The Career Programmer: Guerilla Tactics for an Imperfect World

CHRISTOPHER DUNCAN

Apress™

The Career Programmer: Guerilla Tactics for an Imperfect World

Copyright ©2002 by Christopher Duncan

ISBN (pbk): 1-59059-008-2

Printed and bound in the United States of America 12345678910

Editorial Directors: Dan Appleman, Peter Blackburn, Gary Cornell, Jason Gilmore, Karen Watterson

Managing Editor: Grace Wong

Copy and Development Editor: Tom Gillen

Production Editor: Sofia Marchant

Compositor: Diana Van Winkle, Van Winkle Design Group

Indexer: Ron Strauss

Cover Designer: Tom Debolski

Marketing Manager: Stephanie Rodriguez

Distributed to the book trade in the United States by Springer-Verlag New York, Inc.,175 Fifth Avenue, New York, NY, 10010

and outside the United States by Springer-Verlag GmbH & Co. KG, Tiergartenstr. 17, 69112 Heidelberg, Germany

In the United States, phone 1-800-SPRINGER, email orders@springer-ny.com, or visit http://www.springer-ny.com.

Outside the United States, fax +49 6221 345229, email orders@springer.de, or visit http://www.springer.de.

For information on translations, please contact Apress directly at 901 Grayson Street, Suite 204, Berkeley, CA 94710.

Phone 510-549-5930, fax: 510-549-5939, email info@apress.com, or visit http://www.apress.com.

A Brief Introduction

If you picked this particular book up off the shelves, it's likely that you either make a living as a programmer or plan to do so in the future. Of course, you could just be shopping for something that's precisely the right thickness to prop up your rickety coffee table, but we'll play the odds for the moment and assume that you spend more time in front of a debugger than you do a television.

The first questions we typically have when looking for technical books relate to platform and language. What kind of computer? What flavor of operating system? Which particular set of languages and technologies must we know to benefit from this book? For *The Career Programmer*, it just doesn't matter. Whether we make our living under the flag of Mac or Microsoft, mainframe or mini, our ultimate goals and desires are the same: all we really want to do is make good money, spend our days writing the next killer app, and get our pizza delivered in thirty minutes or less.

However, for those who possess the ability to turn screens full of cryptic statements into state-of-the-art software systems, there's only one slight complication. In order to get paid for doing what we love, we find ourselves working deep in the heart of Corporate America. Nothing we were taught in school prepared us for the illogical, inconsistent, and sometimes downright silly business practices that are the norm in software development shops both large and small.

For instance, deadlines are declared in an almost arbitrary fashion, with no real consideration of what's involved. We're expected to produce specific functionality from vague and ever-changing requirements. We're given little to no time for proper analysis and design, and, when we ask management about hiring professional testers and setting up a QA process—well, I've seen deer in my headlights look less stunned. Internal politics are rampant, threatening everything from our current project to our staplers, which for some bizarre reason seem to be a precious commodity in the cubicle world. In short, from a software developer's point of view, the environment makes no sense. Unfortunately, no matter how unrealistic the deadline, we're expected to work day and night to make it happen, only to have the product shipped with less than fifteen minutes of testing. Care to guess who's going to get yelled at the first time it blows up in the field? I can assure you, the deer have leapt safely out of the glare of the headlights and there's nobody here but us programmers.

Anyone who hasn't worked in our field will by now have labeled me quite the cynic, but I suspect you were shaking your head while reading the last paragraph and remembering the insanities of your own company's releases. As passionately as we want to do good work, it seems that we're checked at every turn by corporate bureaucracy and management that can barely spell the word *computer*, let alone manage a software development process. Of course, we learn all of this the hard way. All of the books on the bookstore shelves teach us how to program, not how to fend off the lunacy of the business world so that we can actually program and deliver excellence.

That's why I wrote this book. I've spent the better part of the past ten years as a mercenary. For the uninitiated, that's a contract programmer, and it means that I've seen a lot of shops. Over the years, I've learned some tricks to help take control of my programming life again, along with how to further my career in general. (I do like to eat well.) For a long time now, I've been working exactly the kinds of jobs that I like, doing the techie stuff that I enjoy, concentrating on the coding, and actually delivering on schedule, not to mention getting paid well in the process.

Much of what I know as a programmer I've learned from other guys who were nice enough to share their experience, and so this is my way of giving a little back. No matter where you fit into your project, you can learn some tricks from this book to help simplify your life and get you back to concentrating on the code. That's the best thing I can think of to give to a programmer.

Assumptions about the Reader

I assume that you already know how to program and that you either currently make your living programming in the business world or soon will be. The platform or language you work with doesn't matter. This is a book about overcoming the obstacles we encounter as programmers in the real world, and these obstacles are the same regardless of what type of software you write. It probably goes without saying, but I also assume that you rail against any and all limitations that you encounter on the job and that you find anything that interferes with your ability to deliver high-quality software extremely frustrating. That's where I hope to help.

Who Should Read This

The issues addressed here affect developers at all levels. If you work as a project manager or team lead, you're already a veteran of many battles with management and are always looking for new ways to talk sense into these guys. If you're a front-line programmer, you're the one who ultimately has to pay the price for bad decisions further up the food chain, whether it's from the Suits in the front office or your own project manager. The tactics and techniques work at both levels. If you're not happy with the way things are run and want to change your job so that you can focus more on software development and less on damage control from dubious decisions, this book is for you.

A Note to Women Who Code

Not everyone who stays up for three straight nights chasing a bug is male. Many women also make a living as professional developers. When the fingers hit the keyboard, it doesn't matter if the nails are polished or not. The compiler doesn't care and neither do I; good code is good code and none of the topics I cover are gender specific. However, the English language simply does not have an easy way to address both genders in a conversational manner. While I applaud the sincere intentions of those who insert a "his/her" or "he/she" into everything they write out of consideration for equality, in practice it can make the text a bit tedious to read.

I'm no Nobel laureate. I'm just a programmer, and I write like I talk. Although the issues we'll cover are serious ones, my priority is to keep this a light, conversational, and easy read. I'm more interested in helping people overcome the many obstacles to good software that Corporate America continually throws our way than I am in being considered a scholarly author. Therefore, to keep it simple, I made a conscious decision to speak to a single gender for the sheer literary convenience of doing so. Because the programming community is overwhelmingly populated by the male of the species, you'll see references to "he" and "him" rather than an attempt to speak to both genders in every sentence. If this is perceived as a lack of political correctness, I hope that the easier flow of words and the matters upon which they touch will compensate. This is a book for programmers. The specifications of your particular body are irrelevant.

What's Not Addressed

This is not a language or technology book. No matter what programming technique you want to master, plenty of books are available to teach you. This book is about overcoming the obstacles you face on the job that ultimately result in release disasters, stressed-out development experiences, software death marches, and bad software that could have been good. It's not a treatise from the ivory tower of academia. It's a field manual for the software developer grunts who relentlessly toil away in the thick of it, day after day.

What This Book Brings to the Party

If you worked in a perfect world, you'd have plenty of time for gathering requirements, for analysis and design, and for implementation and testing. You'd also be in charge of what went into the product, the overall design, and which technologies were used, and you'd release it when you were darned well ready. Management would not interfere, and you wouldn't have to contend with any office politics. Everyone would listen and do things exactly as you suggested. Small, furry creatures from Alpha Centauri would be, well, you know.

If you live in such a world, go ahead and use this book for that wobbly coffee table. Oh, yeah, and save me a seat. I'd love to have a job there.

For the rest of us, this book is a collection of very simple and practical observations and techniques for putting us in as much control of the development process as the real world is going to allow. A number of hurdles must be cleared when shipping a good product, and some of these can be handled by modifying the approach we take to design, estimating, and other matters of process.

Other issues result from bureaucracy and politics. No design methodology in the world is going to help you there. The higher-ups tend to ignore the opinions of programmers partly because we've never learned to speak their language or communicate in a way that is meaningful to them. Consequently, our thoughts and suggestions—the very things that could save our projects from disaster—are ignored even though we're the specialists. Before we can show them how we'd like to do things, we must first acquire the skills necessary to make them hear us. In short, we need to learn how to manage our management so that we can get back to doing the job that we love.

You don't need an MBA to figure this stuff out. You just need to pay attention to how things work and modify your approach to one that is realistic and effective in your environment. The bottom line is simple: whether we agree or disagree, more often than not we're simply told to get the job done in the time we're given, or else. Consequently, the approaches that work when we have the luxury of time fail utterly when we have the ability to implement only a quarter of the process. In such moments, we need simple and practical approaches that get the product delivered.

In the chapters that follow, I'll be addressing these issues with the assumption that you don't have time for the perfect solution. That's why I refer to them as *guerilla tactics*. They're direct, effective, and they're not always pretty. These tricks are all taken from real jobs with real pressures. When you have to deliver, or else, neatness just doesn't count. Getting the job done is all that matters.

A Quick Peek at the Chapters

Here's a look at the rest of the book. Part I explains the problems prevalent in our jobs, and Part II speaks to the issues and their solutions.

Part I: Software Development in an Imperfect World

Chapter 1: Welcome to Corporate America

After landing their first job, many programmers are shocked by the reality of life in the corporate world. Your initial dream of sitting undisturbed each day, kicking out clever little apps, is continually disturbed by unrealistic deadlines, unreasonable decisions, bureaucracy, politics, and crisis after crisis. Any of these could reduce your current software project to a pile of smoking rubble reminiscent of the latest Godzilla movie. They don't teach this sort of thing in school, and even seasoned developers have difficulty knowing how to cope with elements that seem beyond their control.

Chapter 2: Business is War. Meet the Enemy.

To successfully deliver the next killer app, you must fight many battles, the easiest of which is with your debugger. Whether you're a systems architect, project manager, team lead, or full-time coder, your ability to do your job and enjoy your pizza without indigestion is going to be continually assaulted by a host of business-induced problems. The first step in building up your defenses is simply knowing your enemy. Consequently, we'll highlight the problems that most software development teams commonly encounter.

Chapter 3: Good Coding Skills Are Not Enough

In gazing at the enemy, it's tempting for many programmers to simply shrug off management problems as not being a part of a programmer's job description. However, this will be of little consolation to you when you're plucking the arrows out of your posterior. Software development is a team endeavor. If you don't work at your level to help combat the problems that threaten your project, you'll all go down together. If you don't do anything but code, here's why you still need additional skills to survive.

Part II: Guerilla Tactics for Front-Line Programmers

Chapter 4: Preventing Arbitrary Deadlines

It's three o'clock in the morning, your system just crashed again, your debugger shrugs its shoulders and asks for a coffee break, your eyes are so bloodshot that they look like a roadmap of midtown Manhattan, and the product must ship tomorrow if you wish to continue your employment. How did you get into this mess? At this point, there's not much you can do about it beyond persistence, excessive caffeine consumption, and simply hoping for a lucky break. The time to prevent this disaster was much earlier in the game. Where's a good time machine when you need one?

Chapter 5: Getting Your Requirements Etched in Stone

Scope creep is not the title of a bad science fiction movie involving mutant gunsights from outer space; rather, it's one of the foremost reasons that software projects are not delivered on time and within budget. If your features seem to continually evolve as the project progresses, or you find yourself trying to provide well-defined functionality from vague specifications, here's how to nail your requirements down firmly, right at the beginning. If they wiggle, use another nail.

Chapter 6: Effective Design under Fire

The only problem with many design methodologies is that it takes a significant time investment to work through the entire process. Unfortunately, out here on the front lines, we typically have a hard time convincing management that we need a little time away from the compiler for sleep, let alone for months and months of abstract drawings that they don't understand. Consequently, at such times we must break the rules and roll our own design approach, using the best of all that we've encountered in the time we're given to work with. It ain't pretty, but it works.

Chapter 7: Practical Estimating Techniques

Arguably the hardest thing to do in our business (beyond finding pizza toppings that everyone agrees upon) is producing an accurate time estimate for any nontrivial amount of code. Furthermore, many unrealistic deadlines arise due to overlooking tasks other than coding that also eat up chunks of time. In the end, if the estimates aren't real, it's the programmers who pay the price at deadline time. Here's a simple approach to help ensure that your next timeline is an achievable one.

Chapter 8: Fighting for Quality Assurance

No programmer worth his weight in cappuccino ever wants to ship a buggy product. It's bad for the ego, bad for the résumé, and bad on the nerves when your telephone rings in the middle of the night. Amazingly, however, the overwhelming majority of businesses who develop software do not hire quality assurance professionals or otherwise institute any sort of rigorous software testing procedures. This calls for a combination of fighting for change and exercising self-defense wherever possible.

Chapter 9: Keeping the Project under Control

Keeping a software development team running like a well-oiled machine takes work from people at every level. Code standards, version control, technical documentation, organization, discipline, and good communications are but a few of the skills required to keep a project on track. It matters little that you've prevailed in your political battles if your project simply implodes due to poor structural integrity. From programmer to project manager, here's how to keep things running smoothly.

Chapter 10: Managing Your Management

If management is to have realistic expectations and a firm foundation upon which they can plan their business strategies, a little retraining is in order. If it were true that those higher up the corporate food chain were immune to the concerns of rank-and-file programmers, the battle would be lost before it began. However, what we're dealing with here is not an abstract concept but is instead real, flesh-and-blood people. Consequently, they can be convinced, directed, inspired, and motivated to do the right things. You simply need to speak a language that they understand. And, of course, let them think that it was their idea all along.

Chapter 11: Corporate Self-Defense

In companies large and small, internal politics can be the most frustrating and disruptive force that you encounter. People with agendas that are quite different from your own can disrupt, take over, or even completely destroy your project. Many programmers have neither the skill nor the desire to engage in political games; however, just as in the martial arts, many methods of self-defense are available that require little more than attention and redirection—and knowing when to duck. The alternative is to become a professional target.

Chapter 12: Controlling Your Destiny

No matter how permanent you've been told your position is, software developers have about as much job security as a drummer in Spinal Tap. Whether you move to different projects at the same job, change companies as a contractor, or hang out a shingle and go into business yourself, there are no guarantees. If you want to keep paying the rent by making the compiler dance, it's up to you to look after your career. No one else will. This means keeping yourself marketable, knowing how to negotiate, and always staying prepared by looking to the future.

CHAPTER 3

Good Coding Skills Are Not Enough

But I just wanna be a programmer! Why do I need all of these non-coding skills? Can't I just sit in my cubicle and concentrate on programming?

Sure you can. In fact, the overwhelming majority of programmers worldwide do just that. Of course, the overwhelming majority of programmers worldwide also have an extremely common set of complaints about their jobs. The simple reality of the matter is that your job is probably not anywhere near as good as it could be, and neither is your software. We've already identified a large number of culprits that appear to be responsible for the problems we encounter, but, when it all comes down to the bottom line, *it's your fault*. Ouch. Can I say that? Well, perhaps, if only because I'm safe for the moment from the sting of a whiteboard eraser.

How can all of the shortcomings in your software development shop—so many of which are typically caused by managerial decisions that exhibit about as much common sense as a lima bean—be your fault? Simple. If you sit on your hands and do nothing, then you're part of the problem when you could be part of the solution. Wait, that sounded a bit like one of those trendy catch phrases. Maybe I've been hanging out in Corporate America too long.

If I'm suggesting that you take a more active role in dealing with the issues you face as developers, I suppose it's not that different from asking you to storm a machine gun nest. Of course, all those years of dealing with maintenance programmers has undoubtedly prepared you better for such a task. Still, to be practical about it, anyone taking risks should have a reason for doing so. In other words, what's in it for you?

What's in It for Me?

Probably one of the biggest hassles in any full-time programmer's career is sacrificing your life to countless hours of unproductive—and very often unpaid—overtime. It's bad enough that you're given a situation where you can't get the job done working forty hours a week. The way most businesses are run, the end result may well be yet another release disaster even if you put in eighty hours a week.

That's not exactly a rewarding experience, particularly if you have to give up your life for it. When we fire up the editor, what we're reaching for is the next killer app. We are artists as much as anything else. To put blood, sweat, and tears into a project (okay, maybe not the former if you don't have to interact with the maintenance programmer) only to have management ship it in a half-baked state can be downright infuriating, and that's with a full night of sleep. I have no desire to work day and night as it is. Doing so on a project destined for failure adds insult to injury.

Along those lines, one of the things that are in it for you as an artist is the ability to ship a better-quality product. Whether your name is in the About box or not, your signature is on every piece of software you ship. We all tend to take a great deal of pride in our accomplishments, so who wants to be associated with anything other than a spectacular success? Do I work for money or for ego? Both. (In that order, for the record, but definitely both.) If you want to be involved in projects that make you proud, you have to do your part to help them survive in the wild.

Actually, I've always had a pretty bad attitude towards companies that take advantage of programmers and expect them to dedicate their every waking minute to the job. Maybe it's because I've been a musician all my life and have seen how nightclubs and other aspects of the music industry tend to pay almost nothing. They get away with this because they know we love music so much that we'd probably play for free and are usually happy to take whatever we can get. A low-paying gig on the weekend is more fun than no gig. Because of this, bar gigs pay today almost exactly what they paid twenty years ago. Really. It's an unfair and predatory practice but is so common that it's become the accepted norm. If you push for more equitable pay, you're simply told that they're not doing anything different than every other venue in town. That's typically true, but it doesn't make it right.

Many software development companies employ this exact approach in dealing with programmers, and for the exact same reasons. We got into this business because we were passionate about programming. We tend to do it at home in the evenings and on weekends just for fun. With the same predatory attitude, these sweatshops take advantage of our love for development and make continual overtime an accepted norm.

I have a friend who is a programmer working in such an environment. In fairness, I must say that he was told up front in the interview that, due to the stock options giving the employees a sense of ownership in the company, they hired only those people who were willing to dedicate above-average hours to the job. Nonetheless, he's been killing himself the past few weeks working late hours. I made some of the usual jokes with him regarding end-of-the-project crunch time and asked when the release date was. His answer floored me, even though it's nothing new. He said there was no deadline; it was simply a corporate culture. If you weren't putting in all the extra hours, you just weren't working hard enough.

When there's an honest-to-goodness crisis, you can count on me each time, every time. I'll be the guy with the sleeping bag next to my desk. Obviously, my friend sees it as worthwhile, and he's a pretty sharp guy for whom I have a lot of respect. However, this sort of open-ended abuse of programmers constitutes a gig that I wouldn't touch with a ten-foot pole.

Consequently, for years now I have employed a somewhat unorthodox tactic for avoiding sweatshops. I live in a major city, and there always seems to be plenty of work out there for my particular skill set. Consequently, when I go out for interviews, I do so with the desire of landing a job that I really want. By the time I actually get down to the normal face-to-face interaction with the company, we've already done a lot of the dance and it's a foregone conclusion that I'm potentially a good fit. They wouldn't bother to interview me otherwise. So, we go through all the normal motions where we each do our best to convince the other that we're something that life is just not complete without.

When it's just about all said and done and things look good, I ask about the kind of hours that they're working on average and if overtime is a frequent flyer in their world. I've found that a good many managers don't want to be honest with you about this because they figure it would be harder to get people to sign on. They're certainly correct. So, just to make sure that I'm not being suckered into a sweatshop environment, after they've assured me that they don't work much overtime I happily agree with the philosophy, telling them that I have enough going on in my life that I like to get my job done in forty hours a week. I then tell them that as a seasoned developer it's my personal conviction that, if you're unwilling to pull the occasional all-nighter at crunch time, you should get out of the business. However, I feel that any company that has a crisis every week and that requires constant overtime is a company with extremely stupid management, and I have no desire to work for such morons.

The truth is that, if the conversation has indicated to me that I'm not the only programmer in the room who curses like a sailor, I use much stronger language than "extremely stupid" because I really want to make a point. Having done so, one of two things usually results. Either they were telling me the truth in the first place about little overtime, in which case we've agreed on yet another topic, or they're lying to me. If the latter, I have just terminally insulted them and there is no way in heaven or earth that they will hire me. Which is exactly my intent. When times are tough, you take whatever gig you have to in order to survive. However, under normal circumstances, there's plenty of work in our business, and life's too short to work for abusive companies.

Does all of that sound patently unprofessional to you? Perhaps it is. Nonetheless, ask me how many sweatshops I've worked in. Now, I spend my nights and weekends living my life while others toil away hour after hour, pushing themselves closer and closer to burnout. I'm a decent programmer, but many folks in this business are much, much better than I. And yet, I get paid as well as the next guy and I work forty-hour weeks. Why? Because I realize that, to have a gratifying career, good coding skills aren't enough.

The ability to consistently meet your deadlines is indeed another benefit that we can gain by looking beyond our technical abilities. Above and beyond the obvious fact that, if you're hitting your goals in a well organized fashion, you're not killing yourself with pointless overtime. Being successful and productive tends to lower your stress level and makes you less likely to be harassed by management. We get up each morning and spend a very significant portion of our days working for a living. If that experience is unpleasant, then simple math tells us that a very significant portion of our lives is unpleasant. Who wants to live like that?

Of course, if you regain control of your programming life, you can spend more time coding and less time putting out fires. I realize that, technically speaking, coding is coding, but that doesn't mean that I enjoy it all equally. My personal preference is to sit undisturbed writing new code on a project that sparks my interest and enthusiasm. I can assure you, I've spent many, many hours coding in scenarios that were nowhere near my preference. So have you. I could have gone to school and learned to do a great many things for a living. I became a programmer because it was a way to pay the rent that was actually fun. If I'm not having fun, I feel cheated. Consequently, I care a great deal about any aspect of my job that could interfere with the enjoyment of my work, for when I'm enjoying what I'm doing I'm giving it heart, body, and soul. That's good for me, that's good for the project, and that's good for the company. I believe strongly in win-win scenarios.

Naturally, one of the things we want to do is work on the cool projects instead of the stuff nobody wants to touch. Who cares if you're using the programming language and environment of your choice if the task you've been given is dull, tedious, and probably destined to never see a real, live user anyway? The cool projects, as you have no doubt observed, tend to go to the people who make an effective effort to get them. That sexy new project has to go to someone. Why not you?

I once worked a contract with a friend doing development on a data entry product that had an extremely complex list of input validations that were different for each state in the country and for each new customer's needs. The approach that they were taking when we got there was to create a new dynamic link library for each customer/state modification. This struck us as a little cumbersome. We then found that their method of doing this was copying the entire source code base for one library, pasting it to a new directory, going into the code and manually changing anything that needed alteration. Above and beyond the volumes of duplicate code, they even approached the positioning of images by changing magic numbers in the call to display the image, compiling, viewing the image, taking a guess at how much it needed to move, and repeating the process.

My friend, being a serious veteran programmer, observed the obvious that what this really called for was a custom screen editor and code generator, coupled with common code libraries. Of course, we could have solved this problem in other ways that didn't require a code generator, but in talking to the other developers we encountered massive resistance to the idea. They felt that, if there wasn't a lot of code floating around, their job security might be threatened. We both take a dim view of such poor ethics but were realistic enough to know that we were swimming upstream in trying to fight it.

We approached the project manager, who was himself a programmer and a good guy. He was newly arrived to this project and not responsible for the mess of his predecessor. He enthusiastically embraced our idea and told us to get to it. In the end, while the rest of the team slogged away copying and pasting code (my friend also observed that .cpp clearly stood for *Copy-and-Paste Programming*), we were off creating a cool new app using the latest version of the operating system, all the new UI gadgets, and anything else that we wanted to play with that we felt would make a better tool.

When it was complete, work that took several programmers three months to accomplish was done in a week or two by a single developer. After we had moved on to new contracts, we heard that the project manager was promoted. When a new manager came in, the developers got together, scrapped the system we built, and returned to the old ways of copy-and-paste programming. Who cares? We didn't. I've long since spent that money. It's my responsibility to conduct myself in an ethical fashion and do quality work; what the company does with it is its own affair. The point is that, although everyone else was working on dull, boring, and tedious tasks, my friend and I were having a blast kicking out a cool app and earning the high regard of the project manager. Why? Because we both pursued talents beyond the technical.

The last reason I list in terms of what's in it for you is no doubt one of the most important. The ability to make better money has a lot to do with non-coding skills. You do work for money, don't you? I suppose I could have pursued different avenues of programming that might make me a buck or two more, but I like what I do. That's a big deal, and without it I think I'd just go back to playing guitar in smoky bars. Life's too short to spend it doing something you hate, no matter how much it pays. Nonetheless, I've been broke many times in my life (many of which had a curious relationship to the amount of time I spent playing in smoky bars), and I don't care for the experience. Money ain't a bad thing, and, if you want me to write code for you, money is required. How much? Every last penny that I can negotiate, of course. The goal is not just to do what we love for a living, but to get paid extremely well in the process. To accomplish both, you're going to need more than just your technical prowess. If you can code in technical utopia and also have enough money to keep yourself stocked up on the latest bleeding-edge gadgets, isn't that worth a little extra effort?

Who Needs These Skills?

How do these various skills fit into the structure of the development team? You may be thinking that much of what we've discussed thus far is of limited use to a production coder and applies more to those who pursue a management career path. Actually, it's never really that simple. I've never met a programmer whose job could be neatly packaged into one tidy little category. In the real world, throughout the course of the project we end up wearing different hats at different times, even if the job description when we signed on was supposed to be nothing but a coder.

Whether your part of the project is large or small, the same requirements apply if you're to successfully deliver your software. Chances are good that you have some additional responsibilities beyond making sure that your code compiles without warnings and doesn't cause smoke to pour out the back of the box. (It's true, though, that I once came back to my desk in the middle of a debugging session to find a fire extinguisher in front of my keyboard. I can assure you that there were no hardware problems. Honest.) If that's the case, you're going to need skills beyond the technical. However, even if you're fortunate enough to do absolutely nothing but code week after week, you still have other responsibilities. At a minimum, for your project manager to be successful in shielding you from the insanities of the corporate world, he's going to need your support.

You're also going to be involved in meetings. If you never go to meetings, drop me an email and let me know who the human resources person is at your company. In any event, you're going to find that you spend much of your workweek doing things that don't require compiling, debugging, or uttering the occasional programmer's expletive.

The size of your team may shift the types of skills you need, but whether it's large or small you've got to be able to cope with the business world in one manner or another. Small teams with a lot of individual autonomy require individuals with good organizational and navigational skills. If you're working in an environment where you're given a task and are then left alone to make it happen, you actually end up doing a lot of project management whether you realize it or not. (You can think of it as just being organized and focused in your work if the "project manager" part makes you twitch.) Whatever you call it, however, you have many of the same duties. You still need to be able to define your requirements clearly, perform adequate design, and arrive at an achievable timeline with milestones arranged along the way, just to name a few. Your compiler won't help you with any of this.

When working on larger projects with multiple teams, you'll often encounter as much corporate fumbling from within your team as you do from without. You will likely have a dedicated project manager and perhaps a structure of technical leads as well, along with a hefty complement of programmers. Political considerations will be much more a factor in this environment, as will issues such as how well meetings are run, the competency of your project manager in partitioning tasks, how much interference you get from middle and upper management, and many of the other things we've touched on thus far. Remember, you're at the bottom of the software development food chain. Very little happens higher up that doesn't have an effect on you, one way or another.

You may also find yourself working in the capacity of technical lead from time to time. Although it's a testament to the confidence that others have in your technical and organizational skills, this can be a thankless job with great potential for burnout. A technical lead is often nothing more than a project manager with limited scope who carries a full coding load. In other words, not only do you get to do all the work you normally do as a developer, much of it technical and therefore enjoyable, you also get to handle the managerial tasks that are relevant to your team. If it sounds like you just inherited a considerable amount of overtime, you're probably not far from the truth. The trick to working this position with any degree of success, which includes avoiding burnout, is to realize that you can't be a manager at any level, not even the team lead, and get a full day's worth of coding in. Depending on how large your team is and how much organizational work you'll have to perform, you should take your normal level of coding assignments and knock off a quarter or perhaps even half. Unfortunately, technical leads are not always given the power to make such decisions, which is why it's often a real burnout inducer.

Of course, if you have a one-programmer project (and that happens a lot in the business world), you're the project manager, team lead, and coder all rolled into one. If you thought that technical leads had a workload, you'll just love this one. Of course, there are some significant benefits to being a one-programmer team. With no other team members to distract you or call you into endless meetings, you might actually get some coding done. However, never forget that there are always going to be managers above you. What they're called is irrelevant. Any way you shake it, they're managers and that involves all the normal issues of politics, bureaucracy, their effectiveness in dealing with their own management, and all the rest. Additionally, just because you're the only programmer doesn't mean that it's wise to short-circuit the requirements gathering, design, or estimation phases. The rules don't change based on the size of the project, although experience tells us that, if you're a one-programmer team, the chances are good you work in one of those places where they expect to see code flying off your fingertips nonstop. Trying to get a process in place is even harder when you're the only one there.

Taking Control of Your Time

To be successful—and, even more importantly, to be recognized as such by those you work for—you have to get the job done. This sounds too obvious to mention, but sometimes it's easy to overlook the obvious. It's important to keep in mind that business people pay you because they want you to produce something. If you really want to be good at your job, there's more to delivering the goods than coding. You have to keep in mind the end goal of the system you're developing and what it's supposed to accomplish, and do everything within your power and the scope of your position to see it through to completion. You may or may not be recognized for your extra efforts. You may not even want to be recognized, for a variety of reasons. Nonetheless, your ultimate reward will be in delivering quality software, on time and within budget, without overtime, without stress, and without any other nonsense you can avoid. Make this happen, and you put yourself in a better position for the next project that comes up. Everyone loves a winner.

At every level of development, one of the constants is the need for effective time management if you wish to meet your deadlines. Approaching software development in a scattered and disorganized manner is going to significantly diminish your results and increase the amount of time it takes you to get them. Along with that comes a higher level of stress as you've never really quite got a handle on what's going on. This tends to leave you feeling rather breathless and with the nagging suspicion that you're always running behind. It's probably a correct assessment.

I once knew a project manager that actually oversaw several development efforts. This person always seemed to have several balls in the air at any time. His office looked like a whirlwind of file folders, stacks of paper, and various boxes of uninstalled software, and there were probably a couple of chew toys from his dog in there somewhere as well. He constantly had a harried look about him as if he were somewhere on the border between not being able to cope with it all and the sheer terror that someone else was going to come yell at him. All of his projects were behind, and he spent half his time dealing with customers who were upset about it. This, of course, didn't help free up any time for him to solve the problems.

In short, this was one of the most disorganized managers I've seen. Little wonder that his projects were a mess. In fact, what thread of cohesion that actually did run through his various teams was the result of the personal initiative of his developers, who wisely saw that they would get no support from their manager and consequently took matters into their own hands whenever possible. The interesting thing about this guy is that, not only was he overbooked as it was, he never hesitated to take on a new project whenever he could get his hands on one. Could he have handled this kind of workload efficiently? Probably not in forty-hour weeks, but it didn't have to be the disaster that it was. It all comes down to organization. He didn't know how to keep his own ducks in a row, had no skills at planning or running a meeting, and interacted with his developers only in a crisis-driven mode, dashing out in a panic to tell them of the latest fire that they had to work late to put out.

With better time management skills, he could have taken control of the various projects, avoided being yanked from one crisis to the next, and perhaps even have delegated a little. Such things would have settled his projects down tremendously. What's that you say? It's not your problem because you don't want to be a project manager? You're just a programmer? Well, who do you think he had working for him? If your manager is a mess, and many of them are, you're going to need all the skills you can get purely for self-defense.

Enhancing Design

System design is another area in which it's handy to have some facility in something other than compilers. It is certainly not a given that a good coder is naturally a good software designer as well. Although obviously related, they are two completely separate disciplines. However, you don't have to know how to code to work in an architectural capacity, and you don't have to have design skills to write source code. However, you do need to have a grip on the design side of things before you start writing that source code. If you just shoot from the hip and don't think your way through things on a small scale the same as you would for larger tasks, you're likely to encounter difficulties either halfway through what you're coding or the first time someone else has to interface to your code. We typically think of design in terms of mapping out the entire software system, but, when you get down to it, you should always think before you code. Even if management is inclined to believe that you're daydreaming rather than working.

One of the many reasons you need some facility with design is that, to meet your deadlines, you're going to have to have some skills in estimating as well. It's true that an estimate is of little importance if you're given the date before you're given the assignment, but sometimes you'll have a manager who asks you how long a task will take and actually pays attention to what you say. If you can't cook up a good estimate, you're not only setting yourself up for failure, you're doing it to your manager as well. I've found that in general it's a bad thing to make the person who is responsible for your paychecks look stupid to their own boss. People are funny that way.

Improving Interaction

One of the significant benefits to possessing more than merely technical skills comes when you learn to improve your interaction with others. Sometimes it's courtesy, sometimes it's being able to deliver the goods, and sometimes it's just plain old politics, but it's always a beneficial thing that comes back to you. When you come in to the office each morning, you don't deal with a highly specialized class of sentient office furniture. You deal with people. Okay, I did once know someone who spent an inordinate amount of time talking to his furniture but I gave him the benefit of the doubt and chalked it off to sleep deprivation. If you keep in mind that you're dealing with real, live, flesh-and-blood people instead of nameless, faceless coworkers, you're going to have a much better time of it in the business world. The better your people skills, the better your chances of getting what you want, whether it's a new computer, a new project, or more money. If you can also learn to be bilingual and speak the language of business people, there's no end to the enhancements your programming career can experience.

For those of us who take our programming seriously, sometimes just the ability to bring better software to life is reward enough. Interpersonal skills help tremendously in this area as well. If you have great new ideas on how your software should be designed or how a particular chunk should be coded, you still have to be able to sell it to others if you want to make it a reality. Proposing new ideas successfully requires more than flowcharts and whiteboards. Decisions are often made not because the facts overwhelmingly pointed to a particular solution but rather due to the charisma of the individual making the presentation, whether it was a formal meeting or just a persuasive conversation in the hallway. Don't feel as if you're really overflowing with charisma? You'd be surprised how much of that can be an acquired skill. We weren't all born movie stars. Many times, attention to the details of how things work in the business world and learning a few navigational tricks are all the tools you need to gain the respect and admiration of your peers and management. Many programmers feel that they'll never have much luck in the persuasion department because they weren't born natural orators. However, if you learn to speak the language of your audience, understand the things that motivate them, and position yourself appropriately, you'll be surprised how often you win. We'll be going into these things in more detail later, but I'll touch once more on a recurring theme here: you can't win if you don't try.

Probably the bane of programmers and cubicle dwellers everywhere is the dreaded meeting. Some weeks it feels as if all you've done is travel from one meeting to the next. Sometimes it's true. Ironically enough, many of those meetings will be rants from higher-ups who wonder why the heck we're so far behind on our schedule. You're probably not in charge of most of the meetings you attend and therefore have to suffer through someone else's poor skills at organizing and running such gatherings. There's only so much you can do in that scenario, but you can help expedite the process at least a little. Further, if you decide to get serious about your skills in meetings, others will notice and a small groundswell may result. A meeting run by an inept manager can be put back on track, shortened, and made more productive when just a few of the attendees understand some of the basics and are assertive enough to help the process along. If you're actually responsible for reducing the number or duration of meetings at your company, you'll probably become a folk hero among your peers. (Who knows? They may even name a conference room in your honor.)

Getting What You Want

Something that's not really as obvious as it seems is knowing what you want out of life. The truth is that a very large number of people in this world just don't know exactly what they want. Just as it's impossible to meet all the expectations when developing a piece of software with poor or fuzzy requirements, so too is it true in life. That includes your career. If you don't know very specifically what you want, you'll have difficulty achieving it and probably wouldn't recognize it if you did.

My observations have led me to believe that a large majority of programmers got into this business much as I did: I started programming for fun, got hooked, and decided that it would be a cool way to make a living. I then went out and got a job. What exactly was I looking for in a programming career? In retrospect, I had absolutely no idea. I just wanted to get paid to write code.

Having put a number of miles behind me by now, I have a much better idea of what I want for two reasons. First, I've had enough experience to see what's out there, what I like, and what I passionately wish to avoid. Secondly, and I think even more importantly, is the fact that at one point I sat up and realized I was working with fuzzy requirements and decided to do something about it. I actually spent time and gave serious, detailed thought to just exactly what I wanted in my programming career. I then set out to accomplish it.

Naturally, it's much easier to get what you want when you know what it is, and I have had a very enjoyable career thus far. I've spent time doing the types of things I wanted to do and have been paid well for it. This is not because I'm any kind of superstar, one-in-a-million programmer. It's simply because I detailed my goals and desires and then set out to accomplish them in exactly the same way that I would approach turning a requirements document into a good design and ultimately an implemented product. That involves little more than taking one step at a time, always with an eye to the future. Of course, from time to time, I revisit my desires and tweak the spec where necessary, as what I want tends to change. I also spend a fair amount of time reviewing where I am, where I've been, and how things are going so far in my efforts to meet these goals. That helps me to make the necessary course corrections.

Many of us tend to spend our lives on automatic pilot to one degree or another. I'm sure if you took a little time to yourself and gave it some thought, you could come up with a pretty decent list of things you'd like to change about your current job and perhaps your career or life in general. That's a step I'd encourage you to take. Be specific. Be very specific. What you're defining is the perfect world. Don't leave anything out, even if you think it unlikely to accomplish. Once you've done this, you'll have a decent requirements document from which to work.

The next step is to come up with a decent design doc. Take a look at where you are in your career and start brainstorming, just as you would in a design meeting, about how you might get there. Unlike as in software design, in this exercise it's acceptable to leave some questions unanswered. You may not see the solution at the moment, but new input or opportunities could come out of the clear blue sky at any point in the future to help you chart a course for that particular goal. Keep it on the requirements list. For all the other items, you'll end up with at least a beginning strategy and plan of action. Although this doesn't agree with all of the true software design methodologies, in the real world design tends to get tweaked as we go, benefiting from what we learn and steering around problems as we encounter them. So too will your design doc that you create for your programming career be modified as time goes on. I've heard it said that no battle plan survives contact with the enemy. Allow for that flexibility.

Once you've got a good design (which in any business is simply a detailed road map for how to get where you wish to be), it's time to give some attention to implementation. You now know, in great detail, exactly what the perfect programmer's life is, at least for you. You have a strategy in place to make this a reality. The next logical step is to start taking the necessary steps to realize your desires.

When you have both your requirements and design docs sitting in front of you, you'll quickly realize that to meet your goals you're going to need a few more tricks up your sleeve than just knowing how to avoid compiler and runtime errors. That's where we're headed next. You already know how to code. You're good at it, or you wouldn't have a job. Now it's time to hone your skills in all of those other areas so you can effectively combat the insanities of Corporate America and achieve your objectives. With any luck at all, the result will be more coding, more fun, and fewer encounters with nervous little dogs. The last time I saw him, the watchman's partner was sporting a camouflage collar and jacket and was having a whispered conversation with the maintenance programmer about inventory control.