in all things corporate. Human resource's pay matrices, ostensibly designed to prevent injustices in relative

employee salary, actually exist to prevent the lawsuits that arise from these injustices. The organization, an intrinsically pathological construct, simply seeks to depress wages as much as humanly possible. But depressing them by overtly discriminating wherever possible has a negative expected outcome after a lawsuit, so a construct is created to depress equally. And things like dress code or having drink tickets at the Christmas party so no one gets too wasted and drives home? They're really about minimizing organizational cost in the form of messy fallout from human interactions.

This creeps insidiously into even the most benign-seeming corporate institutions, such as morale-boosting activities, company perks, and career counseling sessions. All of these things exist ostensibly to show you that the organization values you as a human and that it will act as a steward for your career. But the real, simple, dollars-and-cents truth of the matter is that somewhere there is a line item on a piece of paper that demonstrates cost of perks is exceeded by cost of turnover. As soon as the math there changes, you will receive an email that a down quarter has resulted in the belt tightening that takes away your free sodas. Some more belt tightening might just take away your job.

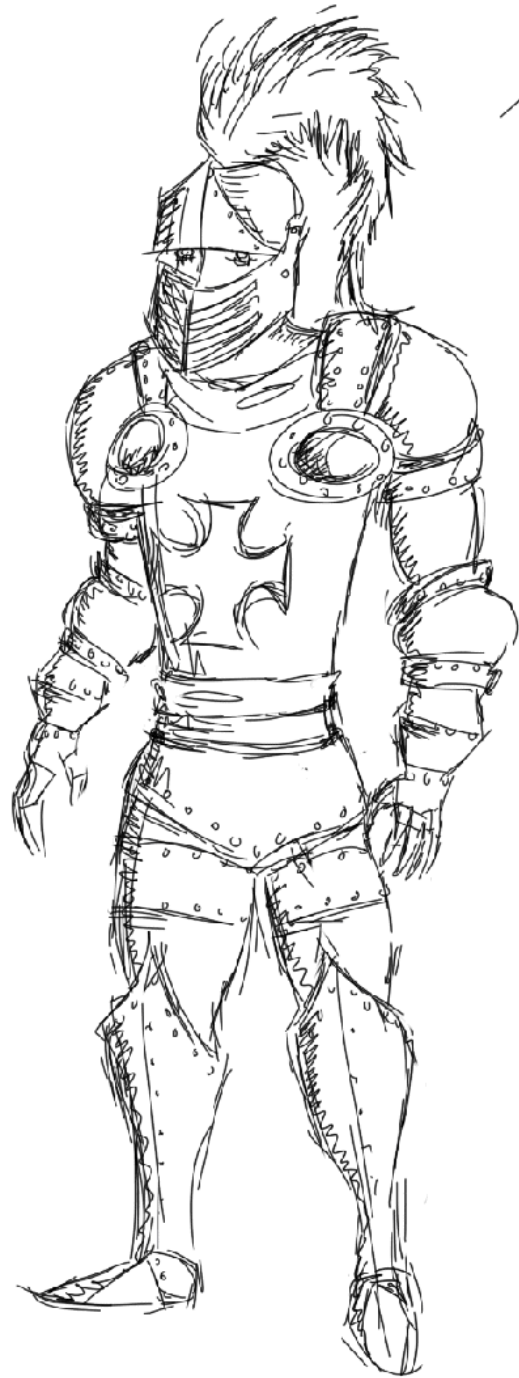
And this leads back to the ultimate truth about corporate America that has led to such widespread disloyalty among the millennial generation: your company does not care about you. And this sad state of affairs is exactly what gives rise to the most hoodwinked idealists and the most jaded opportunists. The very people charged with enacting and enforcing policy to protect companies from their workers are the same people offered as their protectors, confidantes, and advocates. They're asked to tell you how much the company cares about you and values you, and then, later, they're asked to censure and terminate you.

In the aforementioned example of the police department, I talked about how police enforce the law among citizenry and internal affairs departments enforce the law among police. The question "*quis custodiet ipsos custodes?*" thus has a satisfactory answer, at least in

theory. But now imagine if internal affairs had the underlying realpolitik purpose of protecting the police department against citizen complaints. Imagine if those guarding the guards were guarding *for* them and not against them. Quite simply, this would be a society with no empowerment whatsoever.

In such a situation, one can either live in a state of resignation, work one's way into the power structure, or flee and take up residence elsewhere. The majority of books offering career opinions or advice trade in how to do the first two. For the remainder of this book, I will be working toward a detailed treatment of the third option.

PART 3: THE HISTORY OF THE GAME



Chapter 19: Less Than the Sum of Its Parts

The worst of it is over now. I warned you that you were in store for some bleakness and cynicism, and certainly that characterized my assessment of the modern corporate status quo. From here forward, I'll work toward optimism. And the first step toward optimism is understanding.

To understand the corporate beast will require an understanding of its evolution throughout the course of time. Up until this point, we were just looking at a slice in time: right now. But Part 3 will focus on a history of the corporation so we can understand what legacies are carried forward in thinking and practice. After all, it's not as though someone dreamed up the modern corporation yesterday and turned it loose on the world. But before we get into that...

I've told you the story of myself as a fresh-faced graduate, looking with reverence upon the glamor of the corporate world. I've also told you about my first step away from the cave wall, resigning from my first corporate job. What followed, in my own personal story, was a series of shorter stays at different organizations, emblematic of withering loyalty.

I stayed in my first job for almost six years. In the next job, I lasted two and a half years. From there, it was always a year or less. A cursory inspection would suggest that I was either flaky or choosing poorly, but neither of those things holds up to much scrutiny. Sticking with a gig for six years (and simultaneously completing a night-school master's program for almost five of those years) demonstrates that I'm capable of seeing things through over the long haul. And some of the companies at which I stopped briefly were actually good companies. I recall generous benefits packages, friendly coworkers, flexible work arrangements, and all sorts of

perks. They weren't bad companies, and I'm not a bad human. And yet, on I'd move.

The next logical consideration is that I was job-hopping for advancement and profit. This is closer to the core truth. Right from the beginning, I was savvy enough only to jump ship for a nice pay increase. After my first jump, I also learned the importance of negotiating a better title and position each time as well. And so I was able to turn work experience, good references, and accomplishments into successively better arrangements.

The problem with advancement as a lone explanation, however, is that advancement within an organization is possible as well. Granted, the current corporate landscape for programmers is such that it's easier to advance by job-hopping, but my career has not been characterized by the path of least resistance. More money and a better title was attractive, but it wasn't a sole motivator. If it had been, I wouldn't have spent six years working for the same organization without a significant increase in title.

A more visceral force was at play here. The fact of the matter is that I wasn't so much moving *toward* better opportunities as I was moving *away* from situations that wearied me. I was losing faith and leaving, parlaying the departure into something better for myself while I was at it.

I'll oversimplify my mental investment, calling it an endeavor into two competing forces: my belief in the value of what I'm doing weighed against the friction I encounter trying to do it. In this context, my outlier—the six-year tenure with my first company—makes a lot more sense. The overwhelming majority of my projects and tasks were either solo endeavors or, eventually, projects that I led. I didn't view the work I was doing as a cure for cancer, but the friction was extraordinarily minimal. In fact, when I did leave eventually, it was the result of a more mundane concern: the company struggling and cutting our pay and benefits.

After that, however, I was never able to last very long. I would go through the interview process and be sold on the sorts of problems being solved and the approach the companies were taking to solving them. I would hire on, flush with enthusiasm and ideas for how to tackle the challenges facing us. And then I would work cheek-by-jowl with coworkers, navigating corporate processes and office politics that chipped away at my tolerance until I couldn't take it anymore. The force of the psychic friction would quickly outweigh my enthusiasm for the problem and the value I thought I could add. I would lose faith in the company and its people, as constituted.

As I went through this process, I began to harbor doubts about myself. Was there something basically wrong with me? Was I far too picky? And worse, was I limiting my options? Each jump yielded more organizational authority and pay, but the older generation cautioned me that I was going to earn the "job-hopper" label at some point and get stuck. Would the last jump that I made dump me in the most soul-crushing situation to date, and one from which I could not escape?

But, really, the central question was, "Why do I lose faith so easily?" In the context of Part 2, which you just read, the answer is that I was being forged into an opportunist. I wasn't willing to shrug and check out like the pragmatists, and I wasn't willing to put blind faith in organizations like the idealists. In a sense, opportunism was the only option left.

However, there's a deeper philosophical question at play here: why aren't organizations worth our faith? Somewhere between mission statements like "we want to bring the best gosh-darned widgets to the masses" and the realities of these organizations, a major disconnect happens. The corporations are less than the sum of their parts. Why is that?

This part is going to be dedicated to exploring that question. And to do so, I'm going to start from first principles and look at the evolution of the corporation throughout history. If they aren't worth our faith

now, were they ever? Where did these things even come from? And why?

As you read, please bear two things in mind. First, the majority of what I'm talking about here is admittedly Eurocentric, but I feel that this is appropriate given how influential Europe was on the modern corporate construct. And, secondly, while I did a good bit of research for this section, I am a technologist and not a historian nor a PhD in any related subject. Caveat emptor.

Chapter 20: Legacy—Ancient Corporations

To understand the corporation, let's consider the word's etymology.

Just kidding. Sort of. It's not that, like an overeager pedant, I think true understanding comes from the word's root. Rather, I think it's interesting that the word's root is Latin. *Corporare* is a Latin word meaning "to form into a body," and the actual word, corporation, arrived in Middle English, from that root, to mean, "persons united in body for some purpose."

If you did a poll asking people about the history of the corporation, I imagine you'd get some diverse answers. The most vacuous among us might think that corporations got their start only when television became mainstream enough to advertise for them. Others might imagine that modern labor laws and current corporate practices, such as the job interview, marked their birth. Others might cite the robber barons and the Gilded Age as the dawn of the corporation, though anyone that knows these terms would likely also be aware of colonial trading companies and mercantilism. (Don't worry if these terms are not familiar. They'll be covered.)

But the reality is that corporations and commerce go back a long, long way—further than you probably think they do. The oldest currently operating corporation is purported to be [a Japanese organization that has been in business since 578 AD](#). Corporations existed in the Roman Empire, as evidenced by the Latin origins of the word itself. But they go even further back than that, at least to a time when Rome was a small republic, run by senators, and Alexander the Great was rampaging eastward. The empire that eventually turned him back, the Maurya, had [organizations that resembled corporations in the 4th century BC](#).

Let that sink in for just a moment. Nearly 2,400 years ago, commercial organizations existed in such a way that we would

recognize them today. Obviously *commerce* has existed since the first proto-human bartered a shiny rock for a mammoth steak, but we're talking about commercial organizations. They were entities that would create and honor contracts, make loans, pool resources, and various other activities that seem eerily modern. Ancient humans were, as it turns out, pretty clever.

Now, let's revisit the English root of the word. Again, I'm not doing this for the sake of pedantry but for the sake of contemplation. By the time the word "corporation" emerged, the concept had existed for over 1,000 years, but the word's definition contains hints as to its true nature: "persons united in body for some purpose." What "persons" and what "purpose"?

Going back to the Maurya and perhaps other civilizations lost to history, what would have made people unite in some kind of purpose? War and ancient politics certainly had that effect, but we're talking about relatively peaceable, commercial ventures—pig farms, clay shingle shops, wainwrights, and the like. Why would people "unite" in this, rather than just doing it as part of the day-to-day hustle to put a better quality of food on the table?

The answer, I think, is fairly simple: legacy. You can spend your entire life hawking a product or service and ultimately feel as though you wasted all of that time. That can apply even if you were quite successful in doing so, expanding your wealth and influence while mastering your trade. On your deathbed, you're unlikely to be thinking about that awesome day that you sold a lot of things or struck a deal that improved your fortunes. You're going to be wondering about what you'll leave behind that will make people remember you.

The earliest incarnation of the corporation was almost certainly formed as a way to guarantee legacy—a way to establish an institution that survived the mortality of its founder. In a very real sense, it's commercial religion.

Put yourself in the shoes of some Mauryan 2,400 years ago. Imagine that you're skilled in repairing wagons and carts, and you provide ably for your family by offering this service in trade. That's fine and good for the day to day, but imagine that you're doing well enough to want to climb a rung on [Maslow's hierarchy](#). Wouldn't it be nice to feel that what you were building would live on and help preserve the memory of what you did indefinitely? Wouldn't it be nice to think that, 100 years after your death, people would still be talking about you as some kind of visionary wainwright?

It's in this fire that I believe initial corporate entities were forged. Ancient people concerned with legacy enlisted people, in exchange for compensation, to unite in a commercial purpose. The ostensible purpose was commerce and providing for the participants, but the motivational purpose was founder legacy. Think family business.

And so it came to pass in the ancient world that an ordinary merchant could create a way to establish a lasting legacy. They could have a taste of what it was like to be a military hero, a king, or a pharaoh. Their deaths may never be succeeded by legends, currency, or giant pyramids, but they would live on in other media.

Chapter 21: Influence—Medieval Corporations

In certain circles of the modern software development community, there has been a recent revival of medieval commerce terminology. As best I can tell, this orbits around the central notion of software *craftsmanship*.

In this terminology, an experienced, accomplished senior software developer is a craftsman. A mid-level developer is thought of as a journeyman. And an entry-level developer is mapped to an apprentice.

In medieval Europe, these designations described participants in the [guild systems](#). There, all practitioners of a craft within a region had to be vetted by the guild. For entry, one started as an apprentice, learning at the feet of a craftsman until such time as he was ready to venture forth on his own to earn a living. At this point, he became a journeyman. He may or may not eventually reach craftsman status, where the guild decided that he was worthy of highest honor in the field. This rank is also sometimes called “master craftsman” or simply “master.”

If you mull it over a bit, it's easy to see why this concept appeals to software developers and, in particular, high-end freelance consultants and ranking development staff at companies. In the first place, it indulges the pretense that software development work exists in a meritocratic, management-free vacuum. All that matters is one's competence as measured “in-house” by fellow members of the guild. (It is thus not a coincidence that, almost invariably, those proposing these systems axiomatically assign themselves the craftsman status). Notwithstanding any self-indulgence from founders, this system does neatly accommodate a reality: that entry level people are often not ready to handle real programming work without the oversight of someone accomplished in the field. And it also handles

the reality that software developers, perhaps more than just about any other type of line-level employee, bounce around from project to project an awful lot.

There's a halcyon feel to this metaphor, particularly weighted against the libertarian flavor of the IT industry at large. Programming is a meritocracy administered by consensus of the best minds in the field. There is a certain *je ne sais quoi* required for competence and mastery, and this can only be achieved through long, dutiful study at the feet of a master. The best, hardest working, humblest programmers thus make Horatio Alger proud by starting from nothing and working their way up to being at the top of the field. Adoption rates of this terminology are also not hurt by the programmer's love for the fantasy and science fiction genres, the former being permanently stuck in the Middle Ages and the latter expressing echoes of the same with fiction such as Star Wars and its Padawan/Jedi/Jedi Master construct.

Unfortunately for our purposes, the modern incarnation of the guild model with software differs significantly from its counterpart of 700 years ago. At least, it does in terms of motivation. The modern software craftsmanship movement is a call to arms to improve standards and quality whereas the medieval guild movement was in actuality a sort of labor union-cartel hybrid. To understand the impact of the Middle Ages and its guilds on the modern corporation, it is important check your assumptions about guilds at the gate.

The word "guild" itself [is derived from the Saxon word "gilden," meaning "to pay."](#) Note that it is *not* derived from a Saxon word for "to ensure quality." This distinction cuts right to the nature of the beast. The guild offered a service to its members, and its defining and eponymous attribute was that it demanded payment for this service. This is actually a fairly complex transaction. Though guilds established a floor for quality as a byproduct, raising the bar for quality was not a first-class goal.

To be a little more explicit about it, consider first that the medieval guild was somewhat of a fluid construct over the course of hundreds

of years, and there were different sorts of guilds (merchant, craft, and the like). That being said, the implied guild contract always was more or less an exchange of value between the guild and its members. The members gave the guild dues, supplied it with labor (and thus credibility and leverage), and participated in its governance. The guild, in turn, offered its members protection, outsized influence, and the medieval equivalent of marketing. Being in the masons' guild meant that you could count on having a steady supply of masonry work, courtesy of the guild.

The original motivation for the creation of guilds was the imposition of disastrous local taxes within the feudal economy. While not serfs, the merchant class were nonetheless subject to taxes from local lords and in little position to resist when those taxes became excessive. During the eleventh and twelfth centuries, the rise of towns in Europe transformed merchants from transient peddlers to established traders operating in more cosmopolitan settings. There was a natural division-of-labor-based motivation to team up anyway, but the guild was born out of a desire for collective bargaining. No individual farrier could buck the local lord on matters of taxes, but if all of them banded together, the lack of horseshoes would present a problem for him and thus leverage for the guild members.

From this beginning, the organizations grew in power, influence, and scope. Many of them were infused with religious overtones and rituals, and they began to exert influence on the lives of members. As with the legacy nature of ancient corporations, membership began to transcend simple commerce and work its way up Maslow's hierarchy. But beyond the interaction of the membership, the guild was in a position to flex its muscle in an increasingly urban feudal setting.

Guilds ceased to exist merely as defense against noble overreach and began to wield their own power with monopolies on trade. They fixed prices, ran scab labor out of town, restricted membership, placed members in political positions such as local mayors or councils, and exerted societal influence in a variety of ways. In

exchange for all of this, they kept the town serviced and guaranteed a minimum standard of quality. As you can see, this is hardly the modern, libertarian-inspired notion of free agent software craftsmen roaming around delivering the best labor for the best price. This was an organized labor cartel with a monopoly, satisfying its customers enough that they didn't revolt or petition local governments for change.

Guilds in their medieval incarnation lasted in some form or another until the late eighteenth century. As one might expect from an institution granting monopolistic cartel status, the benefits provided became largely outweighed by the rent-seeking behaviors of members and the institution as a whole. With the rise of the patent system (of which guilds, with their conventionally enforced trade secrets, were a forbearer) and competition from more modern manufacturing methods, the guild system wound up seeking to retain relevance by stifling innovation and exerting political influence exclusively, rather than providing any benefit. Eventually, some countries and municipal institutions even passed laws banning guilds. It was an ignominious end to something that had been justifiable, practical, and somewhat ingenious initially.

But the rise and fall of the guild system and the reasoning behind both provides some insight into what the modern corporation took away from the guild construct of the Middle Ages. It was the dawn of commercial entities wielding significant political and societal influence at the local and regional scope. No longer were institutions of commerce a matter of simple founder legacy; they were now vehicles for allowing manufacture and trade of goods to influence political entities and the lives of citizens. This was the precursor to the modern corporation's ability to influence society beyond its own scope of doing business.

Chapter 22: Gestalt—Mercantilism

The idea of guilds evokes medieval imagery, and rightly so. The rise of guilds coincided with a very provincial time in Europe. Local lords and fiefdoms dominated the landscape, and the influence of country and capital was relatively minimal.

But the same wave of urbanization that brought guilds to the height of their power also added a gradual consolidation of national power and influence. Countries went from being loose collections of lord-dominated city states to being the units of geopolitical autonomy that we recognize today. Countries as we know them were born in Europe as guilds entered the height of their prominence.

The result was unprecedented preoccupation with the fate of these nations as a whole, and that included reasoning about economic health. In the 1500s, an economic philosophy that would later become known as [mercantilism](#) emerged. Mercantilism, more or less, held that a country's wealth and success was the amount of gold that it had at its disposal. Protecting the economic health of the nation, then, became a matter of taking certain logically deducible steps:

- Subsidizing exports
- Levying heavy tariffs and other deterrents on imports
- Maximizing the use of domestic resources
- Using a currency other than precious metals for those few external payments that are necessary

Now to us, in the modern world where some nation owns just about every blade of grass and rock, this would seem to be purely a zero-sum game. But in the world of 1500s Europe, there was a lot out there completely open for the taking. You just had to travel a bit to do it.

Specifically, large parts of Asia, the Americas, and Africa were fair game to anyone who might happen by. So if England wanted beef but didn't want to pay the Spanish or French for it, they could dispatch some ships to wherever in the world cows might be, bonk some natives on the head, and simply take it. In the mercantile world, this was a much better alternative than an import. And thanks to the increased nationalization of resources and focus, it was now possible.

But it wasn't exactly obvious what the division of labor might be. Ruling entities were historically military and bureaucratic. In other words, from local lords on up to the king, ruling bodies knew how to fight wars and administer taxes—they were not experts in trade. So, while there was much concern for the good of the nation in the mercantile world, the state would need help to realize the full benefits of colonial plunder.

It was out of this necessity that [the chartered company of the mercantile age](#) was born. At first, these chartered companies were formed in the style of guilds. There'd be the guys that made horseshoes, the guys that made cloth, and the guys that traded with Russia. Like the guilds, members of early trading companies operated as individuals who were part of a company. But given the inherently different model of doing business, this uniform approach quickly diverged. It turns out that running a local service monopoly is much different than brokering international trade.

The chartered companies did two important things that would set themselves apart from their predecessor guilds. First, they secured underwriters (investors) to go after things not achievable by any one individual. (The monarchy and governing bodies often became underwriters of chartered companies.) Second, they introduced the idea of [a joint stock company](#). In short, the new development was that commercial organizations could be financed by non-operating owners, and shares of the interest in the company could be traded like any good or service.

The separation of ownership, and thus profit, from operation is *profound* for our treatment of corporate history. The people that used to be participating guild members now became more passive owners in the form of joint shareholders. They no longer operated as individuals, instead abdicating their decision-making to the company, which operated as a cohesive unit. A business owner has thus evolved over the last few chapters from “Bob Smith, Owner and Operator of Smith & Sons” to “Bob Smith, Craftsman & Guild Member,” to “Bob Smith, Venture Mercantilist.” The business operator evolved in parallel from “just Bob” to “still just Bob” to “a professional operator who says, ‘Trust me, Bob, I know how to earn you a return on your investment.’”

For the first time, the corporate entity has a life of its own, and for the first time, we can think of it as a gestalt. That means it’s an integrated unit whose properties are not derivable from the sum of its parts. The company’s purpose now transcends individual legacy and local politics by virtue of the fact that the company itself is now an ownable, tradeable commodity. In a very real sense, its purpose can be to profit literally anyone in the world.

Of course, the seismic nature of this shift becomes pretty easy to pick out with 20/20 hindsight. Early trading companies may have resembled guilds, but as guild prominence faded, famous mercantile-era companies like the [East India Company](#) rose to astonishing heights of power. These colonial mainstays fought wars, made nations tremble, traded human beings, and redrew the global map.

By the time Adam Smith’s theories rose to prominence in the eighteenth century, the guild system was quite passe, mercantilism was on its way out, and the era of the colonial trade company was about to give way. But the lasting legacy from this time had emerged to shape modern corporations. These entities were capable of global influence by becoming more, economically, than the sum of their parts.

Chapter 23: Barriers to Entry—Industrial Revolution

During the seventeenth through nineteenth centuries, a great race to colonize took place among the powers of Europe. The motivating factors for this were complex, though we've covered them to an extent in discussing mercantilism. Suffice it to say that the era was one in which land was considered to be a great source of wealth. European powers were racing to gobble up land all over the world.

During the eighteenth and nineteenth centuries, however, a change in focus began to take place. Having vast acreage to farm or mine began to take a backseat to the emergence of massive improvements in the production of goods. This came to be known as the Industrial Revolution, and it was characterized by the convergence of key technological advancements.

[New materials \(steel and iron\), sources of power \(coal and electricity\), and machines \(power looms and spinning jennies\)](#) allowed the necessities of life to be generated mechanically in a fraction of the time that had previously been needed. And in a world where investors can finance operations and reap a share of the profit, there were *irresistible* margins to be had here. Forget finding some remote island from which to harvest and ship sugar. The real money was now in building factories that mass-produced pants.

While this all seems very progressive in the context of what we just learned, let's not forget that life for the average peasant was changed only in that they were serially indebted to someone else. Medieval society had been feudal in the countryside, with local lords "renting" land to peasants in exchange for them working those lands. In towns, guilds played a similar role, fixing prices, limiting labor and supply of goods, and imposing strict rules on who could work when and how. In either case, the Middle Age peasant had little choice but

to offer up labor at cost—the game was rigged in that they were forced to spend as much as they earned and do what they were doing until they dropped dead.

The time leading up to the Industrial Revolution saw changes in who controlled the peasants but not so much in the fortunes of those peasants. Society was becoming increasingly urban, and colonialism and trading companies had popped the bubbles required for guilds to operate as effective cartels. The landed aristocracy was having to make way for a merchant class of growing wealth. This led to conditions in flux, but not improved, for peasants.

It was during this time that the [concept of a wage emerged](#). The wealthy merchants became middlemen between producers of goods and consumers, and they started to leverage that position to restrict what could be done by producers. They started paying for orders in advance, then paying and supplying materials, and then essentially paying a “wage” for the labor. But, whether you’re tilling the land in 1400 for your local lord or cranking out aprons all day for your local merchant in 1700, laboring twelve hours a day for someone else is still laboring twelve hours a day for someone else.

Thus while the principals changed, the Industrial Revolution in Europe saw the superposition of a corporate structure atop a feudal one. The serfs of yore became factory workers, and the lords became merchants, but the game was more or less the same. The key difference for our purposes was that this is the first time the corporate structure *included* the serfs. They had become employees, willingly or not.

And so were born the first corporate pragmatists in earnest. Peasants having no practical means of escaping peasanthood was certainly nothing new, but the modern poverty trap was. Theoretically, the emerging corporate game was open for any and all to play—it just so happened that, with the new economy oriented around mechanized productivity gains, only those who already had capital could play meaningfully.

I am not a sympathizer with the socialist/communist cause by any stretch of the imagination. Planned economies and government-compelled incentives simply do not work, in the same way that pre-planned, massive waterfall projects do not work. But nevertheless, I will borrow from Karl Marx in his reaction to the Industrial Revolution and talk about [the means of production](#).

Keep in mind that Karl Marx advocated what he did not in the face of our modern, managed market economy, but in the face of the Industrial Revolution, which represented feudalism *cum* industry. He looked at the possession of land, (wage) labor, and capital by the merchant class and observed that workers lacked the means of production (and means for acquiring those means).

I mention the means of production here as an item of particular interest because I will return to this topic later. Suffice it to say, however, that the Industrial Revolution brought a new concept to the corporate game: barriers to entry. In bygone eras, barriers to entry had existed. In ancient empires, not just any citizen could be a merchant. And in the Middle Ages, not just anyone was allowed to join the guild. But those considerations had always been social and political. By the time the Industrial Revolution rolled around, commerce had expanded and become sufficiently complex as to now have its own barriers to entry. Though they would be allowed, not just anyone could build and staff a factory because few had the money to do so.

By this time in history, the corporation had acquired founder legacy, societal influence, and the concept of gestalt. But these are mainly holistic concerns for the organization. The birth of the hopeless pragmatist has given us the first glimpse into the internal corporate structure that we recognize today.

Chapter 24: Layered Organizations—Taylorism

Organizations gaining pragmatists is an internal concern for them, and the provincially impoverishing tactics of [robber barons](#) generally concern the fate of those pragmatists in society. But let's talk briefly about the changing relationship between producers and consumers during the Industrial Revolution.

Throughout most of human history, the production of goods was a matter of *craft*. That is, any individual good was produced by a craftsman, and no two were exactly the same. Thus the ways in which competitors might distinguish themselves for potential purchasers were both obvious and personal. A maker of furniture, for instance, may have looked to appeal to aesthetics or to quality of workmanship. A different maker might eschew those high bars and go after customers with lower prices. But that was basically it. And where guilds operated in cartel fashion, even these distinctions would have been muted.

When commerce was a matter of craft, each good was unique, and each good was produced entirely by the craftsman. The Industrial Revolution turned that concept on its head via mass production. Factories and the laborers in them cranked out nearly-identical goods en masse, and this changed the nature of competition in commerce. Sure, one manufacturer's product might have a different look or quality than another's, but the goods were being distributed to much larger customer bases and the nature of the competition in production became much more aggregate. In this new industrial world, one person picking product A over product B was irrelevant in a way that would have been inconceivable to craft makers of goods.

As a result of this aggregate competition, new ways for these companies to distinguish themselves emerged. The most notable among these was *efficiency*. Whereas an individual craftsman could

stand to benefit a bit by improving process, at his scale, this wouldn't have mattered too much. At a much larger scale, however, there are equally large gains to be had. It became obvious to industrialist owners that reducing operating cost by 25 percent was as good as increasing revenue by 25 percent.

Thus the Industrial Revolution prompted an obsession with efficiency and economies of scale. You could save money buying in bulk. You could save money assembling parts yourself instead of purchasing them pre-assembled. You could save money moving an operation somewhere that the laborers commanded a lower wage. Or, you could save money by wringing more productivity out of the workers on the payroll.

Historically, management of labor was a fairly ham-fisted pursuit, and this hadn't changed much during the Industrial Revolution. Imagine the construction of the pyramids. You had an owner (pharaoh), his assistant architects, and a bunch of laborers cutting and dragging stones. As the operation ramped up and the owner himself could no longer supervise all parties, it was necessary to manage the laborers to make sure that they didn't slack or fail to show up.

Toward this end, whip-crackers were appointed. Presumably, the largest, most loyal and most sadistic laborers were promoted to "management," cracking the whip and encouraging their former peers to work harder, show up on time, and not waste time at the water cooler. And this style of management at large enterprises persisted up through the Industrial Revolution.

It persisted right up until a man named [Frederick Winslow Taylor](#) came along and led a revolution in management thinking. Taylor was influential during a period known as the Progressive Era, which took place between 1890 and 1930, and it saw a change in approach to industry known as the efficiency movement. Taylor himself wrote a book called [The Principles of Scientific Management](#), and its legacy is absolutely critical in shaping the modern corporation.

I personally think of Taylor as a magnificent bastard.

There was a genius to his work and an undeniable creativity in approach. He observed laborers unloading ore from railroad cars and timed their efforts while noting their approaches. In this fashion, he was able to form hypotheses, such as the best weight of shovel for each person to use. If this sort of concept seems familiar to you, it might be because Taylor was pioneering the lean startup 100 years before [*The Lean Startup*](#) was written. He was applying scientific principles to factories and labor to improve efficiency.

His observations were not limited to efficiency tweaks to labor, either. He was a pioneer in observing that human management by whip-cracking was not effective and introduced, to great success, concepts such as frequent breaks and adequate pay. Taylor was, perhaps, the original management consultant and father of industrial engineering.

So, that establishes the “magnificent” part, but what about “bastard?” Well, that comes in two parts, and the first one is entirely no fault of Taylor’s. I will talk about this later on in the book, but Taylor’s obsession with efficiency and advocacy of micromanagement translate poorly to the modern knowledge work economy. But that doesn’t stop companies from mindlessly carrying it forward in their treatment of staff.

The second part of what chafes me about Taylor was entirely within his control, though it may also be an artifact of the time. Taylor grew up during the reconstruction era and Jim Crow in the US, so facility with categorizing some humans as stupid and beastlike may have been a natural predisposition. Nevertheless, he owned it. To understand what I mean, consider the following quote from *The Principles of Scientific Management*.

“The labor should include rest breaks so that the worker has time to recover from fatigue. Now one of the very first requirements for a man who is fit to handle pig iron as a regular occupation is that he shall be so stupid and so phlegmatic that he more nearly resembles in his mental makeup the ox than any

other type. The man who is mentally alert and intelligent is for this very reason entirely unsuited to what would, for him, be the grinding monotony of work of this character. Therefore the workman who is best suited to handling pig iron is unable to understand the real science of doing this class of work.”

The sentiment here speaks for itself, but the ramifications are both subtle and profound for the purposes of our examination here. Throughout history, there had been owners, beastlike laborers with whips, and normal beastlike laborers. Taylor proposes something different. He proposes another piece of the corporate puzzle that creates the layer cake we know today. He proposes an organization consisting of owners, non-owning but educated managers, and beastlike laborers.

Is this starting to sound familiar? Left to their own devices, those brutes that make up the line level within an organization would come in late, slack off, not know how to direct their efforts, and generally make a mess of things. What’s needed is a layer of more educated, refined, and generally better humans, who understand carrots and sticks and how to apply them deftly to extract the most value out of the quasi-beasts punching in at nine and out at five.

Taylor bequeathed upon us the idealist layer of the organization. These are people without the capital and/or chutzpah to be owners and tycoons, but with the gentle breeding and educated predisposition to have a position of influence within the organization. This position would be at the owner’s pleasure, of course, but it would nevertheless be influential and more respectable than common labor.

Taylor’s legacy was undeniable. If you work in software development or an engineering discipline, you are, no doubt, familiar with the iconic Gantt chart. That chart was named for one [Henry Gantt](#), a disciple of Taylor’s. The tendrils of Taylor’s influence are ubiquitous in our modern corporate world.

Through our historical examination, we've seen the establishment of founder legacy and local influence of corporate entities, and we've observed the global growth that gave rise to the gestalt of the organization. We've now seen the precedent for barriers to entry creating a hopeless caste of workers and the efficiency-driven elevation of an overly loyal, non-owning caste of workers inclined to cede perspective. The organization is truly starting to resemble its modern form.

Chapter 25: Ubiquity—Organized Labor

If you have a rough timeline of modern US and European history in your head, you probably know, without even reading the chapter title, that a discussion of union labor is necessarily coming soon.

Organized labor, in the form of unions, had a major say in the modern corporation. But before we can discuss that as a succession to Taylorism, let's take a look at the history of unions.

They didn't simply spring into life in the early 1900s, ready to collectively bargain for a minimum wage. They had been around, in some form or another, for two centuries. Early unions were known as trade unions, and these stood in some contrast to the later concept of labor unions.

The trade union was a logical successor to the guilds, which had fallen out of favor, been outlawed, or both, depending on location. As the Industrial Revolution progressed, a number of crafts had been displaced by mechanization and low-skill labor. Others, such as carpentry, had not, and trade unions for these occasionally emerged. The first recorded strike for higher wages [occurred in Philadelphia in 1786](#), when the printers in that city opposed a would-be reduction in wages.

In Europe, which had outlawed guild cartels in the years leading up to the Industrial Revolution, these unions took longer to emerge. But in the United States, their formation was more common. There, they attempted to exhibit similar behaviors to their European predecessors, seeking where possible to take advantage of provincial monopolies to bargain collectively. This achieved limited success and generally culminated in some sort of strike action that either reached its goal or resulted in the dissolution of the trade union. And, as with their medieval counterparts, these organizations

often met in secret and had religious or other overtones that went beyond their commercial charters.

This distinction between labor and trade unions is an important one to understand because of the role each would play in forming the modern corporation. Trade unions were guild-like precursors to labor unions, and they concerned themselves with smaller, more well-to-do segments of the population. Labor unions were a broader unification of the downtrodden common laborer.

As the Industrial Revolution progressed, more and more of the population found itself performing unskilled labor. There were some attempts to improve their lot in life, but this was an unprecedented situation, and it was not well addressed by trade union tactics. Trade unions, and guilds before them, sought to use monopolies of skilled trade as bargaining leverage, but that fell short when addressing unskilled trade. Rather, the attempts at life improvement were mainly political. They incorporated the socialist movement, anarchist ideas, cooperatives, and politicized union attempts.

These types of political movements were able to carry out their ideas only by governmental overthrow. In continuing capitalist societies, they had fringe influence only. And since trade unions, with their attempts at monopoly, did not apply to unskilled labor, the beginning of the Progressive Era had seen little to improve the conditions of the unskilled laborer.

These unskilled laborers were in a tough position. Trade unions were more interested in middle class craftsmen than they were in the common man, so there was no support there. Employees were often forced to work long hours in poor conditions, starting at an early age. Even Taylor, a would-be advocate on their behalf for better conditions, viewed them as chattel that should be treated better only for the sake of the enterprise. National labor unions had begun to exist, but membership remained low. In general, it was better for citizens to stay out of the corporate game altogether if it could be avoided.

In a sense, this created a “tragedy of the commons” situation, at least in terms of corporate ubiquity. For any given organization, it made sense to keep costs as low as possible vis-a-vis the people doing the grunt work. But for the concept of corporations on the whole, this approach kept the labor pool necessarily smaller than it might be. Being an employee tended not to be a good deal, and it tended to be the province of the dregs of society, as many saw it.

That would change during the Progressive Era. The unions became increasingly appealing to laborers and membership grew. Even as Taylor’s scientific management principles made labor slightly more tolerable, they introduced micromanagement and separated the laborers further from any sense of pride or accomplishment in their work. On top of that, the political climate created by World War I in the US saw the government begin to endorse, support, and even create labor unions. During the subsequent Roaring Twenties, membership and labor unions would backslide some. But the groundwork had been laid, and there was an increasing legitimacy to being a laborer.

In the 1930s, as the Great Depression ravaged the globe, labor unions would receive a political boon from Franklin Delano Roosevelt. With a government sympathetic to the plight of the common man behind him, FDR passed a veritable goldmine of legislation to protect members of the labor force, and union membership exploded. Suddenly, there were laws limiting child labor; guaranteeing overtime pay; ensuring minimum wages, standards, and working conditions; and other, more targeted considerations besides. These mainstays are quite recognizable even today as the rules that govern modern corporations. They are largely intact from the ones passed to govern the labor economy eighty years ago.

As the Great Depression ended with World War II, labor came to be viewed as the backbone upon which the war effort was built. Manufacturing goods became a patriotic duty, starring the laborer, who was now backed by an influential labor union and popular

sentiment besides. Demand to support the war effort combined with the previous paucity of employment lured millions of people into the workforce. As the war ended, the percentage of people employed by corporations was exploding. And societal regard for labor was growing with it.

There was one final wartime curiosity in the US that would innocuously lay the groundwork for the ubiquity of employment in the half century that would follow. During World War II, the US government was highly sensitive to the problem of post-war inflation that had so affected Germany's economy following World War I. As such, they set a cap on wage increases, but almost as an afterthought, [they allowed a tax exemption on employers providing health insurance](#). The result, unsurprisingly, was an explosion in the amount of employer-provided health insurance plans.

After the war, when the government sought to roll back the benefit, the now-entrenched industry resisted. The result was that an almost-whimsical, temporary tax benefit would establish corporate employment as the key to having affordable health care. As anyone familiar with the US health insurance system knows, this remains true even today.

And so the first half of the twentieth century, largely due to the efforts of labor advocates, saw an absolute explosion in the rate at which people participated in the corporate system. Entering the Progressive Era, corporate work was limited to downtrodden laborers and wealthy owners. By the 1950s, it included owners, executives, managers, and a middle class going proudly to work at factories and plants.

The corporation had acquired the final piece in the puzzle to make it what we recognize today. It was now everywhere. And for the first time, we had a society where the default was that people worked corporate jobs. Whatever effect it may have on the bottom line or artificial wage inflation, there is no doubt that organized labor paved the way toward the corporate entity swallowing everyone, directly or indirectly. The days of the independent craftsman, the worker of the

land, and even the entrepreneur were comparably quite limited. The default position in the age of the company man and the massive global corporation was that everyone respectable got jobs with good organizations, showed company loyalty, and worked their way up the food chain.

Chapter 26: Anachronism—Rise of Knowledge Work

Midway through the twentieth century, the modern corporation was, more or less, established exactly as we recognize it today. And truly, that makes sense. We've examined the beast through more than 2,000 years of history, so it seems reasonable that half a century wouldn't alter it too much.

At least, that seems reasonable until you consider that the rate of technological advancement is following an exponential curve. We've sent men to the moon, mapped our genome, built the internet, made instant global communication from anywhere an afterthought, and eradicated a number of infectious scourges from the face of the earth. Industries are emerging and dying at a dizzying pace compared with any point in recorded history. If you doubt it, consider that print and recorded media are as good as dead and few people develop film anymore. Even building or driving vehicles for a living seems to be on the endangered list. Companies that were blue chippers thirty years ago face existential threats, and today's blue chippers barely existed or didn't exist thirty years ago.

And yet, the corporation is more or less the same as it was sixty years ago, which is all the more amazing when you consider that most corporations were established more recently than that. But let's come back to that in a bit.

In the last chapter, I talked at length about the impact of the labor movement on ubiquity. By the 1950s, legislation and corporate convention had set the stage for a corporate world in which nearly everyone was a citizen. But this wasn't the only change affecting corporations during that time period—just the most instrumental in advancing its DNA to what we've come to know.

Another noteworthy change taking place was the subtle alteration of the management layer. Historically, from the time that mercantilism and the chasing of land brought us expansive commercial entities, organizational leadership was decidedly *martial*. That is, leadership was command and control, in the shape of a pyramid. This held true up until the time of Taylor, which began to usher in a new era of management.

True, Taylor advocated some pretty intense micromanagement and promoted a distrust of the beastlike laborers responsible for the grunt work. But he also introduced the idea of a genteel caste of management, as well as the idea of that caste being responsible for systems of labor. The merit of this thinking yielded undeniable results. So began the transition from martial command and control of organizations to *bureaucratic* command and control.

The leadership structure still retained the shape of a pyramid, but gone was the naked king of the hill approach that had previously existed. In its place stood systems, processes, controls, charts, and management fads, administered by the newly defined idealist class of the workforce. These systems and modern ways of thinking were effective in that they allowed the efficient globalization of manufacturing, but they also neatly separated labor from the obvious measurement of its value. The bureaucratic idealist caste remained molded in a pyramid-shaped structure, but the navigation upward became a matter of intrigue, loyalty, and politics in an unprecedented way.

At this point, corporate structure became remarkably sticky. When the organization scaled up too much and became too complex for even the most scientific-minded person to measure, a reliance on tradition and familiarity took over. Recall that Edison invented the “modern” job interview in 1921, and it has remained intact, doing a terrible job of identifying talent, for ninety-five years. The eight-hour day, five-day workweek was established in the 1930s as a reasonable standard for common manual laborers, and that, too, remains intact for us today. We still use Gantt charts, and we still

tend to assume that line-level grunts are children whose arrival at nine and departure at five needs to be carefully managed, lest they steal precious minutes from the big guys upstairs.

But in spite of the fact that corporations have recently found themselves stuck in quicksand, there is one dramatic change that has taken place. It started innocuously enough and then grew exponentially. I'm talking about the rise of the knowledge worker and, more specifically, the emergence of information technology (IT). Peter Drucker coined the term "[knowledge worker](#)" in 1957 and used it to describe people whose main work function is to think, solving "non-routine" problems. To understand the nature of a knowledge work enterprise, imagine one in which all of the value walks out of the office when the employees leave for home at the end of the day.

As the labor movement made the corporation palatable for the masses, and as management shifted to a kinder, more bureaucratic form of command and control, industry had to be increasingly innovative to keep realizing efficiency gains. Inventing a mechanical loom was easy compared to figuring out ways to automate the construction of cars or airplanes. And even with the genius of folks like Taylor performing industrial engineering analysis, tweaks to human process and efficiency would go only so far.

Luckily for the world and for industry, the middle of the twentieth century saw the advent of a new form of automation: the computer. Initial applications were limited to research and defense, as is the case with many advances. Soon enough, though, mainframes began to trickle their way into industry. The possibilities for automation of work and eventual elimination of manual labor concerns were now limitless.

In order to harness this power, an organization just needed the money to buy a machine like this and hire some geek to operate it so that actual business people could solve actual business problems. This opened the floodgates. As the twentieth century marched toward a close, manual labor was eliminated and operations made more efficient at a jaw-dropping rate. Demand for the weirdos that

could operate and program these devices spiked high enough that it was common for the CFO to start having IT guys reporting to him. They were, of course, line-level employees and one-trick ponies, computer whisperers that could be called upon to fix technical problems and then to go back to their lairs.

Of course, with the turn of the century/millennium, people had started to figure out that this computer thing was here to stay. IT people (and now “software engineers”) of the world could not easily be dismissed. The internet had exploded into existence, the .COM bubble had come and gone, and when the dust settled, the demand for IT infrastructure and automation was institutionalized. Over the next ten years, the demand would become so amazing—so pronounced—that people in this line of work would become annoyed at all the calls coming in, begging them to take job interviews.

Yet for these people and for knowledge workers in general, a heavy, heavy legacy remains to this day. Organizations pay lip service to agile methodologies and to servant leadership, but by and large, bureaucratic command and control structures govern knowledge work. The days of IT guys being social incompetents reporting to the CFO are long gone. The stereotype that developers need “project managers” and “business analysts” to act as translators, however, remains. Corporations hire recruiters to badger and beg developers to take interviews, and then they subject them to Thomas Edison’s idiotic interview scheme to thank them. Just like assembly line workers from eighty years ago, developers are asked to drive to a physical location, punch in at nine, punch out at five, and mind that their breaks not take too long (you know, lest they fail to assemble their expected widget count for the day).

For the knowledge worker, the modern corporation is an all-encompassing anachronism, carrying forward more than 2,000 years of cruft. Its citizens must endure absurd, corporate-religious tales of founder mythos. (“And this is the hoodie he wears every day!”) They are expected to be mindful of the corporation’s place in the community, representing it well and upholding its values. The

corporation itself is also more than the sum of its parts, carrying its founder legacy and its values forward, rising up as some kind of force for that purpose in the broader world. But as good as all of that sounds, not just anyone can start one, as being a founder requires a certain magic spark of genius (and rounds of funding and capital). So it's better just to get a college education to prove that you're worthy of the management caste eventually and then to put your nose to the grindstone, be loyal, and work your way up. Oh, and, by the way, you pretty much have to participate. It's not like you can just go run an ostrich farm somewhere and be taken seriously. Corporations are everywhere.

All of that is our legacy, and all of that is our modern reality. But all of that is a bubble that is about to pop. Ubiquitous and inevitable as it seems, the modern corporation is dying and a sea change is coming. Why do I say that? Well, it's simple. I mentioned the term earlier, promising to return to it, and here I shall. For the first time in recorded history, we easily own our own significant means of production.

Before the Industrial Revolution, craftsmen and traders did not own *significant* means of production—they had no way to scale. And starting with the Industrial Revolution, owning the means of production require massive amounts of preexisting capital. That has remained true up until quite recently. But now, that has vanished spectacularly.

You can go out and get a computer for less than \$200 and use it to start programming. If you're willing to work and to bootstrap, that's it. Your startup expense is \$200. With that, you can build a business that grows into a thriving livelihood, a bustling operation, a national enterprise, and then an *empire*. You truly own your means of production.

Is it likely that you can parlay \$200 into an empire? No. Is it likely that you'll find steady work without prior experience? No, of course not. Barriers to entry still exist, but not the way they used to. You

couldn't bootstrap a car factory because you couldn't afford it. Cost is not a limiting factor any longer in the knowledge worker economy.

At the beginning of Part 3, I relayed how I'd lost faith in modern organizations over and over again, and I asked whether organizations are worth our faith. Looking back at more than 2,000 years of corporate history, we can see that there are complex traditions that make the corporate beast what it is. And we can also see that the corporation as we know it has failed to adapt to the knowledge worker economy and the new pace of technological change.

So, are organizations worth our faith? The answer, sadly, is no. And that's because corporations do a remarkable job of solving the problems of yesterday—problems that we no longer have.

PART 4: HOW TO WIN THE GAME



Chapter 27: Succeeding in the Corporate World, Such as It Is

In Part 3, I took you through a relatively quick history of the corporate beast, focusing specifically on the developments most relevant to the modern incarnation of the corporation. Before I did so, I said that understanding the history was essential to understanding the corporation. That was true. It's also true that this information, current and historical, is the key to your deep understanding of how to succeed from the inside of one of these things.

For instance, it's valuable to know that founder legacy is part of the corporate lizard brain—at least, for those who know what to do with the information. Among other things, it speaks to why many founders are more inclined than you might think to tolerate a layer of semi-competent sycophants in their organization. If they were purely concerned with dollars and cents, they'd inoculate themselves against flattery completely. But founder preoccupation with legacy also speaks to why those types wind up stuck in the middle of the company, not to be trusted with decisions that threaten to torpedo *too* many of those dollars and cents. No more dollars and cents means no more legacy.

We'll get to that in a lot more detail. For now, suffice it to say that understanding the past and creating a taxonomy for the present will make it easier to talk about how to succeed, why the existing success recipe is unfortunate, and how we can do better.

By the time I'd started to figure these things out explicitly in my own journey, I had spent some time job hopping. Now, when I say "figure this stuff out," I obviously don't mean extensive research of the history of the corporation but rather an intuitive understanding of the rules of the game and a crude notion of the taxonomy from Part 2. Bouncing around and starting to consult had furnished me with the

opportunity to make numerous firsthand observations, and I listened greedily to the secondhand tales of others offered to me by friends and family.

At the tender age of thirty-three, I was a technical executive (CIO) of a company with about eighty employees and fairly significant technical operations. I had created this favorable career situation not by paying dues and working hard for *companies*. Rather, I owed my position to working hard for *myself*—putting in seventy-hour weeks to earn a graduate degree while working full time; building a formidable network of people that remembered me well; moonlighting and freelancing; starting a blog; creating developer training videos; and maintaining a constant and open inquiry as to the availability of better paying, better titled jobs, notwithstanding the “disloyal” branding this could have earned me, from the company perspective.

Make no mistake. I got the job done and earned accolades at the office, but I reserved working overtime for activities that maximized *my* position, rather than a company’s. During the course of this time, I did, in fact, exhibit loyalty. It was just that I exhibited loyalty to humans rather than to organizations, being sure not to leave people in the lurch and to keep in touch, providing good references and opportunities where appropriate.

It was in my executive role and later, when leaving it, that I started to think most philosophically about the rat race. That stands to reason, since it was during this time that I would trade the slam-dunk career arc afforded to a thirty-three-year-old CIO for the gigantic question mark that is self-employed consulting. One does not do such a thing without some serious disillusionment with the system—the very system that had served me so well.

But *that* particular story—my motivation for quitting—will have to wait to be told in full. The story here is my rise to the position in relatively short order. This was accomplished through no shortage of hard work, opportunism (in the normal sense of the word), and if I’m being totally honest, some luck. But the general umbrella under which all of

these things fall is that it was accomplished by having a fundamentally different approach to the corporate game than most.

This part of the book is about that approach; it's about how to succeed in the corporate theater as it exists today, based upon the legacy of the corporation in human history. How does one navigate corporate employment to the best possible outcome?

For the bulk of Part 4, I'll assume that success means moving up in rank, getting as close to an owner position as possible. This is pretty reasonable since the overwhelming majority of people playing this game are looking to secure higher wages, more influential roles, and better deals. The fact that a substantial cross section of workers (pragmatists) resign themselves to failure (or not playing, if you will) doesn't actually alter the game, per se. But it *does* create the need to define alternate theories of success—one each for pragmatists and idealists—and devote a chapter to each before moving on to the main course. Bear in mind, after all, that rising from line-level employee to middle and upper management is a very, very first-world concern. There are people out there that say, “I like being a line-level widgeteer and a good one, and I have no interest in moving up.” And that's fair enough.

So we'll take a look at defining success for pragmatists. While we're at it, we'll also stroll through the surreal concept of success for idealists. After that, we'll get serious about winning the corporate game, such as it is.

Chapter 28: Pragmatists Succeed with Alternate Narratives

If you have a mug on your desk that says, “I’d rather be fishing,” then success is fishing.

In Part 2, I talked about what companies take from each archetype. From pragmatists, they remove hope, but that is hope within the context of the company. In other words, the pragmatists are not hopeless humans but rather hopeless vis-a-vis acquiring real power and influence with the organization. And they’re usually okay with this. They’d rather be fishing.

Against this backdrop, defining success is a bit strange. It’s sort of like defining success for someone in a court-ordered anger management course. They’re only there because someone is forcing them to be, and their attitude is essentially, “Let’s get this over with so that I can do other things.” Success is uneventful completion with a minimum of effort expended.

Of course, there’s a difference between a few Saturdays of anger management and needing to pay the bills for the entirety of one’s adult life. The pragmatist’s relationship with the office thus becomes more nuanced, though the “let’s get this over with” attitude tends to pervade all the same. This attitude manifests itself in TGIF (“Thank God it’s Friday”) culture that goes beyond the normal inanities about Friday being wonderful and Monday being awful.

It’s easy to convince oneself, to borrow a term from Taylor’s time, that pragmatists are [goldbrickers](#). Taylor and his ilk assumed that line-level laborers were fundamentally lazy, and they would find inventive ways to loaf unless diligently micromanaged. This is doubly true in a mental model where anything other than full attention for forty hours per week is “misappropriating company time.” But reality

is more complicated, particularly in a knowledge work economy. A Tayloresque micromanager could no doubt have success getting people to assemble more widgets per day by, say, passing a “no talking” rule. But this approach fails badly with knowledge work.

The real trouble for the pragmatists is that they are not invested in what they’re doing, and that investment is critical for knowledge work pursuits. It’s hard to blame them. They’ve ceded hope of meaningful advancement and influence within the organization, so, really, what’s in it for them? Why be invested? Better to invest in other things...like fishing.

The successful pragmatist is thus one that maximizes his non-work interests—the one that sucks the marrow out of life, so to speak. If he’s a family man, this means success is providing ably for his family. If he’s a fishing enthusiast, success is earning enough money and PTO to take some seriously cool fishing trips. Work and the corporation are thus not part of the success equation for these folks, except in a supporting role.

By and large, this explains both the risk aversion that prevents pragmatists from trying their hands as opportunists and the perspective that prevents them from falling into idealism. Work sixty-hour weeks for the hope of eventually being considered for a promotion to a role that expects sixty-hour weeks? No thanks. I’d rather be fishing.

This isn’t to say that pragmatists don’t try at work or that they don’t do their jobs well. If you came along out of the blue and offered a raise and promotion to a pragmatist, the answer would likely be “yes.” Similarly, if you asked a lot of them whether they took pride in their work, the answer would likewise be yes—at least during the time they compartmentalize for the office.

The essence of pragmatism is not a cynical, bitter battle with the company to work as little as humanly possible. Rather, it’s a reluctant but familiar equilibrium in which the company and the pragmatist find one another to be entirely predictable. The pragmatist aims for work

to become part of the routine, like flossing, jogging, or cleaning the house. All of those are things that you might take a certain pride in and that you might derive a certain enjoyment from, but at the end of the day, they're the chores required as part of life's maintenance contracts.

There is a final piece of the pragmatist success puzzle, however, and that is absolutely critical to mention in a book whose audience is software developers (though this could apply to any line-level knowledge worker). In Part 2, I talked about pragmatists *exclusively* valuing outside concerns, such as with hobbies, family, and social lives. By and large, that has been and remains the case. But a growing trend is emerging, particularly in software. A growing contingent of us define success by our proficiency with hobbies *within* the workplace.

I am talking, of course, about programming. Any corporate programmer who intends to continue programming is ipso facto a pragmatist. Why? Well, "programmer" is invariably a line-level labor position. Being a laborer is fundamentally incompatible with the march from laborer to manager to executive to owner. Idealists are striving to be managers (well, theoretically executives, but that's not an attainable goal for them) and opportunists are striving to be owners. There are certainly idealist and opportunist programmers, but neither one is thinking to themselves, "Yup, I'm happy being a software engineer III for the rest of my life, with maybe the option for IV or V down the line."

Against the (office) political backdrop of career advancement, programming is a hobby. If you spend your days slogging through Java 1.5 codebases and go to Scala meetups on your own time, you're living this reality. To take the sting of that blow away just a bit, consider that professional programmers have engaged in a pretty impressive pragmatist life-hack: they've convinced a company to pay them to do their hobby. Not a lot of people with "I'd rather be fishing" mugs manage to convince Acme Inc. to buy them tackle and a boat

with a trolling motor. It's certainly a better deal than the pragmatists had during the Industrial Revolution and scientific management eras.

Nevertheless, the overwhelming majority of corporate programmers content in that role are indeed pragmatists. They value the thrill of flow and the feedback loop of programming over things like autonomy, ownership, and a will-to-power approach. Success for these particular pragmatists, then, becomes that of superior knowledge and practice: knowing the ins and outs of Docker or ridiculous numbers of Regex. They earn a living, enjoy the trappings of life, and are free to pursue their own personal goals and to define their own non-work-oriented success narratives.

Pragmatist success is thus defined by two tiers of goal. The first essential tier is to maintain steady enough employment to sustain good standing on Maslow's hierarchy. The second tier of success is then largely self-defined. A pragmatist succeeds if she stays employed and feels good about her life and what she does with it.

Chapter 29: Idealists Succeed with Merged Identity

If defining success for pragmatists was a little weird, this chapter is going to get a whole lot weirder.

For pragmatists, once table stakes are satisfied, it's pretty open-ended. There's not a great deal of external measurement. Idealist success, on the other hand, is pretty straightforward to measure—at least in theory. Where pragmatists define their success by mastery of hobbies and external pursuits, idealists define theirs by valuation at the hand of the company and their position within the same.

Where it gets weird is that idealists have a concept of success that will prove literally impossible for them to attain. Idealists, particularly those early on in their careers, will tell you that they're gunning for the C-suite at Acme Inc. Having been robbed of rational perspective, they fail to see that this is a fool's errand.

So how do we define idealist success? It's not a simple matter of saying, "It's impossible" and moving on.

To understand why it's more complicated, consider the subtle alteration in narrative for an idealist in the twilight years of his career. He'll say neither that he's still gunning for the C-suite nor that he thinks his career was a failure because he didn't make it. Instead, he'll pivot to a tale of many good years of service, with ups and downs, that made him the person that he is today.

What he's really nibbling at with this retirement speech is the fusion of his identity with that of the company. The idealist will look around the podium at his retirement party, beaming, and regale you with some of the most company-man stuff you've ever heard. If you're not an idealist yourself, you'll probably fidget with your catered lunch

salad in vague discomfort as you wonder whether, freshly retired from the company, he's just going to drive home in his Cadillac and lay down and die.

Idealists early in their tenure with a company would claim that their goal is advancement to executive positions, but that's really because they believe in the infallible corporate meritocracy. Of course, they *would* actually value the perks of being the big boss. I mean, what rational human wouldn't take more money or more power if offered freely? But that's not the core essence of what they want out of a prospective promotion. What they really want, deep down, is acceptance and approval.

Becoming a C-suite occupant means admittance behind the most exclusive of velvet ropes. It means access to the inner circle of advisors who (they believe) drink as deeply of the founder-legacy kool-aid as they do. It means champagne lunches where (they imagine) they can toss around the full lexicon of Acme Inc. insider buzzwords. It means (they believe) putting in long hours and asking for nothing in return but the promise of eventual inclusion in the inner circle. It means founder legend by osmosis. It means accepting payments in carnival cash.

When you then pull back to the young idealist and her goals, you can see the subtle difference between stated and actual. C-suite and promotions are a means to an end only. The real currency at stake is becoming a real honest-to-goodness Acme Inc. insider. That insider status is what success means in the world of the idealist.

Of course, as with the pragmatist, some practical minimum standards must be met. The idealist must not be so wildly incompetent as to be laid off. And they'll need to do some work in addition to regurgitating the company's values and mission. Idealists generally distinguish themselves by overperforming without reciprocity, but that isn't *strictly* necessary for them to succeed in attaining company-man status. They could put in only forty hours a week but be nauseatingly obsequious and enthusiastic about the

organization's higher purpose in life and still get where they're headed.

Success for the idealist thus becomes a matter of doing the work in front of the right people and doing it in such a way as to demonstrate their utter enthusiasm for and gratitude toward the company and their superiors. It involves willingly donating the best years and mental effort they have to what amounts to a corporate tithe above and beyond their expected work input. All of this happens for the duration of their careers, and the whole effort is successful as long as they can keep "I'm a very important person at Acme Inc." as a core piece of their identity.

Chapter 30: Opportunists Become Other

In a sound bite, idealists succeed by fusing their identities with the company, and pragmatists succeed by decoupling theirs from the same. So how do opportunists succeed? *That* is the interesting question in the world of corporate realpolitik.

In the most superficial sense, opportunist success is easiest to define. On the spectrum from grunt to manager to executive to owner, the further to the right you fall, the more successful you are. Beyond that, opportunist success equates most heavily with broad societal recognition of success: wealth and power. But you didn't buy a book to hear someone say "corporate success is working your way up." We're going to need to dive a lot deeper. And while defining success for opportunists is easy, it's going to require the remainder of Part 4 to examine how to achieve it.

I read the essays that would become [Venkatesh Rao's *The Gervais Principle*](#) a few years before I took up residence in the CIO's office, and I would re-read them periodically as I worked my way there. Recall that during this time I was mildly obsessed with cracking the code of corporate advancement. The insight in those essays was powerful, and I consumed it greedily, wondering whether or not I was, in fact, a sociopath (opportunist).

I thought I was. I must have been, right? I mean, I was moving up and securing more power and influence. And I was doing so absent specific company loyalty. And yet, Venkatesh discussed a thing he called "[powertalk](#)" as the language spoken by and among the opportunists. "What distinguishes Powertalk," he explains "is that with every word uttered, the power equation between the two speakers shifts *just* a little." I was moving up, but was my every conversation shifting the balance of power? Maybe not *every* conversation. I started to wonder what I could be doing better, even

when I was sitting in the C-suite. Examples abounded in that and subsequent chapters of *The Gervais Principle*, but there was no how-to, no field guide for aspiring opportunists or even a way to find the “you are here” on the map of your office’s politics.

In retrospect, of course there wasn’t. This isn’t a language that you learn to speak so that you can secure influence, power, and your own goals. It’s a language that you suddenly find yourself speaking once you’ve secured things worth defending and are successfully defending them. Learning powertalk is thus a quixotic mission—the aspiring opportunist needs to learn to acquire power and the powertalk will follow.

In other words, you win the corporate game by becoming an opportunist. If this statement seems fatuous, consider that the advice isn’t “decide that you’re an opportunist rather than a pragmatist or idealist...profit!” The advice is to *become* an opportunist, meaning that the path to opportunism is not one of learning the right things to do or say for a given scenario, but rather it is a path of becoming a different person in the corporate context than you might otherwise be.

If you’re a programmer or engineer (or most other kinds of knowledge workers), you probably want a guide. That’s at the core of our enjoyment of construction, invention, and innovation—building frameworks within which the right action leads to the right outcome: “If a VP says X to me and I respond with Y, then my realpolitik power will increase by three points.” So the rest of Part 4 is dedicated to providing concretions for you so that you can understand this transformation in more tangible terms. Of course, even that will only take you so far. You can’t construct a series of conditionals and heuristics that will take you to the promised land. You need to fundamentally alter how you regard yourself at your office—you need to *become* an opportunist. The rest will follow.

In the novel [*Crime and Punishment*](#), a student named Raskolnikov reasons that society consists of the common man, who plays by the rules, and the extraordinary (e.g., Napoleon Bonaparte), to whom the

rules of society simply do not apply. He makes a ham-fisted attempt to define himself as one of the latter by killing a woman he believes to be a drain on society, and let's just say that it goes poorly for him. To have it go well for you, learn from his mistake. Aping behavior is not sufficient. Napoleon wasn't extraordinary because he could order deaths and get away with it. He could order deaths and get away with it because he was extraordinary.

There's a lesson for us in Raskolnikov's approach and struggles beyond "you shouldn't kill people." Becoming an opportunist isn't a matter of embracing cynicism. It's not deciding that rules don't apply to you and shoving off all loyalties, nor is it any kind of general hardening of spirit. I believe one of Dostoyevsky's points with this book may have been to counteract the standard trope of a character who is only truly able to reach great heights by deciding to become a cutthroat, mercenary type. Deciding to become a mercenary will simply make you a mercenary. It won't make you a CEO.

At its core, becoming an opportunist—becoming extraordinary—is about becoming *other*. But that's extremely abstract. Let's put some meat on the bones.

This transformation into an opportunist is, at its core, about altering your perception of yourself. You need to stop viewing yourself as a software engineer II or a QA specialist or a dev manager. You need to stop viewing yourself as an employee of your (or any) company and start viewing yourself as the owner of your own personal brand and operation. You are an island. You are *other*.

This may seem like a silly mental exercise at first blush, but its impact is profound. If you're a software engineer, your boss asking you to put in a few fifty-hour weeks ahead of the upcoming release is perfectly reasonable, if something of a bummer. But if you're a free agent, your client asking you to work ten hours a week for free is perfectly preposterous. The difference in thinking is that it's now reasonable is for you to ask, "Why would I do that? What's in it for me?"

More likely than not, an opportunist in this position would wind up putting in those hours anyway. After all, he is a business with a single client, and that client is negotiating down the price of his labor. He really lacks the leverage to say no, so he says yes. For now, anyway. But he also files that away and endeavors constantly to acquire more leverage and to improve his bargaining position.

Once this is your mindset—once you're really living, feeling, and breathing it—your conversations will begin to consist entirely of power talk. You are your own tiny company, buffeted on all sides by larger, more powerful organizations and alliances that could seriously ruin your day. Every conversation becomes one of potential consequence with the opportunity for you to advance or lose ground with regard to your interests.

This also explains why, as I claimed earlier, opportunism is a lonely pursuit. Pragmatists have each other and their social groups at the bottom rung of the company. Idealists have the company and bask in its identity. Opportunists have no one. Once you become an opportunist, you understand your fellow opportunists for what they are—not agents of the company but other one-person companies with their own interests to consider. Theirs may or may not align with yours at any given moment, but there is no built-in support network for you. Until you strike out on your own in earnest and start your own company, you're the only one in your corner (and even when you do this, no one will really be in your corner until your company acquires its own idealists).

It's worth stopping at this point to take a breath. Coming to regard yourself as your own tiny, lonely company is your true step away from the cave wall, and there is no going back. What follows is the real, honest key to meteoric advancement at an organization. But it's also ethically murky and high risk. It prioritizes advancement over everything else in your corporate life. Abandon all hope, ye who enter here. In the fundamentally flawed corporate institution, the road to advancement is also fundamentally flawed.

What you'll read in the remainder of Part 4 is not advice. And I ask that you not mistake it for advice. This is not a LinkedIn or BuzzFeed article about the ten ways to shine your boss's shoes so that he'll love you at career review time. Rather, this is a high-risk blueprint for how to wind up in the C-suite. You'll find your way there, if you can stomach it. But you'll also probably lose a job or two along the way and do some things that keep you up at night. And you will absolutely do things that cause people to look at you and think, "That's not FAIR."

Chapter 31: The Realpolitik Tragedy of Corporate Scrum

Assuming you've abandoned hope and entered here from the last chapter, you're probably anxious for the parts where I lay out the success blueprint with concrete examples. Don't worry—those are coming. But first we need to lay out some failure blueprints for clarity and contrast.

After all, commerce consisted only of opportunists for nearly the entirety of civilization's history. It wasn't until Industrial Age merchants put serfs to work as wage employees that we acquired pragmatists, and it wasn't until Taylor tapped slightly more genteel grunts to manage the human chattel that we had idealists. It should be most natural for us as humans to default to opportunist mode, and yet we don't. Somehow, we fail to do that and settle instead into one of these two other unfortunate archetypes. It's essential to know why that happens and why not doing so feels wrong before we can really get into how to do what feels wrong and not fall into these traps.

As a lead-in for describing these opportunism failure blueprints, I'm going to introduce a few brief allegories. It's all too easy to view your situation and the choices you make as inevitable, so opening yourself up to a different view sometimes requires a little jostling.

For the first allegory, imagine that I'm an entrepreneur. I hear about a change in tax laws for some remote island nation that guarantees a glut of wealthy tax-avoiders looking to move down and buy houses. Being a remote island, however, the location does not currently offer a lot of houses. So I decide I'll move down and get into the housing construction game in spite of not knowing a lot about the industry.

Once I get down there, I discover that I'm not alone in having this idea. Competition for the limited number of available contractors on the island is pretty fierce. I get lucky and find one that likes my vision and the idea of being "boss contractor," and I'm in business. Figuring that he knows better than I the details of vetting and hiring contractors, I delegate hiring the rest of my crew to him but watch the process with interest.

He interviews a few people and I'm inclined to make them offers immediately, but my foreman tells me they're bums and not worth the \$50K salaries we're offering. "They couldn't even draw a perfect pool drain on a whiteboard, let alone know the exact water to chlorine mix for a pool in this climate." Confused, I tell him that we're not even going to be building pools, just houses, so who cares about this?

To my bemusement, he tells me, "Well, if you want to get contractors to come work for you, you should build pools along with the houses. We like doing it, so you don't have to pay us any extra."

"Okay," I tell him, "but why are you turning people away if they lack pool skills? It's really about building houses."

His answer surprises me. "For a \$50K salary, any professional house builder should be an expert in building pools. If you want to hire people that aren't, let's just offer them \$20K and tell them that they're lucky to be working somewhere that they get to learn to build pools. It serves them right, the idiots."

"Will that work," I ask, incredulously?

"Oh, absolutely."

Man, I love this country.

The second allegory probably earns a bit more literary cred because it involves animals. Let's say that somewhere in the world is a vast wild forest, untouched by humankind. And at the center of that forest is Wolf City.

In the wild forest, wolves hunt by themselves or in packs. But in Wolf City, there is no organizational structure so vulgar as that of the forest. There are no lone wolves or packs of wolves. Rather, there are packs of packs. And packs of packs of packs. There are packs who do nothing but oversee other packs to make sure that they comply with wolf regulatory concerns and standards and protocols. And order is maintained in martial fashion, with commands being handed down from the mayor of Wolf City in a tree-like structure until it reaches the leaf-packs, who are responsible for hunting to provide food.

A curious thing happens, though. As Wolf City expands outward and grows in population, the line-level packs have a harder and harder time with hunting and providing the city with meat. Increasingly, the city outsources the hunting to individuals and packs of the wild forest. This is necessary because it's common for the city's line-level packs to hunt for days in the surrounding forest without catching anything. Sometimes they even fail to leave the city during these hunting trips, spending all of that time arguing with various wolf oversight committee members about how best to comply with hunting efficiency standards.

The mayor of Wolf City perceives this hunting efficiency gap between his city packs of packs of packs and between the lone wolves and packs of the forest. He starts to do some research, and what he finds is that the forest wolves have an entirely different way of operating. In fact, he finds that they've evolved an entire hunting methodology that was started when high-status representatives of those packs ascended Wolf Mountain far to the west and crafted a hunting manifesto. Codifying these principles allowed the efficient forest wolves to explain the secret to their success.

Encouraged, the mayor of Wolf City summons a pack of these enlightened forest hunters to help Wolf City improve its woeful hunting operations. "Please, share with us the secrets of your success," he entreats these hunters. And for a fee, they do.

The summoned pack has somewhat legendary status in the forest for its hunting prowess. This pack trusts one another implicitly. In fact, this pack has actually ritualized its trust expression into the form of ceremonies. In a ceremony entitled the “subordinate dominance ceremony,” the wolves of this pack all take turns rolling over on their backs and showing their bellies to their pack-mates. Only through demonstrating this level of trust can they achieve the level of pack collaboration necessary to bring down large game every time.

In traditional wolf packs, belly-showing is a sign of pure subordination. Wolf packs are held together with the glue of social hierarchy, established and upheld by a system of dominance stack ranking. There is an alpha, a beta, and so on, all the way down to the omega. But in this enlightened pack, all wolves are equal. All are alphas and omegas, and they achieve dominance through voluntary subordination.

The enlightened pack enters the city and installs among the line-level wolves the trust that has always been missing. They show them the subordination dominance ceremony and turn each and every one of the city’s hunting wolves into subordinate dominators. These wolves, who used to waste excessive energy on infighting, are now efficient teams, slowed only perhaps by mildly excessive belly-showing. But the trust instilled more than makes up for it. It is undeniable that the newly trained Wolf City packs are yielding better results than ever before. Wolf City, like the enlightened pack that taught it, becomes much better at hunting.

Almost as an afterthought, a curious trend emerges in Wolf City. Historically, wolves had ascended into the leadership structure of the city by dominating their packs and then being tapped for the next level of command. But in a city where the model wolf is now a subordinate dominator, that no longer works. Being a good hunting pack member means starting each day baring your belly to the rest of the pack. This renders the group efficient at hunting but the individual inefficient at advancement.

A curious trend emerges. Performing well as a hunting wolf no longer leads to advancement. Whereas in the past, performing well was a matter of sheer physical dominance understood by superiors, now, performing well is a matter of fitting in as a cog in the broader pack. Pack members now aim to distinguish themselves by being the best darned subordinate dominators they can be, showing more belly than anyone else, giving more of themselves, and fortifying themselves in the super pack's teachings. But ironically, this appears not to work for proving themselves. In fact, it seems that some of the least gung-ho team members—the lone wolves—now start to advance.

It turns out that, for all of their enthusiasm about the teachings of the enlightened pack, the mayor of Wolf City and his vast wolf bureaucracy have not fundamentally altered their own perception of wolf nature. They continue to regard winning fights as grounds for advancement and rolling over and showing bellies as signs of surrender. One might say that it's built into their blood and that it's built into the structure of the city. The city itself prospers when the line-level wolves embrace the teachings of the enlightened pack, but the best line-level wolves themselves do not similarly prosper. This new hunting style only works when the pack is more important than the individual.

Two allegories in, we can now get to the opportunism failure blueprints in earnest. The interviewing contractors are pragmatists and the “subordinate dominator” wolves are idealists, but what does it all mean? Let's dive in.

Frederick Taylor, described in Part 3, figured out that whipping the pragmatists wasn't necessarily the best way to extract more utils from their work. Sometimes, they needed breaks, encouraging words, and other slight nods to dignity. But for Taylor and other scientific management disciples, this was not a recognition of human need. It was rather a recognition of efficient beast-care. Be nice to the cat because it's more likely to come to you when your voice is soothing than when you sound angry. So it goes when dealing with

pragmatists. Treat them with dignity so that they get more work done.

In some ways, little has changed since Taylor's (and Edison's) time. Corporations interview and manage in basically the same way. They conceive of work in the same way. Even as the rise of knowledge work transfers us from a manufacturing economy to a global, digital economy, they more or less do things the same way across the board. There is, however, a notable exception—a single way in which the modern corporate workplace has evolved. This exception is autonomy, or the illusion thereof.

Taylor couldn't have scripted the story better himself. It turns out that when you get high enough on [Maslow's hierarchy of needs](#), things other than money start to motivate you. Knowledge workers tend to be well compensated, so they chase new, more subtle goals: [mastery, autonomy, and purpose](#). The modern corporation has not adapted in many significant ways, but it has adapted enough to sustain a status quo where knowledge workers can serve as grunts, the way their manufacturing floor forebears did. The corporation has done this by creating the illusion of line-level autonomy. Gone are the times when a manager like *The Jetsons'* Mr. Spacely harangued a knowledge worker like George Jetson. Instead, the modern knowledge worker can come in anywhere between 8 and 10 AM, wear jeans or other appropriate leg-covering garments, and even, with the proper clearance from on high of course, work from her own house every now and then. What a time to be alive.

What's the significance of autonomy, and why am I suddenly talking about that alongside the failure blueprints? I'm talking about autonomy because it's the essential difference between the forest wolves and the city wolves—between consultant Scrum and corporate Scrum. The former have actual autonomy while the latter have illusory autonomy.

The text now known as the Agile Manifesto was written in 2001 by a group of influential software professionals. They were people that were writing books, speaking at conferences, helping major

companies, and defining the state of the art for the industry. In short, the Agile Manifesto was written by a group of people with *options*. They, like the forest wolves, were self-sufficient. When they introduced the democratic, highly-collaborative, often-role-less agile teams to the world, they did so from a position of strength. But if those companies—Wolf City—had started ordering them around like grunts, they could simply have left and taken their pick of other gigs.

In a very real sense, what you had with the signatories of the Agile Manifesto and with the other early adopters of those practices and processes were groups of highly autonomous, highly skilled consultants. These consultants created the agile movement by explaining the sorts of processes they followed, the sorts of beliefs they held, and the sort of principles that guided them. The industry seems to offer unquestioning acceptance of the premise that these processes, beliefs, and principles made the consultants successful without considering that these consultants would probably enjoy success with just about *any* process since they were really good at what they did. Sure, they're probably all the better for using the methods they had developed through trial, error, and common sense over the years. But these guys were good because, well, they were good.

As the agile movement gained legs, it created a cottage process industry. Some signatories and early adopters got into the business of selling process to companies and individuals alike, trading in training, certifications, and corporate transformations. This had a subtly sad effect in the sense that it shifted the focus from, "You will succeed if you practice and get really good at software and consulting" to, "You will succeed if you have these meetings, follow these protocols, and adopt these techniques." The latter is certainly a much easier sell, but it's also a much easier path to disappointment and mediocrity.

As the years passed, this burgeoning process industry gave rise to paradox. Corporations adopted agile (usually Scrum) in the way that they had adopted other management fads over the years, such as

formal SDLC, RUP, and others—as a top-down management initiative designed to drive efficiency. “You will do this now, peons; let it be so.” But at the core of even the most corporate flavors of agile is a call to dispense with formal hierarchy and roles, to trust the team of knowledge workers and to grant them autonomy. The most basic paradox of corporate Scrum is thus the edict from on high to be autonomous, but not so autonomous as to stop listening to edicts from on high.

In a development that would be somewhat comical if it weren’t so sad, these “You will now do Scrum” teams are usually measured with a great deal of scrutiny. It’s ironic because the lack of trust subverts the foundational autonomy that agile methods call for, and it makes team mediocrity a self-fulfilling prophecy. The company imposes micromanagement on the team to make sure the new agile rollout is going well, and that micromanagement makes the team worse. As the team gets worse, the micromanagement increases. This situation tends to continue and create churn and attrition until someone comes in and finally wins the battle to secure a tiny bubble of semi-autonomy for the team. At this point, output increases and companies are usually somewhat happy. But that happiness is short-lived, as the newly efficient city wolves often like the taste of autonomy and proceed to run off into the forest.

The tragedy of corporate Scrum starts with this autonomy paradox. Scrum is an excellent methodology for the forest wolves who assemble and hunt as self-sufficient peers. If free-agent knowledge workers come together to work on a project and they adopt Scrum, life is good for all. The methodology prioritizes common sense collaboration; tightens feedback loops; gives all team members equality and a voice; and generally makes it easy to manage, track, and pick up work. Scrum is *not* an excellent methodology for people who possess only illusory autonomy. In this context, it’s a farce. In this context, it’s realpolitik tragedy for its practitioners.

As an aspiring opportunist, corporate Scrum is the epitome of a failure blueprint in the same way that belly-showing is a failure

blueprint for opportunists in Wolf City. When you join a large agile company as a developer, you'll probably receive a copy of the Scrum guide along with some training in the ways of agile. You'll see your open plan office space, you'll learn how the team is highly collaborative in the team area/bullpen, you'll come to understand that they depend on you and you on them, and you'll learn that the highest performers are the ones that know agile in and out, live it, and breathe it. If you believe in this, the idealists have a nice, warm seat for you in a meeting room with bean bag chairs.

Participating in agile methodologies as a line-level corporate employee sends intense subordination signals, just as belly-showing does in Wolf City. At daily standup, you're required to report your status every day to be held accountable. You're pairing with another developer at all times to ensure that you stay focused and don't make mistakes. A former project manager converted into a "Scrum master" comes around to make sure you have the meetings that you're supposed to and that you don't get distracted by people from "the business." Every week or two, you demo your progress to someone in management called the "product owner." When each cycle of this is done, you do something called a "retrospective" and voice your feels about the whole thing to the group. If you're doing this as part of a startup that you and a few developer friends co-founded, it's empowering and emblematic of trust. If you're doing it because upper management has determined you will, you seem like a secretary to anyone looking. That means that you're failing at opportunism and failing hard.

Close your eyes for a moment and imagine a boss. Does the boss account for yesterday's and today's work to his peers and superiors every day? Does anyone "pair" with the boss? Does the boss have a "boss master" telling him how to behave during meetings? Does the boss write his phone number and upcoming days off on a whiteboard? Does the boss "demo" things to his boss every week or two? Of course not. These are activities better suited to an intern. If you're doing these things, your behavior is so un-boss-like—so subordinate—that it will be impossible for opportunists to view you as

one of them. If you do it with resignation, you're a pragmatist. If you do it with enthusiasm, you're an idealist.

The more enthusiastically you participate in agile methodologies within a corporate context, the worse your advancement prospects. Sure, all of your enthusiasm will eventually earn you some kind of token nod or promotion, but it is purely idealist-track. You'll top out low and what you do achieve will be agonizingly slow. All the while, you'll be mystified as to why your belly-showing doesn't result in people regarding you as dominant. The reason is that the so-called "servant leadership" valued in corporate contexts is an utter sham. If you think I sound harsh saying this, ask yourself how "servant leaders" get that title in the corporate world. Do you think they spit shine the floors until someone comes along and appoints them CEO? Or do you think they get appointed CEO and then do some token spit shining of the floor to seem humble and to hoodwink the resident idealists into thinking that shutting up and grinding will lead to leadership positions? Servant leadership is a great concept in populist contexts where "lead by example" can define an eventual hierarchical structure. When the hierarchical structure is already defined, "servant leadership" is an insult to opportunist intelligence.

Software folks radiate enthusiasm for methodology (that the organization so adeptly captures and perverts) more than just about any other category of knowledge worker. To put this another way, software developers settle into idealism for far less than other workers. Most employees demand more in the form of carnival cash or even real consideration. They tend to shoot for line manager positions with direct reports, offices, and perhaps a company credit card. We, on the other hand, settle for meaningless titles like "architect" or "tech lead," bragging rights about technical acumen, final decisions on which unit testing framework to use, and the right to gleefully sharpshoot prospective candidates during interviews.

It's bad enough that we team up with our fellow developers at corporations to create and perpetuate subordination factories. It's

worse still that we become idealists so easily that we export our subordination to the broader industry.

This brings me to bookend the chapter by revisiting my first allegory. In it, I hired an idealist who low-balled cowed pragmatists that didn't understand their market worth. This happened on a hypothetical island, but the reality is that it's happening all around us, most prominently at organizations like Google, Amazon, and Facebook. These companies are building pools, and we think it's so cool to be building pools that we put up with absurd interviewing processes and subject ourselves to false scarcity, depressing our wages. There is a massive shortage of developer labor, and the world needs all hands on deck. Yet we operate as if a good programming job were a scarcity that we'd be lucky to have.

If you think that sounds overly dramatic, consider that a few years ago, Silicon Valley giants including Google and Apple [settled an antitrust class action lawsuit for hundreds of millions of dollars](#). The reason for the suit? These companies had entered into off-the-record agreements not to hire one another's employees so as to prevent bidding up the wage of developers. Imagine for a second that CEOs of two major cable companies in your town met for martinis and decided not to accept one another's customers, eliminating competition and forcing customers to pay whatever rate they set. Imagine your outrage. Imagine the blowback. There would be boycotts. But when Google, Apple, and others create a cartel to depress developers' wages across the board, we keep studying up for our algorithm trivia interviews and hoping to make the cut. It's hardly even news.

The tragedy of corporate Scrum is that the very behavior that gets us branded as good developers also limits our career prospects and forces us to remain in subordinate positions. This tragedy is compounded when it bleeds through to the job search and interview process. There, it becomes career-limiting not just at one organization but everywhere we go. Let me repeat that more plainly. Being a good developer—participating gamely in team activities,

learning enough algorithm trivia to pass interviews, being attracted to the best organizations using the coolest tech, and so on—is bad for your career. The real tragedy of all of this is that the current corporate structure forces you to choose between being a good developer and having a good career.

Now you have a firm grasp on the failure blueprint for corporate opportunism. You fail when you show up and dutifully perform at trivia interviews. You fail when you make yourself an open book and indicate the need to account for every detail of your work day. You fail when you raise the organization above yourself. You succeed when you stop doing all this and become other—when you start treating your employer not as a larger group with a larger cause, but as a temporary business partner that is an equal.

Chapter 32: The Programmer's Escape Plan

I want to be very clear at this point. If you're a programmer seeking to be an opportunist, you need to start figuring out how not to be a programmer in the near future. You need to plan for your graduation. If that's unacceptable to you, then you need to settle back into pragmatism or stop working for standard corporations. Opportunism and programming are only compatible in less traditional contexts, such as free agency (much more on this in Part 5).

I'm not saying you shouldn't *program* at work. That's a fine (and, for many of us, fun) thing to do in your spare time. Even programming while at work doesn't have to be career limiting. I've known CIOs and dev managers that occasionally write a script or little program to help themselves somehow. Rather, *being a programmer* is career limiting.

In Part 3, I described how Taylor and scientific management gave us the idealist in the form of the "educated laborer." This rounded out the modern corporate construct of opportunist-idealist-pragmatist, or owner-manager-grunt. Unfortunately, the position of "software developer" developed as and remains a subspecies of grunt. With more than 100 years to calcify, Taylorism and its logical implications have nestled inextricably into the corporate condition: owners and executives hire lackey managers to boss around worker bees. And last I checked, "senior software engineer" or "tech lead" are neither owners, executives, nor managers.

I spoke earlier about Scrum and groveling-centric job applications as forms of corporate belly-showing behavior. But the truth cuts even deeper. Simply showing up to work each day and being a programmer falls into that same category. Servant leadership and team-first behavior in line-level positions is the stuff idealists are made of, but showing up and being a programmer (without an exit strategy) can characterize both idealists and pragmatists. It

categorically does not include opportunists. Opportunists are busy figuring out how to stop being considered programmers, even if it means potentially getting fired.

Let's talk about why an opportunist does this intuitively, even if that opportunist loves programming. It's an innate grasp of the realpolitik shortcomings of pragmatists and idealists.

Up until now, I've talked about the corporate pyramid and archetypes in terms of generic workers. Pragmatists, idealists, and opportunists exist in every type of role within an organization. But let me now get developer-centric. What I'm about to say may well apply to other knowledge work pursuits, but it seems disproportionately common in the developer world.

Pragmatists are pragmatists are pragmatists. Developer pragmatists offer no exception. They are content to show up, work as much as necessary to retain their comfortable jobs, and then go home. They aspire to nothing in the corporate hierarchy and figure to spend their lives programming, switching jobs periodically for a bit more pay or when the opportunity presents itself without too much hassle.

Where things grow more interesting is with programmer idealists, who you might think of as junior idealists, in a manner of speaking. As we've discussed, developer idealists don't even require any significant advancement or marginal pay to feel as if they have arrived. Often, you can just call them "principal" and put them in charge of the coding standards document. If they ever actually make it to a real management role, then they start to look more like generic, over-performing idealists.

Programmer idealists can be found in two flavors. The first is the [expert beginner](#), who really embodies the junior idealist concept. He conflates earning his position by attrition with earning it by merit. And he will force you, the newbie programmer, to use his weird coding standards and dilapidated, homegrown frameworks. The second kind of idealist is curiously developer-specific and confounds the

established archetypes by actually changing companies a fair bit. This character I will call the journeyman idealist.

I plant my tongue somewhat in cheek as I evoke the guilds mentioned earlier, but only somewhat. The title does perfectly describe the mentality. The journeyman idealist swaps faith in the company for faith in the nebulous concept of the technical meritocracy. His company is the programming industry in general, spread across all companies and domains.

This journeyman idealist will bounce around between companies, attracted to shifting “tech stacks,” cultures, and assembled programmer cliques at different organizations. He will accept with reverence the algorithm trivia interviews at major tech companies. If he fails, he accepts that he has work to do, but if he passes, he feels that he has been certified in the industry. He is the island contractor obsessed with pool construction.

It is, perhaps, a testament to the changes that have occurred within the corporate world over the last thirty years. In most ways, the corporate world has remained constant since Taylor’s time. But in some superficial ways, changes have occurred. One predominant example is the dramatically reduced tenure of the average employee. Rather than settling in with a company for life, today’s workers assume that they’ll spend five to ten years with a given company. For software developers, this number shrinks substantially. These days, a software developer probably lasts an average of two or three years before moving on to greener pastures.

It’d be reasonable to assume that this trend would shatter the idealist mentality, but reality is more interesting. In the world of software development, idealism has become something of a co-op. The journeyman idealist does not believe in the folklore of the company but rather the folklore of the software development industry itself. He signs manifestos and deifies the authors of said documents. He accepts that, for some, pecking order is tied not to memorization of corporate pantheon, but memorization of software industry pantheon. He trades the corporate carnival cash of nicer cubicles for

industry carnival cash of higher Stack Overflow scores. He convinces himself completely that applied programming skill translates directly to profitability when this idea is, in fact, absurd. He cedes perspective just as surely as his corporate idealist counterpart does.

You're probably now wondering why, after offering an explanation for the behavior of opportunists, I talked at length about developer non-opportunists. I did so because I needed to explain to you the anatomy of the programmer's bottom-heavy place in the corporate world. If you have a line-level position in account management, sales, or operations, pragmatists sit next to you, idealists above you, and opportunists above that (or else on their way). If you're a programmer, you have two nearly orthogonal sets of idealists above you, effectively creating a gigantic mesh of downward pressure.

As we've discussed, line-level pragmatism is generally a poor economic deal. And it's a terrible one for software developers. Garden variety application development work outside of the high frequency trading space can go for as high as \$150 per hour on the market, and I've rarely seen app dev consultancies sell it for less than \$100. This means an effective compensation of \$200K to \$300K per year, which in turn allows an awful lot of room to vary developer salaries. Entry level developers may start out as low as \$40K per year and range up to \$150K per year. That's an *amazing* range for a line-level position.

This range exists because we, as an industry, are much better at nitpicking amongst ourselves over stack ranking algorithms than we are at capitalizing on our own market value. Predictably, we don't sweat the question of "Why aren't we making \$200K to \$300K?" opting instead to create gigantic hierarchies as an ex post facto justification for our gigantic salary range. Some organizations have software engineer I through as high as VII. On top of that, there's a mixture of fluff titles like "senior software engineer," "principal software engineer," "software architect," "tech lead," and the like. I can't recall ever seeing an organization without at least four levels

(salary bands) of software developer. That seems to be some kind of unspoken industry minimum.

As an aspiring opportunist, the problem starts to become obvious. For me, my first intuition of it, without being able to articulate it concretely, was working for a company with five levels of software engineer and realizing that it took five years to achieve each level. Another inkling of it came when I switched jobs and had a hard time jumping too many of these levels. I was slated to go from software engineer I to software engineer IV at the new company, but those interviewing me decided to dock me a level because I hadn't programmed previously in the new company's language of choice at the time. (Having done so was not a requirement for applying.) I picked up the language quickly and was proficient, leading me to muse about how my career at that point had been largely impeded by "You haven't hung around this company long enough" and "You don't meet the sacred idealist's rigorous twenty-five-point plan for interview approval."

A programmer faces a preposterously uphill battle for advancement, done through normal channels. With most non-programming positions, you take the role, work hard and pay your dues, and eventually get promoted to manager of whatever it is you've been doing. But with programming, you have to run the gauntlet of numbered software engineer positions, earn designations like "senior," then acquire leadership-y titles like "architect" and "tech lead" before you even *get* to the point of "Do it for a few years and then become a manager." It's extremely uncommon to be able to bypass the journeyman idealists or the corporate idealists, much less both.

The normal line-level opportunist's play is some variant of up or out. For instance, perhaps he takes a line-level role and immediately starts making appeals to some bigwig three layers above him in the org chart. If this works, he's on the fast track. If he gets dinged for going outside channels, he simply packs up his toys and leaves for another sandbox. He does things that will either lead to quick

advancement or quick exit, the latter of which he will try to parlay to their advantage with, perhaps, a more-than-lateral transition.

For programmers, there is no up or out because “up” is determined by journeyman idealists for a long time. And there’s no shortcut to their favor—you need to hone your chops, solve their riddles, and best them in a joust of algorithm trivia. You can try to make up or out plays, but it really just turns into “out.” And even though you’ll probably get a pay raise, there’s no given that the next situation constitutes an improvement in standing.

The opportunist has a [will-to-power attitude](#) toward the organization. The opportunist programmer is one that wants to call the shots, the way that a department head or CTO does. And when she sizes up the situation earnestly, she comes to understand that this occurs neither through establishing herself through superior programming knowledge nor through acquiring superior carnival cash. It happens some other way.

The journeyman idealist is eternally mystified by “the suits”—those folks that make the decisions. Go into any sufficiently large programming shop and you’ll hear the philosophical musings of a journeyman idealist. He’ll hold forth (intelligently) on how the productivity boost from buying the dev team extra monitors would let management recoup a return on investment within the first month. He’ll know which programming language would be best to use for the upcoming project and why it would have been better to retire a legacy application six months ago instead of letting it limp along. Yet he’ll be unable to articulate why a person in a suit from “the business” is making these calls instead of him, except for the self-indulgent narrative that he wouldn’t want to sell out.

The budding opportunist quickly comes to understand why the journeyman idealist fails to call the shots. It’s because he’s a programmer, and programmers don’t call the shots.

This in and of itself isn’t especially revealing. Pragmatists, idealists (both flavors), and opportunists alike could tell you that programmers

don't occupy organizational leadership positions and that programmers wanting to do so need to aim for management. But only the opportunist understands that the shortest distance between "junior programmer" and "CTO" isn't through the wide-band spectrum of software development positions. The opportunist bends corporate space-time to allow a shorter path.

In my case, I came to it via good fortune. I was slogging my way along the software engineer roman numeral line when I happened to arrive at "senior software engineer." The next position I took, however, had the job title "senior consultant." And that has made all the difference.

An opportunist view of the programmer condition reveals a sad picture. Some people go to school for computer science, earn computer science degrees, and get jobs as junior programmers. Other people go to school for business, earn BA degrees, and get jobs as project managers. Within a couple of years, the project managers have "dotted line" authority over their peer programmers, since the programmers are another resource in need of management. It's classic Taylor. The opportunist programmer thus learns that it's better to hang out with the project managers and distance herself from her fellow developers.

This vague plan of escape does not demand immediate action or a title change but rather an attitude shift. The goal becomes finding ways not to participate in the software development process; instead, you must manage it somehow. It might be volunteering to represent the software group when talking with the business analysts or serving as the Scrum master. But it might also be looking for roles with titles that sound vaguely managerial or simply ambiguous (like senior consultant). With these types of roles and responsibilities, the programmer can stake a case for leadership positions in a job interview. In contrast, think of a resume full of programming language alphabet soup belonging to someone with a propensity to talk only in terms of algorithms and the like. These are hallmarks of a

journeyman idealist, and no one would be fooled into thinking there was a leadership position in his near-term future.

But there's more to the escape plan than simply targeting non-programming roles and responsibilities. It's in the lexicon and the manner of thinking. You need to learn the language of the business and talk intelligently in terms of profit and loss, return on investment, and the other terms relevant to the company's big picture. You also need to swallow whatever joy you extract from correct, elegant implementations and adopt a willingness to sacrifice Cadillac quality for time to market. You must sell out and *believe* in selling out, taking your joy of working with the tech and completely compartmentalizing it. Fun with toys is for outside the office. Programming is not a calling, and it's not a craft. It's just automation that increases top line revenue through product or reduces bottom line costs through efficiency.

Once you've steeled yourself to thinking that way, the opportunities to move toward leadership will start to become obvious: who to talk to, what to say, and how to say it. Build the resume that you'd need in order to apply for a dev manager job and work backward. Find out how to get yourself in your company's interviewing process. Volunteer to liaise with the project manager closely so that you can list project management activities. Same with the business analysts.

All of this is not going to be to advance your fortunes within your current company, of course. This is the company that brought you in as a line-level programmer, and getting them to see you as anything but that will be nearly impossible. Rather, you're preparing for your next interview—the one in which you'll go from a title that involves “software engineer” or “developer” to one that doesn't. The next move has to be significant in terms of advancement (architect), something involving leadership or management (tech lead, project manager) or something vague enough to serve you well (consultant, associate).

Once you're in a role like that, you can always leverage your technical background as much as possible to further your interests

and career. Many CIOs/CTOs come from a programming background, as do many peripheral and line management types. But in order to get to those roles, you need to first step out of the software engineer assembly line and then step back in somewhere much higher up the food chain. And that first step out requires an escape plan.

Chapter 33: Avoiding the Delivery Trap

Assuming you're still on board the opportunism train, the key lesson from the last chapter is to find a way not to be a work-a-day programmer. The easiest way to make this happen is to make a lateral transition into a role where the title is something other than programmer (project manager, consultant—some kind of lead). And the easiest way to make *that* happen is to work backward. Evaluate what you need on your resume to get an interview for such a role, and then volunteer for those activities in addition to the responsibilities of your current role (or instead of, if you're truly not risk averse).

But let's deconstruct this play and get still more tangible. As I said, programming, per se, does not create problems for you. Being a *programmer* creates problems. You seek to be a leader and decision maker in the organization, and programmers are categorically not that. Sure, they may be experts in their particular niche, but if one wandered into the C-suite with ideas about how to improve the marketing strategy or assembly line efficiency, he'd be regarded as adorable and told to email the corporate "suggestions" email account.

If you're thinking that this requires more than vestigial Taylorism to explain, then you have the right of it. The ongoing corporate love affair with scientific management principles explains why you need to generate escape velocity from the line level to further your career, but it fails to adequately explain the meta-why: why does the "knowledge worker as grunt" model persist so stubbornly, in spite of being a terrible fit? The answer, put simply, is what I call the delivery trap. Let's examine that in some detail.

There's an endless kerfuffle in the software industry on the topic of estimation. In my opinion, this results from the fact that knowledge

work is intrinsically non-linear in nature. You don't solve a hard computer science problem by chipping away at it 2 percent at a time until, after fifty units of time, you finish. You solve it by sitting there, drawing sketches on napkins and paper for weeks, having an aha moment, and then banging out some code for an hour. If you find yourself working on a problem that can be solved using the chip-away method, it's not a sign that you're good at estimating knowledge work; it's a sign that you're doing something that could—and eventually will—be automated.

Yet in spite of the soothsaying nature of estimating complex tasks, business stakeholders continue to wheedle, beg for, and even demand estimates. And programmers continue good faith efforts to provide them. As with anything that requires predicting the future with imperfect knowledge, all parties do their best, and their best is usually pretty bad.

But unlike the quality of the estimate, the badness of the outcome is decidedly asymmetrical. If you browbeat your mechanic into estimating what it will cost to repair your car before he looks at it and then he tells you \$300, you'll be extremely angry if it turns out to be \$3,000. But he's the one that suffers. Oh, don't get me wrong, the shock of \$3,000 in car repairs is definitely a bummer. But it was going to happen to you anyway. It was just a matter of when. He prolonged your ignorant bliss and, in doing so, became the target of your ire rather than your car. For him, however, the outcome is much worse—he now has a customer that regards him as a hack. Had he refused to estimate at all, he'd be in a better position.

Now, imagine if the customer were actually the mechanic's boss. Refusing to give an estimate in this situation represents defying a direct order. Yikes.

To capture this struggle for developer-kind, someone created a meme featuring Dr. Evil and his crew laughing and saying, "We'll ask for estimates and then treat them as deadlines!" I have to imagine that every developer reading can relate to this. Who hasn't done their best to offer a good faith estimate, under the auspices of a

promise that you won't be held to it, only later to be held to it? "You said it would be done this week!"

This seems so cartoonishly evil that the meme resonates. It lulls us into thinking that people who engage in this behavior are snakes. But the reality is that this is a perfectly rational behavior for a manager in the standard corporate context. I'll go even further and say that I view doing this as morally and ethically neutral.

Imagine the manager's position for a moment. The standard belief is that managers handle strategy while teams handle execution. A good manager justifies her salary by getting the most out of her team and enabling them all to be more productive, thus impacting the team's output more than any one individual could. If you accept this concept at face value, then extracting an estimate at metaphorical gunpoint and holding the estimator to it is, indeed, a pathological behavior. But don't accept that concept at face value.

Remember that in our corporate world—one where global scale comes via martial, pyramid-shaped structures—managers give orders and enforce the will of the organization. This charter doesn't operate in compliment to the enabling, force-multiplier concept that the good managers get the most out of their teams. If the manager's role were a pure matter of efficiency creation, then many, many high-performing teams would simply not have managers, since the sizable cost of a manager's salary would not offset the diminishing returns of trying to squeeze a bit more efficiency out of the group.

Let's drop the pretense, then, that the manager makes the team operate significantly more efficiently. Now things become more interesting. The most obvious role of the manager in a large corporation is the paternalistic one of micromanagement in the Taylor sense. But the manager serves an additional function at scale in a corporation—namely, she is an information conduit. For a massive corporation to minimize its natural inefficiency, organized communication channels must be established, and this is, classically, one of the key roles of managers. Managers put the business of their teams into digest format for consumption by higher-level managers,

who then do the same, all the way up to the top of the pyramid. Information then rolls back down in the form of corporate strategy and policy.

In this context, the estimation game makes a lot more sense. What managers really do when it comes to estimates, planning, and the like, is furnish information to superiors. At some level, superiors stake their personal reputations and positions on the outcome of gambling on the future. What is business, after all, if not strategy based on educated guesses about future state? “We should invest in the hardware group because this IoT thing is going to be huge!” “I see big things coming out of China next year, so let’s add to the R&D budget that caters to that market.” A few prescient calls like this can rocket your career into orbit. A few clunkers, and you can find yourself looking for your next role—that is, unless you’re at the line level where this really does not apply.

If you take this paradigm and trickle it back down to the line managers, demanding estimates from developers, you can understand their plight. A developer who has made some kind of boast about getting software done this week can choose to make it happen or not, in many cases. He can react to the work being greater than he thought by saying, “Ugh, I guess I was wrong,” or he can react by pulling a few all-nighters and getting it done anyway. He controls that situation.

But that control fails even to scale to a line manager of a team of two or three people. The line manager is not a horse running in the race. Rather, she’s someone betting on the race in the grandstands. If her horses fall behind, she can’t run onto the track and heroically push them until they catch up. She simply pays the price for a bad bet. So will the manager do everything in her power to get an inside line on the race and then psychologically use whatever means she has to affect that race? You bet. That’s her version of pulling a bunch of all-nighters in the form of a diving save.

Estimates are corporate currency that trade right up the ladder. If your team refuses to provide you estimates, you can bet that your

peers (competitors, if you're an opportunist) aren't having the same struggles. Do you want to be the only one without estimates for your boss, who wants her aggregate estimates from all of the dev managers? Do you want her to be sitting in a meeting with her boss, with a single miss on her estimate sheet where her competitors have all of their estimates in? Someone somewhere in the food chain is making a call based on all of those estimates, and that call is a pure matter of self-interest. That person wants to make a bold prediction or strategic play and turn out to be right. Everyone below them in the food chain needs to furnish the best possible information to make this as likely as possible. Good estimates and predictions are how you scratch a superior's back, with the implicit promise that the superior will then scratch yours.

Do you notice something interesting here? If this were a game of musical chairs, then when the music stopped, the line-level developer would be standing. Everyone else in the corporate hierarchy trades in guesses and predictions, strategy and machinations, orders and instructions. The developers are the only ones that trade in actual *output*. Theirs is the only tangible contribution to the whole pyramid. And significantly, theirs is the only narrative that cannot easily be spun.

Being defined by output rather than spun narrative is the essence of the delivery trap. Opportunists at any stage in their journeys understand this implicitly and seek to position themselves where they can write their ticket by managing their narratives. Idealists also trade in narrative, but only superficially. In idealist-land, the official corporate narrative is the only narrative, whereas opportunists understand that narrative operates at several levels and needs to be managed per audience. Pragmatists are unaware of narrative in any sense that matters. Narrative, in pragmatist land, is simply someone telling the story of how they did or did not produce enough output.

To understand what a bad deal it is to be stuck in the pragmatists' delivery trap, consider how the organization thinks of shortfall, exact delivery, and overrun. In the case of shortfall, the output generating

pragmatists bear the brunt of the burden. Managers talk to them soberly about the problem, and they find themselves subject to performance improvement plans. Hitting a target is celebrated with pizza for the team but treated, organizationally, as the expected outcome. The team gets pizza the same way a Little League baseball team gets pizza at the end of the season. Exceeding the target (which isn't always even possible) usually results in fifty-dollar Amex gift cards or something. If you're doing expected value calculations based on outcome quality, you'll quickly discover that it sucks to be a pragmatist.

If you want to improve your lot, you need to find a way to make your evaluation about *narrative* so that you can spin it. The organization demands its expected output and disproportionately praises the planners when expectations are exceeded. Heads, they win, tails you lose.

There's one narrative that routinely takes place at the line level among pragmatists, and it perfectly exemplifies the problem. Consider two teams. Team A routinely delivers on schedule, like clockwork. They log forty-hour weeks, calmly complete their tasks, and keep things humming. Team B is a high-stress group that tends to procrastinate, fall behind, and then put forth Herculean efforts over the last few days of a project to succeed. This team talks loudly about their stress levels, buys huge quantities of coffee and pulls all-nighters. After each such effort, they regale the organization with tales of their harrowing experience as if they were soldiers returning from war. Which group do you think the organization holds aloft as an example? Group B, every time. *That* is the power of narrative.

Being judged on the basis of output is a massive governor on your career progress, set to a very low speed. As a pragmatist, you shrug and do your best to produce a little more output. As an idealist, you understand the importance of narrative, but only as spoon-fed to you by the organization. As an opportunist, you need to create one for yourself that suits your career ambitions. But to do that, you have to get away from delivering; you have to escape the delivery trap.

For a developer, this is not particularly easy, since your main charter is to produce output. You're not going to do very well if you simply tell all parties that you're done writing code because you want to supervise now. It needs to be subtler.

The first thing to do is to learn from the aforementioned Team B. Delivering something on time isn't a remarkable story. Delivering something on time *against all odds*!? Now that's a remarkable story. Seek to dampen expectations without being overtly gloomy, and then blow those expectations away. Play up how much you've blown them away, too. The ostensible reward will be the fifty-dollar Amex gift card. The actual reward will be your growing rainmaker legend.

But line-level delivery shell games can only take you so far. You're maximizing the narrative of your output, but you're still responsible for output. Start to distance yourself from that.

Any time you have lulls in your expected software delivery schedule, fill those lulls by volunteering for meta tasks around software development. Volunteer to audit the team's use of JIRA and see if you can't come up with a more efficient process. Schedule some talks with business analysts to learn the domain so that you can coach the rest of the team on it. Dig up your team's yearly goals from some back email and do a research project, figuring out how to make those goals more likely. Do this during your lulls and, if you're so inclined, do it during your spare time.

Once that's established, keep it going. Make it visible to management for a while, and it will become an assumed part of your duties. The real litmus test, and the thing that you're angling for, is a situation where you say, "I can't pitch in on that second feature *and* continue optimizing JIRA and coaching the team," and someone that matters says, "Well, we'll just have to assign that feature to someone else." You need to become too "important" for simple delivery.

Don't worry at all about title here. That will take care of itself later, possibly at another organization. The main thing is that you want to begin replacing your delivery responsibilities with other meta-

responsibilities, ala management. Once you do this, there's nothing particularly measurable about what you're doing, and it becomes a matter of crafting your narrative. Did the team's defect count go down? Well, your work with JIRA happened around the same time. Did the defect count go up? Luckily, management was aware of the problem a lot faster because of your work with JIRA. Part and parcel with doing the meta-work is selling that work to those in a position to help you. And you can't do that if you're snared in the delivery trap. The output is the output.

And speaking of output, you have a line to walk here. You don't want to be an incompetent programmer, but you also don't want to be your team's cleanup hitter. If (and I speak from experience here) you become known as the person on a team of ten that delivers half of each release's features, you're fashioning a delivery trap for yourself that's sprung and made of granite. You'll never escape because the organization can't afford to let you. Instead, you need people to say of you something like, "He's a decent programmer, but where he really shines is getting the most out of the other programmers."

In the programmer's escape plan, getting away from delivery is the absolute most critical pillar. There are other facets of that escape, but this is truly thing one. Opportunists in software development establish themselves as other, realize that they need to escape, and then get away from delivery as quickly as possible.

Chapter 34: Partnership and Transcending the Realpolitik Glass Ceiling

So far, I've talked mainly about the natural marriage between software developers and either pragmatism or low-ceilinged idealism. This has included the failure blueprint built into most developers' career paths, in which rewarded behaviors and activities are also career-limiting ones, indicative of subordinate status. That, combined with the curious phenomenon of the journeyman idealist sitting below the standard idealist, creates a situation in which developers serious about their careers need to quickly excuse themselves from their predefined career path. After all, you're unlikely to servant-leader your way past the other pragmatists and through two levels of idealist-mesh before you're eighty if you don't give yourself an advantage. There are just too many people in your way.

In simple terms, this means getting a job other than programmer. But under the covers, this means escaping the delivery trap by any means necessary. To borrow from Mark Twain, you want to be Tom Sawyer, tricking others into painting the fence while you "supervise." Of course, you don't need to approach this quite so cynically; managers and executives really can have outsized impacts on organizations via leadership skills or uncanny strategic ability. But being good and advancing through the corporate world are not the same thing, and this part of the book is about how to advance, not how to be good.

Interestingly, both opportunists and regular, non-journeyman idealists understand the need to escape the delivery trap. Opportunists understand it as getting away from a bad deal. Idealists simply understand it as the reward the organization gives you for overperformance. In other words, their reasoning is more, "If I do a great job for Acme Inc., they'll reward me with a promotion, which I

accept as a good thing because Acme says it's a good thing. Plus I get to tell people what to do." I except the journeyman idealist from this reasoning, since he tends to view delivering software as some kind of sacred calling, to be treated with reverence and guarded jealously against impostors.

The strategies I discussed for escaping the delivery trap were largely tactical in nature, rather than strategic or philosophical. Seek out responsibilities other than delivering software. Seize opportunities to exhibit leadership, real or perceived. Take the standard, LinkedIn-style career advice of emulating your manager's behavior, but with the non-risk-averse, delivery-escaping approach of an opportunist.

I could go on with the tactical, but better to outfit you with the strategic framework and vision instead. You already know the goal (delivery escape), so with a philosophical approach, you can sort out the tactics that make the most sense in your particular context.

Recall that opportunists succeed by becoming *other*. This means that real opportunists do not view themselves as employees of a company but as single-person corporate entities unto themselves, dealing with the rest of the corporation as a series of outsiders. Opportunists thus view all of their relationships at work as various flavors of *partnership*. And this partnership is the key to slamming your way through the various implicit glass ceilings that neither idealists nor pragmatists perceive. Partnership is the key to removing the governor that the corporation places on your advancement.

Let's dissect this idea of partnership in the light of how players view the company. Pragmatists view the company the way spoiled adult scions with generational inherited wealth view their parents. It's the thing that pays their bills and, in exchange, forces them to put up with various discomforts, whims, and indignities. Journeyman idealists also share this outlook and comfort themselves with the fraternity of programming meritocracy. Idealists view the company as a purely benevolent steward of their careers, praising them when they deserve it and appropriately rebuking them when they fall short. The company is still a parent—just a different kind of parent.

Opportunists realize that there's no such thing as a company outside of tax and legal documents. In other words, opportunists don't view themselves as a tiny company dealing with a massive one. Rather, they view themselves as a tiny company dealing with a large population of other tiny companies. The opportunist may as well be in a co-working space, wheeling and dealing with partners and prospective partners.

This is why the concept of partnership is so important. Becoming *other* is a matter of mindset, a matter of leaving the cave wall. Learning to barter, broker, and bargain with partners provides the strategic tools to enable your ascent. And understand that everyone you encounter is a partner with varying degrees of power.

You can understand your relationships by mapping them to the relationships companies have with one another. Your peers are like nominal competitors in an open marketplace. Think of yourself as a Chinese restaurant, and they're French, Indian, pizza, and Lebanese places. You do compete, but can also all benefit simultaneously from a rising tide in the general market.

Your boss is basically your lone client, by default. In industry, you're what's known as a captive shop—you've placed all of your eggs in the basket of your only client. If that client fires you, your business is in deep trouble. The same is true if you have people reporting to you—they are captive shops of yours, where you are a big client that can throw its weight around with impunity. People of influence and "dotted line" relationships are essentially large industry players with the power to indirectly ruin your day. And so on and so forth.

This partnership understanding is critical, particularly the nature of the partnership with your boss. Once you see your boss not as a parental champion of your career, but as the client making you a captive shop, important realizations start to flow. As with any business, that lack of diversification is dangerous. That's doubly true when you're stuck in the delivery trap, being measured in a game entirely of that client's creation.

Imagine that you run a laundry service and your only client is the hotel down the street. The hotel asks you how many sheets you can clean per hour, and you tell it that you can do 200. It says, “We think you can do 300 if you make the following changes, and we’ll be unhappy with you if you can’t hit at least 250.” This is a bad position to be in, and you would need to do something about it. Perhaps you alter your operation or put in a lot of extra hours to keep the client. Perhaps you diversify and look to attract new clients. Perhaps you do your best and hope things work out.

But back in the corporate context, there’s a much better option. What if you could maneuver the situation such that you were not responsible for the number of sheets per hour? What if you became a “laundry efficiency manager” and your job was not to wash 250 sheets per hour but to tell your client that the guy down the street should be able to do 250 per hour instead of 200? That seems like a better, safer deal.

That, viewed through the eyes of the opportunist, is why delivery is a sucker’s game. Of the limited plays available inside the ecosystem of players that others call a company, positioning yourself to avoid failure setups is critical. (In fact, that positioning is exactly what Venkatesh talks about as the core of *The Gervais Principle*: perceiving failure situations early and maneuvering idealists into place to take the fall for you). Sadly, in the current corporation, success can largely be attributed to deftly parachuting out of failure situations. Opportunists don’t buckle down and do what’s best for the company—they count on idealists’ willingness to go down with that ship and oblige them.

The “What would I do if I were a company of one?” exercise is an excellent one for the aspiring opportunist. It provides a mental model for maneuvering away from delivery accountability and escaping line-level roles and responsibilities. It also sets your jaw to do some things that might seem questionable—if the company doesn’t really exist, it won’t prick at your conscience when people say things like, “Is that really good for the company?” There’s an aphorism that says,

“People don’t work for or quit companies, they work for or quit people.” That’s an unusual bit of opportunist advice that floats onto social media alongside all of the crap about working extra hours and dressing like your boss.

But what does this partnership attitude and behavior look like? How do you pull it off? How do you translate it to real advancement? Let’s go into more detail.

As with animals in the wild, you want to make yourself appear big when dealing with various partners in the organization. Other players exploit weakness but defer to strength. The key to survival is strength at your own level, but the key to advancement is the illusion of strength above your level. You want to cultivate a mover and shaker air.

Imagine a world with nobles, merchants, and serfs, who for these purposes correspond to owners, managers, and grunts. You’re a serf, but you want to be a noble. This means somehow getting nobles to accept you as one of their own, in spite of your serf-like appearance. In order to do this, you need to wander boldly into their midst and act enough like a noble to give them pause, and to wonder if maybe, just maybe, you aren’t a disoriented noble that got drunk and lost his shoes. If you fool them just enough to take you in and give you some hot medieval coffee while they figure out your true identity, you have your foot in the door. This is high risk because you’re actually a serf. But you just need that next chance—the chance to convince them that you totally have a minor dukedom in Canada that they probably wouldn’t have heard of anyway, and could they help a fella out with some loaner noble clothes and maybe a place to stay?

Back in the corporate world, you need to start imagining yourself as a displaced owner/executive/entrepreneur with an explanation for hanging out among the grunts. Perhaps you’ve run point on a few startup ventures, but settled into a nine-to-five job for the sake of a little stability for a while. Perhaps you have a side business that’s really taking off, but you’re here, getting some perspective at the line

level. Whatever the explanation, you're an owner and executive among grunts by your own choosing.

You need to do this because advancing requires alliances with comparably powerful partners that see you as an equal in the wrong place. This inspires two sentiments in them. First (and assuming they're opportunists and not idealists), they'll feel a "There but for the grace of God go I" sentiment and have an impulse to balance the cosmic scales by empowering you. And secondly, they'll view you as an ace in the hole. They have official authority but can rely on you as an off-the-books source of advice that everyone else overlooks.

This is an example of a quid pro quo, and it's very much how partnerships work in general. It's also one of the two keys to your advancement. The other key is having options, or at least the impression of having options. This makes sense if you imagine something as simple as dealing with a vendor—say, bringing someone out to fix your furnace. You have money that you need to spend, and you have some options. If one of the prospective vendors offers you some kind of freebie, like a new thermostat, that will make you more likely to pick that vendor. And in terms of options in this situation, you gain leverage by making it known that he's not the only one competing for your business. All of this is how you should approach corporate advancement as well.

At every level, you want to subtly leverage the threat of the market without being so crude as to actually threaten anything. You simply want it to be clear that you have options, and not options limited to your group or even your company. Amateurs do this by securing a competing offer and wielding it like a temporary cudgel. Masters have no need to do this, since they constantly exude options without resorting to obtuse declarations like, "I have an opportunity to work somewhere else, so give me more money."

You can do this in many ways, and you'll need to find the strategy that works for you. Little things like tons of contacts and recommendations on LinkedIn can actually help. Speak extensively about your experience in other places, cultivating an air of the

cosmopolitan that stands in stark contrast to your peers that have only had one or two jobs. Moonlight on the side and talk about your experience doing so. Casually cite experience on the level of your superiors in a flattering way. “I like what you’re doing in terms of dividing up the break/fix work—when I used to run a team, I had a strategy that I thought was pretty good, but I’ve definitely learned a couple of things.” The message that’s actually heard below the superficial? “I’m actually a peer of yours, but I’m not here to threaten.”

One of the best ways to really hint at options is to change the game in stock asymmetrical situations. For example, consider a performance review. Right at the outset, tell the reviewer, your boss, that you’re not really interested in more money. If she wants to show appreciation for your contributions, she could show it instead by letting you do some research into more efficient ways to manage the feature pipeline for your group. Explain that this would scratch a personal and professional itch of yours.

In one fell swoop, what you’re really saying here is, “I don’t need money, I’m intelligently interested in management, and I politely reject this silly performance review process.” There is some risk in an approach like this, particularly if your boss is an idealist, committed to the farce. But as I’ve said all along, this is not a game for the faint of heart. Deftly done, there’s not too much risk since you’re not saying anything untoward. The most likely outcome is that your boss regards you in a new light, as more of a peer and something of an enigma. “I don’t understand...who turns down a raise? That guy must have some kind of interesting master plan, and he probably doesn’t need this job badly.”

Remember, though—never threaten. Even in a situation where what you’re saying is a threat, couch it as “just business.” If you take a gig in another department or accept another job offer, don’t throw it in anyone’s face or delight in it. If anything, present it in such a way that a boss or compatriot is left in the end agreeing that you’re making a great move. You’re not vanquishing foes and you’re not settling

debts—you're playing the game well, simply creating advantages for yourself.

You always want to leave your partnerships in as good shape as you can because of the second ascendant consideration: the *quid pro quo*. If you convince a soon-to-be-former boss that, in your position, she would also take the competing offer, she'll remain open to continuing to have a relationship with you and maybe even look for mutually beneficial arrangements. Having other options isn't a crime, after all—any sensible opportunist does that. But they also constantly look for ways to offer value and then to get a fair price for that value.

Pragmatists and idealists tend to fail spectacularly when it comes to the *quid pro quo* aspect of partnership. As an example, consider something that you would find for a dime a dozen on BuzzFeed or LinkedIn in terms of career advice: at performance review time, you should march into your boss's office with a list of your important accomplishments over the last year. Then you should say, "Here are all of the reasons I deserve a promotion. I have exhibited LEADERSHIP, and I went out and got officially certified in LEADERSHIP, and I am thus ready for a LEADERSHIP role!" Lay out the case for your boss about how awesome you are, preferably in PowerPoint or something. Right?

Goodness gracious, no. Oh, don't get me wrong. That's a great way to secure a 4 percent COLA instead of a 3 percent COLA because your bored boss is going to say, "Oh, for the love of God, fine, because you'll annoy me all year if I don't throw you a scrap." But ascendant opportunists are not looking for scraps. You don't have time.

The trouble here is that the idealist/pragmatist taking this advice is not *offering* anything. He's a student asking for a good grade and some extra credit, not a partner dealing with another partner. If you were running the laundry service mentioned earlier, would you ask your customers for more money because you were "certified in

leadership”? Of course not. That’s worthless to them, and you know it.

Opportunists don’t demand merit evaluations, and they don’t wait for performance reviews. If you want your boss to really hand you some valuable benefits, you need to do the same for her. Make her life better somehow, then look for reciprocity.

You don’t make your boss’s life better by “demonstrating integrity” or whatever pap tends to wind up on official performance review documents. You make her life better by improving her prospects. Find out on what basis she’s being evaluated and what her goals are, and then take steps to make those things happen. The official stuff doesn’t matter. Remember, there is no company. But your boss is your only client, so improving that client’s life will bode well for you—at least until you secure better options.

This quid pro quo sentiment can go beyond your boss, particularly if it is, at least on the surface, good for the organization. And it’s also best if you can incorporate your own marketability and connections outside of the company as well. For instance, imagine that you work for a moderately-sized web development consultancy. Picture the leverage you’d have if you went to your boss and said, “My brother-in-law runs a multi-state restaurant chain that’s looking for a new interactive website. If I can help bring them in as a client, would you let me be the team lead on the engagement?” You’re offering a perfectly reasonable quid pro quo that will make the company money. That’s vastly superior leverage, compared to, “I did pretty good last year at, like, leadership qualities, so I want a promotion.”

Partnership is a subtle, powerful consideration. I’d love to be able to tell you all of its intricacies and give you a foolproof blueprint that will work for you. But sadly, no such thing exists. It simply requires practice.

But if you keep in mind the idea of quid pro quo and the notion of having options, you’ll have an excellent framework to realize the benefits of internal partnerships. And this framework can also help

you navigate political minefields, such as contentious peers or corporate enemies. Partnership greases the skids for your rise out of delivery, out of line-level positions, and up to the highest levels of the company. But in order to keep that going, you have to avoid becoming mired in the trappings of idealism. I'll cover that in the next chapter.

Chapter 35: Avoiding Carnival Cash

Recall that carnival cash is the term for corporate perks devoid of market value outside of the company. This includes things such as special parking spaces, better cubicles or offices, and employee-of-the-month awards.

Implicit in my description of the corporate hierarchy is the idea that, if you're serious about opportunism and success, you want to avoid hoarding carnival cash. This isn't the same as deciding not to *value* the carnival cash. You don't want to have it in the first place. If someone walks up to you and dumps a pile of it in your lap, whether or not you value it will be immaterial (and unknowable) to anyone sizing you up. If you're standing there with all of the novelty prizes from the carnival, a passing opportunist will note you for the idealist that you appear to be, not the opportunist that you're determined to be.

Your path to opportunism is thus fraught with the danger of stalling out. Getting stuck in the delivery trap makes you a perpetual pragmatist. Getting stuck with a gigantic hoard of carnival cash renders you a perpetual idealist.

In the last chapter, I talked about a hypothetical laundry service and how it'd be a better deal to be in charge of managing the laundry service than to be on the hook for actually cleaning some quota of sheets. Think of the laundry service again when considering the dangers of carnival cash to your partnership-oriented mentality. Would a professional laundry service accept a "laundry service of the month" plaque in lieu of a month's payment? Would it view a special parking place in the front of your visitor parking section as acceptable instead of a late fee? Of course not. Would it even accept these patronizing gestures at all? Unlikely.

So, if a small independent business wouldn't do it, you shouldn't do it either. But here's the rub. Large segments of the company you work for believe in the company as an entity and would look askance at you refusing to accept corporate baubles. You need to be subtle when it comes to avoiding suckers' currency.

And by the way, don't underestimate the surprising allure of carnival cash for everyone—me included. It's easy for me, as a free agent and outsider, to tell you antiseptically not to value feel-good office recognition. It's a lot harder when you've been boots-on-the-ground for years and you're earning accolades that make everyone around you impressed and jealous.

I get that. But you have to stay strong. It helps to play the partnership-centric game of asking yourself "What if I were a small business?" Tell yourself that, instead of recognizing you, the people in question could simply pay you. And yet, they aren't.

Before I go further, let me offer an example so that subsequent abstract references have a bit more grounding. A few years back, I consulted for a client with a flat IT organization structure. There were VP-level folks, then a hodgepodge of people at different implied levels but with a uniform reporting structure. Since the Taylor model has entrenched itself so neatly, we tend to recreate unofficial organizational pyramids in the absence of explicitly defined ones. This place presented no exception.

During the IT department's all-hands/department meetings, a feel-good corporate activity would take place, budget permitting. The rules were fairly straightforward. The VP running the meeting would throw a call out to the audience to nominate someone who had gone above and beyond. Game participants would nominate someone and briefly describe that person's heroics. Once the nominations finished, VP would call for a vote. The top three people would get a prize—let's say \$100 gift cards, since I don't recall the exact prizes or figures.

This may seem like a rather mundane, if generous, corporate activity. And against the backdrop of corporate inevitability, this actually *is* far more generous than what most companies do. It came from a good place. But for our purposes, this activity serves as a realpolitik petri dish, where one can take pristine samples of pragmatism, idealism, and opportunism. Here's how it would typically shake out.

The people that would receive gift cards and celebrate them were clearly pragmatists. Pragmatists cede hope while maintaining perspective, so their attitude tends to be, "Yeah, man, I'll take \$100!" (For all I know, clever ones may have entered into partnerships to alternate up-votes for one another, but I wasn't an interested party.)

Now you might think that the idealists would also compete, though more for the recognition than the prizes. That's reasonable to assume. But recall the important detail of the flat management structure here, and the idealist behavior makes more sense. The idealists eschewed the cash prizes and nominations in favor of being *nominators*. Why? Carnival cash! Nominating the little people for recognition is the sort of thing The Boss does. Idealists engage in boss posturing. Being a nominator represents status to them (and worthless carnival cash to anyone looking, since it has neither value nor significant impact on their actual standing). There is perhaps nothing more exemplary of the idealist condition than passing up actual cash for a chance to posture.

What did the ascendant opportunists do? They avoided the whole scene like the plague. An opportunist does not want to be remembered like the pragmatist here, reminiscent of a dog pleased at having a milk bone tossed his way. Recall that opportunism requires you to act as a business entity and cultivate an air of options—partners with options don't get pumped about nominal sums of found money. But the opportunist also does not want to be remembered for chasing carnival cash. Partners with options don't need to aspire to leadership positions with wishful mimicry. So the opportunist stays away—there's no good outcome to be had here.

The opportunist does not want to be remembered for either type of behavior. Indeed, the opportunist generally does not want to be remembered at all in games without power stakes. Would an opportunist seek a gift card as recompense? How about the opportunity to look like a boss by nominating someone for a gift card? Of course not. The opportunist would say, "If I've done something worth calling out at some kind of ceremony, then I've probably done something worth a lot more than \$100. I don't want a gift card, and I don't want the tacit 'perk' of being trusted to decide who should receive gift cards. I want a cut."

Bear in mind that the opportunist's peril in this situation is *not* that she might fail to negotiate a cut. The opportunist's peril is getting swept up in the organization's normalization of not ever being at the negotiating table to try to claim that cut. And there's no faster way to lose your seat in real power discussions than to be viewed as the kind of person that can be bought off for \$100 or, worse still, can be bought off for the prize of giving someone else \$100.

The actual, already-ascended opportunists in the organization participated only in running the ceremony and approving the budget. The ascendant opportunists made themselves scarce by avoiding nominations and avoiding speaking up. I suspect some of them found reasons not to attend an all-hands meeting in the first place. (And being too busy for such an event is actually an excellent opportunist strategy.)

Let's now unpack additional, abstract meaning, using that example to ground things. The first generalized theme is that you need to avoid "'A' for effort" recognition as if it were a pile of dog poop that was also poisonously radioactive. Never, ever be in a position where being patted on the head and tossed a treat is an acceptable substitute for compensation for the value that you bring.

Don't confuse this with advice not to work hard or even having people recognize that you've worked hard. You absolutely can and should work hard when appropriate. Work hard to help your boss advance. Work hard to bring in extra money for your department.

Work hard to generate internal and external leverage to help your situation. But whatever you do, do not work hard to please some superior in the hopes that they'll deem you worthy of a raise someday.

In fact, acceptance of passive reciprocity will bite you. By "passive reciprocity," I mean any situation in which you spew surplus value with no predefined negotiated compensation.

In the last chapter, I discussed quid pro quo and the need to offer something of value before asking for something of value. If you want a raise, don't talk about how you've improved your leadership skills; offer to bring in a new client. The converse also has importance here. Don't expend herculean efforts for free without negotiating some consideration ahead of time. For instance, don't land a major client for your company and then sit back and see what they'll do for you. If you're lucky, it will be a cash referral fee. If you're lost in an idealist wasteland, it will be "employee of the month," with access to a special parking space and a promise that it will be remembered next year at performance review time. Think back to your laundry service. If you wanted to diversify your offering, would you wander out to the parking lot and wash your customers' cars while they waited, hoping they would throw you a few dollars? Or would you say, "you know, I can wash your car while you wait..." and then talk terms?

Now, I know what you may be thinking. "Making demands for doing your job well sounds risky." Well, let me emphasize a few things. First, I said from the outset that opportunism is a high-risk game. Second, we're not talking about doing your job but about going above and beyond your job; you have to do impressive or high-leverage things to move up quickly. And third, you're not going to saunter in and make demands; that's the idealist cartoon of negotiation, not real negotiation.

Present these quid pro quos as win-win, and leave yourself a plausibly deniable out. Consider the example I mentioned earlier. You tell your boss that your brother-in-law runs a firm and ask if you

could be made team lead if you can bring in that business. If boss says yes, then great. If boss is an idealist and says, “It’s your duty to the company to bring in business” or “Bring in the business and we’ll see what happens,” then don’t argue with him. Simply agree, and then don’t bring in the business. If boss asks about it later, just smile a little and say, “I tried, but he just wasn’t that interested.”

To go back and borrow Venkatesh’s term, you’re now speaking powertalk. Superficially, no idealist or anyone else can quibble with you. You went above and beyond your job description to try to do something for the company. You didn’t deliver a new client, which is not ideal, but you also didn’t fail at anything either—this was pure extra credit. Below the surface, however, you’re clearly saying, “I’ll wait for a better offer before we revisit this.” If idealist boss or anyone else wants it badly enough, they’ll cough up an agreement for you to be team lead, and you can “try again” with your brother-in-law.

It’s rare that you have true leverage, particularly in a line-level position. If you’re a positive sum sort of person, look for mutual wins that come from external resources. This helps keep your network healthy and your market value high, as an added bonus. You can also, if you’re so inclined, make zero-sum internal leverage plays. That’s not really my style, so I won’t go into too much detail, but suffice it to say that there are myriad ways to have something that someone else wants or to know stuff that gives you leverage. Be careful exerting pressure on people in this fashion. You make enemies and the stakes go way up. I don’t like that, myself.

When you’re able to manufacture some leverage for yourself, do not squander it. Do not voluntarily accept carnival cash. Do not give anyone the opportunity to swoop in and dump it on you. Avoiding carnival cash as recompense is critical.

But carnival cash as a quid pro quo consolation prize is not the only source of this dangerous currency. You tend to accumulate it naturally if you stay in the same place for very long. Maybe it’s an engraved blender for your five-year anniversary with the company. Maybe it’s a policy that the most senior team member gets the

cubicle with the only leather chair. Or maybe it's just the standard, subtle deference that comes along with seniority. No matter what its source, the game is the same. Like a ship, lack of motion earns you barnacles for your (lack of) trouble.

Let's stick with the nautical theme for a second. You need to be like a shark. You must always keep moving. Always be transferring, switching roles, switching jobs, switching projects. If you start hearing things about yourself from coworkers along the lines of, "Oh, [insert your name] has been saying we should implement that *forever*," then you need to march into your manager's office and tell him you badly need a change of scenery.

This holds true for line-level opportunists looking to break into management, but it holds *especially* true for managers. Shark managers ascend through the narrative control that I talked about earlier. All transfers can be spun as taking on new challenges, graduating to more responsibility, or even staging victorious retreats. Suckers and idealists go down with the ship. Sharks like you swim away and live to fight another day. And most importantly, they don't stick around long enough to get noticed for a hoard of accumulated feel-good carnival prizes.

Another subtle consideration, as you move around and avoid carnival prizes, is to avoid getting too immersed in the corporation's mythology, buzzwords, and backstory. Idealists revel in this sort of filler. Fluency with it is no different than a big stack of "Super Effort!" plaques lining your walls. In fact, there's a bit of power signaling that goes on when you strike just the right balance between operating in the company's interests and skepticism for its folklore and terminology. You're saying, "I'll help out, but on my terms, and without the BS." Do that just enough to let the opportunists know that you have external prospects and expertise to offer (they, after, all, understand the mythology is necessary to keep a healthy idealist layer intact), but not enough so that the idealists start to call for your head as a heretic. And of course, you can tune your message a bit for your audience and ham it up a little to humor idealists, if need be.

I'll close out the chapter by offering one last piece of comfort. If you find yourself in a situational quagmire, stuck in place for too long or pinned down and confronted directly with carnival cash recompense, all is not lost. No single employee of the month award will relegate you to idealist purgatory forever. I'm more talking about the noble act of politely refusing carnival cash. Let's revisit the realpolitik petri dish from earlier. If you're sitting there, minding your own business, and someone nominates you for \$100, you have options. Stand up and graciously say that you can't accept it because it was really Bob who did all of the hard work. Or consider a hypothetical scenario: you're offered acknowledgement at some huge department meeting for landing a new client. You could always say something like, "I really get uncomfortable with that sort of broad recognition for my own work when so many of my teammates contributed to the pre-sales and onboarding processes."

To pragmatists, you seem magnanimous, and to idealists, you seem like an idealist. You're avoiding the limelight and deflecting credit and praise to others. But to opportunists, you're attracting an arched eyebrow of interest. They see you saying, "I don't value this nonsense, so you can keep it." That's going to open you up to discussions of true weight and impact.

So form your partnerships as if you were a business. Offer and demand quid pro quos and understand when you have leverage. And always keep moving, lest you gather barnacles. If you seek and offer legitimate consideration and you avoid carnival cash, your rise will be swift and steady.

Chapter 36: The Art of No

You can simply say no to offers of carnival cash; that's true, and it's important. But saying no is a much larger concern in your ascendancy in general. It deserves its own chapter.

You need to avoid situations that impede your progress, which, beyond offers of carnival cash, might also include setups for failure or obviously subordinate assignments. Steering away from danger and status reduction is an art. Once you've worked your way up into the organization, a lot of this is done by maneuvering idealists into tactical locations. In the interim, you must be resourceful.

In 2016, I wrote a blog post called ["The Beggar CEO and Sucker Culture."](#) The eponymous CEO was "Victoria," who wrote to some corporate Dear Abby on LinkedIn, asking for advice on how to get her employees to put in extra hours at the ol' labor mill. She didn't like that her company's culture was one in which everyone worked "only" their eight hours. She wanted them to be passionate about the organization and put in overtime for the love of the game. I called her the beggar CEO because she essentially asked, "How can I pressure my employees to work unpaid hours?"

I believe the popularity of the post came from my disdain for Victoria's sentiment. I called for a stop to what I described as, "sucker culture," which I view as an apt name for the idealist arms race to work more free hours than those around you. I suggested that we, as a collective, stop wearing mountains of unpaid overtime as a badge of honor. People can read into that as they will, but my objection was less humanitarian than logical. Working like this for pennies on the dollar makes us suckers. Sucker culture normalizes suckerhood to the point that the Victorias of the world whine about it when their employees have the temerity to act in their own rational

best interests. “Can you believe these lazy pragmatists only want to work when I pay them?”

For pragmatists, the prevalence of sucker culture is a real bummer. Sucker culture’s essential mandate is “Say yes to everything the boss wants, no matter what, and then do even more than that.” Pragmatists thus find themselves in situations satirized by *Office Space*, ducking around the office like Peter in the hopes that Lumburgh can’t see him before he leaves.

Idealists relish sucker culture, and I think there’s something of a chicken and egg dynamic. Which came first, sucker culture or the suckers? Organizations in decline tend to become idealist-heavy, having such powerful sucker cultures that pragmatists start heading for the exit through voluntary and involuntary attrition. Does sucker culture drive this decline, or is it simply a byproduct?

Ascendant opportunists thus have a unique challenge when it comes to pushing through the miasma of the idealist layer with its culture. This is particularly true of programmers because of the dual bands of idealism above them: regular and journeyman. Regular idealists think you should put in a lot of extra hours for the love of the company, and journeyman idealists think you should do it for the love of the craft. (The journeyman idealists’ attitudes here are preferable since at least it benefits you to put in extra time to make yourself skilled, unlike toiling away on weekends for the company.)

Ascendant opportunists have to learn the art of no. The art of no is broad and subtle, though. It dives far deeper than simply blurting out the word “no,” which I’d rarely, if ever, advise. And it comes in many flavors.

The first flavor I’ll address is what I’ll refer to as “no by saying yes.” It’s certainly the sneakiest way to say no, but it’s also the one that involves far and away the least amount of conflict. You agree to something that doesn’t benefit you, but instead do something that does.

Here's the simplest form of "no by saying yes": agree to do something and then don't do it. This is often risky and easy to peg, of course. Boss tells you that everyone's going to have to dig deep and come in on Saturday, and you either don't show up or you beg off, saying that you're sick. That's not a good idea, since it makes you look flaky and unreliable.

For this play to be something you should even consider, the asker would need to have extremely low status and the ask would have to be something that would torpedo your prospects. Contrived as it may seem, the best example I can think of is that your boss asks you to march into *her* boss's office and tell him he's making all kinds of mistakes. In this absurd situation, the best option (while you look for other jobs) might just be to tell your boss that you did it and that his reaction was to get angry and demand no one ever mention the conversation again.

More realistically, I'd advise you say yes to things and then advance your own interests. For instance, consider Victoria and her hand-wringing over the organization's lack of sucker culture. She seemed to want nothing besides people physically being at work longer (there was no mention of *why* she wanted those longer hours or what value she hoped they would provide).

Whether Victoria asks for extra hours explicitly or passive-aggressively, you can say yes to staying later while quietly saying no to giving her free labor. Instead of spending those extra hours doing anything for the company, work on beefing up your network, looking for other prospective roles/clients, leveling up a skill, or simply doing some personal errands. You can be there ten or eleven hours a day without working more than eight. No one will know the difference. You've said no by saying yes.

"No by saying yes" gives you a play in situations where assent is hard to verify. Corporate citizens have a complete blind spot for evaluating contribution value, so they use proxies like hours present. With such proxies in use, the law of unintended consequences

becomes a veritable certainty, and compliance while advancing your own interests becomes simple.

In situations with far less muddy waters, you have other options. In the last chapter, I described a specific instance saying no that I'll now call "no by self-effacement or sacrifice." In that chapter, our hero opportunist downplayed his role and refused a cash reward. This is perhaps the best technique for avoiding carnival cash, but it can be used in other situations, too. For instance, consider a dubious "plum assignment" that actually sets you up for failure.

Such assignments represent frequent channels for bounded advancement in the idealist world. If some opportunist mistakes you for an idealist, you might find yourself "promoted" to work on an account or project that will certainly fail. "We know it's a reach, but we feel that, if anyone can do it, you can," they tell you. Counter (and show them that you're not an idealist) by saying, "Oh, I appreciate the offer and am flattered, but I think Jim has been putting in tons of hours and would be a much better fit."

Be careful with self-effacement, though. The last thing you want to do is receive some kind of offer and turn it down by saying, "You don't want me. I'm a moron." Rather, you want to self-efface using an apparently beneficial but really suboptimal trait—hence the "tons of hours" that Jim's been working.

Telling someone they want Jim because of all the hours he puts in is really saying, below the idealist radar, that you recognize you're being set up for a slog, and you're offering Jim instead because he works harder and not smarter. Beg off in favor of others because of traits they exhibit like dedication, working long hours, loyalty, tenacity, and the like. Be sure you're talking about things that tend to exhibit diminishing or negative marginal returns. Don't tell them that Jim is more strategic or smarter than you.

A related but more compelling form of no is "no by counteroffer." With this tactic, we begin a dive into better examples of powertalk

and maneuvering. Saying no with a counteroffer is really just saying yes to another question.

In the world of improv, as I understand it, actors avoid the word “no” in favor of “yes, and...” when continuing the improvised dialog. For instance, let’s say that the first actor says, “I’m a serial killer responsible for gruesome crimes,” and let’s say that the second actor prefers more lighthearted topics. He wouldn’t (indeed, can’t, per the rules of the game) say no. Instead, he’d say, “Yes, and after undergoing a revolutionary new brain surgery, those impulses have been replaced with a desire to be a philanthropist!” The second actor basically says no via redirected narrative.

In the world of negotiation, offers and counteroffers represent a course correction of shared narrative. If you have your house on the market for \$200,000 and I make an offer of \$175,000, I most likely don’t say, “Your valuation is wrong, so I’ll give you \$175,000.” Instead, I say something like, “\$200,000 is outside of my budget,” or, “I can get a similar house for \$170,000 two towns over.” I offer additional information and steer the shared narrative toward the outcome I want. If the other person counter-counteroffers, she might say something like, “Ah, but that house two towns over is part of an inferior school district, so it makes no sense to sell for less than \$190,000.”

Control your situation via narrative-altering counteroffers. If your boss wants to bury you on some backwater legacy project, don’t accept that. But don’t resist by pouting or by kicking and screaming. Instead, offer something else. “I’d really prefer to see my current project through. What if I stuck with it 75 percent of the time and dedicated another 25 percent plus some overtime to getting someone else going on that project?” You’re not simply prevailing upon your manager’s sympathy or offering the unspoken threat to be disgruntled for a while. Rather, you’re implicitly expressing your preference, but doing so in value terms (value to the current project) and offering some actual skin in the game (your overtime). You’re changing the narrative—implying that your project will suffer without

you and that the alternative project can still flourish with you in a reduced role.

“No by counteroffer” can be effective without being sneaky, but it can also fall flat. If the request comes from your boss, then your attempt to say no can fail to hold up. She might simply say, “Sorry, you’re switching projects.” Certainly it does not provide you a bulletproof solution, but it does offer you at least an alternative to simply resigning yourself to fate or to begging and sulking.

You can do a bit better than “no by counteroffer” if you’re clever. Specifically, I’m talking about the next form: “no by better idea.” It’d be fair to consider it a strict subset of “no by counteroffer,” in which the counteroffer is a better idea. But I recommend considering it separately.

“No by better idea” happens when someone throws something at you and you offer a demonstrably superior solution. If you look at “no by counteroffer,” the counter isn’t necessarily superior. Just different. Better ideas don’t tend to involve phrases like “I prefer,” and they don’t really amount to quid pro quos or horse trading. They involve addressing the same problem, but in better fashion. This requires root cause understanding.

For instance, with the previous example, did you stop to wonder what the official reason was for the boss sticking you on a backwater project? You should start thinking in these terms, if you aren’t already. Yes—I’m saying that, in this case, the surface reason matters more than the underlying reason. (That is, if one exists. Your manager can’t say, “I’m putting you on this project to punish you for being personally annoying to me.”)

Let’s say that you ask that question and the response comes back, “Well, it’s a C++ project, and nobody here knows C++. Since you know C, you’re the closest we’ve got.” You counter with, “Steve two departments over is actually a C++ guru, and he’s just rolling off of a project. I bet you could borrow him for a while. That way you’d get someone better on that project while not losing steam on the one I’m

assigned to.” Unlike your previous horse trading, this is a superior proposition in all ways. Both projects have better suited staffing models and the organization avoids an idle developer for any amount of time.

Now, I imagine you’re wondering what to do if you don’t have a Steve in your back pocket or if you don’t think well extemporaneously. I’d say that’s fine. There isn’t a particularly critical shelf life on this form of no. You can always gameplan for a few days and see what you can come up with.

The potential political fallout from this play is too fractured to address definitively. It can really vary, depending on to whom you say no and the nature of the idea in question. It could be your boss, and your boss could respond with, “Wow, thanks—that’s great!” Of course, if the boss was trying to bury you to satisfy a grudge, they might simply refuse, or they might say, “Wow, thanks” angrily, through gritted teeth. Same kind of gamut applies to people not directly above you in the org chart.

The one word of caution here is to check your own ego. If you come up with an inarguably better idea and present “no by better idea” to your boss, he may simply look at you and say no anyway. Don’t blow your stack and sputter about his shortsightedness, like Andy Dufresne in *The Shawshank Redemption* calling the warden obtuse. Realize going in that there may be more at play than what happens on the surface and that superior arguments may not carry the day. If they don’t, it’s a data point.

And finally, realize that this “no” play can apply to major shakeups or just the day-to-day. Boss wants you to put in extra hours slogging through a log file, looking for certain kinds of events? Write a script to do it instead and present that toward the end of the day. Go enjoy your free time.

Now, let’s get into even more political forms of no. Consider the case of “no by negative sell.” In case you’re not a connoisseur of sales techniques, I’ll briefly explain the concept of a negative sell.

You might think of it as reverse psychology. When someone tries to sell you something, particularly something of which you are skeptical, the encounter tends to follow a predictable script. They try to convince you to buy the thing while you list reasons for your skepticism. In the case of a negative sell, they confuse you by suddenly switching gears and asserting that they no longer want to sell to you. The idea is that you become subconsciously indignant and start to convince them that the sale makes sense.

Here's a pretty simple example. Imagine that you're picking out a piece of jewelry for yourself or a significant other. The way this typically goes is that you walk into the store and the people there try to convince you to shell out a lot more money than you want to. After a bit of this, you might decide to leave and go to the central mall kiosk with the costume jewelry or something. But now imagine you ask a salesperson for help. He sizes you up and says, "I think you might have better luck at the costume jewelry kiosk." I dare you to tell me that at least some tiny part of you wouldn't want to reach for your credit card to spite him and show him what kind of spending power you could bring to bear.

In the context of saying no to people in the organization, this one is best applied to people with ambivalent interest in what they're asking you to do. Consider the simple case of avoiding carnival cash like an employee-of-the-month award. You could do a "no by self-effacement," demurring by saying that you aren't qualified. But that's actually a positive sell, in which you successfully sell the idea of being denied the award. The negative sell would be to push them in the opposite direction by being unseemly eager for it. This is a slightly weird example, but hopefully you get the idea.

Negative sell is subtle and not always relevant to saying no, particularly in the case of having some bummer assignment dropped on you. You'd need to flip the discussion back to the opposite outcome (you *not* receiving the assignment) and then try to prompt the asker to convince you that the opposite outcome actually makes sense. For instance, in response to the bummer assignment, you

might say, “I can take it on, though Bob seems more qualified... but I know how hard it can be to convince Bob to do anything, even if you *are* his boss...” As you can see, pulling something like this off is not necessarily easy when the other person has already thought this through. Still, this technique can come in handy from time to time, particularly if you do subtlety well (preferably better than that reverse psychology example about Bob).

A negative sell, even when not successful, can generate leverage and better offers. For instance, when I negotiate with clients who ask me to take on work that I don’t really want, I often negative sell. But it’s in earnest. I genuinely don’t really want the business. Sometimes they’ll bargain me up to the point that I acquiesce. An interesting thing has happened here, though. They got their way (I failed to say no), but we’re both aware that they need it more than I do. I acquire general leverage in the relationship.

“No by leverage” is probably the most powerful form of no that I’ve listed so far. I’d say it’s also the most perilous. This is where you say no by virtue of being able to create consequences for the person asking something of you.

To understand why this is dangerous, let’s consider the employee-of-the-month award in lieu of money situation again. You could exert some very self-defeating leverage in the situation by, say, hinting to the boss that you’ll get drunk at the awards presentation and make a ridiculous speech. Your leverage is the boss looking stupid. But to acquire it, you offer the disproportionately greater leverage of termination for cause. Better leverage plays take the form of stuff that doesn’t blow back on you.

I don’t want to spend a ton of time on this one because it can get fairly dark. After all, the line between leverage and extortion or blackmail is a fine one. If your boss, in a moment of weakness, vented to you about her boss via email, that gives you leverage. Whether you attempt to use it the next time she asks for something is your peril, both ethically and career-wise. I won’t advise here

except to say that the situation can give you a very powerful way of saying no, and it also can set you on an ugly path.

The last one that I'll mention can also range from simple to ethically perilous. I'll call this "no by anticipation." If you get out in front of situations, you can stack the deck pretty heavily in your favor. For instance, if you sense a low-status death-march slog coming on, you might differentiate yourself with a voluntary slog for a bunch of weeks before the main one arrives. Once it comes, you may have positioned yourself for some reprieve.

On the flip side of this comes some behaviors that are politically smart but rent-seeking in nature. The Daily WTF has an entry called ["The Speedup Loop."](#) It describes a team of developers working on an application and adding code whose only purpose was to slow it down. During slow weeks, the tale goes, they'd remove a bit of the offending code and tell management that they'd been hard at work speeding up the application.

I will say unequivocally that I find this behavior unprofessional. But I will also say that it represents an effective political way to say no via quid pro quo (i.e., by better idea or counteroffer). I mention it here because it's *anticipatory*. Sooner or later, management will want a slog of some kind. The people doing this have anticipated it and armed themselves with a different sort of good outcome that they can summon for bargaining whenever they please. Again, you have to evaluate this for yourself.

Now you have a number of tactics you can use for saying no, even in situations where you seem to have little choice. This list is not exhaustive, but it is a tangible one that you can bring to bear and flesh out. If you want to generalize beyond what I've done here, then keep in mind the power balance between you and the company. As an employee, that power balance is intensely asymmetrical, and the scales don't tip in your favor. The people in organizational positions above you hold all of the cards. Saying no tends to require outsize expenditures of what little capital you have.

The key is thus to acquire capital. As you've read here, that can happen in a variety of ways with a variety of ethical implications. You can do it by having great ideas or highly in-demand skills. Or you can do it with blackmail and trickery. But however you do it, you do it best by tipping the power balance scales in your favor and then judiciously parachuting out of suboptimal situations by saying no to them.

Chapter 37: Advancement

The entirety of Part 3 traced the evolution of the corporation to its current state. The entirety of Part 4 to this point has defined how to attain success in that current corporate state. Mainly, this revolves around coming to think of oneself as “other”—a lonely business entity moving among others that see themselves as part of a larger, illusory whole.

Think back to the beginning of Part 3 to understand the deep irony at play here. One succeeds in the modern corporate context by rejecting the very bedrock upon which the whole construct is built: founder legacy. In a sense, you might consider yourself something of a Trojan horse with this approach.

I mention this because I get that the mission seems daunting. Throughout this part of the book, I’ve suggested that you isolate yourself while engaging in high-risk and possibly depressing activities. Maximize external options, steer clear of organizational praise, figure out good ways to say no to those around you.

The question becomes how, specifically, does one pull all of this off? What does it look like to climb inside this horse, have Acme Inc. wheel you past the pragmatist guards and idealist bureaucrats, and to hop out among the inner sanctum of opportunists? In this chapter, I’ll lay out a more tangible approach. This chapter guides you from line-level programmer through avenues of advancement.

As I mentioned earlier, the first thing you need is your escape plan. Take a deep, sad sigh, resolve to leave your professional programming days behind, and pick out something else to do. Specifically, go job title/description shopping. You do not want to shop for the job of your dreams. Rather, you want to plan your road map away from delivery.

Here's the best way to way to approach this. Look around you for opportunists not directly responsible for delivering anything. These will be folks with management or higher roles who do not guzzle the founder legacy kool-aid or worry about the company's mission statement. Pick a few of these folks out that are not yet in the corporate stratosphere; this won't work if you pick the CIO of a 40,000-person company. Study them a bit. What titles do they have? What educations? Past jobs?

Using these folks, build a composite of what you want to be doing in, say, five years. Then work backward. If you see a dev manager role five years in your future, what would three years in your future need to look like? And whatever you do, don't answer something like that with "principal architect" or anything else that requires piercing the journeyman idealist veil. I said three years, not thirty.

In fact, that's the key thing that separates this blueprint from garden variety high-powered career advice. Anyone ambitious and goal-oriented will tell you to have vision and work backward. But the reality is that you need to have vision, work backward *and* avoid the self-defeating, stack-ranking world of techie chest thumping.

I can get you started with two fairly obvious paths to pursue: consulting and project management. Both of these promise to let you step out of the software-engineer-I-through-VIII conveyor belt of pseudo-meritocratic turn waiting. You can step back in as the boss of the people doing that. If you can come up with other options that suit you better, then by all means do so. But I'll proceed talking about these relatively standard options.

If you want to be a dev manager within five years, it's probably safe to assume you should be a consultant or project manager within two or three. Now iterate a step back and think of what needs to be true in order for you to have those titles and responsibilities in two or three years. That should lead you to actionable tasks in the here and now. You're ready to get to work remaking yourself.

At this point, I need to mention something absolutely critical. As a programmer, you are used to very objective, measurable criteria populating your resume. You talk to recruiters in terms of estimated proficiencies with languages, frameworks used, methodologies followed. Leave that world behind and start thinking in terms of narrative and what yours needs to be. You want to go from what you're doing now to dev manager in five years via a role where your title is "consultant." But when interviewing dev managers, organizations deal in narratives and feelings far more than you're used to. Even if you have the title "consultant" but you're just banging out code same as always, you're still in better shape. It's of course better to be doing actual consulting work you can speak of, but you can work a narrative around "consultant" that you can't around "software engineer III."

When you look down the road, imagine interviewing for dev manager. Imagine the experience you want to have going into that interview and contrive of ways to make it plausibly true. Then do the same for the immediate advancement and the consultant or project manager roles. Imagine the things you'll be asked and the answers you'll give. Then imagine how to make those answers as, well, truthy as possible.

On this point, I cannot overemphasize the importance of faking it until you make it. As an opportunist, you simply do not have time to manufacture all of the experience you seem to need in order to get these two roles in the next five years. Doing so would force you to wait a decade or more as people retired and you backed into positions by default. You're going to need to jump way out of your comfort zone and you're going to need to be okay with a bit of creative framing. To lay the groundwork for that, you're going to need to use your current job as a gamification engine to earn all sorts of project management and consulting badges.

Get yourself in a position, if only for a day, where you're supervising something. Contrive of a way to be part of your department's candidate interviewing process. Create some inane thing called the

“committee for competitive salary review” and somehow get your boss to buy a lunch for its quarterly discussion of nothing of any import whatsoever. Make twenty-five Gantt charts. Seriously, make a list of these types of things that you want to be able to tell the next person considering you for a role, and go at it like you were gaming Stack Overflow for some arcane gold badge, upvoting long dead answers or something. You’re building the narrative that you’ll be pitching to your next boss.

With the seeds planted, it’s time to plan ahead to harvest. The things you’ve manufactured will sound contrived and hollow, so the first thing you need to do is come up with subtle ways of making them sound better. Then practice. If you can convince the people around you in your current group and company to remember a more impressive version of you, you’ll be in great shape for an internal transition/promotion or an interview with another company.

Here, you need to make yourself look big, as if a bear were looming nearby. Generalize singular experiences to multiple experiences without technically lying. You know people that do this sort of thing all of the time. Instead of talking about the one time you had supervisory responsibility, say instead, “Every time I’ve had supervisory responsibility for a team, I’ve...” If this were a US political debate fact-check, the panel would call it “true but misleading.” And that’s fine because “true, but misleading” is opportunist for “I’m about to get a better job.”

If this feels icky to you, it’s because that’s what the programmer’s natural enemy, the sales guy, does. You come to the client meeting prepared with facts and figures that suggest a much more conservative timeline, and the sales guy interrupts you to say, “Don’t listen to him—we’ve never failed to deliver a project this big before.” Before you stop spluttering, the deal is inked. Afterward, you say to the sales guy, “We’ve never failed to deliver a project this big before because we’ve never *had* one this big before.” Sales guy grins at you, goes home, and collects a fat commission.

If that seems horrible and unfair, then the opportunism game will not be for you. Let's be clear about something—the *entire world* that you're venturing into with these ambitions is one of sales and nothing besides. One of the main reason that engineers are so undercompensated is that we opt to create a cocoon for ourselves where we can indulge delusions of meritocracy and skill directly correlating with value. The rest of the business lets pragmatists and journeyman idealists exist in this warm cocoon, and they only charge us a 200 percent markup on labor for it. If you want to start getting some of that back, you're going to need to get comfortable with creative embellishment.

Turn single instances into generalities. "Every group I've managed projects for has done X." Make unusual sound routine. "Every time I've found myself managing a group, we've done quarterly forecasting." Speak flatteringly in upper and lower bounds. "I've never delivered a project that went more than 2 percent over budget."

Once you're comfortable socializing narratives like that, test the bounds of those around you. Push until someone calls BS, and then back off. If you understand the limits, you can capitalize on human cognitive biases to normalize the stories. Extrapolate and upsell your experience routinely with those around you, and your tale will start to become part of the general, accepted corporate canon. This is your entree into creating corporate culture for idealists and pragmatists, and you're well on your way to opportunism. A technique that you might contemplate is to weave them into the story in an extremely flattering light. This makes them all the more likely to recall your narrative of events and to go along with your creative enhancements.

I'd like to briefly point out here that you don't want to actually make stuff up. That can backfire. The trick is to do this without saying things that are technically untrue. Manufacturing pure BS is a higher risk, higher reward game, but in my experience, it alters the expected value equation against you. People are more likely to call

you out or dismiss you if you make things up. So I'd advise a loose but consistent relationship to the truth.

In an earlier chapter, I talked about moving around like a shark as a key to avoiding carnival cash. That strategy confers another benefit here: it facilitates your upward trajectory. When you're hired as a software engineer II, it will be tough for the department or company that hired you to view you as a project manager or dev manager. You tend to be anchored in the caste to which you're hired, by default. As a software engineer, that makes you a pragmatist. It'll be much easier to get a fresh start somewhere with a blank slate to work your way into the management layer.

Think of it this way. As you wander around your developer job, shamelessly collecting experiences that would sound nice in an interview for your next position, you're building an alibi of sorts. You're like the kid in high school that went on vacation to the beach one year, met a girl from Canada that he took a walk with on the beach, and came home claiming to his disbelieving friends that he had a Canadian girlfriend. Your current job is like the vacation—no one watching you take that walk will believe she's your girlfriend. You need distance and a lack of firsthand witnesses to embellish your story into being the flattering one it should be.

Each transfer, promotion, or company switch becomes a plausible point of narrative enhancement. Your resume and reference checks offer the illusion of official validation, whereas the lack of witnesses to speak in detail gives you something of an opportunist tabula rasa. As you keep moving, you can have more Canadian girlfriends that are increasingly fabulous.

This strategy will pay off for you. If you carefully build the resume of the person you want to be when you next interview, you will land those interviews. Learn from your mistakes, figure out the gaps in your knowledge, and correct them. Sooner or later, you'll get that offer.

Now, given that you've audaciously worked your way out of your comfort zone and into a role potentially beyond your capabilities at the moment, impostor syndrome may kick in. You'll look at your new responsibilities and think, "My God, what have I done?!" But when you feel outmatched, remember...don't feel outmatched.

Whatever you do, accept more responsibility, authority, and organizational power. Always say yes to opportunity, even if you have no idea what you'll do to make it work. You can figure out a way. Most ascendant opportunists are smart, and most programmers are smart. It's likely that, if you're the type of person who wants to read this book, you're smart. No doubt you'll be able to figure something out to get the job done, even if it's not ideal.

But if you can't—if you're truly lost—that's okay, too. I say this because we're at the end of Part 4, the section about how to succeed in the corporate world by being an opportunist. There would be no more appropriate way to wrap this playbook than to bring you to the defining play of the opportunist playbook. If you've bitten off more than you can chew, the solution isn't to nobly offer your own head for chopping and cede responsibility. The solution is to maneuver an idealist into the firing line.

Just as you want to create success narratives in order to grease the skids of your ascendancy, you want to create failure narratives to prevent backsliding. If you take a project manager role and see that you're off track for success on the new project, figure out a narrative other than "the project manager couldn't get organized."

Perhaps it's "the team consistently overcommits." With this new narrative in mind, find an overperformer on the team, take him aside, and have a frank discussion about how you think the team needs a superstar like him to goose them in the right direction, to reach further than they think they can hit. Now you have a team member consistently and eagerly telling you, in a public setting, "Don't worry, guys, we've got this!" The only record of the conversation where you encouraged this outcome is between you two. And this type of overperformer will heroically and stoically sit on that, in all likelihood,

while you soberly tell upper management that you really dropped the ball by not recognizing that the team was overcommitting and underperforming.

If you've ever read the Orwell novel *Animal Farm*, you can probably recognize exactly what I'm suggesting you do in order to become an opportunist. If you haven't read that novel and don't mind a spoiler, it's an allegory where a group of revolutionaries slowly evolve into the exact same oppressive governing structure they overthrew. So yes, you have it right. I'm saying that to get ahead, you need to become that which you probably hate right now.

But really, could the outcome have been any different? In a corporate setting where the upper echelons tend to be populated by people that you view as self-serving and self-promoting, did you think you'd get there another way? Did you think it would happen with overperformance, scrupulous honesty, loyalty, and waiting your turn?

It couldn't possibly go any other way. The corporation has evolved to its current state over the course of thousands of years. And that state is one in which you have to behave like a self-interested sociopath to enjoy sustained, rapid success. If that sounds like madness, it's because that *is* madness.

Chapter 38: The Madness of it All

I arrive at this last chapter of Part 4 with a sigh of relief. As I mentioned earlier, the contents of the preceding chapters do not constitute career advice, though one could take it that way. My description of how you could claw your way to the top of the corporate pyramid is not an endorsement of your decision to do so.

On the other hand, I *do* endorse righteous indignation at what I've typed throughout this entire part of the book, and I *do* endorse demands to know why these techniques should be effective at all. Why should programming be bad for a career in the software industry? Why do the denizens of the corporate world value narrative spinning more than software creation? Why do we resign ourselves to playing zero sum games at the behest of paternalistic institutions? Why do we cannibalistically drive our own wages into the basement? Why does behaving like a decent, collaborative human being signal to executives that you're a subordinate? Why do we view it as some kind of moral duty to work endless free hours for the weak promise of future money? Why do we work for micromanagers we don't respect and companies that inspire Stockholm syndrome? And above all, why are we so conditioned to think it could not possibly be otherwise?

I'll return to my own career journey here for some perspective. As I made my way through the corporate world, I did so with opportunistic behavior. But I did it without the specific, milestone-oriented game plan I laid out in this part of the book. Instead, I implemented some career hacks of my own, did a whole lot of observation, and eventually ran thought experiments as a consultant. Only in retrospect do I have access to such a mercenary play book.

I don't currently play it, however. My career ambitions lie along a different path. In fact, I have set a goal for myself to get back to

programming.

Given all that I have told you, doesn't this seem insane? I just got through explaining in detail how programming, a subordinate pursuit in the corporate world, slams your career into low gear and keeps it there. Am I aspiring a return to being nagged by project managers? No, of course not.

I currently split my professional time between executive consulting and creating content that generates passive income (for instance, writing books). The consulting pays me well, but it can get tiring both being in the corporate setting and keeping my pipeline full of work. Also, I miss programming. So my goal is to keep ratcheting up the percentage of my income that comes passively from content I generate. Once I hit critical mass with that and no longer need to consult, I intend to return to software development, building things that I find cool and may, as a bonus, also make me money. I will return to programming, but I'll only build what I think would be valuable and interesting, on my own terms.

I say all this not to lay out my goals before you as if this were some kind of unusually honest performance review. Rather, I say this to help you understand how insane the career path is for someone who loves programming. Should I succeed in executing on my plan, I will have left the ranks of lowly pragmatists, budged through two layers of idealist, and topped out with an executive position. From there, I will have gone off on my own, building a consulting practice while spending my spare hours generating content. Eventually, I will have gradually, painstakingly shifted the balance between consulting and content to the point where I can semi-retire and program on my own terms.

I will need to have pulled off a rather amazing career feat in order to both program and not to be a low-status grunt. In a world where nothing is in higher demand than software, that is utter madness.

But the madness goes beyond simply the programmer's place in it all. It's not as though having all developers report directly to the CEO

would fix all forms of dysfunction in the corporate structure. There's still plenty to go around. In the global, high-tech, twenty-first-century knowledge work economy, the pyramid-shaped megalith struggles to keep up. It bogs down the world with its struggle.

Let's revisit corporate history a bit and consider what all we take with us as we dutifully carry on the commercial approach passed down by our parents and their parents. The modern corporation is ubiquitous. Even an independent consultant and content creator like me can't escape it since I have clients. For most, there's absolutely no question of what happens—go to college, get a corporate job. It has spread to every corner of our society and brought with it Taylor's concept of layering. Only the inconsequential peons actually do the work that operates the business. Important people supervise those goldbrickers and *really* important people sit back and pull the strings of those managers between rounds of golf. For all of its khakis and team building activities, modern corporations look like feudalistic societies, but with donuts in the break room.

Reaching back even further, we see all of the historic properties of the corporation evident today as well, even when it makes no sense. In spite of owning our own means of production as programmers, we still perceive enormous barriers to entry when it comes to entrepreneurship. And even besides its ubiquity, the corporation plays an enormous role in our culture, our personalities, and our self concepts. We still measure our influence and position in society with their metrics, and we still perceive corporations as a gestalt. Liberals often invoke the political barb that "corporations are people too," but whatever your politics, the US jurisprudence does recognize the concept of ["corporate personhood."](#) They have personalities, they throw their weight around, and, internally, they have mythology and culture. And all of *that* is driven by founder legacy (ego).

Boiled down to its essence, our corporate structure may perhaps be the ultimate expression of will to power—a single individual setting out to build an empire and be honored with statues and stories, whispered in hushed tones. This is a quixotic goal, to be sure, as it's

unlikely anyone will remember a corporate founder in a few hundred years. The whole thing evokes the imagery from Percy Shelley's poem "Ozymandias."

I met a traveler from an antique land
Who said: Two vast and trunkless legs of stone
Stand in the desert. Near them, on the sand,
Half sunk, a shattered visage lies, whose frown,
And wrinkled lip, and sneer of cold command,
Tell that its sculptor well those passions read
Which yet survive, stamped on these lifeless things,
The hand that mocked them and the heart that fed:
And on the pedestal these words appear:
'My name is Ozymandias, king of kings:
Look on my works, ye Mighty, and despair!'
Nothing beside remains. Round the decay
Of that colossal wreck, boundless and bare
The lone and level sands stretch far away.

The founders among us—the successful ones—dedicate themselves to being Ozymandias, King of Kings. The rest of us just help them build statues of themselves.

This foundational hubris requires two components that the modern corporation has in spades. Those concern legacy and ubiquity. Together, these give rise to the essential characteristic of a modern corporation: growth for growth's sake.

If you want to understand how much this governs our fate, consider a business model that I have not mentioned but that is also ancient in nature. I mean the solo practice—doctors who spend their lives

ministering to patients or lawyers who operate as Joe Smith, Esquire. Whatever ego these sorts may or may not have, they do not chase viral growth and ubiquity. But you know what else they don't do? They don't answer to layer upon layer of lawyer managers and doctor executives, and they don't exist in a world where the *lowest status* of practicing law and administering medicine is the *practicing* of the law or the *administration* of the medicine. Only we corporate programmer citizens find ourselves in that tragicomic situation.

The modern corporation's pathology derives precisely from the mandate to scale at all costs. The scale started with ego, expanded to politics, spanned the globe for the sake of territory, capitalized on automation and economies of scale, build commercial empires that rivaled militaristic ones, and has culminated in the Silicon Valley dude-bro as the idol of our age. But what need for scale for its own sake is there, truly, in the knowledge work global economy? Do you need to be part of a massive megacorporation to design cloud hosting software? Do you need fifteen layers of boss to build a document database?

The assembly of empire and the glory of the emperor don't come cheap. As the sales folks say, "You've got to spend money to make money." It would look weird if the CEO of a hotshot company didn't fly around in a corporate jet, and clearly you need an office space that humbles the Ritz Carlton. Everything that follows inside of the organization builds on and reinforces centuries of corporate stratification. Taylor's scientific management made industrial-age companies more efficient. It also provided a great sieve for filtering people to the lowest effective pay bands.

I very much enjoy my life. As a consultant, I frequently travel and work remotely. I can create content from anywhere. This creates a life where I bounce around the globe, working from wherever I happen to be and at whatever hours I please, all while earning a respectable living. This is possible because of our modern global knowledge work economy. We have technology that allows us to be effective from anywhere in roles with high market value.

Yet we still work as though we didn't.

The madness of it all is that we're prevented from living this sort of life *not* by logistical obstacles but by inertia and cargo cult living. We're factory workers that can't figure out that we own factories. We're serfs that can't figure out that we own manors. We're founders that haven't figured out that our legacies are freedom and self-determination.

PART 5: HOW TO STOP PLAYING GAMES



Chapter 39: Where We Go from Here

Throughout this book, I've woven in my own personal narrative among my observations and research. Hopefully this humanizes what I'm saying a bit, but in truth, I consider that an ancillary benefit. I do this because I cannot meaningfully separate my experience from my opinions.

Even though I rose through the corporate ranks, I ultimately wound up leaving the corporate world and staying away. By most measures, this was a pretty questionable play. As a thirty-three-year-old CIO, I had a nicely manicured path to a lucrative corporate career—spend a few years as the CIO of a small company, then interview to be the CIO of a medium-sized company by forty, and perhaps a global organization by fifty. I'd more than figured out how to step off of the career conveyor belt and reinsert myself somewhere more favorable. I was looking forward to decades of leadership, affluence, and comfort. And I tossed it in favor of the giant question mark of self-employment.

This came from a place of serious disillusionment. I had done some job-hopping in the preceding years, and my disillusionment with each individual job began to generalize into disillusionment with the corporate condition. Each time I landed and found the situation lacking, I became more and more convinced that I would find any and all such situations lacking.

Recall the description of the corporate hierarchy in terms of what they give up. Pragmatists give up hope, idealists give up perspective, and opportunists give up their ethical compass. As a member of this last group, I continuously found myself choosing between what I thought was right and was best for my position. This wasn't just a matter of having to stack the deck to get ahead. It was also a matter of navigating the waters to stay once you're there.

At every job I took, I wound up in a position where my own best interests were at odds with those of customers, clients, coworkers, and my charter with the organization. Someone wanted me to bill a client for busy-work instead of saving them money by telling them how to automate or eliminate the task. Someone wanted me to fudge some data to make our offering look more attractive than it was. Someone wanted me to give people reporting to me titles that would make them less attractive on the open market. The list goes on and on and on.

None of these conflicts of interest in and of themselves created a crisis of conscience in my career. Sometimes I would push back for the right thing, and sometimes I would even win the day. Sometimes I'd hold my nose and do what a higher-up wanted, preserving harmony and good graces. And sometimes I'd get creative. Opportunists ceding ethical high ground generally don't experience a crossing-the-Rubicon moment, like Anakin Skywalker killing Mace Windu. The death of their ethical purity is one of a thousand cuts.

Without hope of advancing, pragmatists can either adopt a "just following orders" mindset, or they can take self-destructive principled stands, depending on their personal appetite for risk. Idealists, by ceding perspective, don't have to worry about this. They resolve the resultant cognitive dissonance by assuming those giving the orders have special knowledge that secretly justifies the apparent conflict. Opportunists have none of these luxuries. They will be subject to this essential conundrum until such time as they have a mandate as CEO or they own the organization.

This was my essential realization. Until I owned the operation, I would necessarily face choices between what I felt was right and advancing my career.

It's easy to look at your manager (and then her manager and on up) and assume those people have the power to do the right thing. But recall that this is the organizational layer of faux owners. They get to tell you what to do only at the pleasure of the owner. They face the same conundrum that you do but with higher stakes, and often with

the burden of presenting policies that they hate as if they were fully on board. The pyramid-shaped corporation is an autonomy iceberg. For everyone below the waterline, there is an omnipresent understanding of where the door is if you don't agree, and there's not much room above the water.

Ironically, the way to get above the waterline and into a position of true autonomy requires substantial ruthlessness. Those who climb the iceberg to get to the top are the ones most likely to choose advancement over the right thing every single time. And when you consider that not everyone looking to claw to the top is interested in using the position to do what they feel is right, you can see that becoming the big boss to do the right thing is a truly quixotic concept.

The best you can hope for is bouncing around from place to place, hoping to find a structure above you that is mostly ethical and competent. From there, you hope that your disagreements with them are largely minimal and easily resolved. Oh, and you also hope that you actually enjoy the job, the work, the benefits, and the organization.

As you can see, if you want to keep hope and perspective, your best play is to continually minimize guilt. I didn't like these options or this perspective. I wanted hope, perspective, *and* to feel ethically content with my life. So I quit the game altogether. I left the certainty of a good career for a different certainty—that I would feel good about what I was doing.

I've heard it said that, apart from being independently wealthy, no one ever really has no boss. As an entrepreneur, your boss changes from someone in a corner office to your customers/clients. There is truth to that. As long as you exchange your labor for money, those with the money can attach terms to your receipt of it. But the wisdom that you always have a boss misses a fundamental distinction. As an independent or entrepreneurial owner, you're participating in a system where you're incented to have as many bosses as possible.

In the corporate context, you're incented, if not mandated, to have only one.

In an investment portfolio, financial advisers will tell you to diversify, with the idea being not to place all of your eggs in one basket. This way, if the bond market in your country tanks, you take a hit without losing your entire portfolio. I look at the market for my labor the same way. With a portfolio of clients and prospects, I can push back or refuse requests without putting my entire livelihood at stake. In the corporate context, I cannot.

These days, I find myself in a position where no single boss dominates the landscape of my life. I can and do say no to things that I think provide no value or that don't align with my ethos and integrity. I no longer cede hope, perspective, or ethical certainty. My course is one of many hours and relatively large financial risk and uncertainty, but I still can't help but feel that I'm on the right track.

The depressing parts are over. For the remainder of the book, I'm going to talk about where we are and where we're going. I'll talk in terms of optimism and improvement. And the positivity makes sense: for the foreseeable future, the demand for technologists will be virtually unbounded. I think we can take advantage of that to improve our situation. I think we can get away from pyramid-shaped mega corporations, moral hazards, and layers of idealists between us and control over our ethical destinies. And I think we can normalize controlling our own destinies now that we own the means of production.

To paint a picture of what's ahead for developers, I'm going to rely partially on my own observations. But I'm also going to broaden the scope of focus. My path and preferences will resonate with those of you that have similar outlooks, but you needn't like the same things as me to venture onto a new path. That's why I'm going to profile a number of technologists that have succeeded outside of the standard wage world and build my ideas of what comes next atop that foundation.

Chapter 40: The Coming Crunch

Before I get to these profiles, let's consider certain mechanics of the software industry as it exists today. We tend to view organizational structures as relatively stable, but think of them as glaciers instead. They move constantly, even if it's too slow to be immediately perceptible.

First, consider the nuts and bolts of a pyramid-shaped, Tayloresque organization. I imagine you'd consider it common knowledge that the people in any given layer of the pyramid make more money than those below them. It's a very symmetrical facsimile of organizational value.

Given the value perception and the laws and regulations around fair pay, this puts a number of weighty constraints on organizations when it comes to staffing. To get concrete about it, let's say that I start an app dev shop, and I hire a manager and a team of developers. I pay the developers \$100K each per year and the manager, say, \$110K. Life is good, and fair pay requirements are met.

Over time, my organization grows and I have five managers of five teams reporting to me. I hire a VP of software development because this is more involvement than I want. Now all of the dev managers report to her, and she makes \$120K. The pyramid remains intact, and no one can gripe about fair pay.

But let's now say that I start to have trouble hiring. I've gobbled up all of the developers in the area, and yet more demand still exists. If I want to get any more, I have to lure them away from other companies with incentives—incentives like more money than they've been making.

I want to start offering them \$110K per year, but now I have a problem. I can't do that without bumping the pay of my five

managers and my VP as well. In other words, I can't adjust line-level salary in a vacuum to respond to ebbs and flows in labor supply. I have a growing pyramid-shaped dog being wagged by that tail.

Now take this dynamic and apply it at a Fortune 500 company with thousands of line-level employees and nine layers of management. My goodness.

The strategic difficulty that organizations face comes from multiple angles. They don't have endless upward flexibility to adjust salaries. They have to consider cost, both direct (laborers) and overhead (management), and operate within a budget. Reverberating salary increases thus become non-starters in many cases, forcing organizations to get creative.

I won't go into all of the details, but this can take any number of forms. Most commonly, especially with union-heavy or government organizations, they can entice laborers with hefty perk packages. You can't pay them more, but you can give them twenty-five company holidays a year. Another common form is to look for commodity staff augmentation labor, often using H-1B visa programs and offshore labor. A third approach is to tilt at the windmill of magical management trends—bring in somewhat underqualified, cheap staff and then hire expensive consultants selling the snake oil of a magic process that does more with less. (Sadly, this is often the guise under which “agile” and “lean” and the like are sold to large companies.)

Suffice it to say that large organizations have a management weight problem. Their massive revenue figures allow them to defray some of this by paying more excessive salaries to upper management than other organizations would. But it still adds up, and it still puts a heavy deflationary pressure on the wage of line-level employees. Add to that the soul-crushing bureaucracy and risk aversion in those companies, and you can say that, on the whole, they have a hard time attracting, paying, and retaining talented software developers.

Let's put a pin in that for the moment. I want to talk now about another dynamic in our industry: the growing software developer shortage. About a year and a half ago, I attended the Pluralsight Author Summit, where they made a presentation on the importance of teaching people to program. They cited an economic projection that, by the year 2020, the shortfall of software development professionals against available jobs would reach *one million people*. The world's appetite for automation seems to know no bounds, and we do not have the people to satisfy it.

You probably understand this dynamic yourself, despite the sometimes-glacial nature of industry trends. I bet you get a pretty steady stream of recruiters calling you, even when you have no interest in looking for new work. They do this because the unemployment rate for experienced software developers is basically zero. The only way to fill their position is to steal someone from another company. As fast as we can train entry-level developers, more demand opens up for the creation of phone apps, websites, and refrigerators that tweet.

If you have even a passing familiarity with market economics, you know what this means for salary. You've probably experienced this firsthand as well. When the market shows intense demand for labor and a shortage of supply, a competitive bidding war ensues. This puts upward pressure on our salaries across the board.

Coming back to the topic of heavy, pyramidal organizations, this situation leaves them in a crunch. When developers have eight different bosses, paying all those bosses makes it harder to pay the developers. That's pressure from above. And as competitive smaller firms have increased need, they start filching those same developers, exerting wage pressure from below. Something has to give.

I spend time with a wide variety of large organizations. Based on what I've seen working with these places, and based on my understanding of industry trends in general, I think I know what will give. Those companies will move away from employing software

developers as wage laborers. The economics simply don't make sense.

Don't get me wrong. These companies will absolutely continue to want and to need software and automation. They just won't continue to staff it with salaried exempt employees in the same kind of numbers.

Large organizations, especially ones that do not directly sell or monetize software, have a number of factors working against them. I've spent this chapter talking about the titanic downward pressure exerted on wages, and that reigns as factor number one. All of the rules about what employees have to make relative to one another completely evaporate when it comes to contractors and custom app dev shops.

But it goes deeper than that. Massive companies have bureaucracies in place that serve two principle purposes: risk mitigation and communication at scale. Both of these considerations are anathema to skilled software development. These companies make huge targets for lawsuits, so they implement cross-cutting policies. They're concerned about what software is allowed on what machines; who can download what and when; what IDEs, frameworks, and languages people can use; and on and on and on. And when you lumber along like this, working with ten-year-old language versions and tooling with no internet access, other companies come along and drink your milkshake. Good software development doesn't happen in environments like that.

The crunch to which I refer is the one that will effectively expel developers from the ranks of large, unwieldy organizations. It will happen in fits and starts and will appear slow to develop, to the naked eye, but make no mistake. It's in progress and will continue.

The trend will drive software developers to three main camps: large software companies, startups and small product companies, and custom app dev shops (including solo freelancers). That's how I see the short to medium term playing out, anyway. Over the long haul, I

think you'll even see fewer developers working for massive tech companies, since they have their own forms of bureaucracy and risk aversion, albeit less obtrusive. When you really dig into the tech titan organizations, they tend to innovate via merger and acquisition more than by homegrown groundbreaking stuff that hearkens back to their leaner startup days.

It's against this backdrop that I predict the freelancer and custom app dev shop start to dominate the technology space. A picture emerges of organizations having their automation and development needs met by legions of freelancers and specialty software shops of various sizes. It's a more harmonious form of labor specialization.

And in that world, a new entity reigns supreme: the autonomous developer opportunist.

Chapter 41: Studies in Success—Those Who’ve Already Won

With everything I’ve said so far, the developer opportunist seems like something of a unicorn—that is, unless I’m talking about an erstwhile developer looking to escape the delivery trap. But that’s because the entire book has thus far focused on the standard corporate world. By stepping outside of that world, one can have both descriptors without looking to escape writing code.

People have already done this with a great deal of success. I suppose I can count myself among those ranks. As a solo consultant and entrepreneur, I do a variety of things to earn my living. Sometimes I write code, sometimes I produce content, sometimes I advise, and sometimes I teach. All of these things I do while happily wearing the badge of opportunist. After all, it’s not a stretch to imagine myself as a solo entity, fending for myself amidst a sea of other entities. That is literally my professional life.

In contemplating exactly what makes me prefer this arrangement and what I recommend for all of us, I have to consider what the corporate world takes from us. We must choose to give up hope, give up perspective, or to give up high-minded ethics. In charting a course forward, I propose that we give up none of these.

The developer opportunist, an entity apart from salaried development in the corporate world, cedes none of these. To meet these criteria, she most likely operates as a free agent of some sort. But she might instead job hop so frequently as to resemble a free agent. Or she might operate as part of a highly unorthodox corporate structure. It is my hope that the coming years will give us a rise in this last option, but for now, we have to accept that blasting off on your own (or job hopping) is easily the most reliable choice for becoming a developer opportunist.

I don't want this part of the book to be me preaching from some sort of implied pedestal. I never set out to make this book's tagline, "I've made it, and you can too!" There's no bulletproof ten-step process I can hawk for \$39.95. You're in uncharted waters here.

The best I can do is to provide a lot of case studies for pattern matching, an idea I lifted from [developer opportunist Gayle Laakmann McDowell's interview on Developer On Fire](#). I've conducted interviews with a number of people that I would consider to be developer opportunists, and I'd like to share them with you.

Let me be a bit more specific about the criteria that I used when approaching the people you'll meet. The developer opportunist must retain hope, meaning that they retain the notion that they control their destinies. They must do this while maintaining perspective, unlike corporate idealists who keep hoping only because they haven't figured out that hope is not warranted. Developer opportunists can hope for meaningful advancement because it is within their power to execute. Lastly and most importantly, the developer opportunist's advancement must not be predicated upon the ethically questionable proposition of gaming the system. They are not required to play games if they do not want to.

Now, all of that is fairly philosophical, so I distilled it into some more tangible criteria. I looked for people whose careers could be defined by their success and autonomy and who enjoyed both those things outside of the standard corporate arrangement. This is not to say they aren't employees or people otherwise affiliated with corporate entities. Rather, they do not work in environments where you could map them to the standard corporate pragmatist/idealist/opportunist roles.

Here are the people that I interviewed, in alphabetical order by last name.

David Boike

[David](#) works for an organization called [Particular Software](#). “Works for” carries an interesting connotation in this case, however. He works for Particular the same way that Tom Hagen in *The Godfather* worked for the Corleone family. He has a very special practice; he handles one client.

David, like others affiliated with Particular around the globe, has his own company, and Particular is a client. In this sense, David’s arrangement resembles contract staff augmentation—but with a key difference. Generally, staff augmentation firms prefer to deal with individuals as employees rather than as businesses. Particular prefers to deal with David as a business.

This enviable arrangement (having his own business and working for a highly respected organization in the .NET architecture world) came about sort of by accident. David had a few different jobs over the years that follow the standard corporate narrative. But during the course of one such job, he had occasion to begin using Particular’s [NServiceBus](#) to help with scaling challenges faced by his team. During that same time frame, he had also had occasion to start a blog, giving rise to the first happy accident. Since NServiceBus was not yet commercialized at the time, a few of his blog posts started to become de facto documentation for the product. Through this blogging effort, he became what Particular thought of as an NServiceBus champion.

Sometime later, David went to work for an app dev consultancy that was amenable to him moonlighting and forming non-traditional arrangements. One of the things he did in his spare time was agree to an offer from [Packt Publishing](#) to write [a book about NServiceBus](#). This prompted [creator of NServiceBus and Particular CEO Udi Dahan](#) to reach out and ask David to help formally with NServiceBus training. David’s employer actually facilitated this, letting him use their office to conduct training.

After a while, this led to the offer that created David’s current state of affairs, partnering with Particular. Though he was happy at his the consulting firm, he considered it “an offer I couldn’t refuse.” But this

time, it wasn't like *The Godfather*—no firearms were required. He couldn't refuse because it was a truly great offer.

David now has an obvious developer opportunist setup. He has a remote-first arrangement with a developer-founded, well-respected organization. All of this is done as the founder of his own consultancy but with a good amount of security and no pressing need to worry about pipeline management and marketing.

James Grenning

James is the founder of [Wingman Software](#), a software training/coaching consultancy focused on embedded software. Specifically, he helps bring agile software development practices to that space.

He has written a book for the Pragmatic Bookshelf called [Test-Driven Development for Embedded C](#), as well as numerous white papers. James speaks internationally about related topics, and he also invented the agile practice of [“Planning Poker.”](#) Many of you are no doubt familiar with the [so-called Agile Manifesto \(actually the “Manifesto for Agile Software Development”\)](#). James was one of the authors of that particular document.

In high school and college, James tried to avoid computers (“no windows in the computer room, stay away, I thought.”) But he discovered that programming could be fun and that people would pay him to do it, so he began a career in software development. He enjoyed his initial jobs in this line of work, solving interesting problems and collaborating with good people. He picked up responsibilities as his career progressed, going from individual contributor to leadership roles and eventually managing a product engineering team, working under a general manager of the organization.

Unlike some colleagues who let go of their technical skills in leadership roles, James wanted to keep his skills sharp. He remained involved in technical details with his team and took on

some moonlighting work using C++. But he came to realize that there was no way for him to remain technically focused while getting ahead, a conundrum that should no doubt be quite familiar and predictable to you after reading most of this book. James knew that he needed to choose a different path.

James's friend ["Uncle" Bob Martin](#) had originally recruited him to join his company. But Bob had left a few years earlier, joining a startup and then consulting. Bob had contacted James a few times about working together as consultants, and now, at this career crossroads, he was ready in spite of the risk (and fortunate to be supported by his wife, despite having three children and a mortgage). He made the jump out of the standard corporate gig and has been happy ever since.

Matt Heusser

Matt is the founder of [a company called Excelon Development](#). Excelon helps with software delivery services—mainly programming and testing. They do some consulting, a bit of training, and a lot of writing, helping software product/tool vendors with content marketing. It's an interesting mix that furnishes exposure to all facets of the business of software delivery.

In 2008, Matt moved from more traditional industry work to taking a work-from-home gig with a venture-backed company. This introduced him to a higher risk/reward paradigm in which a quarterly sales review would inevitably make him worry about the security of his job. After twelve quarters of that, he was accustomed to this sort of risk. He also had a decent freelance portfolio and some savings. Starting to do contracting work seemed like a logical option.

He started doing this on his own and then, after a while, brought in contractors and employees to work for Excelon. It's a small shop that keeps overhead low, allowing them to provide high-end services for relatively low cost to clients. This creates high satisfaction rates and an interesting variety of work.

Thorben Janssen

Thorben Janssen is the founder of [Thoughts on Java](#), an organization that provides independent training and information products in that space. Under the umbrella of Thoughts on Java, Thorben offers courses and content that focus on the persistence tier of Java-based applications.

Early on, his career followed a fairly typical arc for a software developer. He started as an intern and worked his way onto larger, more substantial projects. As he was doing this, he began to assume some project management responsibilities. He coordinated with customers and with his team in addition to software development work. This went on for several years and with two different companies.

Eventually, the natural pressure of the corporate world had him doing more and more management and less and less programming. He missed the programming, so he started doing technical things in his spare time, including starting his blog, Thoughts on Java. This writing would be the beginning of a new career, even if he didn't know it at the time.

Over time, he perceived that the spare-time work on his blog was a business opportunity and negotiated a work arrangement that reduced his hours and allowed him to concentrate on Thoughts on Java. As his success grew, he needed even more time away from his normal job. The employer balked at this second request for reduced hours, but Thorben had seen the writing on the wall. He eventually put in notice and started working full time for himself, producing content.

The game-changing difference? During the reduced work arrangement, he'd launched a couple of info products and enjoyed a lot of success. He now realizes that he was cut out for entrepreneurship all along.

Sally Lehman

Sally Lehman is a software developer that has historically focused specifically on the reliability and scalability of email structures. She has served in this capacity for an impressive array of companies, including GoDaddy, GitHub, and Auth0.

Earlier in her career, she concentrated more exclusively on the email niche. But she has more recently broadened her focus, allowing her to add value to organizations in a variety of ways. She does this currently with Auth0 as a production engineer, working as part of their infrastructure team to diagnose and prevent service interruptions. Sally realizes this goal by making use of and building fast, reliable monitoring tools.

Sally was bitten by the automation bug at a young age. Her first computer experiences were with EMACs, MS-DOS, and Ski Free when she was less than five years old. As an adult, she spent time at GitHub, improving their email infrastructure and delivery for bulk email and notifications. This involved building tools with Rails and directing a team of customer support specialists. Then, at GoDaddy, she built out the Puppet and CI/CD server infrastructure for their email marketing and for Madmimi.com before going on to work for Auth0.

Unlike others I've interviewed, Sally has exhibited developer opportunism in a purely employed context. But she has done two things that relatively few normal developers ever do: established and leveraged a niche within the corporate world and worked for non-traditional companies approaching knowledge work in a new way.

Eugen Paraschiv

Eugen is the founder of [the site Baeldung](#), which produces large amounts of informational content for Java developers. This includes many free blog posts and information, as well as paid courses about the Spring framework. On top of that, Eugen also has a consulting

practice and is currently serving in an architect capacity for the firm [Uptake](#) in a remote arrangement.

Eugen got a computer science degree and then “did the corporate thing,” as he put it, for a number of years. He began to feel bored with enterprise Java development, so he rather spontaneously decided to start a blog. What he lacked in upfront planning, he made up for in enthusiasm for writing, and this allowed him to grow his audience. As his visibility grew, opportunities for more interesting freelance work opened up, particularly since his reach made geography a non-issue. He could work with clients all over the world.

As he was parlaying the reach of his site into increasingly favorable programming work, he also grew Baeldung, opening it to many authors and producing more content than he could have on his own. This led to the hiring of technical staff, the development of paid content, and the general operationalization of his site. This allowed the growth of a substantial income stream on top of the consulting practice.

Dave Rael

Dave is a freelance software developer and architect, and he is the creator of [the popular Developer on Fire podcast](#). He divides his time between serving the audience of his podcast and taking on consulting engagements.

While his educational background was not a computer science one, Dave learned on the fly working as a programmer during the dot-com bubble era. When the bubble burst, he remained employed, continuing to develop his skills—though not with the same relish. As he became comfortable there, his learning slowed and his tolerance for corporate bureaucracy decreased.

Dave stayed longer than he otherwise might have because of the nature of the economy at the time. So he was pleased at the opportunity to switch to a smaller, fast-growing company. There, his career grew along with the company’s fortune, resulting in more

learning and eventual promotion to architect. But unfortunately, the company's growth landed him back in a familiar position of being comfortable within a now-large and relatively bureaucratic organization.

He stayed longer than retrospect tells him he should have. When Dave eventually left, he carried with him lot of dissatisfaction with the corporate working condition. He decided to try his hand at independent work, taking on a few app dev contracts. When a family situation pressed him into taking some time off, he began to blog and record podcasts. He has since returned to consulting work but is more excited about the content creation and entrepreneurial ventures that are no doubt in his future.

John Sonmez

John is the founder of [Simple Programmer](#) and has worn a lot of hats over the years. Past titles include software developer, architect, trainer, coach, consultant, author, and entrepreneur.

John started out in a way that will likely feel familiar to most of us: as a software developer working in the corporate world. He did some work as an employee and as contractor, doing staff augmentation. But after a while, he felt that he'd hit the natural cap I've discussed in this book. So John, an entrepreneur at heart, began looking for alternatives to augment the standard career track.

This prompted him to start building mobile apps and to start a blog (the same site earlier). He made a bit of money with the mobile apps but nothing that was going to imminently let him quit his day job. The blog didn't make him money—at first, anyway—but it proved a more valuable long-term commodity in that it began to generate him notoriety. Over the course of a few years, he received an increasing number of emails and calls offering him interviews, jobs, consulting gigs, and other sorts of opportunities.

John seized on this momentum. He began to get his name out there in other ways as well, appearing on and making podcasts, speaking

at conferences, and leaning more toward consulting work. This led to an opportunity to work with [Pluralsight](#), creating courses for software developers to learn about various technologies. In fact, he dove into this opportunity so intensely that he stopped working a corporate job and made an astonishing fifty-five full-length courses.

As this point, money was starting to roll in through his blog. This led him to start creating products. He wrote his first book, [*Soft Skills: The Software Developer's Life Manual*](#), which established a niche for him as someone who could help software developers with more than just tech. He now has a wildly popular YouTube channel, many more products and courses, and a thriving site with a host of contributing authors.

For the full transcripts of my conversations with these studies in success, please see Appendix A.

Chapter 42: Building a Composite—Developer Übermensch

In his work *Thus Spake Zarathustra*, Friedrich Nietzsche stirred up a bit of controversy by proclaiming, “God is dead.” I have no interest in courting such controversy in this book. But if you can look past whatever preconceptions you might have about this philosopher and his work, you’ll find a concept in his book relevant to our own journey here. I’m talking about the übermensch, which translates to “overman” or “superman.”

In the time that followed Nietzsche’s work, this concept has been appropriated by all sorts of unsavory people, most notably the Nazis. But let’s distill the concept to suit our purposes.

First and foremost, the term most emphatically had nothing to do with race or any other category of dividing humans beyond morality. Nietzsche believed that the next step in human evolution would be categorized by those that rejected preconceived moralities in favor of their own. Rather than call for moral bankruptcy, the message could be interpreted as this: let’s say that you remove God and any concept of objective morality. The übermensch is the person who staves off nihilism by filling the resultant void with morality that results from a love of life. The übermensch, then, radiates morality in the absence of an externally defined and imposed morality. Think of him as the moral equivalent of a rogue planet in deep space with its own, internal heat source.

My choice to use the term übermensch may be a touch dramatic, but it is no accident. “Developer übermensch” characterizes the developer opportunist. Absent the standard corporate framework, the developer übermensch creates her own and radiates it as an example to others. The developer opportunist, motivated by (but not

myopically obsessed with) a love of her calling and craft, defines a new way to operate.

All of the people I've listed in my studies in success share this defining philosophical characteristic, but they also have other characteristics. In this chapter, let's take a look at a composite of the people that I've interviewed to define the developer opportunist, or developer ubermensch, in earnest.

First and foremost, all of them rejected in some way or another the standard prevailing corporate narrative. Whether due to boredom, random opportunity, specific goals, or luck, all of them created their own arrangements. They thought, "This isn't going to work for me, so I'll blaze a new trail instead." Such is the nature of developer opportunism.

But let's look at some other defining characteristics, some of which are more tangible than philosophical. This should yield a nice profile of a prototypical developer opportunist, and we can use that for the rest of the book to define where we go from here. I'll base the remainder of this chapter both on the biographies of the folks in question and the answers to additional questions that I asked them during the course of my interviews.

Developer opportunist trait 1: they don't spend all their time programming

The first thing that bears mentioning is that I asked everyone how much time they spent programming. The responses ranged from "a good bit" to "almost none."

I understand that this is probably true of *any* corporate programmer to some degree, but we're not talking about also going to that weekly status meeting. In all cases here, folks have responsibilities that go well beyond writing code. And while some expressed a desire to write more code, no one expressed a desire to do nothing *but* write code.

The first property of the developer opportunist is that he recognizes writing code is a means to an end. Whatever that end may be, other means matter as well, and those are also worth seeing to. Writing code may be an enjoyable means to an end, but it is not an end itself. Even in my own quest for independent wealth that allows me to program to my heart's content, it's not really the programming I love as much as it's the tackling of projects.

Getting this wrong will land you in an easily-manipulated position within a corporation. If you think that writing sublimely factored functional code is its own reward, some upwardly mobile opportunist is going to give you exactly that reward in exchange for seventy-hour weeks and 100 percent of the surplus value you create.

Developer opportunists do not allow this to happen. John Sonmez summarized it nicely: "You should definitely expand your abilities beyond just programming code and become a more valuable software developer by adding communication skills, architecture, and true software development expertise into your arsenal."

Developer opportunist trait 2: they market themselves

Speaking of John Sonmez, let's talk about the idea of marketing oneself. I say this because John found the concept so important that he actually built a course called ["How to Market Yourself as a Software Developer."](#)

Discussing marketing and sales for a moment will help us establish definitions, and I'll return to this subject in a lot more detail later. I consider sales to be a subset of marketing. Selling is the act of convincing someone to buy something. Marketing is the gamut of all that you do to raise awareness of and sell that same something. If I call you up and ask you if you want to buy a used pair of socks from me, I'm engaging in a sales call. If I build a website extolling the virtues of my socks, create a supporting Twitter account, and publish

blog posts about the most awesome used socks of 2017, I'm engaging in marketing.

I wouldn't specifically even try to *sell* the socks at first. I'd just raise people's awareness of them, establishing mindshare. My goal would be that, when you think about possibly needing new socks, you think of me. I'm investing in a long game, with the idea that giving away a bit of content now will lead to easier sales down the line.

Consider how this matters to a professional software developer. I won't be so idealist as to suggest it matters to you because increased revenue at your company matters to you. Rather, it matters to you in the sense of how you sell the only asset you have—your labor.

Most likely, you sell your labor by doing the equivalent of cold-calling someone and trying to get them to buy your used socks. To wit, you put on a suit, go into some random building, BS for a couple of hours using words like “utilize,” and hope to close the deal. That's right. An interview is sales. I'm guessing you're also pretty bad at it since you only spend a week doing it once every three years. Luckily, it doesn't matter that you're bad at it. Everyone is desperate for the used socks of your labor these days.

Now consider what *marketing* yourself would look like. You'd head to user groups and introduce yourself to people. You'd build a website with a blog and publish to it. You'd establish yourself as having expertise in some particular area or niche. And you'd do it all knowing that the payoff would come much later.

If you *market* yourself, the labor-sales process goes much differently. I can speak from personal experience, and I know the folks I've interviewed can testify to this as well. The idea of giving a resume to a recruiter and hoping for the best would never occur to me, even if I wanted a salaried job. I routinely get offers to shortlist my application and even offers for jobs with no need to interview. And these are not prefaced with, “I saw on LinkedIn that you have seven years of XML.” Rather, these people seek me out, specifically, through my

website because of something they read on my blog, in a book I've written, or in a course I've published.

David Boike similarly attributes his current situation to this form of marketing. In our interview, he credited his blog as “the single biggest reason I am at Particular today.”

Everyone I interviewed markets themselves, with varying degrees of deliberateness. But *all* of them have made names for themselves. Blogging is, perhaps, the most common vehicle and was frequently mentioned. But podcast appearances, user group participation, video creation, and speaking at conferences all factor heavily into the equation for the developer opportunists. They have made themselves known by marketing themselves.

Developer opportunist trait 3: they create content

Speaking of blogging, another universal thread running throughout developer opportunists' stories is content creation. The people to whom I spoke create all kinds of content for the consumption of others. Some write books, some make videos, some blog, some create training courses.

Creating content requires a great deal of effort, but it repays that effort by offering one of the most effective marketing opportunities imaginable. Creating content positions the creator as an expert on the subject. Make no mistake. The primary benefit of this content is rarely the actual royalties involved. James Grenning said that his royalty check from his well-known book “is not a lot of money.” But he also said, “*Test-Driven Development for Embedded C* is my main marketing tool.”

I've already stressed that marketing oneself is essential for developer opportunism. Creating publicly available content makes a great centerpiece for that marketing.

Speaking at conferences, attending user groups, appearing on podcasts—all are important. They get you exposure and afford you

opportunities to meet interesting people and find mutual benefit. But without something tangible to point at during those interactions, some of the luster gets lost.

To understand what I mean, imagine a developer who acquires some niche expertise on a subject. Perhaps she winds up optimizing database schema across her company for performance. She then takes the important step of creating a talk for conferences and user groups. Let's imagine what unfolds from there as two contrasting scenarios.

In scenario one, she gives the talks, which are incredibly well received. People mill around afterward, keen to ask her all sorts of questions and pick her brain. Many ask how they can follow her and get more information. She answers the questions and gives people her Twitter handle and website URL. On her website, visitors can find a few abortive attempts at blogging and a slightly dated copy of her resume. In scenario one, the best likely outcome for her is to be invited to interview for a job similar to the one she has, but that pays a bit more or affords her better benefits. It's theoretically possible that people approach her about consulting, but they're likely to be dissuaded by her full-time employment status and lack of any real marketing of her skills outside of a salaried context.

In scenario two, she also gives the talks and they are also well received. The same people mill, asking the same questions. This time, instead of giving out her Twitter handle, she says, "I've created a page on my site that you should check out: mysite.com/talk-followup." This page on her site has a few sections, which include "Where to follow me on social media," "Check out the book I've written on this topic," and "Hire me for consultations." In this scenario, she also has a thriving blog addressing the same topic as her book and talks. The opportunities are likely to flow instead of to trickle haltingly.

There are a few differences here. In scenario two, she has a book and a blog (both content) but also a generally better marketing plan. And she explicitly expresses a willingness to consult. All are

important, but the book and blog really drive it home. These say, “I am a full-time expert with bonafides on this subject, and here’s proof you can trust me.” Contrast this with scenario one, where the follow up seems to send the message, “I’m good at this because my employer told me to get good at it.”

Having publicly available content definitely establishes you as an expert and as a standalone presence in the community, current employment specifics notwithstanding.

Developer opportunist trait 4: they are literal opportunists

So far, I’ve covered the importance of carving out time to manage your career with marketing and content creation. All of that falls under a common heading. The people I’ve talked to attack their careers with deliberate action. But this is not to say that everything happens according to a master plan.

Another important component of the developer opportunist is literal opportunism. I say literal because I ask you for the moment to disconnect from the terminology of this book. Consider that these people all take opportunistic approaches to their career.

David Boike was approached about writing a book on NServiceBus and figured he’d take a shot at it. Matt Heusser looked at his time in a relatively high-risk full-time job and realized it had armed him for a foray into freelancing. Dave Rael made the best of a tough situation by building an audience via podcast and becoming more entrepreneurial.

Switching back to the book’s lingo, developer opportunists need to keep their eyes out for advantageous situations and pounce on them. I don’t mean that they need to act like sharks, constantly circling the water looking for blood. They just need to have an attitude of willingness to leave their comfort zone when something with a high upside presents itself.

In this scenario, the same instincts that would allow you to succeed in the dysfunctional corporate entity also allow you to succeed independently. This openness to improving your situation requires thinking of yourself as your own tiny entity and not part of any company. Rarely will your best opportunities come from the company at which you currently work, and when they do, it's not a hard decision. If your local VP comes along and says, "How would you like a promotion and a 25 percent pay increase?" agreeing is hardly a gut-check moment, unless you've had your eyes opened to the fact that this is an idealist-forging offer.

The people to whom I spoke and the developer opportunist in general constantly monitor their surroundings for chances to improve their lives and careers. *All* of their surroundings.

Developer opportunist trait 5: they manage the pipeline

A developer opportunist is actively reviewing his own interests at all times. If you're technical and not buried beneath layers of idealists in the corporate context, you're in that position because you've charted your own course. You're not passively waiting for recruiters to come along looking for JavaScript ninjas.

We can formalize this idea a bit. Developer opportunists manage what I'll call the work pipeline. For example, consider Eugen, who does both architectural consulting and content creation in the form of courses on the Spring framework. He's not looking for assignments. Instead, he's constantly monitoring his business interests and considering which opportunities make the most sense.

James and Matt, both principals of consultancies, have the same conceptual approach. They orient their marketing around content and use this to generate prospects interested in doing business with them. With inbound prospects, James and Matt spend time convincing them to buy consulting services, converting prospects to customers. And the whole time, they qualify or disqualify prospects

as they come in by carefully evaluating what would and wouldn't be a good engagement.

All of this falls under the heading of managing the pipeline. It's an absolutely essential activity for any business. It's also a big part of the reason that developer opportunists don't—and can't—spend all of their time programming. They need to spend some of their time acquiring work.

But this isn't just for the solo consultant or head of a consultancy. A developer opportunist that moves from freelance contract to freelance contract, or even one that jumps from full-time job to full-time job, is managing the pipeline. Identifying prospective employers and doing the interview dance requires a lot of effort. If you're doing it with any frequency, you're already spending time managing the pipeline.

I would also submit that people actively participating in the community are doing this as well. If you're spending time blogging or speaking, you're identifying audiences to appeal to, and you're accepting or disqualifying opportunities on the basis of what kind of fit they are and how beneficial they would be.

It's only the pragmatists and idealists of the world that don't do this, in fact. Pragmatists resign themselves to whatever work gets dumped in their laps, telling themselves that work is work is work. Idealists enthusiastically accept whatever gets dumped in their laps, reasoning that their superiors and the organization know best.

To join the ranks of developer opportunists, you need to start qualifying your work and managing your pipeline. If you're just starting to do this as a wage employee, the best way to get off the ground is to simply identify whether or not the work you're being given is good for you and your career. Once you've established a habit of critically qualifying your work, you can set about adding progressively better options to your pipeline.

Developer opportunist trait 6: they have options

The mention of options brings this developer ubermensch chapter to an appropriate close. Everything I've mentioned here and the stories of all of the developer opportunists I've interviewed unite under that common theme. Developer opportunists have options.

Programming in and of itself doesn't really generate any options for you. Let's say that you come to work and, day in and day out, you deliver high quality code on or ahead of schedule. Depending on the internal politics of your organization, this may or may not be favorably recognized and that recognition may or may not make it into the form of some kind of public praise. And even when all that happens, the read from most external parties is going to be "laborer succeeds at laboring." In other words, all of that hard work and excellent code establishes that you, as a programmer, can be relied upon to program. At best, this gives you the option to trade your current programming job for a different programming job. But with the market what it is, you have that option even without doing good work.

Developer opportunists recognize this reality subconsciously, if not consciously. They figure out that they need to spend somewhat less time programming and somewhat more time making sure they have options. Typically, this takes the form of marketing that I mentioned in the chapter. As John Sonmez points out, and as I can echo, marketing himself generated all sorts of job and consulting opportunities that didn't previously exist.

Now, you're probably thinking that I'm being contradictory. I just said programming doesn't generate options, except for the option to switch jobs, which you have whether you program well or not. And now I'm citing consulting and job offers as John's options. This would be contradictory if not for a subtle but real difference. The workaday programmer uses his programming resume to ask *the favor* of a job. Contrast this with John's situation, in which a company approaches John, asking for *the favor* of his services.

Let's put this another way. Workaday programmer has the option of job A or job B. John has the additional option of neither, because C, D, and E are always imminently on the horizon without much effort.

When options come to you with relatively little effort, it has a multiplicative effect on the flexibility with which you approach work. It means you've diversified your portfolio, as I discussed earlier, in Part 4. And if you've diversified, even when you seek out options, any particular one becomes less drastically important.

Matt, Dave, and James have consulting practices, which means that they have books of business that render no individual client of paramount importance. Losing a client is a bummer but survivable. Eugen and John have multiple lines of business, giving them the same sorts of options. David Boike, as a consultant with a single client, is a bit more invested with that one client. But he's made a name for himself, has tons of industry contacts, and already has a consultancy legally setup and ready to operate.

Developer opportunists do not get caught flatfooted the way a single prospect employee would. They continually cultivate and maintain options so that they can opportunistically choose from among them and create their best situation.

When you get down to brass tacks, all of the non-programming activities they take on are about option generation. They self-market, building names for themselves in the industry. They create content and grow their networks. They keep their eyes open for what opportunities may come along, in any form and from any angle. And they actively oil their work pipelines.

At a granular level, they may engage in these activities simply for the sake of enjoyment, and they may do some or all of these things subconsciously. But the end result is the same. The developer ubermensch needs the world less than the world needs her.

And the reason for this is critically important to keep in mind for the rest of the book and in contemplating your own fate. She's in such

demand not necessarily because she's the most skilled, smartest, or most technically knowledgeable. She's in such demand because she does an incredible job of identifying what others value and positioning herself uniquely to deliver that value. Superior technical prowess helps, but developer opportunism, predicated upon options and granting autonomy, comes from understanding of and fluency with business.

Chapter 43: The Developer Ubermensch is a Businessperson

If you made your living developing software following the release of the original iPhone, this will probably sound familiar.

For a period of time, every non-programmer on earth would approach someone they knew who could write code, saying, “I have an idea for an app!” They would then proceed to lay an idea on you that went something like this: “It’ll be like Facebook, but for parking spaces! And you can take pictures and stuff. And it’ll have GPS. So I figure, you build the app, and I’ll do all of the business stuff. Since it was my idea, we’ll partner up seventy-five/twenty-five. You in?!”

Seriously, anyone who has been in the programming game for a decade has listened to some variant of this pitch. Perhaps someone even hinted that you should sign some kind of non-disclosure agreement before they lay this top secret idea on you. It annoyed me then, and it annoys me now, but for different reasons.

Back then, it annoyed me as a matter of gall. I’d have to restrain myself from saying, “Oh, okay, let me see if I have this right. You just spent five minutes dreaming up a half-baked scheme. I’ll do all of the work of making the thing that will be the centerpiece of our business, while you ‘supervise.’ And you’ll have a majority ownership stake because it’s *your* half-baked scheme. Remind me why I need you for any of this?”

I was annoyed because if I were going to try my hand at app-based entrepreneurship, I’d do it myself with my own ideas, thank you very much. I was annoyed because I knew I could do this alone and I knew they couldn’t, and yet the best arrangement I’d ever get pitched was a fifty/fifty one.

These days, I'm annoyed because we as software developers have helped create a business reality where these half-baked-idea pitchers are right. They're not right about their ideas being good ones; they're right about how much the relative contributions are worth. They're right that building the thing is worth a quarter of the equity, on the generous side. And they're right to offer us that because of our longstanding, self-defeating tendency to turn our noses up at the worth of other aspects of the business.

Now, years back, I wasn't so obtuse as to think that nothing besides building the app mattered. For me, the galling thing was the aforementioned "Why do I need you?" I could handle all the other aspects of launching an app myself. I'd make pitches to investors, keep the books, and get the word out. Surely that's worth about 10 percent equity to the 90 percent I should have for building the things.

At least, this is what I thought back in 2009, when I worked full time as a salaried software engineer. Eight years later, more than five of which I've spent running my own business, I realize that my take was only slightly less naive than thinking nothing besides building the software even mattered.

Sure, I *could* spend 90 percent of the time building the thing and handle the rest of the business affairs during the other 10 percent of the time. I'd then have a recipe for a terribly run business, albeit one that I built all by myself.

The thing is, the idea-havers also had it wrong. Having some crackpot idea for an app isn't worth 75 percent of the business. It's not worth *any* of the business because ideas for apps are a dime a dozen. The thing that's worth 75 percent of the business is the 75 percent of the business that goes above and beyond banging out some code. And this is true even for a pure software endeavor like an app.

To convince you of the merits of my argument here, let's dive into the anatomy of a business a bit. To do that, consider for a moment the

most code-oriented business imaginable: a freelance software developer.

Assuming that you form Workaday Dev Inc., wherein you offer staff augmentation contracting services to anyone with a spec and a need, you still have *a business*. “I just want to code, man” is fundamentally incompatible with developer opportunism. The coding part is just 20 percent of a very simple business equation I’m going to offer you.

Because this is not an MBA course, I’m only going to toss out a quick summary of the nuts and bolts of the business. This summary is based on my own experience, though I have found [sources like this one](#) that roughly map to what I’m saying. You can generally divide a business enterprise into the following components.

1. Product/service creation (“We make pizza.”)
2. Operations/delivery (“Customers can come in and pick the pizzas up or we’ll deliver the pizzas to them.”)
3. Accounting (“We need to sell 200 pizzas per night to cover cost, payroll, and taxes.”)
4. Sales (“We sell pizzas for \$15.99 each but offer a buy one/get one half off deal on Fridays.”)
5. Marketing (“We get the word out via website, fliers in local mailboxes, and sponsoring a little league team.”)

For some reason, I find lessons about business seem to go a lot easier when talking about pizza. Hopefully, you get the general idea. The actual product, pizza, is accomplished with the first part of the business, concerning the product or service. The rest of the business concerns itself with the logistics of delivery, keeping track of the money that you make, getting people to buy from you, and making people aware of you.

Let’s take a look at what this looks like applied to our world, in the form of a single freelance software developer.

1. Product/service creation (“I write code.”)

2. Operations/delivery (“I’ll deliver the code to you by January twenty-fifth and check in with you every two weeks in the interim.”)
3. Accounting (“When I’ve delivered the code, I want you to pay me, and I’ll keep track of whether you have or not.”)
4. Sales (“I charge \$100 per hour, but I’ll offer a weekly rate of \$3,500 and a monthly rate of \$12,000.”)
5. Marketing (“Check out my blog and website for tips on writing good code.”)

If you steadfastly refuse to have any interest in any of this outside of coding, you’re saying, “I only care about the first item.” To illustrate how problematic this is, consider a hypothetical scenario.

Let’s say that you’re at home writing code one night, and some future version of you steps through a hole in spacetime. Future you pairs with you, and the two of you together write an awesome product that will analyze a codebase and find 100 percent of the bugs in it every time, without exception. Future you then steps back into the hole in spacetime, leaving you to do your best Biff imitation using this deus ex machina. What now?

Well, obviously, you have to think of a name. How about something awesome, like “Bug Whacker”? Wait, maybe that’s not so awesome. You spend a few hours trying to think of a name, and then those hours turn into days. After all, you don’t want to launch with a bad name. Unfortunately, your only metric for naming something is to decide if it sounds appealing to you. If only there were people out there with experience picking things like this in ways that drive results.

Whatever. You pick a name and go with it. After all, this product is so good—so perfect—that it really doesn’t matter. Literally everyone who writes software will want it. But wait. How do you even get it to them? And how do you collect money? You could compile it onto a thumb drive and mail it to them, you guess. Should you do that before or after you receive the check in the mail? Huh. If only there

was someone that knew how these types of operational things worked.

After a few days mulling that conundrum over, it occurs to you that you should probably have a website that you do this through. After all, you're a software developer. You'll just make a website that accepts credit card payments and lets them download the product from behind a paywall. Things like that might exist, but you don't know much about them. Better to code it by hand, right? Or is it? If only there were people out there with experience quickly and inexpensively setting up websites to accept payments.

So you spend a few months building a site by hand and using APIs for the credit card processing and the paywall. You had a few snags, like paralysis by analysis with hosting options and then not realizing you needed an SSL certificate to have a site that handles payments. Who could have known? Good news is that your site executes flawlessly, since you [dogfooded](#) Bug Whacker on it. Oh yeah. You went with the domain bugwhacker.com because no one advised you not to.

Finally, after months of spare time invested on top of your day job, you have the site up and the payment processor working. Time to sit back and bask in the profits. Just need to decide what the price should be. Does \$100 sound about right? That's a nice chunk of change in your pocket, but not so expensive it will discourage people from buying it. You wouldn't want to turn away customers.

But wait a minute. You need to license it in some way that isn't just "one per organization." After all, you want freelance software developers to be able to afford it, but when Google comes knocking, you figure Google should pay you more than \$100. Oh, gosh, and how do you stop a company from buying a single copy and just using it over and over again? If only someone specialized in knowing how your customers would use the software and who would pay what and how.

Eventually, you say “screw it” and just start selling. Several months in, having tens of thousands of customers paying you sounds great, even if you could have gotten more than \$100 from them. It’s not like you’ll be hurting. So you throw the switch on your site to send it live, and then you tell the world about it.

The only problem is that the world involves your forty Twitter followers, most of whom are bots, and people that you know on Facebook. You make the big announcement, and nothing happens. Not a single site visit. Okay, fine, you need a better platform for this. You try to tell people on Stack Overflow about it. That results in a ban. You approach some prominent bloggers, meetup organizers, and other people with audiences about promoting it, and no one believes you. And that makes sense because “Bug Whacker fixes all of your bugs, guaranteed!” sounds like a two-bit scam. If only there was someone that could have seen that coming and advised you on how to establish credibility with your target market.

I think you get the point implicitly. But let me say it explicitly. You could have the absolute perfect technical product, sent from the future, indistinguishable from magic, and, without the other components of a business, you will not make any money.

We come from a corporate history in which many software developers don’t understand this fundamental truth of business, or else they don’t much care. But in the coming age of the developer opportunist, that changes. Developer hegemony will be ushered in when we reach a critical mass of understanding and caring about this truth.

Understanding that writing code is only a piece of the puzzle does not mean that you need to like all of these other activities, nor does it mean that you need to divide your time equally among them. It simply means recognizing that they have value. James Grenning said, “I really hate the record keeping. One minute of it is a burden.” Yet he still does it, rather than, say, not collect payment or sign up for a pragmatist role within an organization. Developers in his position can put up with it, automate it (James’s choice), or delegate it. But

whatever they do, they need to recognize the worth of the activity and they need to recognize that it is not a second-class citizen to the business, even when the business is making software.

When I asked Dave Rael why he thought there was such a ceiling (both in pay and responsibility) for software developers in the corporate world, he had the following to say.

There are several reasons, [and] probably chief among them is that a significant portion of software developers are terrible marketers. [They're] not only terrible at knowing and articulating their worth, but [they're also] emotionally tied to an idea that marketing is beneath them.

I concur with his assessment, and I think this last sentence is of absolutely critical importance. As developers, we not only eschew things like marketing, bookkeeping, operations (and even quasi-software activities like QA and UX)—we exhibit the attitude that they are *beneath* us. We look at the corporate world and demand that it accommodate us with roles where we need not be burdened by thinking of the business. The corporate world is all too happy to indulge our demand, pay us a third of what we're worth, and let us exist in a bubble of illusory superiority.

I've talked a lot about marketing yourself, creating content, and giving yourself options. If you start to do that, you'll naturally develop an appreciation for the other parts of a business. After all, you'll be operating, collecting money, marketing yourself, and selling products/services. But whether or not you decide this path is for you, the path to collective developer opportunism starts with looking at the whole of business in a whole new light. Accept its necessity, embrace its existence, respect it enough to learn about it, and make yourself into a complete businessperson whose trade happens to be software.

As long as we fetishize unmarketable levels of knowledge of arcane technologies, we will continue to be managed. When we start

geeking out not about tech but about delivering value and participating in commerce, we will flip the script and start to manage businesses.

Chapter 44: Developers Becoming Efficiencyers to Take Control

Please set aside for the moment the fact that I invented a word in the chapter title. I'll get to that shortly. First, I want to tell a story about something that happened to me this morning.

My cell phone rang about mid-morning, and it presented me with the ultimate conundrum for an introverted entrepreneur/consultant: an unrecognized phone number from my area code. On the one hand, it could have been a prospective client or generally someone whose call it would make sense to take. On the other hand, I didn't recognize the number, which usually signals some kind of solicitor or scam. I decided to answer because I thought I'd seen the number before. Either someone really wanted to talk to me or I'd have to tell them to put me on their do not call list to get them to leave me alone.

I instantly regretted my decision to answer. A criminally cheerful voice said, "Is this Erik Dietrich?!" You'd think he'd just gotten his personal hero on the phone.

I sighed, recognizing the forced, chipper tone of someone who lives by the motto "Always be closing." "Yes," I said guardedly.

He told me that one of my friends had referred him to me, saying that I was someone that would potentially be interested in his services. He was a financial planner, you see, who specialized in helping young couples achieve their dreams—couples like my wife and me, as my luck would have it. "When is a good time for us to get coffee?" he asked after blithely glossing over this dubious introduction. (My friend never had, in fact, briefed me that this guy would call.)

Do you see what he did there? It's a classic sales technique wherein you don't give the ~~mark~~ prospect the option of saying no. This bit of

slicked-back-hair salesmanship is a close cousin of [the “loaded question” logical fallacy](#). “Have you stopped cheating on your taxes?” “Yes, I mean, no, I mean—hey!!” No matter how you answer the question, you implicitly concede a point made by the asker. In the case of our used-car salesman *cum* financial planner, answering his question politely leaves me no choice but to agree to meet him.

I sighed again. Briefly, I thought about setting a meeting at some Starbucks in the area and then never thinking about him again. But that seemed disproportionately cruel, so I broke script and told him that I wasn’t interested. After taking one more stab at me with a “When is a good time to follow up to see if you’ve changed your mind?” he’d exhausted his slimy script and hung up on me with no fanfare. Class act from start to finish. I should call him back, say I’ve reconsidered, and set up that phony meeting after all.

We software developers present as an unusually marketing/sales-averse group. We’re skeptical of crap like this, and the world reinforces that skepticism. On top of that, we also tend to be fairly clever, so we get good at sniffing these things out. “Oh, heavens yes, I could use some unsolicited financial planning from you, stranger,” isn’t something you’re likely to hear out of the same mouth that says, “You should use the builder pattern in that module.”

We connect such instances of disillusionment with the seedier side of commerce, and we use the scars they produce to inform our preferred roles within organizations. Sales is an unseemly vocation, so we stay away from it, even disdain it. We don’t stop to consider that not all sales happens equally. Imagine that I had a buddy sitting next to me when I got that sales call, and he’d listened to the exchange. When I got hung up on, say this buddy said, “Oh, dude, I wrote this app a while back that’ll make sure no d-bag like that ever calls you again. It’s only \$12.” I’d have sprained my wrist grabbing my wallet with too much force.

We don’t really think of those helpful types of interactions as sales. More importantly, we don’t tend to think about how we might apply our own world views to disciplines like sales—what sales would look

like if developers ruled the world. Instead, we regard these other professions the way most of us regard the vocation of plumbers specializing in sewage. It needs to exist, but I want to stay as far from it as possible and not think about it.

Developer hegemony will come from using our leverage to remake facets of business, like sales, according to *our* preferences. If you don't like sleazeball sales, weird accounting practices, and sloppy organizational operation, don't complain, and definitely don't ignore them. Take over and fix it. We're not trying to bamboozle suckers into generating commission for us in our roles as middlemen. We have too much leverage for that to be necessary. And we have this leverage by virtue of the fact that we're efficiencers, whether we realize it yet or not.

I now owe you a definition of this term “efficiencer.” In short, it's the name describing the service that we as software developers provide. “Software developer” is descriptive, but it suffers the fate of being entirely too procedural and focused on details. We perpetuate this problem by being, ourselves, far too focused on details. The value proposition that we offer the world, notwithstanding what we write on our resumes, is not “five years of Python, three years of JavaScript, etc., etc., etc.” The value proposition that we offer has deep roots in *efficiency*.

As software developers, we automate things. This is easy to see when you contemplate a team that saves money for an organization by writing line-of-business web services and the like. It's a team that's automating away the need for manual data entry. That automation enables the organization to do more work in the same amount of time without hiring additional people or buying additional resources. And output as a function of time is—you guessed it—efficiency.

Efficiency equals stuff divided by time. Thus, our true vocation takes two basic forms. We allow people to do more stuff in the same amount of time, or we allow people to do the same stuff in less time. In both cases, we increase the value of efficiency in that equation.

This is obviously true with things like automating data entry, but it extends to all facets of programming, including virtual reality or games. We allow people to “visit” the Grand Canyon without spending money on plane tickets and wasting time traveling. For games, consider fantasy football. My uncle played fantasy football in the nineties before the internet really took over. He was an extreme early adopter because playing fantasy football back then involved league participants manually collecting statistical information from newspapers, compiling it, and using it to score the league’s games. That’s a huge time investment. Now fantasy football takes place in pretty much every corporate office in the US because it’s more efficient to play it.

Time is perhaps the most important commodity of the digital age. If you grab any modern human and ask them to list their current biggest problems, lack of time is sure to rank highly. Books upon books upon books have been written about how individuals can wring a few more spare minutes out of each hour or day by being more productive. When businesses look to do the same, the stakes become immeasurably higher. And that’s what we, as efficiencers, sell. We sell efficiency to business, thus granting us incredible leverage as part of the workforce. We sell, to individuals and businesses alike, the ability to increase profits, to expand, and to scale.

If you were to ask a lawyer about his core value proposition, he’d say that he provides expertise in the law: “I help you claim and defend your legal rights.” If you were to ask a doctor about her core value proposition, she’d say that she provides expertise in your health: “I help you live longer and have better quality of life.” But if you ask a programmer about his core value proposition, he will probably say something about knowing ASP and helping you build websites. “I help your company build nice, responsive design websites that function on a whole variety of blah, blah, blah....”

Wrong. Our value proposition is that we provide expertise in *efficiency*. “I help you have more time and money.” Or, at the risk of

sounding a tad overly dramatic, “I help raise the standard of living.” Sounds pompous, but that’s what we do—eliminate jobs of drudgery and create ones of knowledge work. I’d say that time and standard of living as by-products of our work put us on par with those offering services around rights and health.

But we don’t really have the same sort of place in the world as doctors and lawyers, do we? At least, not yet. Why is that?

The reasoning for this is no doubt complex. But I think we can start by reaching back into Part 3 and considering the history of the corporation. Recall that the Industrial Revolution gave us the rise of efficiency as a serious concern. Until that time, vendors were basically artisans that would compete with individual quality. Then the Industrial Revolution occurred, allowing competitors to compete via scale and efficiency. By the time the first computers came into existence, they provided a natural path toward the ever-mounting demand for efficiency. The vocation of “programmer” thus emerged from the belly of the corporation, as a result of the quest for optimization that was already well underway. Notwithstanding academic research projects, it was the corporation that gave birth to the modern programmer, and it did so at the grunt level.

Contrast this with doctors and lawyers. Both vocations predate the Industrial Revolution and have a lengthy history of being consumed *en masse* by individual citizens. These professions evolved over the centuries, outside of the crucibles of industry and Taylorism. They have rich histories of individuals and partnerships hanging out shingles and dealing directly with customers.

Consider a law firm. Do the founding partners go out and hire a Lawyer Manager to order them around, and then do they hire a VP of Lawyering to order the manager around? Do they then hire a CEO to rule over everyone and a CFO to handle the finances and a COO to schedule court dates and such? Of course not. There’s no historical precedent for all that fluff. Rather, they handle facets of the business themselves. What they can’t or don’t want to handle, they delegate to subordinates that they hire. The partners don’t specialize

in finance, sales, marketing, or operations. That would be silly. But they understand enough about it to act as the boss.

More established knowledge work professions first encountered the corporation as a customer and not a master. Not so for us, as programmers. We came into this world as miscellaneous corporate employees that happened to figure out Microsoft Access pretty quickly. Before we really knew what was happening, the Senior Director of Something was telling the Manager of Whatever to tell us to do some stuff to the Access database that would make them not have to hire an intern to manually enter data all summer. We've historically automated and saved time at the behest of operations strategists that understood how to translate our skills into cost savings and revenue generation.

The time for that is over.

As an industry, we're at that crossroads that happen in sci-fi movies, where the robot achieves self-awareness. The corporate world has become so utterly, irrevocably dependent on ever-increasing efficiency that it absolutely cannot live without what we provide. If we now alter the terms under which we furnish that efficiency, no one is in any position to argue. That is our link to the older knowledge work professions. Is someone being sued in a position to demand that an attorney fill out a TPS report? Is a sick person in a position to order his doctor to participate in a daily status call? Is a company whose service delivery costs more than it makes in a position to demand these things of us?

The next phase of our profession's evolution involves us leaving large organizations, keeping our own counsel, and hanging up our own shingles. On those shingles, we will say, "We specialize in making your business more efficient."

In concrete terms, this means a subtle shift in our focus. Without this shift, you have the same sort of staff augmentation firms engaged in the same sorts of low-leverage commerce that define the app dev consulting space today. We stop accepting specs. We stop receiving

wireframes. We stop dealing with prospective clients that come to us and say, “We’ve had our business analysts figure out everything we need to do, and now we just need some geek to actually write the code.”

“No, no, no,” you say. “That’s not how we do business. You don’t come to us when you’ve planned the details of your site. You don’t even come to us when you think you need a site. Rather, you come to us the moment that someone says, ‘It’d be a lot easier if our customers could order stuff online.’ You’re talking there about making your operation more efficient, and efficiency is our specialty. We’ll be the judge of whether you need a site or not and, if you do, how it should work and what its ‘spec’ needs to be.”

But you only get to have conversations like that when you can understand and wield your leverage. And you only get yourself in a position to understand and wield leverage by becoming serious students of business—all components of the business. As efficiencers, we recognize that code is a means to our end only and that sometimes our clients are better served by not commissioning any code at all. Geeks will rarely tell you *not* to pay them to write code, even if that’s the right call. Efficiencers will always make the right call.

And on top of that, efficiencers will be savvy enough to start and run firms the way lawyers and doctors do. They’ll partner up and run the business themselves, delegating the parts that don’t make sense for them to handle. They still don’t need to worry directly about, say, sales, if they choose not to, but they call the shots. That means that efficiencers can slap and fire people that act like my would-be financial planner. They can chart a course of which they approve, instead. And best of all, efficiencer firms will have an utter monopoly on making their own marketing, finance, sales, and operations more efficient. Not only can they dictate and replace. They can automate as well.

Chapter 45: Turning App Dev Firms into Efficiencer Firms

Up to this point, I've been relatively abstract about the future and what we do to get there. You might just think of what this part has covered as a mental change in our own philosophies as individuals. The developer opportunists I've interviewed have a different philosophical outlook than the workaday developer. Now armed with that outlook, you may change the way you do things toward your own benefit. But how does that change our industry? How does that produce developer hegemony?

In this chapter, I'll talk specifics. What you're going to read next is wishful thinking, encouragement, and prediction all rolled into one. I guess you can accuse me of being an optimist. I think we're heading in the direction I lay out. My goal in writing this book is to help us get there faster.

One of the questions that I asked of my panel of developer opportunists was, "Why do you think there's such a ceiling, both in pay and organizational status, for software developers in the corporate world?" Matt Heusser had an interesting answer.

Programming is viewed as entry-level translation work. It is something to get promoted out of. Most companies want a journeyman programmer with three to five years of relevant experience so they won't have to pay to train them, then lose them. That means there are few good training programs. Plus, the company doesn't want to pay for your twenty years of experience: five of COBOL, five of C, five of early web development, and five of recent relevant responsive design.

That looks right on paper, but the result is that we have a pile of new people who are trained poorly, while kicking good people out around forty.

James Grenning, likewise, had an interesting answer.

The corporate world does not know what programming is. They view it as labor (more hours equals more output, cheaper hour rate means less cost), not knowledge work (problem solving takes time and some people are better at it than others).

I use these two quotes because they do an excellent job illustrating the current, non-efficiency state of affairs. As James points out, the corporate world regards the nitty-gritty of programming as “labor.” And as Matt points out, that labor fits tightly as a cog into a larger machine: “translating” a boss’s strategic vision into the mundane geekery of source code.

Programming in the corporate context involves a neat division of labor. Strategic-minded folks that understand business (usually Tayloresque managers) figure out the “what,” and the grunt developers concern themselves with “how,” to the extent that they aren’t also micromanaged on the “how.” If you work hard and keep your nose to the grindstone, you will eventually be promoted from the latter to the former. But never the twain shall meet.

It’s interesting how the agile software movement has encouraged a blending of all things technical. In the enterprise, I’ve heard the term “T-shaped people” tossed around pretty freely. What this means is that you want people with a wide base of skills but who go deep in at least one specialty. If you have people like this, you can put together excellent Scrum teams. Everyone knows a bit about writing code, testing, operations, and so on, but each individual has deep expertise in one of those things.

As a resource allocation strategy, this is savvy. If all of the development work is done and it’s testing time, everyone can pitch in

with the testing. This is a lot more efficient for handling variable specialty workloads. If you sometimes have nothing but testing to do, you don't want the developers all sitting around twiddling their thumbs. If everyone can help with everything, that's never a concern.

This has given rise to all manner of ceremonies and strategies for cross-discipline collaboration. In fact, the recent love affair with the concept of DevOps illustrates this perfectly. Why silo between writing software and caring for it at runtime? Can't we all participate? And now, in fact, we do.

It turns out that treating knowledge work pursuits like manufacturing operations doesn't work particularly well. But it was the model inherited from the Industrial Revolution, and it's been hard to shake. The thinking goes like this: if you break a complex operation down to its individual components and then have people specialize in those components, batching the work and letting people get good at tiny slices of it leads to greater efficiency. This works well for stations on an assembly line but not so much for writing software.

Breaking down the silos and walls of this type of specialization has counterintuitively led to large efficiency gains in our line of work. And yet, one sacred silo remains: business strategy versus software development.

For their part, proponents of agile software development get credit for the strides that *have* been made along these lines. Scrum calls for a product owner—someone empowered to make any business decision related to the software, to be a first-class member of the Scrum team. People facilitating so-called agile transformations at enterprises stress the importance of collaboration between IT and “the business.” In fact, one of the twelve principles outlined on the site of the [Manifesto for Agile Software](#) holds that “business people and developers must work together daily throughout the project.”

But no one is saying that business people and developers should be the same people. Except me. I'm saying it now. Efficiencyers are to

business and software development what DevOps is to development and operations.

This is the first guiding principle of efficiencer firms. The firm's principals are programmers. They are not former programmers, kinda-sorta programmers, nor any of the other designations that principals of such firms might currently hold. They *still* program. But they also run the firm, and they also speak to clients about their business operations and how the efficiencer firm can make their business more, well, efficient.

A traditional app dev consultancy practicing Scrum would operate by assembling a team of software developers and asking the client to supply a product owner. This serves the important purpose of getting the team a single empowered decision-maker to let them go as fast as possible. It's a smart concept, and it improves by light years the contract/spec-driven waterfall development, which tends to result in more lawsuits than happy customers. But it nonetheless separates business from software. It's the same old "We write the code, you supply the vision."

No more of that. In Scrum parlance, you could think of an efficiencer firm as one that convinces the client to delegate product ownership to a member of the firm. Your client would say, "Hey, Efficiencers Inc., we have the goal of automating customer orders, and you've told us that you'll take on the project and deliver a system that lets us process ten times as many orders per hour as we currently do. The gig is yours—go!" With this mandate, "product owner" is clearly something the efficiencer firm arranges to be internal. After all, they're the ones with the vision for the 10X speedup, so why wouldn't they own any decisions relating to any software that needs to be written to achieve the goal?

To put this concretely, if you started an efficiencer firm, you would not take specs, and you would not take wireframes, and you would not take direction on software. Software (and by extension, automation/efficiency) is *your* area of expertise, not your client's. In the future, this is how more and more software will get written.

There's an obvious hurdle, as you can no doubt imagine, and it's the fact that the world doesn't currently work this way. If you're a freelance developer or a small app dev firm, you're imagining the next time a prospect calls to ask for a website and picturing sitting back, crossing your hands behind your head, and condescendingly saying, "Let's back up and ask if you really even *need* a website." Fair enough; that wouldn't go well. I owe you some explanation as to how we get from our current state to one where they call us *before* they need that site.

To put it succinctly, we make that move by putting our sales hats on and changing our core offering. We'll get there by slowly ceasing to sell app dev and starting to sell efficiency. But since cold calling people and offering "efficiency" sounds a little nutty, let's look at some interim steps so you can see what I mean.

I've done some comparisons of our industry to that of lawyers and doctors. Hopefully, the recasting of our work as dealing in time and efficiency has made that seem somewhat more plausible because I'm going to draw on that again to help illustrate where we need to go.

If you find yourself in the unfortunate position of getting divorced, you start asking people you know for referrals to *divorce* attorneys. If you have problems with your foot, you call up someone with the title "podiatrist," meaning "foot doctor." Those autonomous knowledge work professions organize themselves according to the problems their prospective customers have. They advertise in terms their customers understand.

Software developers and app dev firms fail spectacularly in this regard. We organize ourselves and sell labor on the basis of what tools we use while our customers tell us what to do. Think about how a moderately sized app dev consultancy might look for employees and advertise to clients. "We're an agile Ruby shop that works with small and medium sized companies." This is like an OB GYN practice saying, "We're a practice that does epidurals and Cesarean sections for young to middle aged women" instead of "We help you

give birth.” Current app dev shops also wash their hands of any real expertise in business. Instead of telling our clients “the baby hasn’t turned and you’re going to need a C-section,” we say, “All right, just give us a spec for how the birth needs to go, and we can do anything you want!”

Let’s get back to initiating conversations about whether a website is even necessary or not. How do we get there? Well, the first step is to stop organizing ourselves into groups of people that use common tools and then selling “We’ll do anything you tell us” to customers. When you do that, you’re selling neither a good nor a commodity nor a service. You’re selling humans in the form of pseudo employees. That’s called staff augmentation. And make no mistake: even if you supply an entire team that doesn’t physically sit at the client’s site, you might still just be doing staff augmentation. Staff aug is not a question of how many people you supply or where they work, and you don’t get out of staff aug mode by simply offering advice about what you’re doing and calling that “consulting.” Staff aug is a matter of whose vision you execute. If it’s anyone’s other than yours, you’re augmenting their staff.

When we advertise as a Ruby shop that can build anything you like using Ruby, we’re making two efficiencer mistakes. First, we’re involving the customer in something it shouldn’t know or care about: our tool of choice. Second, we’re leaving ourselves out of a matter we should know and care about: what we’re building and why. Understanding this truth is the first step toward forming efficiencer firms. It’s also a step toward getting clients that don’t just ask who you think you are when you have a frank conversation about whether they even need the software they want.

The first thing you need to do in order to start having these conversations is to put on your sales hat and articulate what you offer. It can’t be your labor. It *has* to be your expertise in making something more efficient. This will most likely take the form of a productized service. A productized service is a sort of pre-baked consulting solution to a tangible problem. For instance, consider the

example problem to which I have referred a few times: an organization thinking a website will allow them to fill many times more orders than they currently can. Instead of making your selling point “We build websites in Ruby for small to medium sized companies,” you switch to “We help organizations without a public digital presence dramatically increase the volume of orders they can handle.”

Notice that we’ve flipped the script and avoided the two anti-efficiency mistakes. We’re no longer involving the customer in something it doesn’t care about (what programming language we use) and we’re no longer left out of the client’s discussion of what they want and why since, by the very nature of what we offer, we’re automatically part of that conversation. This can work even as you transition your business model and develop the offering. If you interject with a question about *why* the client wants you to do what they’re asking, you can follow it by saying, “Please excuse my presumptuousness, but the strategy behind these types of projects is actually a first-class offering of ours.” It’s a bummer to still have to say this in 2017, but adding that last part changes your rank from “coder” to “consultant,” and that makes them far more likely to listen instead of saying, “Shut up and write some code, geek” (even if they say it in more polite terms).

If you look at the panel of people to whom I’ve talked, you’ll find examples of productized services in the mix. James Grenning tells prospective clients, “I’ll teach your embedded software team how to test drive.” Sally Lehman has created a niche for herself that would serve well in a consulting capacity: “I can help you with email delivery.”

But beyond that, look at the folks that offer products rather than productized services. Eugen and Thorben specialize in Spring and Hibernate, respectively, and offer courses. John Sonmez offers info products to teach software developers soft skills. They could all easily compliment their product offerings if they so chose by offering productized services around them. (It bears mentioning that the

efficiency mandate not to burden customers with tools of the trade does not apply to courses on Spring and Hibernate since, in those cases, software developers are the customers.)

If your firm adds productized services to its portfolio, it will, almost by definition, add an alternative billing model to its portfolio as well. And alternate billing models are another important step in the efficiency direction.

Let's consider how app dev consultancies bill their clients today. Generally, they have two standard options: fixed bid or hourly/time-and-materials. Fixed bid offerings mean that the customer comes to you with a spec or a request for proposal (RFP). You size it up, estimate that it will cost you about \$300K to develop, and then say that you'll do it for \$500K. If the customer agrees, you now absorb all of the risk with relatively limited reward. After all, you'll get the \$500K from them whether it costs you \$300K or three million. For this reason, most firms avoid fixed bid arrangements for anything complex, and they foist all the risk onto their client.

That brings us to time and materials. In time and materials, you, as an app dev firm, say, "Gosh, I dunno—that's a complex project. We'll just start working and it'll cost \$100 per hour. We think it'll take 5,000 hours, but that's just a guess." The customer now agrees to a rate for the work rather than a price for the deliverable.

Guess what. That makes the customer *really* interested in what you're doing during those hours. It's the only measure they have for managing risk. Guess what else. Heavy interest in what you're doing each hour makes you look an awful lot like one of their employees and lands you right back in staff augmentation mode. As I explained earlier in the book, salaried employment is a zero-sum game. So is hourly billing.

But if you're adding productized services to your offering portfolio, you enter a different mode of operation. You're doing something repeatable enough that you don't need to completely punt on effort/cost estimation. In other words, you'd never agree to build

someone a massive custom website as a fixed bid project because no one has ever done that exact project before and neither party has the faintest clue what it will cost. Not so with a productized service that you deliver over and over again. You can quote a price on that.

Even better than quoting a price based on your cost is quoting a price based on the client's realized value. For instance, let's go back to the example of automating order processing. If you've done ten of these automation projects before, you might know that it takes you about two weeks a pop. If you're accustomed to earning about \$5,000 per week, you could quote a fixed rate of \$10,000. But you can also try reasoning about what the project is worth to the client and charging on that basis. For instance, if you know that processing ten times the orders they've historically been able to handle would mean that they'd go from \$100K in monthly revenue to over a million, I think it's fair to say that the automation is worth more than \$10K to them. Heck, it's worth more than \$10K to them the first day after you deploy. You could ask for \$300K for the project, and they wouldn't bat an eye. Neither party cares about your cost during pricing—only what the thing is worth.

This is called value-based billing, and while it's a nice alternative to other billing models in its own right, I don't mention it here to say that it's a prerequisite for transitioning toward being an efficiencer firm. Rather, I mention it because it gives you yet another pretext for having the strategic "why" conversation for the project. You can't attempt to ascertain the value of an initiative without having a strategic discussion. And again, if you explain that the strategic conversation is part of your discovery and quote process, clients will be a lot less likely to take it out of turn.

You can even continue to offer custom app dev as an upsell to your normal offering. That's less strategic in and of itself, but the expertise required to furnish a productized service offering will ensure the discussions remain strategic. The most important thing is to position yourself as a *business* expert that can bring automation to bear. Then you can build a suite of offerings around that common theme.

Hopefully the move toward expert productized services addresses some concerns about asking clients the “why” question. If not, you still have one more card to play, assuming that you have a good financial outlook or an appetite for risk. It’s the negative sell.

Going back to the order-taking website example, here’s what that might look like.

I can see that you’ve gone to a lot of trouble to create an RFP and all of those specs, so I can definitely understand that you want to proceed along those lines. Unfortunately, we don’t advise that course of action based on our experience with these sorts of initiatives. If you want to do that, I’d suggest going with another vendor, but keep us in mind down the road if you do. What usually happens to clients in situations like yours is that they issue an RFP, get about five bids, pick the second-least expensive one, spend six months arguing about fonts and colors, and wind up paying 50 percent more than the firm quoted for something you’re not thrilled about. If you sense that happening, let’s talk.

You can be politely firm about your position and demonstrative of a lot of experience all rolled into one conversation. It’s also powerful when you’re able to read or forecast your prospect’s pain points and when you refuse to be the source of one of them. Efficiency firms, niching around a specific area of expertise, will be able to do all of this.

Obviously, for efficiency firms to become the norm and replace the “We’ll build anything as long as it’s Java” firms, we need to change from software pros to efficiency firms. That means a lot of change in the way we do things. But I do foresee another development greasing the skids as well.

The need for software grows stronger as organizations seek more and more efficiency and time savings. And the gig economy continues to gain steam. Whether we become efficiency firms or not,

there is money to be made in catering to these freelancers and gig-based firms. There will be more money if they're efficiencer firms since they will be savvier about growth, hiring, and delegation. But either way, a market continues to emerge.

I foresee companies springing up to offer all sorts of services to tiny software development shops. For instance, accounting firms and marketing firms could easily niche into helping only small custom app dev shops. Legal is another big one as well. A company could propose that, for \$1,000, they will incorporate you, start you off with a set of boilerplate contracts for clients, and give you a road map for keeping yourself out of any legal jams.

The sky is the limit with these sorts of things, so it's not like I'm giving away billion-dollar trade secrets. More and more, software developers are going the freelance route. Even more than that are flirting with the idea. Removing barriers to entry will be a growth industry since there's a huge untapped customer base.

Here's why this development is so important. As change happens, it will mean a cottage industry of people flipping to a mode where software developers are the boss. They won't demand to speak to a project manager or assume that business is a foreign language to you. Money talks.

If you want to see some early signs of this transition—of the rise of efficiencer firms and developer entrepreneurs—look no further than people selling things to developers. Scratching our own itches is the beginning of developer hegemony because it's a low friction way to cut out needless interposition. Look at John, James, Eugen, and Thorben. All of them sell to software development organizations. All of them delegate business operations to others. All of them are the boss.

Developer hegemony will start with small firms and developers catering to one another. It will gain steam as more leave traditional work roles and solicit help from the emerging industry of catering to these firms. It will become dominant as we offer more niche

productized servitiices and as we scratch more business itches.
And it will culminate with efficiencer firms as the new normal for
telling businesses where, when, and how software should be written.

Chapter 46: Anatomy of the Efficiency Firm

If we take inventory of what we've got so far, it's the distinction between efficiency work and app dev work. Developer opportunists care about other parts of the business, and they interact with firms at a more strategic level. Hopefully this gives you an idea of the form and shape of developer hegemony. Developer opportunists that solve business problems and leverage automation to do so will own the future, and they have a competitive advantage in the present.

With that in the books, let's spend this chapter looking at efficiency firms from a more concrete, nuts-and-bolts perspective. To do that, we're going to need to toss out some sacred fixtures of the corporate world. I assume that, if you've read this far, you're probably open to considering that.

I'll get the hardest one to swallow out of the way first. Forget about the notion that organizations need to scale, in terms of salaried employees. In fact, forget the notion that doing so is even desirable.

Setting aside high-minded mission statements and pitches to investors, the primary motivation to scale an enterprise is to allow an owner to get paid for someone else's work. Now, I don't find anything intrinsically onerous in this. I've done it myself. But it's important to be clear about the goal, whatever other motivations may exist.

Let's say that you strike out on your own as an application developer, and you initially charge \$75 per hour on the freelance market. At first, you're scraping by with only a few clients and a few hours worked per week. But over the course of time, you do well, secure more business, and grow. You raise your rates steadily over the course of a few years and get to the point of being about 80 percent billable at \$150 per hour. Kudos to you, as you've elevated your income from, say, \$40,000 per year to nearly \$250,000 per year.

That's a heady feeling, and if you're used to doubling your income every couple of years, you might want to keep doing that. No judgment here. Maybe you tithe, pay for a relative's medical bills, and then donate the rest to puppy shelters, for all I know. Whatever the motivation, you're bumping up against something of a cap. You're not going to be able to raise your rates too much for stable app dev. You'd need to do something more specialized, and that would likely mean reducing your billable hours and possibly even backsliding. And say you eventually got to \$250 an hour for IT management consulting. You'd just wind up hitting another cap years later. The cap exists, and it's real, so let's not worry overmuch about where exactly it falls. Suffice it to say, you want more.

As luck would have it, there's an easy, tried-and-true way to do this. You'll hire a second developer, and you'll even offer him a generous salary of \$100,000 per year, which totals out to \$50 an hour. You bill him out at \$150, just like you, netting you a cool \$200,000 extra per year. Nice!

Except, er, wait. As I mentioned in Part 2, you have to spend about \$25 an hour of that on benefits for this developer. And since you're not exactly GigaTech with paid masseuses and chefs, you need to make the benefits really good in order to compete. Fine. A rate of \$75 per hour still nets you a not-too-shabby \$150K per year. But wait a second. Darn it. You have to spend an entire week doing nothing but onboarding this person. And then you have weekly one-on-ones and status calls and the like. Also, his work is occasionally not up to snuff, making you spend non-billable hours redoing it and teaching him what went wrong. Oh, and now you have to do payroll, benefits administration, and a whole host of other stuff you never had to do before. Wow. You're still making \$150K per year for that employee, but your own billable percent and income has taken a sharp downturn. It turns out that hiring another person isn't even as profitable as that time you started billing \$25 more per hour.

Okay, but wait a second. If you're going to do all that employer/manager/benefits stuff, it's not that much harder for five

people than it is for one person. So you take *another* hit to your billable hours and go out in search of bigger clients. You land one and hire a team of five developers. Now, you have to spend more time managing them, but earning that extra \$50K per year on each of their billable work plus, say, \$100K on your own work has you at a comfortable increase.

Wait a minute. Now that there's a whole team of them, they expect an office to come to. Plus you have to adjudicate conflicts, do more onboarding, and buy them all stock machines because having them use their own personal computers no longer cuts it. And you're suddenly taking on clients you don't much care for and would have refused before, but you have five people's mortgages to worry about and not just yours. *Sigh*.

You *can* grow an empire by earning margins on the labor of progressively more people. But it's an ongoing battle, and it's never going to be as pure, simple, and satisfying as driving up profitability of your own one-person operation. When you grow to expand your revenue as an owner, you'll always be chasing that dragon.

As efficiencers, I propose that we avoid viewing margins on other people's work as the only (or even a desirable) way to scale. With that central assumption cast aside, we can revisit others.

For instance, without the margin-scale imperative, we can much more easily toss out the job interview. An efficiencer firm need not participate in this farce. To understand what I'm driving at, ask yourself why organizations rely on hiring complete strangers that nobody there knows. Oh, sure, most companies would prefer to interview and hire a referral or a colleague of an employee, but absent those options, they just hire random people after pretending that a two-hour conversation will help them know whether or not that person will work out. Why do they do this? For the same reason that, in my hypothetical scale example, you would do it after landing a contract with a big firm that you need five new people to service. You tie your own hands with the scale imperative.

If we allow ourselves as efficiencers to grow comfortable with the constraint of not engaging in marginal scale, we are freed of all sorts of corporate pap that frustrates. If we dispense with the stranger interview, we eliminate the need to work through the corporate recruiter. As we interact with our coworkers, we don't have to play politics over pay bands and mutual status while also playing zero sum negotiating games with a boss. In the US, without mandatory scale, we don't need to contend with as many labor laws, obviating concerns like human resources and convoluted legal-driven bureaucracy. On the ownership front, without the scale imperative, we also don't need nearly as much venture capital to get started, leaving us less beholden to outside influencers.

I won't list everything here. But if you need a reading break for a moment, I invite you to let your mind wander over all that becomes possible if we consider that throwing more people at a corporation isn't the only way to get more profit out of it. Doesn't that make sense? After all, we are *efficiencers*. Scale is the opposite of efficiency. It's brute force.

Now that I've primed the pump a bit with the questioning of basic corporate assumptions and practice, let me introduce a set of principles by which I propose that efficiencer firms operate. I'm a big believer in the idea of introducing creative constraints, and that's what I aim for here. If you set forth core principles and then let people innovate within those principles, their cleverness can produce amazing results. The key is getting folks to believe that just because we've been doing something for a while doesn't make that something inevitable.

Here are my principles:

- Efficiencer firms are bootstrapped and self-sufficient, meaning they're profitable from the outset.
- The members of efficiencer firms are partners. All members execute on the organization's value proposition and have skin in the game. Instead of employing pragmatists, they delegate to vendors.

- They don't rely on absurd practices like job interviews to scale up their delivery capabilities. That happens via trial, recommendations, and networking.
- Everyone's contributions to the organization's profits can easily be measured. They only grow as long as that remains true.
- In short, the firm is comprised of opportunists. Anyone who would normally be a pragmatist is instead a vendor. This in turn eliminates the need for an overenthusiastic buffer of managers and senior people.

You might wonder why I propose that efficiencer firms should operate this way. And indeed, this is something that I'm *proposing* as a sustainable, dignified model for operating in the future. You could easily specialize as an efficiencer consultant and hire an army of random people to support you, including enthusiastic idealist acolytes that hang on every word of yours. I don't like that concept, and I won't recommend it. But I also won't ask you not to do it. That said, I'll resume advocating for my vision.

First and foremost, this lightweight, nimble model seems entirely twenty-first-century appropriate for people who won their means of production and can work anywhere with a Wi-Fi signal and their laptop. You can engage in what Tim Ferriss calls "lifestyle design" in *The 4-Hour Workweek*, wherein you decide what you want out of life and then build a business that supports those goals. In other words, if you leave behind the assumption that "work" means commuting to some physical location every Monday through Friday from 9:00 AM to 5:00 PM, you have a lot more options for happiness. Maybe you travel a lot. Maybe you live somewhere remote and perfect for a quiet personality, or maybe you stay in a foreign city that energizes and delights you. Maybe it's just a matter of skipping a commute and seeing your family more or working hours that are more compatible with those around you.

Whatever you might have in mind, working as a partner in a small firm makes this much more likely. If you grow larger, start employing people, and looking more like a traditional company, you're the boss.

The boss has to be present to supervise the grunts. Before you know it, you'll be only nominally less trapped than you are in a typical nine-to-five gig.

But that's really just the tip of the iceberg, and it may not even be for everyone. Some people like structure and going to a specific place, and more power to you if you're not looking to do what I've been doing and migrate to warmer climes for the winter. A key property of all of this taken together is that you avoid being subject to stuff that makes you want to pound your head against your desk until you see stars.

If you're a partner in an efficiencer firm, you have, by definition, partnered with people you believe are compatible. That alone is a step up from getting hired at Soul Crushers Inc. and stuck on a team run by some expert beginner and a bunch of checked-out people that roll their eyes when you tell them you think they should start using source control. You have the ability to seek out and collaborate with people that have similar taste in circumstances and compatible beliefs.

The principles that create drag on growing the firm pay off as well. When I consult with companies, I frequently explain that trying to automate a process as you figure it out is a mistake. Get the process right first, then simplify it to core principles. Only *then* do you automate. Same thing with expanding your company. If you have to ask strangers why manhole covers are round in order to grow, you're failing hard at this. "Well, Ms. Smith, we need to add to our headcount to get this software in front of our biggest client by October, and we think you can help us with that... if you don't mind explaining why manhole covers are round." It's a wonder that commerce is even possible in our society.

What does runaway growth in pursuit of headcount that blows [Dunbar's number](#) out of the water get you? It ensures the emergence of bureaucracy and the friction that comes with strangers trying to collaborate at scale. Those things in turn create the Dilbertesque pain points for which the corporate world is known. If

you abide by the principles I've laid out above, you'll never have to suffer through another paternalistic performance review, let alone sit through some kind of company-mandated sexual harassment video from the seventies.

Speaking of the performance review, can you imagine how refreshing it would be to operate in such a way that everyone's value to the enterprise was transparent and obvious? Compensation structures would be clear up front. So too would the amount of money that people brought in. Discussing performance in any assessment capacity would be redundant, since you could basically create the equivalent of a build radiator for it. Having an objective measure of your performance, the way you do with metrics on code, would be highly morale-boosting.

To zoom out, what I'm talking about cuts down massively on waste. You're free to only add things if you need them. This includes functions like HR or legal, but it also includes more mundane stuff like office space and the gas or train fare it costs to commute. All of that is within your control.

Enterprise clients trying to go agile often ask me what the best way to do agile at scale is. I generally hedge when asked this directly. You will, no doubt, understand why now. My honest opinion on the best way to scale agile is "you don't." The core principles laid out in the manifesto and reinforced now all over the world emphasize common sense, human interaction, fast feedback, and adaptability. None of that applies to enterprises and programs lumbering along like brontosauruses because no one has conceived of how to break them into manageable bites.

Near the beginning of Part 2, I said that your odds of becoming Mark Zuckerberg are the same as your odds of becoming the next LeBron James. When you shoot for Zuckerberg, what you usually wind up presiding over, after a career of blood, sweat and tears, is one of those brontosauruses. If you grow within the framework I'm outlining, you know that sanity and dignity will accompany you at every stage. You'll have sustainable growth when you grow, and the things that

make you want to facepalm will be isolated to the occasional client interaction that you can always opt out of.

I'll close my pitch for these principles as guidelines for efficiencer firms with a literary reference. I don't want our future of developer hegemony to look like *Animal Farm*. I don't want us to use our leverage to become the opportunists presiding over the pyramid instead of the pragmatists at the bottom of it. It's time to stop working as if all commerce were nineteenth-century textile factories.

Let's switch gears now and consider what an efficiencer firm might look like. It could certainly be a solo endeavor, though I'd advocate considering partnerships as this becomes more common. Speaking as a solo efficiencer that is starting to partner, I can say that partnership pools the risk, sort of the way an insurance company does. It's a more stable arrangement.

Efficiencer firms could form easily enough (in the US, at least) under a partnership LLC structure, though I won't get too far into specifics. The key thing here is to define an operating agreement that covers explicitly what happens in the event that the partners want to stop working together at some point. It should also spell out what adding new partners looks like, in terms of who has to approve it and what kinds of revenue splits are on the table. This overhead is not intended to create unneeded bureaucracy but to prevent future messes. In a traditional corporate context, termination is a stock procedure (both voluntary and involuntary). The efficiencer equivalent should be the same. It's good not to pretend that every commercial arrangement we enter is the last one we'll ever want.

In terms of the actual work conducted by these firms, it would center around efficiency (meaning, in all likelihood, automation and a lot of code). I encourage you to anchor these firms around a niche—a productized service or at least a highly targeted service. This is not to say that these firms shouldn't take custom app dev projects. Far from it. The need for custom code isn't going away anytime soon. But what you need to do is get away from saying that you know Python and you'll do anything, just please-oh-please give you a job.

That's subordinate behavior and it's unfair to clients since it forces *them* to figure out how to use *your* expertise. Yes, everyone's used to that. But 200 years ago, everyone was used to going to the bathroom outside. It's not a reason to continue the practice.

When you start to define your niche and offering, managing work volume becomes a lot easier. You're going to have fewer situations like the one that smallish app dev firms have no doubt all faced: "ZOMG, Godzilla Corp. signed a huge contract with us, which is awesome, but we need five new people YESTERDAY!" Nevertheless, there are situations that will come up where you need to work beyond your current capacity.

In the future state, I see addressing this in a variety of ways. First of all, you could add a new partner to the firm. I would do this only if it addresses capacity while being a sensible partnership. What you want to look for in a new partner is someone with decent knowledge of your niche, a track record of understanding all facets of the business, and compatibility with the existing partners. Oh, and the prospect bringing along a book of business, or at least a lot of leads, should be considered table stakes. Draft some kind of agreement that may include trial partnership, and bring her on.

If you have a more tactical need for automation work, then I'd look to the freelance market and bring on subcontractors for spot work. Remember, you have in-depth knowledge of the niche/domain and deep technical knowledge, so you have a great leg-up on Bob's Pizza Shack looking for a contractor to build a website. Go to your network and hit it hard. Look for independents you've worked with in the past or full-time employees with bandwidth for moonlighting. If you're a partnership of well-traveled efficiencers, you will know *hundreds* of possible people.

If that doesn't work, you have other options as well. I've found subcontractor work through people that follow my blog, local networking things, and even Twitter. Look for people active in the community with reputations to protect. They'll almost certainly do good work.

Now, when you find these people, contrive of experiments that will let you see if they'll work out and fail fast otherwise. Peel off a tiny slice of work, pay them to do it, see how it goes, and keep an open dialog. If it goes well, offer up consecutively larger slices. This works. I've subcontracted labor both to former colleagues and strangers without anything resembling an interview. Do you know what I did instead? I asked them if they thought they could do the work. And do you know what else? They answered honestly. You might think I'm a sucker, but this has never once come back to haunt me.

That covers general operations related specifically to clients, growth, and dissolution. But what about the internal operations of the efficiencer firm? That's going to be pretty fluid and depend a great deal on the skill sets of the individuals involved. Generally speaking, as a bootstrap enterprise, you'll handle all of that internally, with the most efficient person handling internal operations through a division of labor. For instance, if you have a partner that's really good at marketing and another one that's really good at finance, they'll take on those tasks as overhead to the enterprise. They'll also have periodic sanity checks to make sure they wouldn't be better served to automate or delegate the things they're doing, letting them focus more on revenue-generating activities—in other words, is the opportunity cost of spending all day in QuickBooks higher than what you save by not paying an accounting firm?

Collective introspection and evaluation isn't hard if you keep things in perspective. Remember one of my essential principles of an efficiencer firm: that you continually make sure everyone's contributions are easy to evaluate. If you govern yourselves according to these principles, conversations about how to improve the business's efficiency and productivity are easy to start, even if perhaps the solutions present difficulties. It also makes conversations about performance easier to have since the issues should be self-evident, the way problems become obvious when your code doesn't compile.

Now, there's another important consideration for all of this to really get jump started and hit critical mass. We need some changes in the general workforce, some of which I think will come regardless of how intentional we are about moving toward efficiencer firms.

First of all, we could certainly use less anachronistic labor laws, particularly around who is and who isn't considered an employee. As the increasingly digital world hurtles toward the gig economy, these laws will creakily catch up, even if progress is slow. For instance, Illinois, my home state for the moment, has a labor law on the books that more or less says, "We'll penalize you if you use a subcontractor that we think is really more like an employee, but we won't tell you what criteria we use to make that distinction—it's a case by case thing." They did this in response to years of companies using loopholes to get out of paying benefits, qualifying their laborers as contractors instead of employees. So while I'm sympathetic to those workers and to the state's frustration, closing the loophole by making the thing completely subjective is hardly a great solution. Our world changes quickly, and the old men that make laws struggle to keep up. But they'll get there, and we'll benefit.

Another critical component to developer hegemony via efficiencer firms is the supporting businesses that I mentioned—accounting firms, law firms, marketing firms, and the like that cater to us. Right now, it feels like going into business for yourself makes you choose between overwhelming and expensive. You have to *manage* the business and *be* the business, or else take on the prohibitive cost of outsourcing some of this work. But that will change as more people make the leap that I and that most of the developer opportunists mentioned earlier did. Firms will perceive a market. Efficiencyers like us might just turn around and scratch that itch for would-be efficiencyers like you, automating some aspects of the work. Some of them may do it for pay, and some may even come up with creative arrangements, like offering the help in exchange for equity in your firm. And I think you might see a shift in what recruiters do. Instead of trying to staff developers at MegaTech Corp., they may start

matchmaking between efficiencer firms looking to scale up and down for projects.

The last sea change that I'll mention is the critical mass that will take place at some point. As developers flow out of large firms and into small ones—into independence—the vendors catering to these markets will grow. But so too will the support networks and infrastructure. For instance, consider something I've always thought of as a great conceptual alternative to the programming interview: hire someone for a week and see how it goes. Currently, that's unworkable. Few staff software developers will want to quit their current job to try out another one for a week when the new one might not work out. But imagine if *thousands* of firms hired this way. Now the risk is gone, since you can just temp-hire on at a different one each week until you find a good landing spot. This, in a nutshell is the sort of critical mass paradigm I'm talking about. The more people that leave standard arrangements, the easier it will become for you to leave as well.

I'll close this chapter by addressing a series of objections that people may have at this point.

Objection 1: I get what you're saying about Zuckerberg and the evils of scale, but I want to get rich!

Bear in mind that I'm not saying efficiencer firms can't scale, and I'm certainly not saying they can't make unbounded amounts of money. I'm saying that they can scale, but with some stipulations:

- They avoid venture capital
- They don't employ pragmatists and idealists.
- They don't scale sloppily (e.g., via job interviews).
- They don't obscure individual contributions as they grow.

This list might seem prohibitive, but it really just means they avoid the mindless, grow-like-cancer mandate of the standard corporation.

They can't bring on headcount so that the owners can profit off of the margins generated by grunts (by definition, pragmatists).

Consider two examples. First, as an example of unbounded money, the efficiencer firm could collaborate on a book or video series. If that asset went globally viral and raked in seven-figure sales, they'd be rolling in money without the need to add massive headcount and bureaucracy. This may seem a bit facile, but it illustrates that you have a lot of options for revenue generation that don't require capitalizing on others' labor, some of which you may never have considered.

The second example I'll offer is how you could scale and meet all of these criteria. You could scale by franchising the partnership into other geographical markets. In other words, you could establish a line of business that brings in aspiring efficiencers in other markets and they pay you to train them and lend them your brand. They set up shop in another city and fend for themselves, giving you an ongoing cut of whatever revenue they bring in. I'm not endorsing that model, per se, but rather pointing out another less traditional way to grow without committing all of the sins of a Dilbert comic.

Objection 2: I need the stability of a paycheck and efficiencer firms don't offer that.

Unlike the conundrum of how to grow without grunt profiteering, I think industry trends will play a big part in addressing this one. Right now, you're somewhat out of luck. But as more developers go the small firm and efficiencer route, you'll have a new class of entrepreneur with disposable income, and that will attract people to serve them.

Down the line, I anticipate that the unknown factor of logistics will be eliminated for those going on their own, to a large degree. Where you get benefits and for how much, how to legally establish the business, how to market—all will be much better defined. Your

calculus will be simplified to “Do I think I can get enough clients to generate X in revenue?”

Of course, that’s still a pretty big unknown. But imagine that you weren’t striking out on your own and were instead discussing partnership with an established efficiencer firm that had been around for a while. I imagine the risk would seem less, well, risky.

I’ll conclude answering this question with some blunt wisdom from Matt Heusser.

Financial security is an illusion. Worse, being an employee is a huge risk.

Consider two people, both making about the same per year. One is a freelancer who has about ten customers, with the largest being no more than 25 percent of his income. The other is an employee with ONE customer who gives him 100 percent of his income. Who is more at risk?

The employee.

Objection 3: What about stuff that requires a massive amount of capital, like a Mars robot? You can’t bootstrap that.

I wanted to throw this in here to make it clear that I’m not saying that *all* software development will be the product of efficiencer firms. It’s not as though, at some magical date in five years, all software developers will get up and leave their jobs at large companies as if some kind of brain control device had just been activated. I’m speaking in broader, more general trends.

Companies will continue to operate the way they operate. They’ll have investment capital, massive payrolls, mind-numbing status calls with 200 people on them, and all of the things that we know and love

about them. And they'll continue to need software, including companies that build Mars robots.

I just think that, when they do, they'll call more and more frequently on efficiencer firms instead of on recruiters to find them rockstar ninja embedded heroes with twelve years of C.

Objection 4: What about entry-level developers and upskilling in this new way of doing stuff?

I'm glad you asked. I'm including a chapter on this topic, which starts now.

Chapter 47: Becoming an Efficiencer and Joining an Efficiencer Firm

As I mentioned, I went to Carnegie Mellon University (CMU). During my tenure there, the school of computer science had an ongoing rivalry with the Massachusetts Institute of Technology (MIT) for the distinction of “top CS university in the world.” Go Tartans! (Yes, really, Tartans.)

I say this not to brag but to illustrate a point. In the debate on whether or not one needs a CS degree—a debate that serves as incessant background noise for our industry—I theoretically should come down firmly on the side of needing a CS degree, particularly when you also consider that I earned a master’s degree from another prestigious university, the University of Illinois at Urbana-Champaign (UIUC).

The CMU undergrad degree *opened doors* for me, as explained to me by companies inviting me to interview. Organizations like Google and serious Wall Street firms would periodically contact me, explaining that they really liked grads from “top five” schools. In fact, they would sometimes boast that they *only* considered people from these schools. For a quick rundown of the names we’re talking here, [2014’s US News ranking](#) has the following top five, with a four-way tie for first...somehow.

1. CMU
2. MIT
3. Stanford University
4. University of California at Berkeley
5. UIUC

So, six to ten years ago, you had some of the most sought-after employers basically saying, “We only consider people that came out

of CMU, MIT, Stanford, UC Berkeley, or UIUC.” Maybe, if they were really in a pinch, they’d lower their standards enough to consider people from Cornell or Princeton. But don’t count on it, you Ivy League also-rans.

I interviewed at some companies with this elitist hiring approach (though I could never really understand why, if that’s what they valued, they didn’t just ask for SAT scores and not waste their time and energy on interviews). I never once took a job with any of them. Some I wound up passing on. Sometimes, they passed on me, generally because I hadn’t relived my CS451 algorithms class recently enough for their liking. I had two outstanding degrees that opened plenty of doors for me, and I never once walked through them.

Don’t get me wrong. The degrees didn’t hurt anything at the places that *did* hire me, and I value them. I value them for what I learned at those schools, for the life experience, and for what they meant on my resume at one time (after a bunch of years, it kind of stops mattering where you went to school).

Why am I going on about my own degrees and background? Because I don’t think that I could, in good faith, recommend to an eighteen-year-old aspiring programmer that she follow the path I did. After all, some of the developer opportunists that I interviewed have done quite well in the industry without CS degrees.

Even defraying the cost of my degrees with academic scholarships and tuition reimbursement, they were still quite expensive. In today’s dollars, the retail cost for my education would be \$367,144. I got to watch guest lecturers who had won Nobel Prizes and I got to play with some really awesome robots during my coursework, but nonetheless, that’s a staggering figure. If I were to pay it off over the course of a forty-five-year career, I would pay \$8,158 per year, or \$679 per month.

I don’t know exactly what the earning splits are in the programming industry between people with no degree, a BS degree, and an MS

degree, but I cannot believe they cover that amount. So, my degrees? Much like eating a garlic and jalapeno pizza and washing it down with a beer last week, I enjoyed the experience at the time but do not recommend it to others. I don't think you're likely to see a return on that investment, especially now that those same erstwhile elitist companies have moved away from that hiring practice.

Nevertheless, the historical impact of the CS degree on our industry looms large. And it has its fingerprints all over the journeyman idealist layer of organizations.

Decades ago, earning a CS degree meant that you might reasonably learn how a computer worked, soup to nuts. Early computers had more in common with programmable graphing calculators than with your MacBook Pro, so this wasn't as crazy as it sounds. Even in my day, this broad understanding came easier. I learned a lot of math and algorithm theory as an undergrad and rounded it out as a graduate with "closer to the metal" courses where I studied computer architecture, how processors work, and other interesting and foundational things.

Today, however, that end-to-end understanding asks too much from us mere mortals. Now there are entire complex languages and frameworks that run in the browser. When I was an undergrad, "browser" meant Internet Explorer 5. From a computer science coursework perspective, CMU's unofficial take was, "We do C and C++, with which you're going to write a garbage collector. You can learn fluff like HTML in your spare time if you want to slum with hobbyists." A decade before that, the conversation would have been, "What's a browser, and what's a Windows?"

If you got started all of those years ago and amassed a complete understanding at that time, and you've also stayed close to the keyboard, *and* you've been able to assimilate new technologies as they come, you're in an elite class. If you didn't get started all of those years ago, forget it—it's hopeless. And rightfully so. Labor specialization is the reason we're not all still wandering around

wearing animal skins and eating wild edibles that we gather from fields. Civilization requires specialization.

But that doesn't stop the pervasive overvaluing of the unassailably complete knowledge. We humans have a cognitive bias known as [the endowment effect](#), wherein we value things more when we own them than if we were buying them. Those of us into the CS education system for a quarter of a million dollars have a whole lot of endowment, and we are adamant that you should know how to whiteboard an alpha-beta pruning algorithm and tell us its worst-case runtime. Are you going to use that in your day-to-day work? No, never. Of course not. But we had to do it for our junior year midterms, damn it, so it must be important. And we're not hiring you unless you also spend a bunch of nights cramming like we did. You too must perform and then promptly forget how to do it three weeks later.

It's interesting that, in a quirk of history beyond the scope of this book, the computer science degree has drifted gently away from actual programming work over the years. The degree emphasizes highly mathematical principles and academic concepts (which I loved, for the record), but it spits out graduates that have only a rudimentary grasp of the tools of the workplace trade, like source control, build machines, group collaboration, and legacy rescue. This results in you amassing four years of theoretical background. How things really work is something you learn on the job.

CS degrees, as they are, have therefore created a vacuum ripe for filling with vocational programs. Generally speaking, vocational programs aim to distill a well-rounded college curriculum, which tends to include general education requirements, to a minimum viable education. These schools teach you precisely what you need to know to get hired in the field, and no more. Many programs for chefs and auto mechanics work this way. But CS vocational programs are different.

Instead of one- to two-year-long vocational programs, what computer science has gotten is [the relatively shorter \(and theoretically more](#)

[intense\) boot camp.](#) These programs have a much more palatable price tag, running in the low five figures or perhaps taking a cut of your early salaries. From an ROI perspective, this is a no-brainer, if all goes according to plan. If you occupied a miscellaneous business job at \$40K per year but then sunk six months and \$20K in learning to program and landed an \$80K per year job, you'd be in the black six months into your new career.

I won't bother to speak to the efficacy of boot camps since, with fifteen pro years and two degrees, my personal path is about as far from that one as you can get. For our purposes, let's assume that they're highly effective at turning would-be entry-level programmers into real entry-level programmers, capable of finding a text box on a website and filling it with data that came from a database.

This newbie is now the perfect sidekick for the journeyman idealist, and it explains the rising popularity of the guild metaphor. You now have a second stream of human beings flowing into the mix. The first stream has four years of academic theory and X years of sink-or-swim on-the-job learning, and they are ready to algorithm-trivia-haze the crap out of anyone following in their wake. The second stream emerges from the crucible of the high intensity, forced-bonding experience of learning to code the way a piece of coal learns to diamond. They are the fraternity pledges, ready to be hazed by seniors wanting to know how much memory Quicksort consumes.

This is the professional world, of course, so the dynamic assumes a much statelier form. The newbies form up as apprentices, ready to learn at the feet of true masters who say, "Patience, young grasshopper. I will teach you teh codez. You probably don't even know that you should only ever use a string builder to concatenate more than two strings. But I forgive you."

What you have now is two sets of people arriving along two very different paths but coming together in the shared illusion that getting really, really good with code can and should be its own commercial reward. Figuring out who will pay for that skill and why is someone else's problem. It is precisely this kind of dynamic that leads to the

reverence of programming-as-art types of affairs that essentially amount to collective tech-chops navel-gazing.

“Check it out! I just hacked an old Super Nintendo to make Mario naked any time he shows up in any game for the system.”

“Dude, why?”

“Because someone had to.”

“Awesome! Your industry cred is at an all-time high.”

To be clear, if someone actually did that, I would be highly amused. I would also recognize that it takes a good bit of skill and notable (though questionably aimed) determination. But what I would *not* do is venerate this activity as anything other than an aesthetic pursuit. This is art, or some definition and interpretation of art. And it's nothing besides.

Art is worth appreciating, and it has a place in the world. But art should not be conflated with anything productive in a commercial sense. It's not even related. It's not a sign that you want this person helping you write line-of-business apps. It's not a sign that this person would make a great addition to a team. While it *may* be a sign that this person is a skilled programmer, it's not a sign that he is a *profitable* programmer. It's definitely not a sign that this person is an efficiencer.

What our naked Mario artist actually represents is an ongoing reinforcement of a dynamic: we, as programmers, can be distracted from our own fates and autonomy by asinine gamification.

Now, there is a third stream of people that enter the ranks of professional software developers. And I'm going to use them as the cornerstone of industry onboarding in the efficiencer era. If they don't come from boot camps and they don't come from CS programs, they must float in from another line of work somehow or another.

Dave Rael is an example of someone who came into our world via this third stream. It can happen in any number of ways. Someone starts automating things to make a job more efficient, takes to the automation, and slowly becomes a professional programmer. I've watched people do that successfully. Sometimes it happens more deliberately, with on-the-side learning and formal interviews for positions and trial periods. I've both seen these and approved them as a manager. I could go on, but I doubt I could list every conceivable path to programmer that starts somewhere else in the professional workforce.

Here lies the key.

What use does an efficiencer firm, containing developer opportunists like the ones I've interviewed for this book, have for entry-level CS grads or boot camp grads? Not much, honestly. Not as a partner, anyway. Both these sets of people come in able to do a bit of programming but know little or nothing about the other aspects of a business that the efficiencers might find useful: sales, marketing, operations, and finance. They won't have industry contacts, and they won't have insight into offerings the efficiencer firm could make. As it stands, these folks would be best served by getting a foot in the door at Humongous Inc. and treating that as a paid efficiencer school, plumbing the cavernous depths of that organization for efficiencer opportunities while collecting a below-market wage.

But the people with industry experience? Give me those to partner with any day of the week. Those folks can come to the efficiencer firm with plenty of product and service ideas, industry contacts, and sales leads. Picture someone coming to an efficiencer firm with this:

“Hey, you guys specialize in simplifying telephone switchboard menus? I used to work the support desk at a company that made those systems, and I have hundreds of leads I could share with you. I've been working on my programming and have even prototyped some things. What do you say? If I open up my

lead book to you and hand over what code I have, will you work with me on improving my skills and teach me your playbook?”

Now I’m interested. Go on, sir.

Is that to say that no path exists directly from a boot camp or college to an efficiencer firm? Certainly not. But the path isn’t obvious. Both of those programs fail to spit out people ready to contribute on day one, so their initial salary is always going to be an investment from a company. That means that the company needs to either be able to bill them out to a third party to learn on the job, or else they need those people to stick around long enough to recoup a return on the investment. In both cases, the rational move for the company is to depress those people’s wages as much as possible for as long as possible. In fact, if you’re one of those people, I would almost categorically recommend hopping sooner than later so the company you work for doesn’t quasi-permanently view you as a reclamation project. But I digress.

An efficiencer company does not have pragmatists, and it does not play the “Let’s hope to get an all-star rookie that we can underpay for a while” game. Efficiency firms should not enter the business of human capital investment, in my opinion. That means the would-be efficiencers at the entry level need to bring something to the table. And let’s face it: that something most likely won’t be efficient and sustainable automation. You don’t come out of college writing simple, elegant, maintainable code.

One option you have is to *try* with code. You could write something, however unsure you may be of your skills, that scratches an itch related to the efficiencer firm’s purview. For instance, perhaps you cobble together a piece of software that migrates data from QuickBooks to new upstart accounting software. If the efficiencer firm specializes in accounting software migrations, they’ll no doubt be interested in talking with you.

Another option that you'd have in this future is to try to approach efficiencer firms on the sales side, but in the sense of identifying and articulating an itch to be scratched. Perhaps you have enough passing conversations to learn that a lot of companies hate QuickBooks and are interested in EasyBooks, the new kid on the block. Most firms don't pull the trigger, though, because EasyBooks doesn't port a lot of data over automatically. Aha. Maybe you don't write any code. Maybe you take a page out of *The Lean Startup* and do a concierge offering wherein you actually just transcribe the data by hand over the course of a few nights. Tell the firm you'll do the port for \$2,000 for them and do the grunt work. Now you can go to the efficiencer firm with a client, an idea, and important insight into a market segment.

The third thing that strikes me as reasonable at the entry level is to interact with the efficiencer firm in a moonlighting, subcontracting capacity. You can, of course, layer this with the last two options. But you could also offer to do some pretty small, low risk programming for them as a subcontractor and see if they bite. I can tell you offhand that if you approached me about a bit of moonlighting, with a reasonable project and small rate in mind, I'd probably toss some automation work your way as an experiment. I'd be even more interested if you value priced it to me. This option fits pretty nicely with the profile of an entry-level person since those with no industry experience are less likely to have a mortgage and children to support.

Lastly, since we're likely talking about a period of your life when you can gamble with less risk, you could simply try to start your own efficiencer firm. People like Mark Zuckerberg (and countless others whose names are lost to the dustbin of history after they failed to go supernova) did this, in a sense. They identified an itch and scratched it. They did it with high stakes, angel investors, years of labor, and a series of grand unveils to various people. As I've said before, you won't be Zuckerberg. So aim at something sustainable and bootstrapped, but recognize that all it takes is to understand a need and fill it. (As an aside, mentioning the Zuckerbergs and Gates of the

world is particularly appropriate in discouraging you from getting swept up in developer skill—those folks are heroes and gods of our industry, and it’s not because they were so skilled that they could write a twenty-seven-line Lisp compiler while hungover and talking on the phone. It’s that they opportunistically wrote code that was good enough to get people to gladly fork over money.)

I’ll close out the chapter by saying that I think our industry needs a new form of education to coincide with the rise of efficiencers owning their means of production and calling the shots. Let’s call these “effiencer boot camps.” I’d say “effiencer degrees,” but if I make so bold as to suggest that I’ve coined a line of academic study, I may get struck by a divine bolt of humble-lighting.

I firmly believe that we need to start thinking of ourselves as automation professionals—efficiencers. Learning to write code alone does not come close to equipping us to deliver on this charter. As I said earlier, you also need to understand marketing, sales, operations, and finance. At least, you need to understand these things well enough to delegate them.

Some boot camps give you components of that, perhaps. I’ve heard of them requiring you to start blogs, so that’s something. But the duration is such that you acquire just enough programming skills to keep your head above water at a corporate programming job. It wouldn’t be reasonable for them to expect you to learn a bit about bookkeeping, marketing yourself, identifying automation opportunities, and selling that automation.

That is, it wouldn’t be reasonable unless you made the “boot camps” longer and had them line up with the duration of more traditional vocational schools. In the programming industry, they get to be shorter because the world is a desperately thirsty and ragged man wandering in the automation dessert, looking for water. Six months? Heck, six weeks is fine. Whatever, just give us someone who can write a for loop. But we’re not looking to slake their thirst—at least not in the traditional “Turn this spec into code, grunt” sense that they want.

An efficiencer program could be longer. It could give you enough time to start with the bigger picture concern of identifying automation and efficiency opportunities. It could send you forth into partner companies to interview people working there and discover pain points.

Once you'd gathered up those needs using the interview techniques you were taught, you could bring them back to home base. There, people would show you how to identify good and bad candidates for automation. They could show you how to write user stories and tell you *why* you do this, instead of just teaching you *teh agilez*. They could also show you how to write value stories that express your proposed automation in terms of the amount of money it would save your client, and over how long. They could also show you how to value price the offering and perhaps generalize it for other prospective clients.

Now that you'd have a money-making opportunity to chase with your automation, you could learn to write code toward that end. By all means, learn to write clean code by studying software craftsmanship principles. Learn that you need to keep design flexible because your next client will almost certainly want some changes. Learn to partner up with other efficiencers to get the job done quickly. Learn all of the traditional boot camp stuff. Don't learn any of the O-notation stuff. You can always pick that up if you become an efficiencer whose client needs a string manipulation library sped up slightly.

Once your product is built, learn to sell it. Learn to make those calls and deal with prospects. Learn to use a CRM and do enough marketing to be dangerous. Learn to give webinars. Learn that if you build it, they won't necessarily come.

And then, maybe make a sale or two. Learn to keep some books, pay subcontractors and such. Learn a bit about taxes. Perhaps most importantly, learn to analyze your operating costs and revenue to see if you can sustain or grow with your product as-is or if you need to do some retooling.

Finally, gain some experience with the operational aspects of working with a client. Do you deliver and call it done? Do you maintain it and perform feature requests? Who handles bug fixes? What do discussions about the end of support sound like?

I don't know exactly what this looks like, but I do know that some variant of this brings us a lot closer to being a first-class knowledge work profession instead of an auxiliary cost center nestled in the bowels of ConglomerInc. Earlier in this chapter, I said something offhandedly that carries a great deal of significance: we have two main ways of preparing people to be programmers, and neither one of those produces people ready to program on day one without a *lot* of hand-holding.

It seems to me that this is fundamentally flawed, even if the software labor shortage takes some of the sting out of it. As an industry, we have immense leverage. Yet our preparatory institutions fail to prepare us for jobs. And even when we finally get those jobs, we voluntarily cough up all of our leverage, with the help of the “masters” to our “apprentice.” As a friend of mine once said, “Yep, that’s all the way broken.”

I think a shift in education and training is coming in the next decade. While I don't know what form that takes, I hope it looks something like what I've laid out in this chapter. But more importantly, I hope it achieves the broader goals I have in mind. I would like to see our industry become one of professional automation—of efficiencers. I would like to see people who train for this vocation enter the workforce well equipped to get started, and what's more, to do so dealing with businesses as equals. I encourage all programmers to identify both mentors to emulate and people you could partner with, and to keep identifying these at all stages of their careers. But that doesn't mean that we have to enter the workforce as supplicants, trusting others to validate us and tell us when we're ready.

Famous tech startup icons didn't wait, and they didn't enter our industry ready to “sweep the floors” and “pay their dues.” Don't set your sights on billion-dollar market cap, and don't delude yourself

into thinking you'll rule a massive empire. But learn from them that you can enter and work on your own terms. We own our means of production and have the ability to easily bootstrap. Developer hegemony involves capitalizing on those goals, and it means you don't have to wait for anything in order to do so.

Chapter 48: But How Do We Fix Non-Efficiencer Corporations?

Let me get out of the weeds of efficiencer firms a little bit and speak to the broader corporate world. I've spent a few chapters and a lot of words diving deep on efficiencer firms, but make no mistake, I don't think that *all* software development will be done by these types of organizations in the future.

I mainly think that the efficiencer firm will start to replace the staff of developers that you see at non-software companies: banks, manufacturing companies, retailers, and the like. At organizations like that, software development (and IT in general) is what's called a *cost center*. That term describes a department within an organization that costs money to operate but does not directly contribute to revenue generation. Other cost centers include things like human resources and accounting. Incidentally, at any decent size, all layers of management are also cost centers.

Unlike revenue generating activities and R&D, the sky is not the limit when it comes to cost centers. They have to stay lean and mean, lest a wave of management consultants be called in to trim them as fat. To picture this, imagine you're a freelance dev or, better yet, an efficiencer firm of one. With each new contract, you do discovery, wherein you take a bunch of notes about their problem and turn the resulting thing into an official PDF contract. Transcribing the notes and typing up this contract is not a great use of your time, so you'd probably be amenable to paying someone else to do it, assuming it were both inexpensive and effective. You have a tolerance for delegation and automation of this task, but you'd have the project on a short leash, so to speak.

That's the fundamental condition of corporate IT at non-software companies. They're important but on a short leash. In these sorts of

organizations, corporate IT typically serves two purposes. One is internal cost-saving automation. For example, IT at a bank might be asked to automate the help desk system as much as possible to save on phone support salary. IT's second purpose is product enhancement. Developers, for instance, might be asked to implement a "manage your account" website because customers consider that a fundamental service of modern banking. This is not the bank's *product*, per se, so it qualifies as product enhancement—it makes the actual products, like savings accounts and money markets, more attractive to customers.

For this reason, the company takes an understandably philosophical approach to in-house software: we *probably* want to do this, but only if we absolutely minimize the cost, justify every penny, and actively manage risk. They view the whole process with healthy skepticism and have a quick hook at the ready. Keep that in the back of your head, because I'll return to this shortly.

There is a tried and true organizational play for cost reduction: converting something a vendor does into something an employee does. The reasoning here is the same reason that you should buy in bulk when you can or sign a multi-year lease instead of doing something month to month. When you have enough demand, a supplier will supply a conceptual bulk discount. Now, imagine that we're back to your tiny efficiencer operation. Remember, you're paying someone to type up discovery documents and make PDFs. If this task occupies a few hours per month, you might pay someone \$30 per hour to do it. But if you grow to the point where the task is occupying twenty hours per week, farming that out has a cost of more than \$30K per year. "This is stupid," you say. "I could hire someone to work for me forty hours a week at that rate." So you do.

Writ large, this is the approach to software developers when IT is a pure cost center for a company. High minded rhetoric about all companies being software companies notwithstanding, the DNA of that decision and practice has a double helix ladder made of "Full-timers are cheaper" and "Let's find the cheapest full-timers we can

get by with.” Conceptually, this has, will, and should always make sense. In practice, this used to make sense and no longer makes sense at all.

One important consideration throws a Paul-Bunyan-sized wrench into the spokes; the only thing more expensive than a good software development team is a bad software development team. The colloquialism that comes to mind is penny wise and pound foolish. It describes cost center IT quite effectively.

Returning to the cost-minimizing theme earlier, these organizations do the ostensibly sensible thing by using salaried full-time software developers (often inexperienced ones) instead of much more expensive app dev firms. The problem is that they wind up doing their own app dev *so badly* that it has a higher total cost of ownership than it would have if they’d farmed everything out to an app dev shop.

Part of this problem comes from companies thinking that they can sprinkle a few high-priced “architects” among legions of half-trained entry-level developers and achieve the same outcome as they would sending a couple of industrial engineers to make factory workers on an assembly line more efficient. But part of this also comes from the culture of their risk-minimizing, efficiency-obsessed quick hooks. They heap restrictions, committees, audits, and all sorts of other organizational cruft on top of these legions of developers and then demand to know why they’re going so slowly. “What do you mean it’s hard to get the software you need?” the company says. “We approved that text editor you wanted in just under six weeks!”

One thing I’ve seen time and again in my travels through enterprises is a pair of sequential aha moments. “Aha! We’re really bad at this, so let’s bring in some consultants to figure out what’s wrong and demonstrate how to do it right.” And then, “Aha! Those consultants are really good at this—let’s just pay them to do it instead.”

As those “ahas” blink into existence like stars across the night of the corporate landscape, staff software developers will flow out of those

organizations and into efficiencer firms. There, they'll flip from cost centers to revenue generators. And they'll participate directly and meaningfully in the running of the business. So goes the rise of the efficiencer firm.

That settled, let's talk now about the fate of non-efficiencer firms in a future of developer hegemony. These consist of three interesting types that will probably not ever become efficiencer firms: non-software companies, software product companies, and traditional app dev shops ("consultancies," with quotes used very deliberately).

Why am I claiming these will never become efficiencer firms? For non-software companies, we have an obvious answer that I can express rhetorically. Why would they? They have no more incentive to specialize in helping customers get more efficient than they do to help them extract molars or fight parking tickets. These companies will interact with efficiencer firms in a pure customer-vendor capacity. They won't have any desire to compete.

Secondly, consider software product companies. Again, why would they become efficiencer firms? Granted, their motivations might be a touch grayer since they do have automation at their core and they do trade in efficiency. For instance, Amazon has a massive product/service that helps you buy things more efficiently. But the efficiencer firms of the future exist by fleeing the role of cost center and flipping their former employers to customers. Efficiency firms will offer business specific solutions and play almost exclusively in the B2B (business to business) market. Software product companies may target businesses as customers, but they also heavily target consumers. The difference lies in the model. Software product companies target personas representing the masses and have relatively low-touch engagements. Efficiency firms partner with businesses in a higher touch capacity, offering more targeted help.

Finally, consider the traditional app dev "consultancy." This type of organization seems most ripe for the conversion to efficiencer firm, but I submit that most will never transition. Instead, they will live on as the perpetual budget-brand version of the efficiencer firm, taking

in specs and spitting out matching software and proclamations of “Don’t blame me—that’s what you asked for!”

To this, I attribute the principles of tomorrow’s efficiencer firms I laid out earlier. Recall that:

- Efficiencer firms are bootstrapped and self-sufficient, meaning they’re profitable from the outset.
- The members of efficiencer firms are partners. All members execute on the organization’s value proposition and have skin in the game. Instead of employing pragmatists, they delegate to vendors.
- They don’t rely on absurd practices like job interviews to scale up their delivery capabilities. That happens via trial, recommendations, and networking.
- Everyone’s contributions to the organization’s profits can easily be measured. They only grow as long as that remains true.
- The firm is comprised of opportunists. Anyone who would normally be a pragmatist is instead a vendor. This in turn eliminates the need for an overenthusiastic buffer of managers and senior people.

Efficiencer firms necessarily need to interact with organizational bosses—generally opportunists. Recall that they’re making *strategic* recommendations oriented around automation. They say, “Tell me your goals, and I’ll lay out and oversee execution of your software strategy.” That’s a conversation that happens one opportunist to another. Efficiencer firms, consisting of only opportunists, can do this easily. App dev consultancies? Not so much.

Traditional app dev consultancies bill hourly, and they grow by capturing margin on employee labor and reinvesting it in the company (or keeping it as owner/shareholder profit, but let’s forget about that for the time being). In other words, going back to the pyramidal organizational structure, they grow by adding more pragmatists at the bottom of the pyramid. They also need a proportional number of idealists to manage the margin generators at the bottom. Growth—nicer offices, new branches, additional lines of

business—involves legions of pragmatists, a skeleton structure of idealists, and the occasional (very busy) opportunist.

In a firm with this success story, the opportunists who are capable of strategic organizational thinking must concentrate on solving their own firm's problems at the exclusion of solving the client's problems. To put it more bluntly, if you sell someone else's hourly labor as a product, the person providing that labor is hardly credible as a partner. They figure that, if the laborer was actually strategic, why would he be making \$50 per hour for strategy work worth at least four or five times that amount? That doesn't seem very strategic.

Of course, the client opportunists probably don't think of it quite this bluntly. It's more of a subconscious matter of perception that echoes the tragedy of applied Scrum from Part 4, where we talked about how to win the corporate game. However much one might talk the game of a dominant wolf, the client opportunist has a hard time seeing past the fact that these developers are doing so while lying on the ground, exposing their bellies submissively.

When app dev consultancies interact with clients, success means expanding the grunt surface area as well. They might start with a few consultations or a small team, but as they open up the account, they'll flood the client with wave after wave of belly-showing pragmatists. The gestalt is that the client inevitably views them as a non-strategic supplier of cheap labor, effectively rendering moot any chance the app dev shop has of doing effective efficiencer work.

Philosophically speaking, traditional dollars-for-hours app dev shops have an Industrial-Age growth model. The closer they get to selling a cheaply produced, fungible commodity, the bigger their valuation and the higher their profits. It's a tried and true model. The problem is that you can't exactly expect it to pivot to a model where the cheap, fungible commodity wakes up and starts offering strategy advice. If these firms want to be efficiencer firms, they will need to conceive of foundationally different ways to scale. And that's a pretty unreasonable thing to ask of an organization that has already achieved success. It may not seem like it, but asking large,

successful app dev shops to become efficiencer shops isn't actually that different than asking banks or product companies to do so.

So, we've established that efficiencer firms will emerge besides companies that currently exist rather than replacing them—at least at first. Eventually the app dev shop will dwindle beside the efficiencer firm. But that's probably a long way off, so let's not worry too much about it.

The question then becomes, "What does the interplay look like between these companies and efficiencer firms with the rise of developer hegemony?" Simply put, efficiencer firms will offer their services to non-software companies and software product companies. Other organizations become clients. With traditional app dev shops, efficiencers will sometimes compete, sometimes partner. But they will fire a shot across the bow of all three types of organizations in the form of driving up the market value of software work. These firms will offer more profitable, more autonomous, more dignified, more fulfilling work to software people. And that, in turn, will make it more difficult for these other companies to staff software people.

This might sound like a brash prediction, but even without many efficiencer firms today, this already happens. Software developers are often held to different rules within organizations, simply because they're hard to replace. We're already unwittingly using our leverage to get our way. Imagine what it starts to look like when we do it *deliberately*.

For the rest of this chapter, I'll talk about fixing the existing corporation to make it more developer (and aspiring efficiencer) friendly. In the world of developer hegemony, the same old crap just isn't going to cut it. If you *try* to make Taylor's system of labor work, you'll get slaughtered in efficiency by your competition, who won't bother to grumble about catering to their software people.

I'll list some basics first.

First, the nine-to-five work day needs to go away. There are two principle factors at play here that tend to anchor this in place: child care and old men. The world has loosely agreed to a public babysitting service (school) that occurs during the work day, and that pressures people to conform to the mean. Secondly, old men are in charge of traditional corporations, and old men have circadian rhythms that get them up and going early. Unfortunately, [human circadian rhythm doesn't make a nine-to-five work day make sense until you're fifty-five years old](#). Corporate workers have had to put up with this forever. Software developers don't. Word to corporations and potential firms alike: if you want to remain competitive, stop trying to make them comply with unnatural schedules.

Speaking of throwbacks, do you still require everyone to be present at the office? Have you ever heard of this thing called Skype? Or the internet? Have you thought about the fact that it's 2017? Because developers have. Suffering through commutes and choosing only from among the jobs in a twenty-mile vicinity are problems that historically have been unavoidable. They're not anymore. And if you try to force the people within twenty miles of you to do that, some company on the other side of the globe is going to sink you before you even know it by hiring the most effective people right out from under your nose.

The common theme here, if you'd like to extrapolate, is "That's the way we've always done it, by God!" no longer cuts it. Organizations can remain competitive by being willing to put aside traditional, paternalistic practices and trusting software pros to get their work done on their terms, in a way that makes sense to them. I understand it's worrisome to them, but soon it will be more worrisome not to. So let them come in at 10:00 AM, work from home, wear jeans, and eat at their desks. If you're a corporation and you're trying to fight and win those battles with software developers, you're soon going to start winning them by forfeit. No developers will be in your company to fight back.

All of that applies to the here and now. These things are *already* table stakes for attracting tech talent. But more is coming on the horizon.

To get ahead of that curve, here are less obvious ideas.

Organizations must remake their interview process. By now, you understand that I view job interviews as a complete waste of everyone's time. But I also understand that, until the gig economy becomes significantly more mature, companies can't just do away with them. What they can do is expel as much of the stupid as possible. Don't ask about algorithm trivia. Don't pelt interviewees with Edison's brain teasers. Put people in situations that mimic their target job and review their performance with them. Ruthlessly eliminate everything from the process that makes the interviewers feel as though they're part of some exclusive club. And for the love of God, stop saying, "We only hire the best." That is ipso facto not true, for every company that everyone reading this has ever worked for. Platitudes like that only reinforce the interview process as a vanity exercise and encourage expert beginners and organizational rot.

Instead of ranting about more problems with the current beast, however, I'll turn now to what some interesting organizations are doing or have done to help them remain attractive destinations during the days of developer hegemony.

First up, consider [GitHub](#), where Sally Lehman once worked. She described what she looks for in an employer this way: "First thing I look for is a team that I would enjoy working with, and secondly I look for technical and company structures that are conducive to being productive." GitHub once had a policy known as [open allocation](#), wherein the individual employees choose which projects to work on and how to spend their time. This intense trust in the workforce will attract products of developer hegemony, as it closely mirrors the opportunist-only partnership model of the efficiencer firm. Beware, though, as this can become a victim of the scale imperative. After Sally left, she heard that a lot of the management policies

changed as the organization doubled in size. The more folks, the harder to trust them.

Another interesting model is the one I described earlier: David Boike's arrangement with Particular. He has his own consultancy but only a single client. Particular scales by partnership, in both the conceptual and the legal senses. I love the message that this sends against the backdrop of developer hegemony. It reinforces the partnership concept, obviously, but it also presents a way to scale that doesn't involve the introduction of pragmatists and idealists. Extrapolating, the firms of the future would serve themselves well by exploring ways to partner with software developers that actually demonstrate the programmers' skin in the game. Companies with pap on their website's culture page like "Everyone here is as important as the CEO" are a dime a dozen. Companies that *demonstrate* partnering with developers in a meaningful way are not.

Speaking of partnering, I'd like to talk about an organization called [Pillar Technology](#) and tell a bit of my own story. Pillar provides consulting and app dev services to its clients, and I subcontract for them sometimes. An executive there once described my arrangement elegantly, using a great analogy. He said that newspapers tend to have staff writers, who have traditional salaried jobs. But they also have contributing writers, who have a standing relationship with the paper but operate as independent contractors. He then described me as a contributing consultant, which I thought was wonderful. That's a singularly progressive way to partner with people in a way that blurs the line between employee and independent.

Organizations could do worse than to mimic Pillar in this regard. If you look at the panel of folks that I've interviewed and at some of the bigger names in the software industry, you'll find people for whom it would not make sense to accept a salaried position. Firms that figure out how to *partner* with these sorts of people create positive sum scenarios where both benefit from the relationship. That's an

excellent way to stay relevant on the rising tide of developer hegemony.

I'll close the chapter with one last, more philosophical bit of advice for today's companies. A lot of what I've said up to this point falls under its heading, but it's your thematic takeaway:

Stop pretending that software developers are doing labor work for you, and stop pretending that they'll work for you forever.

I can hear the protests. "We know they're super-valuable members of the team." "We know there's a lot of turnover in the industry." But the company still has "career growth" activities and performance reviews whether the employee wants them or not. The company and its people still talk in hushed tones about Bob potentially leaving, then send awkward emails and have an awkward send-off lunch when he does. The company still paternalistically talks about being "a family" of 500 people and insists on requiring permission for authorized absences, vacations, and even showing up late one day. I could go on, but you get the idea.

The organizations that succeed in the world of developer hegemony will be the ones that find creative ways to break this mold. Software developers need these companies less than the companies need them. If an organization can find ways to treat them as partners, as efficiency experts, and as autonomous, independent humans, it'll be just fine in the years to come.

Chapter 49: Concrete, Immediate Steps You Can Take to Become an Efficiencer

I'd like to get a little bit more grounded now. I've talked in great detail about where I think the industry is going, how efficiencers can take us there, and what principles they should have. But creating alternate organization structures and radically altering how and for whom you work might seem a bridge too far. This would have been a reach for me for most of my life, too.

For this chapter, I'd like to return more to the tangible present for most of you reading. What can you do in the here and now to change the game for yourself, and to a lesser extent, for the industry at large? How can you participate in bringing about developer hegemony and the rise of the efficiencer?

To that point, I'm going to focus on actions that involve nothing more dramatic than applying for your next job. That means that, individually, nothing here should be too far outside of your comfort zone. (I'm assuming just about everyone reading has applied for a job at some point.) But in aggregate, these actions will set you on the path to developer opportunism, and in some cases, to participating in the efficiencer movement.

What you can do now: train yourself as an efficiencer at your current job

Speaking of efficiencers, let's start off with that. One of the lowest risk things that you can possibly do is start to operate as if you were already an efficiencer within your current company. Consider it a form of on-the-job self-directed training. You'll be developing a skill set that helps you in any organization, and you'll be learning how to be taken seriously?something that doesn't tend to happen with workaday developers outside of the development group itself.

Recall that efficiencers position themselves as automation experts with a full understanding of the business around that automation. This involves an ability to audit organizational processes and assess whether automation of those processes would pay for itself. To put it concretely, if Bill from two cubes over spends half his day filling out digital forms by typing, you should be able to speak to whether automating that work makes sense for the organization, financially.

So, first things first. Stop geeking out about how you could use some JavaScript framework invented yesterday to automate what Bill is doing. And stop diving in or volunteering to do it just because it's there and you can. That's the sort of non-strategic, subordination-inviting behavior we need to avoid, collectively.

Don't volunteer anything at first, in fact. Just get practice observing what people are doing and assessing how difficult and expensive automation might be. Learn how to do gap analysis—find situations where actual performance falls short of what's desired. Do a gap analysis that focuses on automatable activities. This could mean thinking about Bill and his manual data entry or the gigantic defect tracking spreadsheet that the QA department maintains. It might be something as simple as people routinely printing out emails and handing them to one another instead of using email. Remember, your goal isn't to find things that you can write code to solve. Your goal is to find where you could eliminate inefficiencies with automation.

Once you've begun to recognize these, learn to size up what they cost the organization. This means ballparking Bill's salary and calculating how much of that goes toward typing things in manually. Or it might mean doing some thinking about how much time QA spends updating their spreadsheet. In the manual printing example, calculate time, but also add in paper, toner, and printer maintenance cost. Gain experience estimating fixed and recurring costs of processes.

This experience will serve as the basis for deciding whether automation is worthwhile or not. In the event that automating the

process requires custom app dev, you can price that expensive intervention according to your own salary cost. But remember, your goal is inexpensive automation, and paying for custom app dev is not inexpensive. Look first for pure process solutions. Throw it out there to a coworker: “Hey, why don’t you guys email those documents instead of printing them out and walking them over?” If you convince them with a single sentence, your solution just cost about four cents and saved who knows how much. Look for process improvements and existing solutions first. Then contemplate what sorts of commercial off-the-shelf products could help. Only after that should you let your compiler finger get itchy as you contemplate writing some code to fix the problem.

With all of that experience in place, start practicing your sales pitch. This is where you take your business case, demonstrating a return on investment, and pitch it to a decision maker. You’ll be surprised by how often you get shot down, even with a bulletproof case. This is to be expected since you’re a developer and no one is used to business strategy coming from developers. You’ll get the message that it’s cute that you’re trying, but people who “know business” have already figured out how you should spend your time. Don’t let that attitude dissuade you. Keep at it. Pitch it to different people. Eventually, someone will bite.

Once someone bites, you have successfully turned yourself into an efficiencer without ever doing anything more risky than going above and beyond for your company. Practice this as much as you’re comfortable. There is absolutely no downside.

What you can do now: avoid draconian non-competes and other nasty corporate policies

With the low hanging efficiencer fruit out of the way, let’s talk about one of the easiest things to implement the next time you go job searching (which can be now, if you’re so inclined). You can certainly practice efficiencer behavior to your heart’s content within a company, but you need to position yourself to practice it externally as

well. You need to market yourself, build your brand, and maximize your options. And it's really, really hard to do that with a company that's forced you to sign a contract laying claim to all of your intellectual property. To further explain, I'll tell a story.

When I was a kid, my little brother watched Disney films constantly between the ages of one and six. As a result, I have an embarrassingly encyclopedic knowledge of the plots and songs from these movies within a fairly narrow time window. *The Little Mermaid* fell right in the middle of this range, and to this day I giggle thinking about that crazed chef chasing Sebastian the crab all over the kitchen.

I also remember the witch Ursula and the Faustian bargain that she offered Ariel. Ursula would transform Ariel into a human for three days to win the man she fell for. If she secures a kiss from this man, everyone lives happily ever after. If she fails, she's a mermaid again, and now Ursula owns her.

Many companies behave like Ursula. They offer you deals like, "We'll pay for your tuition as a perk, but we'll claw the money back from you if you don't continue to work for us for years," and, "We'll employ you, but we own everything you come up with during work hours. And also at home, in your spare time."

When you come to work for the company, they sing to you in Ursula's dulcet tones.

Poor unfortunate souls,

In pain, in need!

This one longing for a paycheck,

That one wants a new degree,

And do I help them?

Yes indeed!

Of course, they warn you about the potential consequences of this help. In the fine print. It's hardly even worth mentioning because, if you're a decent human being and a good worker, you'll never even have to think about it. But still, you should know...

Now it's happened once or twice,

Someone couldn't pay the price,

And I'm afraid I had to rake 'em 'cross the coals.

Yes I've had the odd complaint,

But on the whole, I've been a saint,

To those poor, unfortunate souls!

But hey, like I said before, it's really not even worth thinking about. Think instead about the new job we just offered you. The pay is great, and the perks are to die for. There's a chef onsite, and you can have your dry cleaning done with no surcharge. You don't want to be unemployed, do you? You don't want to pass up your chance to work at such a competitive, destination employer that cares so much about its employees, do you? Remember, this offer letter doesn't last forever, so make your choice. All we're asking is that you give your full creative energies to us at all times!

You poor, unfortunate soul,

It's sad, but true,

If you want to get ahead, my sweet,

You've got to pay the toll,

Take a gulp and take a breath,
And go ahead and sign the scroll,
Flotsam, Jetsam, now I've got her boys,
The boss is on a roll!
This poor, unfortunate soul!
Mwwwwaahaahaaaa!

But you're not desperate. As a knowledge worker and programmer, you absolutely have the upper hand. Ours has been and remains an extreme employee's market, no matter what anyone tells you.

Do not accept roles with companies that play games like this. David Boike talked about an employer that told him he couldn't moonlight. "That was first blood," he said, referring to his eventual departure. I had a similar experience once, being forced to sign a non-compete on my first day—after I'd quit my other job and made future plans based on this new employment—rather than as part of the paperwork discussed upon being offered the job.

This sort of thing is a sign of bad faith. Make no mistake. These policies are not designed to "protect" the company. They're rent-seeking strong-arm tactics designed to discourage you from exercising your options. They mean to keep you employed. "Oh, thinking of leaving? That's fine if you want to owe us \$10,000. Oh, thinking of having a commercial life outside day company? Nah, we own you."

When you're contemplating new jobs, ask about this up front. Tell the recruiter or hiring manager that any non-compete of this nature constitutes an instant deal breaker for you, so it's better to figure it out sooner than later. Two things tend to happen. First, you avoid wasting your time with large bureaucracies. Second, some firms will actually modify the agreement or waive your signing of it.

If you're already at a company and under the dubious restriction of such an arrangement, start looking for a new job. Tuition clawback arrangements are usually borderline unenforceable, at least in the US (go figure—courts tend not to like indentured servitude), so you needn't fear much. If you're under a draconian non-compete, ask them to let you out of it in parallel with your job search. If they agree, great. If not, you can explain during the exit interview that there's no hard feelings but you want your freedom.

Under no circumstances should you sign your life away like this, nor should you put up with it already having been done to you. We have far too much leverage in our line of work. And if you're looking to take reasonable but real steps toward developer opportunism, you need to be freed up for the hustle.

I will offer one closing note of moderation to this point of view. In her interview with me, Sally Lehman mentioned that she didn't think it was unreasonable for companies to put this restriction on you, *assuming you're well compensated*. I can concede that point. If you know what you're giving up and you negotiate accordingly, this may make sense for you. But I nonetheless think that it's bad for the industry as a whole, and it's a relatively short-term consideration in either case. As John Sonmez puts it, when you work for someone else, you're building their empire instead of yours. When you sign an agreement like this, you agree *never* to build your own empire.

What you can do now: stay away from big companies

Another similar policy that I'd offer is to stay away from big companies, as a software developer. To some extent this might be unfair, but I'm talking law of averages here. The larger the company, the necessarily thicker you'll find the idealist layer. But that's not really even the worst of it, since you're hopefully not looking to play the ethically questionable opportunist game after reading this.

The problem is that companies have only ever managed to bloat beyond a certain headcount using the traditional pyramid model. At best, they just divide themselves into a bunch of smaller pyramids. If you want to escape the pathological Taylor concept, you have a few options: small, startup-like firms; smallish custom app dev shops; non-traditional outfits like GitHub and Zappos; and free agency. And you're not going to find any of those (by and large) on the Fortune 500 list.

If you work at Juggernaut Inc. now or if you're in a fluid state, scoping out the next thing, don't go the traditional route. In fact, you *can't* go the traditional route if you're seeking the non-traditional. When looking for a different culinary experience, don't head down to the local strip mall and hunt down the McDonald's.

Also of note, big companies limit the amount of internal efficiencer practice you can really get. Burdened by massive bureaucracies, it becomes harder for anyone there to sign off on an improvement initiative that you pitch.

Like avoiding the non-compete, this speaks to improving our collective move toward developer hegemony as much as anything else. The future of software development does not lay inside of companies like this but rather inside of efficiencer firms (and realistically, custom app dev shops) that sell to them.

What you can do now: politely end interviews that involve trivia

Another relatively low impact way you can make a difference comes when you face the prospect of job interviews. Refuse to put up with journeyman idealist garbage in the form of the trivia interview. (You should also refuse to put up with regular idealist garbage, which generally takes the form of different company-culture-focused inane questions, but that's not specific to our cause.) To put yourself in the right frame of mind, imagine how a true efficiencer would react to being peppered with minutiae about some programming language.

He would think, “Yeah, that’s cute, but can I talk to your boss so we can get to how this project impacts the bottom line?”

This means not participating in interviews during which you’d be quizzed on things like “List four namespaces in the system library” or “Describe the builder pattern.” I’m not sure what people think they’re learning about your ability to help a company make money by asking you to produce knowledge that could be acquired in four seconds with a search engine. If you want to really go out in a blaze of glory here, pull out your phone and say, “Okay Google, describe the builder pattern.” You get bonus points if you “okay Google” the question during an interview with Google. Just say you’re interviewing *them* by way of seeing if their products are up to your high standards.

Thwarting the journeyman idealist also means saying no to algorithm/whiteboard interviews. Unlike trivia interviews, which aim to see what you’ve memorized, whiteboard interviews are meant to see how you think. Or so the story goes. Really, it’s just an analog of fraternity hazing. They want to see if, to earn your stripes, you’ll jump through the same hoops that they had to jump through.

This is a human cognitive bias known as [effort justification](#), wherein your value of the “in” crowd and its selection process goes up substantially in proportion to the barriers to entry. Here’s an example. If I ask you to eat a flavorless gelatin, you’ll turn up your nose. But if I ask you to whiteboard a doubly linked list in order to qualify to eat that same dessert, you’ll eat it and say, you know, it’s actually pretty good, and people really should have to do a whiteboard exercise to earn it.

Don’t drift into this trap. Whiteboarding things or solving problems using commodity algorithms has no bearing on your ability to do a programming job unless the job involves going back in time to when Quicksort wasn’t a thing. If this type of interview really worked, why wouldn’t companies just administer IQ tests? At least that test is *actually* designed to see how you think.

Now, snark aside, I don't propose that you raise a fuss in the middle of an interview or get up and leave in a huff. I'm saying that you should politely but firmly refuse to participate.

The first step is to ask up front. If a recruiter from Best 'n' Brightest Programmers Inc. gives you a call and invites you to interview, reply that you're definitely interested, but that you'd like a bit more information about what the interview covers. Explain that you have a policy to not participate in trivia or algorithm interviews. Nine times out of ten, that will stop the conversation right there. "Welp, the process is the process," they'll say.

If you nonetheless find yourself being grilled by the Alex Trebek of journeyman idealism during a phone screen or interview, you can stop the process without being impolite. Apologize heavily, but stand your ground. "Oh, I think we've gotten our wires crossed somewhere. I've participated in the hiring process on your end plenty of times and have had some pretty bad luck with this style of interview, to the point where I just have a policy not to participate in it on either side. So I don't want to waste your time or mine by going any further. If you want, I can reach out to some people that might be a better fit."

The reason I recommend this route and not a combative one is relatively simple. The combative route smells like sour grapes. This one is a very polite statement about your opinion of the way they do things. You're saying, "Oh, I don't really want to work here anymore because your interview process is dumb. But I probably know some dummies that are more your speed." That advances our cause and gets us away from ludicrous hiring practices.

When you do something like this, you might feel self-conscious, worrying whether they think you're deflecting from an inability to answer the question. This might hold doubly true if, in fact, you cannot answer the question. But in either case, resist this impulse. Keep focused on the broader goal of getting away from this practice and finding work somewhere that doesn't demand entry as a supplicant subordinate. Answering these questions has no long term upside, so don't bother.

For your purposes, this polite refusal at least moves you away from gigs with two layers of idealists. At worst, you'll have regular idealists to contend with, and that's a step in the right direction. Ideally the organization where you land will have a process of finding talent that's saner than the traditional job interview. Maybe you'll find yourself discussing whether you and the prospective organization can make money together, like efficiencers.

What you can do now: realize that the tech giants aren't that great

This piece of advice builds on the last two somewhat. The tech giants definitely fall under the umbrella of "big companies," but I feel that they bear special mentioning because we humans tend toward the exceptionalism fallacy (as in, "Oh, I like Gigantech Inc., so Erik must mean all of the big companies except them.") Also, those companies seem to love journeyman idealist interviews.

But let me work my way back to that. Do you remember contemplating when you were younger what it meant to go to an institution like, say, Harvard? As kids thinking about secondary education, we'd look at that and think, "Wow, if I can get in there, I can write my ticket anywhere." And there's a decent chance we were right.

Then, around when we were in college or maybe a bit afterward, a new entity entered our consciousness. We'd look at Microsoft and Google and think, "Wow, if I can get in there, I can write my ticket anywhere." Start with Ivy League, head for Big Tech, and then the world is your oyster. You probably won't even have to interview at places after that—you can just wander in, pick out a cubicle, and get to work.

That thinking framed my career outlook, and I'd imagine it describes some of yours as well. Plus there's an [ant-instead-of-grasshopper](#) thinking to it. Prove yourself early and enjoy cred and stability later in life. Or something like that.

But the last two decades have shot all sorts of holes in that thinking. If you want a programming job these days, you don't need Harvard (or MIT or CMU or Stanford) and Microsoft. You don't even need college or a prior software development job. Demand is such that doing some crazy stuff with Excel macros can get you a steady job writing code.

Also, "write your ticket anywhere" means a lot less against a backdrop of serial job hopping. Software developers come and go from jobs like booth attendees at a trade show. Searching for jobs no longer scares the pants off of people—it's just what you do every year or two to get a raise. There's less reason than ever to go into an interview as an experienced developer wanting to stack the deck in your favor down to the last detail. If DiscerningCorp only wants Microsoft, Google, and Facebook alumni, you can just wander into the building next door and get a job there instead.

And finally, realize that these big tech companies are not the blue chippers of old. A job at Yahoo would have been a pretty sweet plum when I graduated college. But if that were on my resume now, I'd be explaining to people, "Yeah, but I was there when it was impressive to be there! No, seriously, that was a thing once. And I also wrote client side stuff for Myspace after that!"

Hopefully, we can now dispense with the "write your ticket" mystique around working for these places. Consider that, ironically enough, getting an offer from Google would mean that you didn't need to take a job with Google. After all, you've just demonstrated to yourself that you can secure an offer from the most selective place around, theoretically. You can *already* write your ticket.

This leaves only one (important) argument for taking a tech giant job: you want to work there. If the work, the culture, and the people appeal to you, far be it from me to try to stop you. Those are legit reasons to go do something, and I wouldn't deign to say there's anything wrong with that. But you won't be advancing developer hegemony, which is the impetus for the advice I offer here. If you go

under that mandate, you are embracing the life of the pragmatist, for better or worse.

Even at a large software product company, you're essentially a cog in a very large machine. Business strategy is handled elsewhere. You probably get more respect for your technical chops alone than you would at most places, but you'll still hit the glass ceiling of "That's nice, geek. Now let the executives talk." The efficiencer calling is not to be had here.

What you can do now: find a job that gets you out there

At this point, I think I've said enough about the companies that you should probably avoid. You don't want to go places that restrict you from conducting your own affairs, that bury you below layers of journeyman and regular idealists, and that aim to make you devote your life to the company. All of these things prevent you from advancing your own interests and speaking at the strategic level.

Let's turn our focus instead to the sorts of companies that you *should* work for. And I can sum that up succinctly, echoing the advice offered by David Boike. Find a company that lets you get your name out there and raise your own profile.

What does this mean, exactly? I'll perhaps have an easier time offering examples at opposite ends of the spectrum. At the inhibiting end, the one that you should avoid, resides SecretCo Inc., who asks you to toil away in anonymity. Under no circumstances can you ever show anyone outside of the company examples of code you've written. Heck, their restrictive non-disclosure agreements with employees barely let you admit to *working* there. Your service to that company is entirely opaque to external parties. And your interaction with outside entities? Non-existent. At the end of working there, all you'll have to show for it is the text on your resume that you must leave suitably vague. This is what you don't want. *They* control *your* narrative.

Contrast this with working for an app dev shop, a consultancy, or a developer tools/software company. These organizations pay you to go out and publicly interact with other companies. As a consultant, you move from organization to organization, helping them solve problems and building quite a network. If you work for a company that makes and sells software, you help customers solve their technical problems. In those cases, you build a network of people that will vouch for you. In essence, you build your brand. And best of all, you do it on someone else's dime.

As you apply for your next job, looking to escape Gigantech and its draconian NDAs and non-competes, actively look for a place where you can start making a name for yourself. No matter what the future looks like for us and for you, you're going to need a reputation and a network. If someone will pay you to start building that, then you've effectively worked out a deal where you get paid to market yourself. If you ever go solo or run a small business, you will appreciate how valuable that is.

What you can do now: apply at non-traditional companies

Earlier, I mentioned some companies that have non-traditional arrangements in terms of how they interact with their employees. These are companies who have found a way not to make their org charts look like predictable pyramids or who have found ways to more partner with people than employ them. In this list, I include the aforementioned GitHub, Particular, and Pillar. But there are plenty of others out there as well. Zappos is famous for its [holocracy](#), and I'd imagine that there are others that are less famous but no less interesting. Go check these companies out.

Don't confuse this advice with me suggesting that you just go work anywhere unusual. That's not what I'm saying; novelty doesn't always equal innovation. Rather, I'm suggesting that you find organizations that don't involve the pragmatist-idealist-opportunist

pyramid. Look for organizations willing to embrace you as an efficiencer and a business partner.

If you work at organizations like these, you'll have a lot more control over your own destiny and you'll be free to market yourself, raise your profile, and define your own version of developer hegemony.

What you can do now: apply at smaller app dev shops

The last type of place at which I'll encourage you to go work is the small app dev shop. Ideally, you should find one run by a software developer or a recently-former software developer. If the owner used to kinda write code once like twenty years ago, that's not the same thing. Proceed with relative caution, since that person probably views you as a one-dimensional geek who hasn't managed to escape the delivery trap.

At smaller app dev shops, you're likely to be client facing, and you're likely to *matter* to the organization. This gives you some leverage and the ability to act like a partner and to speak up about issues beyond code. At organizations like this, it is relatively easy to establish yourself as an efficiencer.

I would also add the caveat that you want to look for a place that doesn't intend to grow by turning today's line level contractors into tomorrow's pure managers. That organization is just going to mushroom into a pyramid shop with you in the idealist layer. Make sure that you'd be keeping your finger on the true pulse of automation.

What you can do now: start working from home

Those of you familiar with *The 4-Hour Workweek* find yourselves nodding along to the title of this section, no doubt. I can't and won't dispute any of Tim Ferriss's wisdom on this subject, notwithstanding the unique subject of lifestyle design.

Tim Ferriss advocates a work-from-home arrangement in large part so that you can travel where you want, when you want, without your job holding you back. This is great. I've personally lived this reality, spending the entirety of last winter in the southern part of the United States for the specific purpose of avoiding winter. But I'm not recommending the work-from-home arrangement for this purpose. Instead, I'll talk about some more practical concerns for someone looking to become more autonomous and independent.

First things first: you're working toward autonomy, and working from home, by its very nature, grants you more of that. It starts to remove the "butts in seats" attitude of employers, at least as it concerns you. They're less likely to evaluate your performance the same way they would the guy who takes orders at a pizza place. When presence melts away as an evaluation criterion, they get closer to reasoning about the value of your contributions. All of this has the effect of raising your profile in the eyes of those you work with—provided you still manage others' perception of your contributions, of course.

Next up comes the productivity consideration. Being present at the office from nine to five essentially gives you a carte blanche to waste as much time as you can reasonably get away with. This isn't to say that people come to the office thinking, "Let's see how much time I can spend in the break room before someone yells at me." Rather, it's that little accountability exists for good usage of time since ipso facto anything you do while at the office during those hours is construed as work, productive or not. This results in the average worker spending a depressingly small fraction of the day at productive work. This probably includes you.

Think of Peter in the movie *Office Space*, describing how, each day, he does maybe fifteen minutes of actual work. This may represent a slight caricature of your own life, but does it miss by a lot? How many hours do you spend coding, in a state of flow? How many hours do you spend hanging around a friend's desk, visiting the break room, attending pointless meetings, going to lunch, having "strategy" discussions that devolve into gossip, and going outside to

get steps on your Fitbit (or to smoke)? If you abandon the notion that any time spent at the office represents valuable time, how many hours of your day could actually be construed as valuable? How many hours would you feel good claiming as “work” if you were doing them from home?

When you shudder a bit and do some soul searching, I suspect that you’ll come to a realization. You likely spend two to four hours of your day doing something productive. The rest of the time, you content yourself with collecting a paycheck simply for presence. Hey, it’s not like you’d spend time at work if you had a choice, so you oughta get paid regardless of what you do there, amirite?

Let’s get back to working at home. After years of feeling good about two hours of productivity and six hours of presence, you wake up at 7:00 AM, skip your forty-five-minute commute, and arrive at “work” (your home office) a little early at 7:45. Normally, you’d get in at 8:30 and gossip with Susan, your fellow developer, at the Keurig for twenty minutes or so. But your home office lacks Susan. Instead, you just get to work. And you work from 7:45 to 9:30 without any interruption, actually getting into something of a state of flow. At 9:30, you realize you need to dial in for a daily status meeting, and you’re amazed at the nearly two hours of completed work. Normally, your parlay with Susan would have taken you until about 8:50, at which point you would have rearranged your Outlook folders a bit, since there’d be no point getting to absorbed between 8:50 and 9:30.

And so it goes. This is a fraction of your work from home day. But it’s a representative fraction. By 9:30, you’ve accomplished almost as much as you would in a normal *day* of presence, with no one to distract you and no “points for participation” time wasters. Tim Ferriss posits that you can do your forty-hour-per-week job in five to ten hours per week, and I feel inclined to agree, based on experience. If you pack up your salaried job and take it home, you’ll save yourself the commute and an amount of fluff that would shock you.

But I'll take it a step beyond Ferriss and his lifestyle design. Go home to build your business. If you can negotiate work from home, you'll free up the fluff hours and establish some cachet. Even if you spent your time the exact same way from nine to five, at least you'd get a half an hour to an hour per day freed up from commuting. But I promise you, you'll get way more. You'll get the same pay to do the same thing in a lot less time, freeing you to pursue developer opportunist ambitions. And on top of that, the people in your office will start to assume that you bring a lot to the table to have a special arrangement.

How should you secure such a sweet deal? As in-demand as software developers are, you probably just need to ask. Tell the boss that something has changed in your personal life and that you really don't want to leave but your hands might be tied. If they consider you even somewhat of a decent performer, the conversation will likely end here with a grudging, "Ugh, okay." If not, you could always take Tim Ferriss's advice and build a business case, asking to try it one day per week for a trial period and see if productivity doesn't improve. If that strategy's not for you, just go out searching for a remote job. They're everywhere.

This arrangement gives you back a lot of your time, and it acclimates you to thinking in terms of value rather than mortgaging most of your waking hours for someone to pay your bills. Thinking of and selling value is what will allow you to start selling as an efficiencer and claiming your own autonomy.

What you can do now: start a blog

Let's switch gears with a piece of advice that is succinct and, if implemented, will create little disruption in your life. Go start yourself a blog.

Don't get caught up in details. They're just ways to procrastinate. Should you host it yourself, use a hosting provider, or use WordPress? Should you build it by hand or use a CMS? Will people think less of you if it doesn't somehow involve GitHub? Stop it.

Seriously. Recognize this as the bike shedding that it is. You're obsessing over shop because shop is familiar and comforting. Get out of your comfort zone.

Do the easiest, lowest friction thing. If you go to wordpress.com, you can have a blog going in literally three minutes. You can sort everything else out later, including figuring out a better name. Just get some momentum now.

The biggest barrier to blogging success is stalled momentum. No doubt you've stored up a few rants and a handful of how-to posts over the years. But once you expend that initial store of ideas, it can be hard to keep going. To combat this, I suggest limiting your initial cadence. Decide to do a post per week. If you're inspired early on and want to do more than that, then queue them up instead of sending them live.

To have a blog that actually helps you requires two things: actually starting it and maintaining cadence. Just dive right in with the former, and have a plan for the latter. You can always iterate and improve as you go, but you can't go back in time and have started a blog years ago.

Don't expect blogging to pay off right away. This is a long play designed to open doors and opportunities over the course of time. But eventually, when you've successfully positioned and marketed yourself, you'll be happy you started one.

What you can do now: start the side hustle. Like, now.

To build on the idea of your blog helping with marketing, I'd also advise you to start some kind of side hustling venture. This may go hand in hand with your blog, or it may be an entirely separate affair. The idea here is for you to start to own and understand all facets of business. And recall my piece of advice about draconian non-

competes—you don't want to do this if you've signed your life away saying that you wouldn't.

I've saved this piece of advice for last because I consider it perhaps the single most important thing that you can do. Take a product or productized service, build it entirely end-to-end, and sell it. I cannot understate how educational this is for grokking all aspects of a business and putting yourself in a position to call the shots. By doing this, you are graduating as an efficiencer in a way that even the playing-an-effiencer-inside-your-company tactic won't allow. You have built a business.

This will teach you from the ground up about marketing, sales, finance, and operations. You already know tech, and when you add a working knowledge of these other elements of business to the tech, you become CEO material. It frees you to choose the elements of your business that you want to retain and those that you want to delegate.

Build something small. Maybe it's an online course, an e-book, or a little app that you sell in an app store. Just pick something and ship it. The amount of education that happens in that alone would amaze you. You'll need to consider what sort of business entity to form and set up a way of dealing with your expenses and revenue. You'll need to figure out how to market what you've done—how to get it in front of interested parties. On top of that, you might find yourself making sales pitches. The list goes on. Not all of that is the deeply technical work that we're used to doing, but it's complete, informative work.

If your side hustle project turns into something major, then great. You've cashed in a big score on your first try, which would put you in extremely rarified air. For most of the rest of us, that first end-to-end thing will range somewhere between total flop and making enough revenue that your time investment was worth \$4 an hour. You'll build something that no one ever sees or buys. You'll attempt a product launch and get it all wrong. I can't count the ways that you can (and I have) screwed something like this up.

The point isn't to do well enough to quit your day job. Not yet. The point is to make mistakes and learn from them. It's to develop an understanding of the world of business that only experience can teach you. It's to establish yourself as an efficiencer before you take any risks with full-time work.

I'll close out the chapter by mentioning one final thing about this. Even if you never make it on your own as a solopreneur, founder, freelancer, or efficiencer, this still has a ton of value. Earning promotions and carving out territory in even traditional workplaces becomes astonishingly easier when you know the ins and outs of running a business. You can speak to pretty much everyone at any company on their terms and in their language, at least to some extent. And that is worth its weight in gold.

Chapter 50: Full Circle—The Fate of Pragmatists, Idealists, and Opportunists

I just finished giving you advice that I would go back in time and give myself. Given that I left a slam dunk corporate career arc for the uncertain world of free agency, you might believe me a risk taker. I assure you this is not the case. In reality, I am pretty fiscally conservative and risk averse. I like to win, but I don't like to gamble.

For that reason, I took a path recommended explicitly by John Sonmez and referenced by some others. Before taking the plunge to go off on my own, I worked tirelessly to make everything line up just so. I marketed myself to an audience, spent years moonlighting, established expertise and extensive context, and even lined up serious work ahead of time.

I now likewise recommend the same to you. I feel like the tangible tips that I've given you for moving toward the world of efficiencers and autonomy is the equivalent of moving you toward a ziplining expedition hundreds of feet above a picturesque jungle in some warm country. Together, we're checking and double checking your safety equipment, making sure you understand protocol and procedure, verifying that you are not pregnant and have no heart condition, and inching you carefully toward the release point. And then, once an incredible amount of preparation has happened, whoosh, you're off taking what seems like a risky plunge but is actually carefully choreographed.

I've been giving you advice on how to stop being a pragmatist (or idealist) and start being an opportunist. I offered a rather grim play book for becoming an opportunist and dominating the pyramid-shaped corporation. I view this as grim because of both the malevolent paternalism of the corporate structure and because it means you need to stop programming. Now, I've offered a different,

more uplifting path to opportunism: the efficiencer route. Regain autonomy and dignity by becoming an automation expert instead of a JavaScript geek.

In both cases, I exhort exodus from the bottom layer in an up-or-out movement. Picture an Egyptian pyramid. I'm counseling each grain of sand at the base to move toward the top of the pyramid or to move away from it. This will leave in its wake crumbling pyramids and liberated grains of sand.

Let's keep going with this Kansas "Dust in the Wind" vibe and get even more philosophical. I want to tell a brief story about the history of automation—one that stretches further back than you've probably considered.

Think back to the beginning of the Industrial Revolution. The means-of-production-owning capitalists of the Industrial Revolution—the opportunists of that age—were, in a sense, the very first efficiencers. Prior to the Industrial Revolution, individual artisans competed in simple, localized markets on the basis of cost and quality of workmanship. Most likely, they eked out their livings by having better quality products or else selling to more price-sensitive customers at cheaper prices. Their individual efficiency might vary a bit, and that might make a bit of difference in their lives.

Then along came the Industrial Revolution, which allowed opportunists of means to completely obliterate the artisan markets. By leveraging significant resources, they were able to use mass production to produce goods of predictable, decent quality. These goods were so cheap that they drove the mom and pop operations completely out of business. Think of it as an early predecessor to the "Walmart vs the Little Guy" paradigm.

The capitalist merchants achieved this outcome through automation and efficiency. They created systems of human laborers. Actually, let's rephrase that. They *programmed* systems of human laborers, who became modern day pragmatists.

After a century or so of this dynamic, along came Taylor to introduce idealists. “Don’t worry about dealing with the laboring man-beasts in your employ,” Taylor told the owner opportunists. “You need a manager layer of people who specialize in doing that exact thing. They’ll handle it for you so you needn’t concern yourselves.” The programming of systems was thus delegated to the idealist layer around the time of Taylor, with owners and opportunists washing their hands of the pursuit. Efficiency became the province of middle management.

That continued for a bit more than half of a century, until the computerization of the workplace and the advent of industrial engineers at the line level came about. Idealists began to bring in engineers and programmers not only to produce output but also to increase internal efficiency. The idealists began to delegate care for internal systems to the laboring pragmatists. Efficiency became the province of line-level engineers and programmers.

In the years since then, the opportunist layer of organizations has long since gotten out of the efficiency business, except as consumers. Same goes for the idealist middle management layer. Oh, they understand the broader notion that if they stopped paying people to do data entry, they’d save money. But they’re completely out of touch with the implementation.

At the behest of the upper layers of the organization, we programmers have been dutifully automating legions of pragmatists out of jobs. The aforementioned data entry jobs are ephemeral and increasingly rare. Many factory jobs and manual labor tasks have gone the way of the dodo. And in coming years, more pragmatist jobs will follow suit: driving trucks, waiting tables, working as cashiers, and many others.

But a notable thing has simultaneously happened. During the days of the Industrial Revolution, only a handful of already well-heeled capitalists could bring the means of production to bear. It cost a lot of money to assemble pig iron and copper, and it cost additional money to assemble the system of humans required to turn those resources

into profit. Only a select few in the organization could create efficiency. And efficiency is perhaps the single most valuable commodity in the modern world, considering time is the nonrenewable resource for which all of us would go to the ends of the earth to get more.

The opportunists delegated efficiency creation to idealists, who in turn delegated it to pragmatists. Along with all of that delegation came the huge, unintentional windfall of also passing the means of efficiency production. Today, we live in a world where the pragmatist engineers, developers, and designers create all of the efficiency and have all of the means of production for doing so. This, in turn, means that *we have all the leverage*. Let me now state the implication that generates in no uncertain terms.

We have absolutely no need for owners and managers, for traditional opportunists and idealists.

And we're just starting to figure that out.

So what becomes of these three codependent archetypes and of the pyramid organization itself? Let's return to the metaphor of an Egyptian pyramid, where the base begins to conduct an exodus up and out. The pragmatists themselves simply leave. This creates a situation where savvy opportunists also leave for greener pastures, perceiving a degenerating situation.

I said before that the end of a company's life wasn't especially interesting. Indeed, only the idealists (and pragmatists with either no options or no sense) stick around for the inevitable bankruptcy, buyout, or closing of the doors. They imagine themselves as ship captains in a world overseen by a benevolent higher power. That power will intercede to reward their faith and loyalty or else the captain go down with the ship, dutiful to the end. Sad, but as I said, not especially interesting.

What is a bit more interesting is that the idealist condition is not possible without a company to overpromote based on seniority and

enthusiasm. Since idealists, following the collapse of the company, do not literally die, they are collected, commercially reconstituted and spat back into the workforce. This will produce at least cynicism, if not wisdom. The erstwhile idealist may be hesitant to ever love again because it just hurts too much, man. In fact, if it hurts enough, the idealist becomes an opportunist. If not, he repeats entry as a pragmatist to be groomed for idealism.

Idealists cannot exist without three essential components: faith in the wisdom of the corporate entity, pragmatists to compare themselves to in order to manufacture meritocracy narratives, and opportunists to manipulate their naivety. If we build a true efficiencer movement, we turn legions of pragmatists into opportunists. The idealist, then, just sort of fades into the background of history. The pragmatists exit, making overpromotion moot. The opportunists, many of whom are former-pragmatist efficiencers, recognize a more efficient path to ownership than pyramid climbing. They also exit. With the absence of the other two layers and the crumbling pyramid, the idealist faith will not last.

So, moving away from the philosophical, let's close out this chapter with what happens to these archetypes in real terms, considering them as people. In another nod to symmetry, I'll flip from talking about what corporations take from the archetypes, as I have for the duration of this book, and talk instead about what corporations provide to them. For pragmatists, organizations take hope but provide stability in exchange. From idealists, they take perspective but provide a feeling of significance. From opportunists, they take the ethical high ground but provide lower risk opportunity.

Right now, pragmatists and risk-averse opportunists alike are dissuaded from the free-agent/efficiencer route. Going off on your own means spending a *lot* of extra time on risk management and the hustle. It's like moving from a life of raising crops in a temperate climate to a life of nomadic hunting and foraging. But early efficiencers are blazing a trail.

The programmer community, for whatever flaws it may have, is wonderfully collaborative and instructive. We flood the world with language guides, framework tutorials, codecasts, books, and everything else you can imagine. We may get a little elitist, but we help one another.

The same thing is happening with the efficiencer playbook. We're branching out from the technical into the practical. People offer their stories as instruction: "Here's how I went from building WordPress sites for \$60 an hour to offering a productized service that nets me three-quarters of a million in revenue." Each one of us that flips the bozo bit on the pyramid corporation and heads off for the world of efficiencers makes it just a little easier for those contemplating the trip. And each one that contemplates the trip creates a little more market demand for existing efficiencers to service.

One of the biggest growth industries imaginable is going to be products and services that help make it easier to go efficiencer. This will include tutorials and guides, as well as service offerings to set up a corporate entity for you. But it will also evolve into more elaborate and complex offerings, such as single-payer benefit groups and risk-pooling mechanisms. What this looks like, I don't know exactly. But it's coming. The demand for lifestyle design is simply too high for it not to, when we're talking about a population segment with all of the leverage and the means of production.

And so at some point, a switch will flip. Enough people will go efficiencer that founding or joining a small, nimble efficiencer firm will be no more risky than pyramid corporation employment. And when that happens, there will be absolutely zero incentive for pragmatists to stay. The entire advantage of the corporation is tied up in risk mitigation, so when it no longer provides that, it will no longer have pragmatists. They will move to the world of efficiencer firms and partnership-oriented knowledge work offerings. Inasmuch as pragmatists in the future can continue to do manual labor, that will more and more frequently take the form of contractors and subcontractors.

Next, let's talk about idealists. Idealists tend to be hardworking and well intentioned. They simply get too caught up in the rituals and mythos of the pyramidal corporation and founder BS. This is hardly going to survive the pragmatist exodus toward autonomy and the fragmentation of today's juggernauts. People predisposed to this mix of flattery and illusion may be bamboozled in more egalitarian, value-exposing arrangements, but I don't think it's likely. I think they'll take the tendency to overperform and apply it in contexts that actually benefit them.

To get a bit more grounded, imagine a corporate idealist that you know: maybe a person that works sixty-hour weeks to impress a higher-up and believes that "junior vice president" is something earned through a combination of long hours at the office and regurgitating the company culture. This is a *driven* person, willing to work hard and chase the only key performance indicators they have at their disposal. In the context of a massive, lumbering corporation where an individual contributor's value is utterly opaque, they charge at what the organization tells them is valuable.

But in the context of a small efficiencer firm where an individual's value is obvious, can't you imagine them charging at something that actually matters? The overperforming idealist will make for a true high performer in a situation with rational cause-and-effect scenarios. These folks will incur some scars and come out the other side making pretty good partners, complimenting highly strategic opportunist types.

And what of the opportunist? Well, they'll be more or less the same, but without the ethical conundrums forced on them by corporate gamesmanship. They'll embody the simple, literal definition of opportunist and reserve the gamesmanship for advancing their own or their firm's cause.

Will there continue to be politics in efficiencer firms? Sure. Any time you have more than two humans in a room together, you have politics. And will self-interested opportunists continue to be able to take advantage of people willing to give up hope for security or

perspective for illusion? Of course. For better or for worse, that's part of human nature. But that doesn't mean we need to create a ubiquitous institution that normalizes it and raises it to a perverse art form while presenting a benevolent face to us.

We have created and nurtured a technology that democratizes opportunity. All we need to do now is throw off the shackles of an institution that resists that democratization. Pragmatists, idealists, and opportunists will all play their part.

Chapter 51: What This Looks Like, Long Term

Remember Emma from all the way back in Part 1? Let's think about the slice we saw of her life.

She fixed her small app dev firm's build, then boarded a plane to negotiate an extension of an existing contract. From there, she turned around and had a pre-sales meeting with a prospect firm before heading home to relax. She also delegated tasks intermittently that included project management (operations), accounting, and legal concerns. We didn't get to see her do any marketing, but four out of the five components that I mentioned isn't bad for a couple of days' work.

Emma the efficiencer either implemented or delegated in almost all phases of the business. Adding all of the phases might seem gratuitous, given that she was part of a partnership of unspecified quantity. Emma, you see, is not a CEO but a partner. It stands entirely to reason that Craig or one of the others might head up the marketing.

There is a discrepancy you may have noticed, though. Emma's firm doesn't offer a productized service or value billing, per se, both of which I suggest as excellent goals for aspiring efficiencers. Emma's firm bills for feature delivery, which is cost-based. Since their cost is basically time, this isn't too far removed from hourly billing. And their main area of expertise seems to be software development rather than solving a specific business problem. This is something her firm could work on changing as they moved toward being true problem solvers.

But recall the principles of efficiencer firms I outlined:

- They're bootstrapped/self-sufficient
- They're a partnership without employed pragmatists/idealists

- There's no scaling for scale's sake, via interviews and other silliness
- The value contributions of each partner can be measured
- Only opportunists are allowed

Emma's firm checks all of the boxes, imperfections and all. And I think that's powerful.

The reason I say "powerful" has to do with the way reality will unfold. No one is going to walk out of Huge Pyramid Inc., announce to the world, "I offer a productized service wherein I speed up your relational database by at least 50 percent," and partner up with three or four other people well suited to do the same while divvying up the work of running a business. It's going to be way, way messier than that.

Some of us will stumble and fail and try again. Others of us will strike off on our own, only to slide back into staff augmentation app dev when the bills come due. Some of us will start efficiencer firms and wind up hiring pragmatists and promoting idealists after all. Everything you can think of in between will happen, too.

App dev shops that convert specs into software using Gantt charts will stick around for a long time. So will giant tech companies with business hammocks, lots of cachet, and algorithm trivia interviews. I doubt we'll supplant the journeyman idealist layer in the industry any time soon, either. All of this is a long play.

But buried in the fits and starts will be success stories. Efficiencyers will emerge, and we'll head inexorably in that direction. Emma's firm thus makes a great example. They do a Scrummy form of app dev, but they talk to the business in terms of value and engage in negative sells when they think the project won't have ROI. They know and understand business.

They also hit every single principle of efficiencer firms. Emma's firm is self-sufficient: a partnership consisting only of opportunist partners and contractors to whom they delegate work. They scale revenue in

creative ways, taking in finder's fees and residual revenue streams rather than paying you to write bubble sort on a whiteboard with your opposite hand. They keep things lean enough that measuring each partner's contribution to the work is relatively easy. And they seek no grunts to toil away on their behalf.

You may wonder how Emma's firm formed. I don't know exactly, despite the fact that the characters are mine to imagine and control. But I can venture a guess as to what probably happened. One of the partners, let's say Emma, went off on her own and got some contract work. She used her knowledge of all facets of the business to parlay this into better, more strategic deals, eventually doing well enough that she landed gigs needing more than one person to help with automation.

At this point, she reached out to her network of friends and former colleagues, and she found some people who were game. And just like that, an efficiencer partnership was born. They drafted an operating agreement and set about dividing up responsibilities, refining their operation, and earning a living.

If we zoom out from Emma's firm and into generalities, what does all of this look like in the long term? How does developer hegemony take root and spread?

In the most general sense, it involves the steady flow of software developers who will leave organizations that regard them as cost centers and fungible commodities. Those organizations tend to believe you need two categories of people to implement software: business people who think, and grunts who, as James Grenning suggested, do low-level translation of natural language instructions into source code. And historically, we've proven them right to an extent. There are plenty of reasonably well-compensated programmers out there who content themselves with this golden coffin arrangement.

But here's the thing. That approach produces inferior software.

The agile software movement suggested that we break down the barriers between business folks and IT folks so they can work more effectively together. I say we reject that premise in its entirety and go forward believing that business folks and IT folks should be *the same people*. This is, of course, a disconcerting proposition for the business folks. Managers and former developers will need to come face-to-face with an uncomfortable question, and that's "What do you need me for, then?" My honest answer to that is, "I don't know. You'll probably figure something out."

As we begin to have automation experts—efficiencers—that understand both business and software development, we're creating a legitimate profession. And we're creating a profession that doesn't make sense to staff in-house. If you're a company that makes dishwashers, stick to making dishwashers. You know how to market and sell them, and you know how to keep the books. You're obviously going to need machinery and software capabilities. But remember, you make dishwashers. If you want to use the internet as a sales or distribution channel, does it make sense to hire several different types of specialized people to manage projects, write code, write tests, design "user experiences," and do "business analysis?" Or does it make more sense to call someone that specializes in automating sales and distribution of appliances to take care of it for you?

Over the long term, we will find our niches and realize our leverage. The number of dishwasher companies out there stumbling their way through massively inefficient implementations is staggering. Likewise staggering is the amount of money to be made by showing up, shaking your head, and saying, "What you're doing is ridiculous. Let me help." Right now, there's a whole transformation industry out there dedicated to quixotically helping them get better at it. The next wave industry will be the one that helps them realize there's a better division of labor.

Dishwashing companies will continue to employ software folks the same way that they employ a staff lawyer, if they're big enough. But

those staff members will become generalists that coordinate with efficiencer firms and figure out who specializes in what. The way these companies consume software and implement programs will become more distributed and decomposed, kind of like the way that microservices have replaced monoliths.

Every enterprise I've ever walked into has asked how they can scale agile. When companies try to do this, it usually involves a methodology named something that sounds comforting to an enterprise like "SAFE" or "LESS" or "MORPHINE." (I may have made that last one up.) When I'm asked how to scale agile, I simply answer, "You don't." You see, when you try to scale agile, it gets complex, process-heavy, and massively inefficient. My more nuanced answer is that you slice up the work into loosely-coupled autonomous chunks that don't require coordination, thus obviating the need to scale at all.

That is what I see happening in our industry, simply due to market forces. Companies that fail to adapt will be bested by companies that enlist efficiencer firms. Or they'll try a few times, fail, and reboot by dealing with efficiencer firms. As that goes on, efficiencer firms will learn to leverage targeted and well-marketed offerings: "Do you have the hardening sprint blues for your fourth consecutive mess of a quarter? Call us for simplification!"

If you look at a lot of larger successful tech product firms, they seem to succeed with similar loose coupling philosophies. As I understand it, departments or teams at Amazon, Google, and Microsoft all operate with a large degree of autonomy and independence. To a certain extent, you might think of those as organizations comprised of smart folks capable of forming excellent efficiencer firms if they weren't more content with pragmatist and journeyman idealist career scripts. But any way you look at it, the decomposed, targeted, automation-expert path is going to win.

Globalism and control of our means of production are here to stay in an increasingly digital, increasingly knowledge-work-driven economy. Humans will collaborate in corporate structures more reminiscent of

atoms assembling into molecules and decomposing than of the early-twentieth-century global conglomerates. In a world where communication and autonomy are easy, pyramid structures make little sense. We're ready for something new.

And that something will not need to rob its commercial participants of essential parts of their makeup. Efficiency firms and things that look like them will not steal anyone's hope, because the partners can always earn advancement in direct proportion to the measurable success of their efforts. Likewise, no one will be robbed of perspective, because that's simply impossible with structures simple enough to prevent a divorce of outcomes from actions, thus allowing narrative to replace measurement.

And opportunists, whose ethical high ground has been taken from them? I bet you expected me to say that wouldn't happen anymore, either. But as an opportunist, you can always have hope and perspective; the ethical high ground is yours to cede or claim on your own terms.

The pyramidal corporation *forces* you to choose. Cede the ethical high ground and advance, or keep it and stay where you are. The market for your labor doesn't force this choice on you, but neither does it save you from it. You may, for instance, need to pay your mortgage and decide to take on an hourly contract even though you know it will probably not help your client. It's ugly to have to make a choice like that, but I'm proposing a world in which at least you always have that choice to make.

In fact, I'm proposing we usher in new world entirely. It's time to look critically at the nature of the game and, like Neo facing agent bullets at the end of the first Matrix movie, say no and watch the bullets drop.

No, I will not work endless overtime for the same amount of pay. No, I won't accept that I can't enjoy making a living because "that's why they call it work." No, hiring authority, I do not need to be thankful to have a job. No, I will not accept 2 percent pay raises per year,

independent of how much value I add to a company. No, I shouldn't need to commute, dress up, and punch a clock to perform knowledge work.

I set before you a world where we're *all* opportunists. What's more, in this world, we're all *adults*, conducting business on our own terms. We're all knowledge workers, having replaced non-thinking work with automation. I'm proposing a world where those of us who trade and specialize in humanity's most valuable commodity are justly compensated. I'm proposing developer hegemony as the future of labor.

What Now?

If you've enjoyed this book and are interested in hearing more, I've set up [a landing page on my site](#). Check it out for next steps in how you can help yourself and others reach a state of developer hegemony. I will over the course of time be adding relevant content, courses, and general information on how to liberate yourself from outdated commercial models.

Appendix A

Below are the full interview transcripts for each of the developer opportunists I interviewed for Part 5 of the book. Minor editing has been applied, correcting only typos or misspellings.

David Boike

Q: First off, if you have any reactions/impressions to what I've said [outlining the message of this book], that's certainly welcome.

Seems to make sense to me. I certainly see a trend with my peers locally all going independent eventually, and once you do, you never go back. Those that stay in companies tend to be either not as skilled, or they stick there more due to inertia than anything else.

Q: Can you walk through your background a bit? How did you come to be doing something other than following the standard corporate software developer career arc? What made you decide to do something different?

Editorial note: I redacted some material for this answer that might have been overly specific about individuals or organizations. The entire response, verbatim, was not "on the record."

Well, none of this happened according to any sort of plan. Really it was a bunch of happy accidents.

I started at a small company in Clear Lake, Iowa, where I had interned while in college. I stayed there a year and left because I was getting married. It was small enough, though, that I knew everybody and the CEO was maybe three management levels away from me. I walked by his corner office several times per day, and if I wanted to have a conversation directly with him, that was totally possible and happened on more than one occasion.

I moved to ClearChannel Television, which used to be a thing and now is not. ClearChannel Television got sold off to a separate company, and we were the IT wing, independently branded as Inergize Digital, and we had customers outside of the ClearChannel/Nexstar stations. In any case, even though ClearChannel was huge, our department was physically separated and felt like a small company. The Senior VP was, for all intents and purposes, the CEO for all it mattered to me most of the time. He was two org chart levels from me, and just like the small company, I had direct conversations with him frequently.

Because we were building websites and other products for TV stations, magazines, newspapers, etc. we were dealing with a kind of scale I'd never encountered before. So that's where I learned about NServiceBus out of necessity, otherwise I would not have been able to build those systems. During that time I also started my blog, which is the single biggest reason I am at Particular today. I blogged about NServiceBus stuff quite a bit, and at a time when NServiceBus wasn't commercial and there wasn't a lot in the way of documentation, some of my blog posts became *de facto* documentation.

At some point, NServiceBus commercialized. Because of my blogging, they made me an NServiceBus Champion (think like Microsoft MVP except smaller scale) and then when NServiceBus commercialized, Udi [Dahan, founder of Particular] reached out to me to see if I'd want to help out with round-the-world support as a moonlighting sort of thing. I asked my boss and got turned down, basically because "they owned me" and they always had to come first.

There was a bench consulting firm in town (ILM) who sponsored the local .NET User Group and recruiters were commonly there saying if you'd like a salary review, swing on by. So one day I did. They said a lot of things I liked so I decided to jump ship the instant my contract was done. I had a newborn at the time so the pressure of going

independent at that point (basically millions of unknowns) was impossible, so a bench consulting firm was perfect.

I actually enjoyed that quite a bit. I got exposed to a bunch of different things, networked a lot, and never had to put up with any one client's corporate BS long enough to really tire of it before it was on to the next thing. I quickly became a principal consultant, headed the interview team, did more strategic things, etc. All good.

I was still an NServiceBus Champ, but I wasn't really doing much with it. But then Packt (my book's publisher) started hunting around for someone to write Learning NServiceBus by emailing all the champs. I had thought before I had the skills to do that but wasn't ever sure how to get started. So when I replied, I kind of started going down the rabbit hole. They asked for a table of contents. I whipped one up in about five minutes I was sure they would rip to shreds. Instead they replied back, "Great, can we send you a contract?"

After the book was published, Udi reached out. He thought that since I had written the book it would make sense for me to do trainings for them. Unlike my previous bosses, ILM was like, "great, let's do a partnership." So I did trainings out of the ILM office two to three times, as well as a few occasions where Particular would contract with ILM to have me go do a training at a customer site. Unfortunately, it was difficult for ILM to leverage that partnership in terms of NServiceBus projects, so I was still never really working on or with NServiceBus during my day job.

When Udi reached out about joining Particular, I wasn't even unhappy at ILM, but I wanted to do stuff with NServiceBus whether I worked for the company or not. It was kind of an offer I couldn't refuse.

Q: What does your day to day tend to look like in terms of the kind of work that you do?

It's definitely quite a bit different than it used to be. I used to, for the most part, sit down and code things for 8 hours per day. For Particular, I'm a member of a couple squads (a small group charged with co-managing an area of strategy, in lieu of company directors) and maintainer groups, so there's a lot more collaboration. And because we're a dispersed organization, that's all in GitHub or video conferences. So I'm in more "meetings" since there is no "turning around in your chair" to talk to people. But the good news is that because we're fully dispersed, with no home office, everyone is on the same playing field as far as communication goes. There's no possibility of missing a conversation that "accidentally" happens in the hallway because there are no hallways.

Many of the people I need to collaborate with are in Europe/Israel, and some in Australia. I don't have much crossover with Australia just because of geography, but I have about two to three hours of crossover with Europe. So most of my meetings tend to occur in the morning. A lot of times my mornings consist of triaging my inbox and accomplishing small tasks where I can, with some meetings interspersed, and then I reserve my afternoon for tasks that require more attention and focus.

Q: How would you categorize the nature of most of your work: Entrepreneurship? Training/coaching? Content creation? Consulting? App dev?

Because of my particular skill set, I have been spending far more time doing content creation and much less time deep in Visual Studio coding. When I am coding, I'm very infrequently banging out a bunch of greenfield code. Because we maintain a framework that people are depending upon, it's much more targeted, and the focus is definitely on quality over quantity.

I also do customer training and education, customer consulting (mostly remote in small chunks), customer support, and it feels like a million other things too. The most important thing I seem to need to do is to limit the things I'm doing.

Q: Specifically, do you spend a lot of your time programming?

No, not a ton. I run on a MacBook Pro with Parallels for Windows, basically just for Visual Studio. There are some days I don't crack open a Windows app.

Q: How do you balance your time between looking after business interests (overhead) and the work you do that offers value? Does the overhead ever seem burdensome?

Because of my arrangement with Particular, this isn't something that's really a problem. I don't have to market my own company or network or any of those things. I have to manage my own payroll and accounting stuff. But from the start, I got a good accountant/lawyer hybrid who helped me incorporate. I have ADP to handle my payroll. (My accountant told me "Get a service to manage that for you. The rules are complex, you'll screw it up eventually, and when you do, the fines are not fun.") I submit an invoice to Particular once a month and pay myself with ADP with a few clicks. It's really not much to manage every month. The biggest problem for me is remembering to send in estimated tax payments to the IRS at the right time.

Q: In what profit structure(s) do you typically make money (e.g., hourly consulting/contracting, fixed bid consulting/contracting, passive, royalties, dividends/salary from your company, etc.)?

The payment from Particular is wired into my business account once per month. I take about half of it as W2 income through ADP. The rest I transfer with my online banking as a dividends payment, probably similar to anyone else incorporated as an S-Corp.

Q: Do you think there's a ceiling, both in pay and organizational status, for software developers in the corporate world?

Probably. I think I was about there before I jumped. At least, there probably is for most developers that don't have a very specific combination of talents that is very necessary for some organizations.

For me I think it's coding + architecture + writing + teaching that puts me in a very small percentage of developers.

Q: There seems to be a common canard in corporate software development that programming is a game for the young and that a career-oriented person needs to stop it and become a manager. What do you recommend for someone that is ambitious but not willing to stop programming? How do you avoid the stigma that programming work is "line-level grunt work?" How do you suggest others avoid it?

Wow, that's a thinker.

I think normal salary ranges for developers and managers tends to back that up. Some big places (Microsoft comes to mind) have career advancement opportunities for the "hardcore nerds" who don't want to give up coding. But little places do not. So you get "quasi-coding" managers that won't give it up but are getting worse and worse at it every year. I think the stigmas are starting to reverse though. Maybe not in big, slow-moving, old-school corporations, but people are starting to figure out that becoming a manager isn't necessarily the best progression for everyone.

Uncle Bob had [something to say about this too](#)—the perception that all programmers were young and there were no old programmers. His take was that it's not that there are no old programmers. They're still around. It's just that the number of total software devs on the planet is doubling every five years, and that means that at any given time, half of all developers have less than five years of experience.

In any case, the best thing I think a programmer can do is keep your options open and always keep learning and making yourself better. Being a "grunt" isn't something you have to put up with forever because your skills are in demand, and there are companies starting to catch on to how developers should be treated.

Q: What advice do you have for readers of the book that want to get out of the nine to five programming game but continue to earn a

living as technologists?

Well, maybe management is right for you. I'm not sure I can offer any advice because that's not how I see myself. If you're stuck in a big corporate developer mill, you might not meet very many people or get very many new experiences. Going to a bench consulting firm can be a great way to learn a bunch of new things quickly and "accidentally" network just because of all the different places you're going. Also, if you don't have a blog, start one yesterday. Communication is such a premium skill amongst developers, and a blog will give you a way to practice that skill and market yourself at the same time.

Q: A lot of readers wanting to be independent might be worried that they're taking a big risk. What would you say to someone at this crossroads?

I can relate. I was there. But if you're a capable developer, you're in demand. There are more opportunities for people like you than there are people like you. I didn't realize before I joined the consulting firm how many recruiters there are out there looking for people to fill six-month contracts. Granted, I've never had to do it, but I don't think it would take very much effort to keep yourself busy. You're not going to get rich giving those recruiters their big fat finders fees, but you won't go hungry either. Of course, this is all assuming you're a competent developer. I've interviewed tons that aren't. You have to know yourself and be confident in your own abilities.

Q: On top of that, I'm curious about Particular's arrangement with people globally. As I understand it, don't you guys that work for Particular incorporate and then technically have a B2B relationship with Particular? I ask because I think that's actually really great (forces developers to learn skills necessary for being independent in a less risky environment), and it could potentially be a model for organizations in the future. Whatever you can tell me about this, I'd love to hear. Does your relationship with Particular look a lot like a standard employer-employee relationship in general, or is it looser than that?

Yep, I am CEO of David Boike Consulting, Inc. (I wish I'd picked a different name), and I have a contract with Particular. Technically, I guess they're my customer. I already talked about invoicing them monthly up above. The contract is pretty simple. I provide them with software development, they provide me with cash.

The people in Israel are direct employees of NServiceBus Ltd, and there's a few policies that apply differently to them because of Israeli law. Simple stuff, like they track sick time in a spreadsheet and the rest of us don't. Nothing major.

Particular did make it easier to incorporate, since I knew money would be coming. I didn't have to tell my wife "I'm going to go independent" and then have her ask "OK, who's your first gig going to be with?" and then answer "No idea." And now, if Particular would cease to exist tomorrow, since I'm already incorporated, going independent is exactly what I'd do. I'd leverage the skills I have as best as I could, look for those three-to-six month contracts, probably suck at it for awhile, and really hate having to drive to work again. But I'm confident I'd continue to feed my family. I'm also fairly confident I wouldn't have trouble getting another good more long-term gig if I decided that's what I wanted. It would be a matter of finding the right long-term gig, rather than, "Oh crap, I need a job yesterday."

James Grenning

Q: Can you walk through your background a bit? How did you come to be doing something other than following the standard corporate software developer career arc? What made you decide to do something different?

In high school and college, I tried to avoid computers. Punch cards, no windows in the computer room, "Stay away," I thought. Then I discovered that programming was fun. And someone would actually pay me to do it!

My first two “real” jobs were great. I was solving interesting problems and working with good people. I worked my way into more responsible positions as my career evolved, starting as an individual contributor, then recruiting and leading others, eventually managing a product engineering team including software and hardware, and finally a systems engineer working for the general manager of our organization exploring a new product idea. I learned a lot in my progression through the different jobs and responsibilities over my seventeen years in corporate America.

Several of my colleagues, when they got into management roles, let their technical skills go. They were managing, so they did not need to know about OO and C++. Later, they had to leave technology as they became stale and not so employable. As a manager, I still loved programming and did not want to lose it. It was probably annoying to the people that worked for me—I got into the details with them. I also had a side job doing some contract programming in C++. I wanted to still program, learn C++, and have some extra money. (It’s always good if an activity you do has multiple benefits.)

In my last corporate position, reporting to our GM (I learned a lot from him), I worked very hard; but there was no way to get ahead. Come review time, I found from my GM that I did not do enough and did not do it well enough. As he would say, “Let’s focus on the weakness.” I figured that I needed a different economic and enjoyment path. That moment got me back on track for a technology-focused future.

My friend Robert Martin (Uncle Bob) worked at this company. It was the interview with Bob fourteen years earlier that convinced me to join the company in the first place. Bob had left a few years earlier than I did. He joined a startup and later started consulting. He had contacted me a couple times to join him in consulting. I was not ready at first, but having learned what I could at the big company and starting to question my path, I was ready to take the risk. Luckily, my wife, Marilee, was supportive. It was a big risk, as we had three children and a mortgage.

I look back on this move of twenty years ago and I am very happy I made it.

Q: What does your day to day tend to look like in terms of the kind of work that you do?

Last week I was in Slovenia teaching TDD for Embedded C. This week I did my billing, returned emails and talked to some perspective customers. I also worked on my side project. Today, I'm replying to a request from a friend to answer some questions for his book. That is not unprecedented, but it's a little unusual. My day to day activities are not always my own, but here are some of the things that I do as an owner of a tiny consulting company.

My business focus is teaching test-driven development and design to embedded systems engineers. To support that business, there is of course the dreaded bookkeeping and accounting! I have an accountant, but I need to keep all the records to make his job possible. This is one area where I am not proactive enough. I don't like the work, though it is a necessary evil. Well, billing is kind of fun, especially when the payments come in. Sending checks to the government to pay taxes is horrible. So is sending money for health insurance. Keeping track of all the business expenses is tedious as well.

Some of the things I do to support the business are fun. I've been evolving my website to better support my business. I have learned my way around the Ruby on Rails web framework and evolved my website to be more than a place to drop my articles. It helps me deliver a better product to my clients and I get to scratch my programming itch learning Ruby and RoR.

I automate a lot of my boring and error prone manual processes. For example, if you fill out my contact form, I can generate various emails (services descriptions, requests for more information and pricing) that are consistent and professional. I used to copy/paste an old email and then edit it for the new client. Inevitably, when there is

copy/paste, I forget something. So my reply to the client looks careless, which is antithetical to my message of software quality.

I've evolved my website to support my training logistics. I once bought tickets for the wrong city because of a miscommunication with my client over email. Now my clients complete the logistics form on my website, giving the location of the training. I make reservations from the info they typed into my website. I can automatically forward that meeting room info to tripit.com. When I am running late on day one of a training week, I have the address on my phone, right where I need it.

I've evolved my website to support my training delivery. During one of my training courses, there are many web links that are needed, and they evolve during the course. I used to write them on the board. That works fine as long as I am in the room with people, though there are opportunities for error (me writing, them reading my writing, their typing the URL). I started delivering my training live via the web as well, so a whiteboard is not much of an option. Now I have a web page that I quickly update during the course, so attendees have the information and links they need just in time.

Then of course there is the delivery of my training. It could get boring, but it has not. I keep finding ways to improve and engage people. My website plays a role here, too, with pre-training surveys, in-training reactions to new ideas, and post-training feedback. All this information goes live on my website and lets me react to what people experience during topic presentations, demos, and exercises.

I also need to keep business coming my way. My book *Test-Driven Development for Embedded C* is my main marketing tool. Engineers continue to discover it. Potential clients contact me via my website, usually. I'll have a call with a new potential client to make sure I understand what they want and they understand what I deliver. Considering that TDD is a challenging and controversial topic, I need to make sure they are ready to learn about it, too. I had a bad experience once when I accepted a job where the client was not ready. The great-grand-boss of the team I was working with hired

me. It was not my usual engagement where the team was open-minded about the problems they had, and at least some wanted to fix them. They did not think they had any problems! Now I am very careful to make sure the team I am visiting is ready.

In my spare time I am exploring technology and building an IoT prototype for a product with my brother's company. I'll also write the occasional blog post or longer paper.

Q: How would you categorize the nature of most of your work: Entrepreneurship? Training/coaching? Content creation? Consulting? App dev?

I wear a lot of hats. Training and coaching delivery. Course development. Article and book writing, and, more recently, IT for my company and some R&D for the new product idea in the works.

Q: Specifically, do you spend a lot of your time programming?

For the first few years of my business (started in 2008), I only programmed with clients or wrote code for my training exercises. Since building my website, I have really gotten a chance to do a lot of valuable programming. I found that in Ruby and RoR, I did not know how to test drive. I have since figured out how to test drive in the most valuable areas of a RoR application.

The IoT project I am working on is giving me a lot of exposure to other areas. The UI is an Android app programmed in Java, the data collection point is a tiny Linux box where most programming for the IoT stuff is done in Python (another new language to learn). The sensors are also programmed in Python. I am figuring out how to write applications and tests in this challenging, asynchronous environment. I can hardly wait to finish writing this so I can get back to it.

The lessons of TDD have really helped me to learn how to get these wildly different environments to do what I want them to do. As the

core of TDD is establishing cause and effect. I use this core activity in every programming activity I do.

Q: How do you balance your time between looking after business interests (overhead) and the work you do that offers value? Does the overhead ever seem burdensome?

I really hate the record keeping. One minute of it is a burden. I put it off. By doing so, I forget how my accounting package works. When I get back to it, it is total drudgery. So now I try to get to it sooner, automating my way out, if possible. For the first six years of my business, I collected all my paper receipts. This was a pain and took up valuable file cabinet space. Since 2014, I have virtually no paper receipts. When I am handed a receipt, I scan it with my phone and name the scan after the client, with the purpose and dollar amount in the name. I save it to Dropbox. When my computer sees the new PDF appear in Dropbox, it date-stamps the file and moves it to my current receipts. Now the receipts are just where I need them, when I need them, if I need them.

Q: In what profit structure(s) do you typically make money (e.g., hourly consulting/contracting, fixed bid consulting/contracting, passive, royalties, dividends/salary from your company, etc.)?

Most my income is from training fees. I charge for the course, not for my time. Coaching is billed as a daily rate, usually. I bundle expenses so my client pays a flat rate and I don't have to mess with expense reporting and receipts in my billing. I get a quarterly royalty check for my book. That is always nice to get, but is not a lot of money.

Q: Why do you think there's such a ceiling, both in pay and organizational status, for software developers in the corporate world?

The corporate world does not know what programming is. They view it as labor (more hours equals more output, cheaper hour rate

means less cost), not knowledge work (problem solving takes time and some people are better at it than others).

Programmers do themselves no favors. Most programmers in my training courses program the way I did at my first job in 1979: Debug-Later Programming (<http://blog.wingman-sw.com/archives/16>). Lots has changed since then. There is a lot to learn. Programmers tend to overinflate their skills and worth. I know that feeling firsthand, like I knew all there was to know about programming when I was a young programmer. Now, I am overwhelmed by how much more there is to learn.

Programmers tend to get through the “app-titude” test (getting an app to work) and think there is nothing else to learn. There is a lot to learn. You do not have to learn it all, but you do need to continuously improve.

Q: There seems to be a common canard in corporate software development that programming is a game for the young and that a career-oriented person needs to stop it and become a manager. What do you recommend for someone that is ambitious but not willing to stop programming? How do you avoid the stigma that programming work is “line-level grunt work?” How do you suggest others avoid it?

I find my management experience invaluable in my work. I can relate and talk to people in all roles. I'd say do some time in management, but keep a hand in the technology. Program for fun and contribute to an open source project. Mentor the people you manage. Know your stuff so you have something to mentor them in.

As a programmer, you need to become a partner with the business people. Say you sign up to deliver some feature in three months and, after two and a half months, announce a delay and ask for an extra month. Then you deliver three weeks late from the revised date. Follow that with code with a bunch of bugs, and how's your reputation doing? Not so great, right?

Instead, what if you got good at producing code that worked and kept working when other changes are tossed at it? What if you could slice your work into small deliverables and deliver several of these pieces every two weeks? What if the business could count on you? If all those what ifs came to be, your reputation would grow and the business would listen to you. You'd be in partnership with the business.

Q: What advice do you have for readers of the book that want to get out of the nine to five programming game but continue to earn a living as technologists?

Read! Obviously, if you got this far, you are doing that. Write! Do you have some insight to offer or can help others solve problems? Get out to the meetups and find others that are trying to improve. Give a talk at a meetup. Give a talk at a conference. Follow and interact with people you respect on Twitter or whatever the social media app of your day is.

My most valuable skill as an engineer is problem solving. Your company may mostly reward you for learning and focus on their product, but you need to also know what the software industry is doing, what advances are being made.

Q: A lot of readers wanting to be independent might be worried that they're taking a big risk. What would you say to someone at this crossroads?

It is not for everybody. Going into business for yourself is a risk. Finding clients is difficult. I was lucky to be able to consult with Bob Martin, giving me time to develop my own reputation. In the process, I stumbled across an interesting niche and have been able to turn it into a nice business. I am very fortunate. "I am a great believer in luck, and I find the harder I work the more I have of it."—Thomas Jefferson

Matt Heusser

Q: First off, if you have any reactions/impressions to what I've said [outlining the message of this book], that's certainly welcome.

Let me start by agreeing that the way companies are structured is a bit silly.

I remember working at an insurance company that assigned a PM, technical PM, analyst, programmer, and a couple subject-matter-expert testers to a project at a time. Really we had six people to do the work on one, plus a lot of translation activities. What we were wanted to do would turn into a business case, which was turned into requirements, then technical requirements, then a Gantt chart, then weekly meetings. All that to do a couple of days of programming work. Except it didn't take a couple of days; it took weeks because the programmer had questions that weren't answered for a week, or the test environment wasn't up, or the programmer had to work on some other emergency. Of course, because of all the waiting, everyone took on additional projects—perhaps six projects each running at the same time. So answers to questions would take a week, so we'd have lots of extra time, so we'd take on more work, and so on.

At one point, around 2008, I wondered if we could just make it a more capitalistic system. There should be a big list of projects on the wall, and technical staff could build a team and propose, "This is my team, and we'll do [project name] in [period of time]." If they failed, we'd keep that in mind at annual review time.

Of course, this is a bit of a chicken and egg problem, as you have to wind down projects and spool up others, and what do you do if no one bids on a project?

The answer, of course, is to shift to freelancers as the company grows.

Q: Can you walk through your background a bit? How did you come to be doing something other than following the standard corporate

software developer career arc? What made you decide to do something different?

I worked from home in 2008 for a venture-funded company called Socialtext. Every three months we'd have a quarterly meeting to talk about sales and I would worry about my job. I survived twelve quarters of that and got pretty used to taking risks. When the job ended, I had a decent freelance portfolio and some savings; it seemed better to go a-contracting than get a job. That was 2011; I haven't been an employee since, though we are reforming my company as a S-Corporation in 2017 and I will be joining that.

Q: What does your day to day tend to look like in terms of the kind of work that you do?

It has really shifted. At this point, a typical business day is either working from home, consulting, or attending a conference. A from-home day means a few hours of billable work, a lot of coordinating resources, and an hour or two of marketing. By marketing I mean writing articles or reaching out to people, through various channels, to help them know about our services. I've also recently eliminated lunch in favor of a protein shake and workout at my local gym; that gets me out of the house. Plus there is creating invoices, waiting for the physical mail and emails to come with the checks, and running to the bank to deposit them. Then there's QuickBooks work, like marking checks as deposited, transferring a set-aside for taxes, and getting a list of past-due accounts and finding ways to remind them we need to get paid.

Q: How would you categorize the nature of most of your work: Entrepreneurship? Training/coaching? Content creation? Consulting? App dev?

"You can have anything you want at Excelon Development." We do software delivery services (mostly programming and testing), consulting, a little training, and a lot of writing. The consulting and training is mostly on lean software testing and delivery, helping companies continuously improve.

Q: Your reputation is in testing and quality. Do you spend a lot of your time programming or testing?

From 2011–2013, I was billing close to 2,000 hours a year. After 2013, we started to bring in contractors, and, regardless of what you've heard, "passive income" is not free money for no work. For every one contract we actually execute, I spend a lot of time going to meetings, drinking coffee, and chatting on Skype and the phone. Then, once we have the contract, we need to either extend it, find the contractor other work, or lose the income. This year I managed to have two part-time delivery contracts, from home, which I would like to keep.

Q: How do you balance your time between looking after business interests (overhead) and the work you do that offers value? Does the overhead ever seem burdensome?

Right now we have four and a half people working full time, including me, and we are looking to move to five. With that kind of size, we can have me spend a great deal of time working on the business instead of in it. Even though that is not always satisfying, it is right for our size. When I started out, for the first two years it was much more like 85 percent billable, 15 percent overhead. Two things to remember: that 85 percent billable was about forty-five hours a week, and the overhead was another six hours of work!

Q: In what profit structure(s) do you typically make money (e.g., hourly consulting/contracting, fixed bid consulting/contracting, passive, royalties, dividends/salary from your company, etc.)?

We're pretty mixed right now, and it changes month to month. In Q1–Q3 of 2016, it really was close between subcontracts, writing, and my own consulting and training. I think your audience is probably more interested in the early years, when consulting/contracting was more like 70 percent of revenue and writing 20 percent more. Training might have been an addition 10 percent.

Q: Why do you think there's such a ceiling, both in pay and organizational status, for software developers in the corporate world?

Programming is viewed as entry-level translation work. It is something to get promoted out of. Most companies want a journeyman programmer with three to five years of relevant experience so they won't have to pay to train them and then lose them. So there are few good training programs. Plus, the company doesn't want to pay for your twenty years of experience that's five of COBOL, five of C, five of early web development, and five of recent relevant responsive design.

That looks right on paper, but the result is that we have a pile of new people who are trained poorly, while kicking good people out around forty.

Q: There seems to be a common canard in corporate software development that programming is a game for the young and that a career-oriented person needs to stop it and become a manager. What do you recommend for someone that is ambitious but not willing to stop programming? How do you avoid the stigma that programming work is "line-level grunt work?" How do you suggest others avoid it?

If you find a niche, you can do well programming into your sixties—but that niche will limit you to legacy programming languages. That might be fine in a major metropolitan area, like Chicago, New York, Dallas, and so on: a place with, say, a lot of banks that are running old technology. It is, however, more than a little boring.

I'd suggest stepping back from technology and solving business problems. There was a programmer I knew on a message board who did fine writing VBA scripts in MS Access to make applications. He would come into the business leadership, not IT, and get them to agree to the project. It's crazy. All us programmers are thinking, "That's crazy, man, you are writing a mess of legacy code." But he

made a good living, kept his customers happy, got repeat business, and was thought of as a business problem-solver, not a “C# coder.”

Q: What advice do you have for readers of the book that want to get out of the nine to five programming game but continue to earn a living as technologists?

Get involved with local small businesses with technology problems. That means your chamber of commerce, that means the JayCees, that means the Rotary. Don't push it. Spend a year or two just grabbing coffee with people and talking about stuff. Figure out the common problems they have and develop the skills to solve them. Eventually someone will need a website or a mobile app or something. Develop a stream of revenue at night that is large enough and predictable enough to quit the day job.

Alternatively, in a big city, you can become a contractor. Just build up an alternative revenue stream so you have something to even out the up-and-down-ness that is contracting and consulting.

Finally, I'd build a cash moat around your business—more than as much cash as you'd get in a year of unemployment, at least six months of expenses. Ideally, you get things so that part time revenue plus the cash means you could go a year. Then you jump into a six-month contract with the opportunity to extend to twelve. After twelve months, you should have a year of cash in the bank that will last further with part-time work.

Q: A lot of readers wanting to be independent might be worried that they're taking a big risk. What would you say to someone at this crossroads?

I would start by saying that financial security is an illusion. Worse, being an employee is a huge risk.

Consider two people, both making about the same per year. One is a freelancer who has about ten customers, with the largest being no more than 25 percent of his income. The other is an employee with

ONE customer who gives him 100 percent of his income. Who is more at risk?

The employee.

The employee has two, perhaps three advantages: the idea that he is secure, so he can breathe, relax, and stop at 5:00 PM. Really, that is permission to be lazy. Second, the employee has unemployment benefits. Once you can earn more part time at night than unemployment offers, that becomes useless. Third, perhaps, is this idea that taxes are easier and you get insurance, planned paid time off, and so on. Most of that is laziness too. Worse, it is your independence as a person. Take it back, man.

Take it back.

Thorben Janssen

Q: Can you walk through your background a bit? How did you come to be doing something other than following the standard corporate software developer career arc? What made you decide to do something different?

To be honest, I somehow stumbled into it. For most of my career, I followed a very typical company career arc. I worked as an intern during my studies and stayed with that company as a software developer afterward. After some time, I got project responsibility for small to medium size projects and tried to be a software developer and project manager at the same time. I really enjoyed it. I talked with customers, coordinated my project team, designed the backend part of the application, and did some programming. I did that for several years and at two companies. All the different tasks made my work days a lot of fun but often also a challenge.

It took me a while to learn that I can't do everything at the same time. I had to drastically reduce the number programming tasks if I didn't want to spread myself too thin. That was OK for a while, but at some point, I recognized that I was just doing project management and

that I wasn't happy with it. Project management stressed me out, and I was missing the programming and software design part.

I started to do more programming in my free time and looked into the relatively new Java Persistence API 2.1 specification. I liked some of the new features and decided to write about them so that I had some notes and examples when I got the chance to use one of the new features in a project. That was the beginning of my blog. It took me by surprise when I saw that other developers liked my posts and the blog started to get some traffic. I never saw myself as a writer. But I enjoyed it and kept writing. All that happened at the end of 2013, and it was the beginning of a new career. But I obviously didn't know that.

During that time, the company I was working for did some internal restructuring, and I took the chance to change my position. That allowed me to do a lot more software design and development again. I enjoyed that for a while, but at the end of 2014, I missed project management, and I was disappointed because a few potential career opportunities in that company didn't work out. I felt stuck in my position, and I had to change something.

A few months earlier, I had found out that some people ran a successful business based on their blog. They offered online courses, books, and classroom training based on the information they shared on their blog. To me, that looked like a great opportunity, and I was wondering if I could do the same. I enjoyed blogging and teaching. I could be my own boss, making a living on my own terms, working on tasks and projects I enjoy.

I talked with my wife about it, and we decided to give it a try. I would invest more time into the blog and try to speak at a few conferences. The blog didn't earn any money, so I had to do that on the side. That approach required a lot of work but didn't involve any risks, which is a good thing when you're the only earner for a family of three. If it doesn't work out, I would at least have something interesting to put on my CV and hopefully some good stories to tell.

The additional effort paid off in 2015. More people read my blog posts, and I published seven articles about different Java EE topics in German print magazines, spoke at two conferences, and gave three paid workshops. My blog was no longer just a hobby. It became a side business.

The speed at which the additional effort brought results was incredible. But it also required an amount of work that I couldn't sustain for long. In summer 2015, we decided that I should try to reduce the working hours in my day job so that I could take the Friday off and work on my side business. Luckily, my boss accepted, and I didn't have to do all the work in the evenings and on weekends.

The additional time gave me the chance to work on my first real product. I created an online training about Hibernate Performance Tuning based on the workshops I had given. I launched it for the first time at the beginning of 2016 and another time in the summer of the same year.

These two launches were the big game changer. They showed that my side business had the potential to earn us a living. I talked with my boss again, with the goal to reduce my day job to three days a week. He didn't want to do that, and I gave my notice soon after. Since then, I'm happier than ever before.

When I look back, I recognize that I had an entrepreneurial mindset which I tried to but couldn't use as an employee. That was probably one of the reasons I often wasn't satisfied with my job. It took a while before I finally understood that I could use all these ideas in a profitable side business without investing tens of thousands of dollars. That was the beginning of a new career outside of the normal corporate world.

Q: What does your day to day tend to look like in terms of the kind of work that you do?

Right now, I'm working on my first book. So I spent most of my time writing, planning, and learning about book publishing. That's probably the biggest change since I left the corporate world. When you start a business and work on your own, you are responsible for everything. I don't have a huge team to whom I can delegate tasks and whose expertise and experience I can use. That makes new projects, like the book, exciting but also challenging and sometimes a little bit overwhelming.

When I'm not working on the book, I create content for my blog and YouTube channel and answer the questions of my readers and online students.

Q: How would you categorize the nature of most of your work: Entrepreneurship? Training/coaching? Content creation? Consulting? App dev?

I categorize most of my tasks as training/coaching and content creation. But all of them, of course, also have an entrepreneurial component. I have to run my own business, and all actions and decisions I make have some impact on it. Some activities influence the positioning of my personal brand or current courses and others to bring new ideas for future content or products. Even actions that just feel right, like giving away a particular kind of content for free or jumping on a free call to discuss someone's questions, are in the end not only a coaching or content activity. They are also entrepreneurial activities that have an impact on my business and brand. So, it's always a mix.

Q: Specifically, do you spend a lot of your time programming?

I obviously don't spend as much time programming as I did in the past. But I still spend quite some time on it. I try new features or dive deeper into the performance impacts of existing, well-known API calls. I need to do that to really understand the things I coach and explain in my blog posts. But I also wouldn't like my job if I didn't spend a good part of my time in the IDE.

Q: How do you balance your time between looking after business interests (overhead) and the work you do that offers value? Does the overhead ever seem burdensome?

I spend almost all of my time on work that offers value, like creating content for my blog and YouTube channel or answering questions from my readers and students. I enjoy this kind of work. It provides value to my community and is the best marketing I can do for my training offers.

But there is, of course, also some regular paperwork and lots of repetitive tasks that need to be done. I try to avoid these things and work with an accountant to do my bookkeeping and tax returns. I have a virtual assistant who handles most of the repetitive tasks. Working with these people is probably one of the best decisions I've made so far. They allow me to focus on the work I enjoy the most, and that creates the most value for my readers and customers.

Q: In what profit structure(s) do you typically make money (e.g., hourly consulting/contracting, fixed bid consulting/contracting, passive, royalties, dividends/salary from your company, etc.)?

I make most of my money with my online training offers. Most people would probably categorize that as passive. But I'm helping my students while they work on the course material and when they apply it for the first time in their projects. That makes them and me successful. So it's not as passive as you might expect.

I also do some paid writing for other websites and publications, which is a kind of fixed bid contracting.

Q: Why do you think there's such a ceiling, both in pay and organizational status, for software developers in the corporate world?

There is a reason why people talk about "climbing the career ladder." In most jobs, you change your position and job title when you improve and get more responsibilities. The new position brings more

status and sometimes also a nice salary increase. You're climbing to the next rung of the ladder.

Software developers most often don't change their position or title. They might become a senior developer and work on the most challenging tasks, but they are still a software developer who's programming something.

We all know that there's a huge difference between the work of a beginner and an experienced senior developer. But there is no representation for it in the typical corporate career ladder. That limits the pay and organizational status, as long as you need to get a new position or title to improve it.

Q: There seems to be a common canard in corporate software development that programming is a game for the young and that a career-oriented person needs to stop it and become a manager. What do you recommend for someone that is ambitious but not willing to stop programming? How do you avoid the stigma that programming work is "line-level grunt work?" How do you suggest others avoid it?

First of all, do what you love doing. You spend a huge part of your life working. Do you really want to do something every day for the next twenty, thirty, or forty years that you don't enjoy?

There are several things you can do to have a career as a software developer. You can become a software architect, for example. A good architect spends most of his time on technical tasks and does a lot of programming.

In a lot of smaller companies, you also have the option to do a combination of project management and programming, like I did for a few years. Or you can become the leader of a small team and still do some programming. You just need to be aware that both options require you to do at least some—and sometimes even a lot—of other work, which will reduce the amount of time you spend programming.

You see, there are a lot of options. It all depends on what you want to do and the company you're working for.

Q: What advice do you have for readers of the book that want to get out of the nine to five programming game but continue to earn a living as technologists?

Ask yourself two questions:

1. What do you want to do?
2. Who's paying for that?

The first question is the most important and often also the most difficult to answer.

Try a few different things before you decide how to change your career. I never expected that I would enjoy writing so much that I would make it a huge part of my daily work. To be honest, I hated English at school and was really bad at it. But after I started blogging, I couldn't stop.

So, go out there and try a few things. Get a consulting project and work on it in the evening or on weekends, publish a blog or magazine article, speak at a local user group or conference, do internal training to share your knowledge with your coworkers. Sooner or later you will find something you enjoy so much that you want to keep doing it.

As soon as you've found that, ask yourself who would pay for it. You can also have a look at what others in your niche are doing. There will most likely be some people who already made a career based on a similar passion.

When you've answered both questions, you can work on your new career. Start small and don't quit when it doesn't immediately work out. It will most likely take a while to make the switch. You also didn't learn to program in a day.

Q: A lot of readers wanting to be independent might be worried that they're taking a big risk. What would you say to someone at this crossroads?

Yes, going independent seems risky. But you also have no job security when you're an employee. I know of several companies that laid off a good number of their employees without any warning. And I don't know anyone in our niche who worked at the same company for twenty or thirty years.

You have to decide for yourself how you want to handle this situation.

Do you want to rely on others to secure your current job, and do you want to prepare yourself to find a new position on short notice?

Or do you want to have all information and be in control of all aspects of your career and income so that you can handle the risks?

Both are valid options, and I definitely know which approach I prefer. What about you?

Sally Lehman

Sally had a limited amount of time and bandwidth to answer interview questions. I sent her the stock set of questions that I was asking everyone, and a few additional questions aimed specifically at her background, asking for higher priority with those. On top of that, as a salaried employee, not all of the questions were relevant to her. The result is the Q&A featured below.

Q: Do you spend a lot of your time programming?

I try to spend about half of my day programming and the other half of my day figuring out what to program and coordinating with people, mostly other members of my team.

Q: How do you balance your time between looking after business interests (overhead) and the work you do that offers value? Does the overhead ever seem burdensome?

No, because the companies I've worked for have been large enough to have people overseeing the business aspect, and they were coordinated enough to not really need me (except to automate specific things). I haven't needed to delve into that much.

Q: Can you tell me a bit about what it was like working for GitHub? From an outsider's perspective, I've heard that they offer open allocation and a whole lot of freedom. Do I have that right? Was that your experience?

I don't really know how it is now as it's been almost three years since I was there. GitHub was an incredible growth opportunity, and there was a lot of freedom and open allocation when I started. What I've heard is that it has changed quite a lot—that it is a very different company than the one I originally started working for. It's over two times the size it was in employees. I know that management structures have changed and morphed since I was there and that there has been lots of churn in c-level folks as well. I think they are still trying to figure out what will work for them.

Q: It looks at a casual glance (LinkedIn) like you've favored working for relatively small, cutting-edge tech companies. What do you look for in a prospective employer? What are your requirements?

I love technology, but I ultimately care about being in a healthy working environment more than technology. So first thing I look for is a team that I would enjoy working with. Secondly, I look for technical and company structures that are conducive to being productive, like a group chat system being the hub of communication (rather than any communication that is by default one-to-one or undocumented). After that I look for a good mix of technologies or problem sets that I know and ones I don't so there is opportunity for growth for me and opportunity to deliver highest value to the company given my experience.

Q: You also seem to have a fairly specific/niche focus in terms of the type of work that you do (working specifically with email). How does this play into your relationship with companies and the hiring/search process? Do you ever think about flipping that to a consulting practice at some point?

Although my previous three job titles to this one have had “mail” in the name, I have worked very hard to widen my job description over the past three years, which is why I am a “Production Engineer” now. I’ve learned that most companies do not value their email reliability and scalability enough to properly sustain full-time email staff for the long term. While I enjoy caring for the email structures and solve problems as they arise using the special experience I have, I’ve consciously made a career decision to continue to widen my range of operations knowledge. Consulting has too much overhead for me personally—as an employee, I have the luxury to think mostly about technology, and I don’t want to have the additional huge responsibility of keeping a business.

Q: In your travels, have you experienced other organizations having structures like (my understanding of) GitHub’s, in that they grant you a lot of autonomy and flexibility?

GitHub has had the highest flexibility of any company that I worked for, at least when I started working there. The other companies I’ve worked for have also had significant flexibility, even the relatively large one, GoDaddy. I have always had the option to be a remote worker and be on a global team, and when I didn’t have the flexibility of choosing what to work on, I’ve gotten large tradeoffs like flexibility in travel and work times and other useful things. I feel that I’ve just been lucky to have always had this. All the stories I’ve heard about restrictive companies have been just that: stories. I think any company with a large global & remote workforce will find hard restrictions difficult to enforce.

Q: Do you do any freelance consulting/moonlighting on the side? Either way, would the various companies you’ve worked for have allowed you to do that?

About half of the companies I've worked for have had that stipulation that prevented a side job/contract. In one case I negotiated the possibility, but didn't end up using the provision. I think if you are well compensated enough, it's reasonable to expect that you devote all the time and mental energy that you have reserved for work to one company.

Eugen Paraschiv

Q: Can you walk through your background a bit? How did you come to be doing something other than following the standard corporate software developer career arc? What made you decide to do something different?

My background is quite typical—I studied computer science and then did the corporate thing for a number of years. A few years in, I was getting bored of typical enterprise Java development, so I started to blog. It wasn't some well planned decision, but I was super enthusiastic about writing, so I didn't really need much to keep my going.

A couple of years later I started to understand that I could do a lot more interesting work freelancing and that geography was no longer a constraint. That meant that I could work with clients all over the world—which made a lot more financial sense than staying local.

Now, I'm working as a 100-percent remote architect with Uptake, a Chicago-based company, which is quite far from the typical enterprise grind. I'm also working on my own products and growing the content team at Baeldung (around 100 authors).

Q: What does your day to day tend to look like in terms of the kind of work that you do?

I try to get some structure into my days. Typically, the first part of my day is work on the site. I help the editorial team, do marketing, record videos for our next course—things of that nature.

The latter half is focused on client work—working with teams, coding, reviews. Uptake has skyrocketed from zero to almost 1000 employees in just two years, so every six months or so, what I'm doing day to day changes.

Q: How would you categorize the nature of most of your work: Entrepreneurship? Training/coaching? Content creation? Consulting? App dev?

That's an interesting question, but one without a simple answer right now. Part of my work is consulting, and the other part is entrepreneurship at a high level—and lots of training/coaching at a more practical level as I build the team here at Baeldung.

Q: Specifically, do you spend a lot of your time programming?

Not as much as I'd like to, but yes, I still do a lot of coding. I do some of that in my client work and some when I'm creating a new product for Baeldung. For example, Spring 5 is coming out with reactive support, so I'm now digging into that and putting together a workshop.

Q: How do you balance your time between looking after business interests (overhead) and the work you do that offers value? Does the overhead ever seem burdensome?

Luckily, I always had long term clients, which may be a combination of my own preference and the fact that I work in the Java space. That means I don't have a ton of overhead when it comes to client work.

Q: In what profit structure(s) do you typically make money (e.g., hourly consulting/contracting, fixed bid consulting/contracting, passive, royalties, dividends/salary from your company, etc.)?

Half of my income is the consulting work. This is not hourly; it's structured in weekly chunks. And the other half is sales of my own courses and products through the site.

Q: Why do you think there's such a ceiling, both in pay and organizational status, for software developers in the corporate world?

Lack of imagination, in a sense. Corporations aren't very quick to adapt to the new realities in our ecosystem, and most developers aren't that quick to take advantage of these, either. When a developer works in that relatively strict structure, yes, they generally do accept a lot of constraints.

But when they step outside of that and either become a free agent or move towards organizations that share both risk and upside with their employees, those constraints very quickly start to go away.

Q: There seems to be a common canard in corporate software development that programming is a game for the young and that a career-oriented person needs to stop it and become a manager. What do you recommend for someone that is ambitious but not willing to stop programming? How do you avoid the stigma that programming work is "line-level grunt work?" How do you suggest others avoid it?

I really hope we're outgrowing that script and start seeing plenty of examples of people that code far past their thirties and forties. Blogging definitely plays an important role here. Right now, I follow the blogs of more developers that are in their forties than those of some young whippersnapper. And all of these people code regularly.

If I were to give advice to a developer that wants to keep coding, it would be "provide a lot of value." As long as you do that, there are plenty of companies that have no problem receiving that value, regardless of your age. That being said, if the driving motivation is financial, there is certainly that ceiling to be aware of. So a move towards becoming a free agent and removing that ceiling entirely is not a bad way to go.

Q: A lot of readers wanting to be independent might be worried that they're taking a big risk. What would you say to someone at this

crossroads?

Given my own trajectory, the obvious advice is to start freelancing/consulting. Now, you can either set money aside and do this abruptly, or you can do this smoothly over the course of a few months or even a year (if you're prepared to push yourself for a while). I personally did it the slow way, mainly because my risk tolerance isn't very high.

One other approach that I've seen produce some fantastic results is productized consulting services. If you've never heard the term, it's a hybrid between custom consulting work and selling a product. It has the advantage of not having a lot of startup cost (the best ones I've seen were done over a single weekend) but can still be highly profitable.

We're living through a significant and hard to miss shift in the way we're developing software. And that basically means that there's no longer one script we're stuck with. There are a lot of opportunities to take advantage of, if we step a bit outside of our comfort zones. It would be silly not to take advantage of them.

Dave Rael

Q: First off, if you have any reactions/impressions to what I've said [outlining the message of this book], that's certainly welcome.

There is no question there is dysfunction in the way most corporations operate. I always think of Douglas Adams and the Golgafrinchans with all their useless meetings and processes in the Hitchhiker's Guide series when thinking of this topic and corporate culture in general. The future you predict would almost certainly be an enhancement compared to what I have experienced—both from the perspective of thought workers having better working conditions, better wealth-enhancement as a result of the value they deliver, better problems to solve, and in terms of better results for the organizations they serve as partners rather than as internal laborers.

Whether the real world moves in that direction is a different question. Trusting an independent expert rather than turning the screws on a subordinate is a better way in just about any measure of which I can think, but convincing people who have already acquired rank, position, and stature in a different model of that truth is a challenge. As more agencies have been popping up in recent years and a few organizations have been emerging with a focus on giving good people the trust to run with what they can do, there's reason for optimism. This is a matter of trying to fight the inertia required to turn an aircraft carrier, though. It's more than that, too. There are vested interests in the power structures of corporations, governments, and social organizations, formal and informal. Trying to improve the lives of individuals meets lots of resistance from people with contrary incentives.

Q: Can you walk through your background a bit? How did you come to be doing something other than following the standard corporate software developer career arc? What made you decide to do something different?

I had a small number of long tenures in jobs. I worked for a large telecom company as a programmer having not had an education in computer science in my first software job. I learned on the fly at a time when the dot com bubble was inflating and programmers were in great demand. My first project there was great and wonderful and I loved the team and learned at an incredible rate. Later projects were exactly what you'd expect at a huge corporation with all the bureaucracy, overbuilt process nonsense, misaligned incentives, hurry-up-and-wait nature, and lack of impact. I stayed at that job, though completely bored, for a long time due to the bursting of the bubble and the complete reciprocal of the previous demand for programmers.

When I did get another job, it was for a small-and-growing company. I joined a team of two operations guys and two programmers and grew with the company to a time when there were over thirty developers and many other IT personnel. I was promoted through

different developer titles and ultimately named architect. I stayed there longer than I should have as it became a large organization because I was comfortable and didn't make the effort to find new challenges I knew I needed to make.

It was a great feeling of dissatisfaction with the lack of growth I had been undertaking and a feeling like I had so much more to offer that (much later than it should have) ultimately prompted me to leave and try independent work. I worked a few app dev contracts and had some family health issues that prompted me to take some time off. During that time off, I started blogging and later podcasting. When savings ran low, I returned to doing some contracting/consulting, but with less relish.

Q: What does your day to day tend to look like in terms of the kind of work that you do?

It varies now. I have broken out of the idea that I need to be continuously employed and move from one project to the next. Increasing rates has certainly made that more realistic. Since establishing an audience via podcasting, I spend quite a bit of time trying to serve that audience.

I now seek shorter engagements and value downtime. I intend to move toward more entrepreneurial endeavors, including monetizing content creation and product development, but progress on trying to approach earning a living has been slow. I continue to create audio content and a little bit of text, and I like to work on streamlining and automating my processes for that.

Sometimes I have clients with whom I am working, and I do most of that remotely. Sometimes I go onsite for limited engagements, but I have come to dislike doing so and will probably be avoiding that in the future. In a given day I may be recording interviews, writing code, meeting with clients, editing audio, writing show notes, interacting with podcast listeners and community members, and writing blog posts.

Q: How would you categorize the nature of most of your work: Entrepreneurship? Training/coaching? Content creation? Consulting? App dev?

I have mostly done staff-augmentation labor work as a contractor, which is different from full-time employment, but not that different. I have consulted a little and intend to do more of that. Earning some entrepreneurial income resulting from having an audience has started, and I intend to expand that. I have done some training, but only a little. Coaching is of interest to me, but I have yet to explore that route—I'd probably do that more on an individual basis than with organizations.

Q: Specifically, do you spend a lot of your time programming?

Yes. Not most of my time, but a significant portion.

Q: How do you balance your time between looking after business interests (overhead) and the work you do that offers value? Does the overhead ever seem burdensome?

The overhead is not really burdensome. I don't do a lot of seeking opportunities. Most of them find me. Using software for accounting and such makes things pretty easy.

Q: In what profit structure(s) do you typically make money (e.g., hourly consulting/contracting, fixed bid consulting/contracting, passive, royalties, dividends/salary from your company, etc.)?

Mostly hourly contracting, but I'm moving hourly stuff toward consulting. I have written proposals for fixed bid consulting/contracting and would love to have that replace hourly entirely, but I've yet to have any takers on one of those proposals. I've made a little money from podcast sponsorship and donors, but that's so far pretty insignificant.

Q: Why do you think there's such a ceiling, both in pay and organizational status, for software developers in the corporate

world?

There are several reasons. Probably chief among that is because a significant portion of software developers are terrible marketers—not only terrible at knowing and articulating their worth, but emotionally tied to an idea that marketing is beneath them. Combine that with a technical focus and an emotional undercurrent in thinking that making money in ways other than delivering software is a lesser pursuit finds developers tending to limit their own value.

Programmers are seen by many as being interchangeable, too. When viewed as a commodity, we are treated as a commodity.

Q: There seems to be a common canard in corporate software development that programming is a game for the young and that a career-oriented person needs to stop it and become a manager. What do you recommend for someone that is ambitious but not willing to stop programming? How do you avoid the stigma that programming work is “line-level grunt work?” How do you suggest others avoid it?

Entrepreneurship is the obvious answer to a place where creating software can have an unlimited upside. It also has unlimited downside, though. I have absolutely hated my experiences with being a manager and had mostly just accepted that that meant there were going to be limitations on what I could do.

I do think that creating an audience via content creation will ultimately lead to alternatives, but it's a long road that pays off only after significant investment. A future with the freelancer-driven organizations in the gig economy would provide a great alternative with less risk, the ability to be rewarded while on the climb, and the opportunity to make great impact. There are probably opportunities like that now, but I don't know that I've seen them.

Q: What advice do you have for readers of the book that want to get out of the nine to five programming game but continue to earn a living as technologists?

First and foremost: patience. Escaping the rat race does not happen quickly. I still have a long way to go. Next, building an audience is a great way to enhance everything you have to offer. It can expand your options for sources of income and ways to find clients, and it can help you learn about parts of your personality and expertise you didn't know existed. Creation of content in some form is a must in this new world where creating content is easy. Building an audience is not easy and it is slow, but creating valuable content is something you can do today. If you are consistently serving communities, you can start to build one of your own.

Q: A lot of readers wanting to be independent might be worried that they're taking a big risk. What would you say to someone at this crossroads?

Being an independent laborer is not greatly different than being an employee. It's not greatly better or worse and the risk is not terribly different, especially in a high-demand environment. After establishing high enough rates/bids, one is able to sustain downtime with stress and may even come to welcome and anticipate it. Additionally, full-time employment is not without risk. It's not even certain the risks there are less.

John Sonmez

Q: First off, if you have any reactions/impressions to what I've said [outlining the message of this book], that's certainly welcome.

I definitely agree with your viewpoint on this. Most in-house software projects fail and they fail miserably. But companies keep doing them because they don't know of an alternative, and they think that their environment and domain is so unique and specific that outside consultants can't understand it and build software for it.

Salesforce.com has already started to challenge this kind of thinking by replacing a large amount of in-house software with a commercial off the shelf solution, which can be customized by consultants.

Q: Can you walk through your background a bit? How did you come to be doing something other than following the standard corporate software developer career arc? What made you decide to do something different?

I started off as the typical software developer, working both in the corporate world and the small company/startup space. I also spent a large amount of my early career in contract positions, which were really staff augmentation for larger companies.

I did pretty well by most people's standards. I had good jobs. I made really good money, but I eventually hit a cap. I hit a point where it didn't matter how much more experience I had or how much I developed my skills, I just wasn't going to be paid a higher salary or contract rate. So I started looking for alternatives—not consciously at first, but I was just trying to do more things and figure out if there were other ways I could make more money on the side.

I ended up creating some mobile apps and creating my blog at <http://simpleprogrammer.com>. The mobile apps made some decent money but nothing to write home about. But the blog—now that was interesting. It didn't make me money directly, but it got me notoriety.

Suddenly people knew who I was. Suddenly I had some authority.

I say suddenly, but it definitely wasn't suddenly. It took a few years to get real traction on the blog. But when I did, BAM. I was getting emails and calls with all kinds of job opportunities, to work directly but also to do real consulting. I started doing podcast interviews, making my own podcasts, speaking at conferences, and other activities to really market myself, and the opportunities just kept increasing. One of the best ones was the opportunity to create developer training courses for Pluralsight—I ended up creating fifty-five courses for them.

Eventually though, the blog itself started making money. I released my first product, called "How to Market Yourself as a Software Developer," based on the experience I was having with branding

myself and increasing my reputation as a software developer. I started to realize that more and more of my popular content was not just the technical material but also the content focused on soft skills and personal growth. So I made a pivot and shifted my Simple Programmer business to helping developers develop their soft skills in addition to their technical ones.

I published my first real book with Manning, *Soft Skills: The Software Developer's Life Manual*, and then I basically became “the guy,” in terms of teaching software developers soft skills.

And that's what I do now. I've created several other products that I sell through Simple Programmer, I produce two to three YouTube videos a day, I still blog, I'm working on another book, and I do podcasting and speaking, all related to the idea of teaching software developers how to achieve their goals by focusing on personal growth and development.

Q: What does your day to day tend to look like in terms of the kind of work that you do?

Well now, that depends heavily on the day.

Most days, I spend the first hour of my day writing. [I heavily use the Pomodoro technique, along with a Kanban board to manage my week.](#) So, I'll usually spend the first two Pomodori writing. I find that it is best to start with the biggest or most difficult task of the day—and for most people, that's usually writing.

Then, I'll usually move on to whatever projects I have going. I always have some other project going, whether it be recording a video course, writing a guide, optimizing something in the business, hiring someone, or doing interviews. I try to tackle one project at a time, but I have a list of at least 100 that I know would benefit the business.

Right now, I also record two to three new YouTube videos each day, and of course, no matter what I do, I have plenty of email to deal with. I'd say that most of my work is creative work, producing tons of

content, and the rest of it is working on the business to improve it, optimize sales, reduce costs, and grow.

Q: How would you categorize the nature of most of your work: Entrepreneurship? Training/coaching? Content creation? Consulting? App dev?

Definitely mostly entrepreneurship and content creating at this point. I very rarely write code anymore, and I do limited training and coaching now.

I try to focus mostly on things that scale and will create a long lasting value for me. That is why I focus on content so much.

Q: Specifically, do you spend a lot of your time programming?

Almost zero. I may change that next year as I get into more technical topics and create more programming related content and products. But, right now, it just doesn't make sense for me to code—even though I miss it a great deal.

Q: How do you balance your time between looking after business interests (overhead) and the work you do that offers value? Does the overhead ever seem burdensome?

It's difficult. Especially in regards to email. I get tons of email. I have an employee who answers as much of it as he can, but there is still a ton of it.

There are also plenty of other overhead activities that only really I can do and that need to be done. So it is burdensome, but I always schedule the value work first. That's why I spend the first hour of my day writing. I'll even skip email for a day or two if there is some project I am working on that I don't want to interrupt. I also make sure that I'm constantly producing content, and I create quotas for that content each week or day so that I can't get off track and spend too much time on overhead.

But running a business will always have overhead; that's just part of being an entrepreneur.

Q: In what profit structure(s) do you typically make money (e.g., hourly consulting/contracting, fixed bid consulting/contracting, passive, royalties, dividends/salary from your company, etc.)?

A huge chunk of my income comes from royalties, both the royalties I make from my Pluralsight courses and from my book. I keep that as a separate business from Simple Programmer, though.

I also make a large amount of my personal income from my real estate investments. I've been investing in real estate since I was nineteen.

As for the business—specifically, Simple Programmer—we get the largest amount of revenue from product sales of digital products. After that, advertising is probably the second biggest income source, and then affiliate commissions (Amazon referrals and the like).

There are a few other minor income sources. I'll do some coaching and consulting. I'll get speaking fees for speaking at conferences or events.

I take a salary from Simple Programmer, but I try to reinvest profits into growing the company—at least right now.

Q: Why do you think there's such a ceiling, both in pay and organizational status, for software developers in the corporate world?

It all has to do with risk. Entrepreneurs will always have the potential to make more money because they are willing to take on more risk. People complain all the time about big, greedy corporations and how they should pay their employees more, or they treat business owners like the devil, but they don't understand risk/reward. The more you risk, the higher the potential reward.

As a software developer working at a corporate job, you have zero risk. In fact, you have negative risk because you are pretty much going to get your paycheck no matter what, the chances of you being fired are small, and if you do get laid off, you are probably getting a healthy severance package. I'm not trying to knock employees, by any means. Being an employee is a perfectly valid choice—especially if you are highly risk adverse.

But if you want to make more money and smash through the glass ceiling, you are going to have to either take more risk or provide a huge amount of more value—or both. That's just how the world works. I didn't invent it. If you go out on your own, if you freelance or start your own business, you are going to be taking on much more risk, but you can also make a lot more money.

Q: There seems to be a common canard in corporate software development that programming is a game for the young and that a career-oriented person needs to stop it and become a manager. What do you recommend for someone that is ambitious but not willing to stop programming? How do you avoid the stigma that programming work is “line-level grunt work?” How do you suggest others avoid it?

Well, first of all, that's ridiculous. I know plenty of “old” software developers. Look at Bob Martin (Uncle Bob).

I think a good amount of older developers try to claim age discrimination and say that software development is only for the young, when in reality software development is only for those who choose to never stop learning and keep their skills up to date. Software development is more of a meritocracy than most other professions, so it kind of pissed off people who've “paid their dues” but haven't stayed up to date on technology.

But, to address the bigger issues, I'd say that a software developer who wants to continue to advance their career past senior software developer or some other such level has two choices:

1. Go out on their own and freelance, and charge a rate that is much higher than they could possibly get at a corporate job or even contract position.
2. Join a very large technical corporation like Microsoft, HP, IBM, Google, or Apple, where they specifically have technical tracks where you can continue to advance as a software developer without going into management. Then you can get a cool title like, “Senior Fellow,” or “Distinguished Technologist.”

To answer your question a third way, yes, programming *is* grunt work. That is, in comparison to running a business or commanding an entire division.

I’m not saying this to knock programming. I love programming. I am a programmer at heart.

But I am saying this from practical experience. Obviously I had to stop coding because it was not the best and most profitable use of my time. I can’t change the world by writing code, but I can by doing what I am doing. It doesn’t mean I don’t love to code or that I’ll never do it again, it just means that I realize that programming is a technician activity—and it always will be.

Expect programming to become more and more commoditized over time—especially as general education increases. It doesn’t mean you can’t do it. It doesn’t mean you can’t love it.

And you should definitely expand your abilities beyond just programming code and become a more valuable software developer by adding communication skills, architecture, and true software development expertise into your arsenal. But you should realize that there will always be limits to what one programming can do because programming is not an activity that has a huge amount of leverage on its own. Yes, creating products that change the world is high leverage, but you don’t have to know how to program to do that.

Programming is a tool that gets other things done. Programming itself cannot change the world.

Q: What advice do you have for readers of the book that want to get out of the nine to five programming game but continue to earn a living as technologists?

Start with a side project and/or a side business. Work your nine to five, and then when you come home, work your side business for four more hours. Do this for several years. Feel what it is like to work on weekends and to spend that much time working. If you can't hack it, stick to the nine to five. If you can, you should be able to generate enough income to barely cover your expenses and quit your regular job.

What are you talking about John? That's crazy!

Yes, it is crazy, but that is what it is like to start your own business—at least the first few years. If you want to become an entrepreneur, you have to make sure you can do it first, and if you can't hold down a full time job and build a side business, you can't build a full time entrepreneurial business—you'll wash out. Better to wash out while you still have that paycheck coming in and before you've mortgaged your house and sunk all your savings into your brilliant idea.

Oh, and I'm an optimist.

Seriously though, you can do it, and I want you to do it. But it's going to require about 10x the work you think it will and it will be 10x as difficult.

Read the books *The 10x Rule* by Grant Cardone, *The E-Myth Revisited* by Michael Gerber, and *The War of Art* by Steven Pressfield before you even think of "leaving the nest."

Q: A lot of readers wanting to be independent might be worried that they're taking a big risk. What would you say to someone at this crossroads?

Yes, you are. Remember what I said about risk versus reward. But there is good news. If you do it like I said above, you'll be reducing

that risk to almost nothing. You'll be trading pain, time, and hard work for risk—oh yeah, forgot to tell you you could do that. It's called elbow grease and it's the greatest risk reducer of all time.

Yeah, there will still be risk, but life itself is risky. Always take risks—just take calculated ones.

When you walk across the street or fly in an airplane, you take a risk. More so when you drive a car more than a couple hundred feet. Too many of us are too afraid to risk. We are afraid of uncertainty. We want it all nailed down and charted out.

Well, let me tell you something from experience. Life is boring that way—it's not worth living. Don't be stupid and quit your job, max out your credit cards, give your boss the finger and head to Starbucks to work on your next great idea, but don't be so afraid to live that you watch all your dreams die on the vine. I'd rather see you fall flat on your face and biff it and not follow any of my advice than to live a boring-ass mediocre life full of "what ifs." Excuse my language, but fuck that.