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 attitude affects both your productivity and your future all day, every day.

In the 1990s Bare Bones Software released a text editor called BBEdit with the tagline "It Doesn't Suck." It retains the tagline to this day. Truly brilliant marketing. Who's their market? Programmers.

Programmers are a pessimistic and sarcastic lot. The vast majority will tell you about 100 things that suck for every one thing that doesn't. The highest praise a programmer will give a product is, "It doesn't suck."

Pessimist programmers are in good company. I've read various reports saying the vast majority of projects fail,[1] 80 or 90 percent. Adding insult to injury, it's not the good 10 or 20 percent that succeed; it's some hodgepodge of good and bad. It almost seems that bland to downright crappy products are more successful, on average, than really good ones. When a programmer says such-and-such sucks, chances are she's right.

The gambler would simply say everything sucks—playing those odds isn't rocket science. But here's the problem: you don't win anything for picking losers.

Balancing the Odds

When I was an industry newbie, my first manager told me, "Being a pessimist is the easiest thing in the world. It's the easy way out. It's much harder to be an optimist." Those words—and that challenge—changed my tone and my career.

When you shift your perspective from "It sucks" to "Wouldn't it be cool if…," you shift from a defeatist mind-set to a creative mind-set. The best you can do from a *sucks*

1. The most common source is the Standish Group's CHAOS Report, http://www.standishgroup.com/. However, there are numerous challengers of the CHAOS report, e.g., http://doi.acm.org/10.1145/1145287.1145301 and http://dx.doi.org/10.1109/MS.2009.154 and others.

attitude is create something that sucks slightly less. From a *cool* attitude, you can create something completely new.

Creating new things is a practiced skill. Starting from school, you've been trained to follow examples and create small bits of new work. With practice, you'll create larger works and also deviate further from prior examples into work wholly of your own imagination.

Your rate of learning is largely determined by how much you want to push yourself, and that push comes from your attitude.

Structure for Creation

Robert Fritz, in his book *The Path of Least Resistance* [Fri89], identifies two *structures* for how we interact with our world.

Reactive/Responsive

This is our default structure, where we react to circumstances. To a programmer, this could go something like this: you want to reduce the number of bugs in the product (responding to testers), and likewise you want to get the product out the door (responding to management pressure). Pulled between these forces—forces pulling in opposite directions—you make fixes that are good enough to fix the bugs without jeopardizing the schedule.

This is also known as *fire-fighting mode*. You may put out the fire today, but you never get around to addressing systematic, big-picture problems in the product.

Creative

Rather than immediately responding to present circumstances, in the creative mode you acknowledge the present state and visualize a better future state. For example, you visualize a product that is more modular and therefore easier to test and reason about. Guided by this creative vision, you go into the code looking for opportunities to make it modular as you're fixing the bugs. (See Tip 7, *Improve Legacy Code*, on page ? for some advice on *finding seams* in legacy code.)

What's the difference? In the long run, the reactive/responsive bug fixing will leave you with a code base that's even

more gnarly to maintain than when you started. The *hard* bugs will just be patched over, not really fixed. In the creative structure, on the other hand, you have a guiding vision that will improve the code base—including the hard bits—over time.

Creating a vision engenders—in fact, it *requires*—a positive attitude. You can't create anything out of pessimism. It also requires hard work to bring that creative vision into reality, but the work is no harder than you'd be doing anyway. The bonus is that when you're driving toward a vision of a better future, the hard work is fulfilling in a way that you don't get from reactive work.

Evangelism

The next level of creative vision is bringing others along for the ride. Early in your career, you may be on the receiving end of technology evangelism. I would hope so, because it's tremendously fun to *believe* in what you're doing. Later, you'll create and evangelize on your own.

Evangelism is tremendously underrated in the technology world. People think of computers as, well, boring machines, so what's there to get excited about? But think about your early, wide-eyed play with computers: didn't you dive in with vigor and passion? Of course…that's why you're here today, reading this book.

An evangelist reignites that passion and directs it to a vision of something new and cool. (It may be a vision that would be profitable once created, but the focus of evangelism is more about the spirit than the dollars.) This isn't just the job of CEOs and marketing folks; programmers can be just as talented at evangelism as anyone. It's that conversation that starts with "Wouldn't it be cool if…,"

There's absolutely nothing deceitful about evangelism; it's just painting a picture of a better future state so that others can understand it and believe in it. Terence Ryan's *Driving Technical Change* [Rya10] is a great resource for learning this skill. Guy Kawasaki, one of Apple's early evangelists, gives a big-picture view of evangelism in his book *Selling the Dream* [Kaw92].

Now "It Doesn't Suck" doesn't seem like such a brilliant product tagline after all. (To its credit, it still makes me laugh.) Think hard about the reputation you want to establish. Visualize—yes, create—in your mind the *you* five years from now. Are you the naysayer, or are you the one inspiring people to make something cool?

Actions

Think about your most inspiring teacher (in school or otherwise). What was it about them that made them so great? Write in your journal characteristics of how they talked about their subject and how that inspired you to think about the subject.

Watch some recorded presentations of great product announcements, for example Steve Jobs introducing the Macintosh. Observe how the presenter doesn't sell the product so much as the *vision* of the product—the *dream* behind the product. That's evangelism at work.