Extracted from:

New Programmer's Survival Manual

Navigate Your Workplace, Cube Farm, or Startup

The Pragmatic Bookshelf

Dallas, Texas • Raleigh, North Carolina

The Pragmatic Programmers

New Programmer's Survival Manual

Navigate Your Workplace, Cube Farm, or Startup

Josh Carter

Edited by Susannah Davidson Pfalzer

New Programmer's Survival Manual

Navigate Your Workplace, Cube Farm, or Startup

Josh Carter

The Pragmatic Bookshelf

Dallas, Texas • Raleigh, North Carolina

Our Pragmatic courses, workshops, and other products can help you and your team create better software and have more fun. For more information, as well as the latest Pragmatic titles, please visit us at *http://pragprog.com*.

The team that produced this book includes:

Susannah Pfalzer (editor)
Potomac Indexing, LLC (indexer)
Kim Wimpsett (copyeditor)
David J Kelly (typesetter)
Janet Furlow (producer)
Juliet Benda (rights)
Ellie Callahan (support)

Printed in the United States of America.
ISBN-13: 978-1-934356-81-4
Printed on acid-free paper.
Book version: P1.0—November 2011

For Daria and Genevieve.

[*White Belt*] Your effectiveness as a programmer is, to a degree, gated by your ability to work with your team.

In most cases, you'll be one programmer among several, and like rowing a boat, the sum of efforts depends on everyone rowing the same direction. This is trite and easy to say. In practice, coordinating a programming effort is less like rowing a boat and more like herding cats.

Good programmers are opinionated and strong-willed. Ask any two programmers to solve a problem, and they'll solve it differently. Yet the product depends on several (or many) programmers working together and creating a cohesive whole.

Divide and Conquer

The average product requires many talents, and every programmer has unique skills, interests, and expertise. Understanding what *you* bring to the table is essential to contributing the most to the product.

When you're just starting out, your industry programming experience is zilch, but you have enthusiasm. OK, let's work with that. Is there a part of the product nobody else has experience in, either? This is usually in the gnarly parts; a common example is software packaging and field upgrade. It's a hairy mess, and nobody wants to touch it. It sounds like a good place to dig in and earn your stripes.

Taking on gnarly problems is how you build your expertise and credibility within the team. Sticking with easy parts of the project doesn't. (Balance this, however, with some early wins, as discussed in Tip 17, *Be Visible*, on page ?.) Look for credibility builders when the team is divvying up work; where is an unmet need that you can tackle? Consider the following group planning meeting:

> *Manager: Any volunteers for the 3D flying icon rendering tool?*
>
> *Dave, Emma, Frank (in unison): Me.*
>
> *Manager: Now what about the part that imports 1986-vintage DXF files and converts them to our current format?*
>
> (crickets chirping)

You: If I take the lead on this one, can someone back me up if I get stuck?

Frank: I wrote some DXF stuff a long time ago, and then my cat choked on the floppy disk containing the code, and I'm still very upset about that, but I should be able to help.

You: OK, I'll take it.

Now you have a gnarly project and also saved Frank—who would have gotten the assignment otherwise—from a month's worth of lament over the fate of his cat.

Pair Programming

Sometimes you set out to tackle a problem and it tackles you instead. No worries, it happens to all of us. Pair up with another programmer and try again. Often, a second set of eyes and a fresh perspective are all it takes to make the breakthrough you need.

Pair programming is effective enough that some teams *always* pair, one typing and the other observing and commenting. Other teams will work individually and pair only when someone gets stuck. I'll sometimes do a hybrid, with each person on a laptop in a common area, each attacking a different aspect of the problem.

If your team doesn't have an established practice for pairing, it's usually easy to get some time from a co-worker. Here are a couple tips:

- Try to find someone with experience in the area you're working in. ("Frank, I heard you know a thing or two about DXF. Could you watch over my shoulder for a bit and advise me on this gnarly part of the DXF importer?")
- If you need more than a few hours of time, run it by your manager. ("Can I borrow some of Frank's time? I'm in a tough spot and could use his help.")

The only unacceptable practice is floundering on your own, not asking for help. Yes, some problems require tedious investigation that takes a long time. Recognize, however, when you've gone past the point of diminishing return and it's time to get another set of eyes on the problem.

In the Center of Things

My very best working environment was precisely what most programmers dread: a low-wall cube in the middle of the office. I was a couple years into my career, and I wanted to sit with the main software team, so when a cube opened up—any cube—I grabbed it.

My new cube was right in the middle, by a common area. Interesting conversations would pop up, and I could participate effortlessly. I got involved with many parts of the product, gaining tremendous expertise and credibility very quickly. As it happens, the easiest way to get in the middle of things is to *physically* get in the middle of things.

I've been on the opposite end of the spectrum, too, working in my basement office a thousand miles away from the rest of my team. That was the worst job of my career—no matter what, I couldn't get into the action and participate in the design of the product.

Americans value the corner office with the window, but my own experience says the Japanese have it right: the most prized location—the location with the most influence—is the one in the center of things.

Concentration and Interruption

Collaboration necessarily involves both dividing work between individuals and working together. In most teams, this is a fluid thing, with unpredictable mixing of alone time and collaborative time. It's where those meet that can cause friction.

In programmer-speak, the "context switch overhead" required to flip between modes can be very high, depending on the person and the task at hand. Programming often requires intense concentration, and getting into that state—the *flow* or *zone*—takes time. This is why some companies give programmers offices to minimize interruptions.

On the other hand, collaboration requires interruption. This is why other companies put programmers in a large, open room to maximize collaboration. Both philosophies have truth to them, but you need to be conscious of the interplay between concentration and interruption to perform well in either environment.

First, when another team member is "in the zone," try not to bug them. Closed office doors or headphones are a good clue. When you need some in-the-zone time, turn off email, instant messaging, and your cell phone. If your company culture allows it, work from home or a coffee shop.

Second, get used to interruptions. There's tremendous value in collaboration, and shutting your office door shuts *you* out of the interactions going on around you. There are a number of productivity techniques you can use to minimize the impact of interruptions; *Getting Things Done* [All02] is a good place to start.

Actions

Here's an easy one: most of what we've been talking about boils down to *talking*. If you're in an office, make it a point to chat with another programmer each day. Coffee, lunch, and—in start-ups—dinner should provide ample opportunity to chat. If your company is physically distributed, get on instant messaging or the phone at least once a day.

The other part is dealing with interruptions. Take a survey of some productivity techniques—ask your co-workers, read some blogs, check out David Allen's book[6]—and pick one that you think meets your needs. Try it for four to six weeks; that's how long it takes to establish a new habit. If it's still more hassle than it's worth after that time, chuck it.

6. *Getting Things Done: The Art of Stress-Free Productivity* [All02]