Extracted from:

The Developer's Code

What Real Programmers Do

The Pragmatic Bookshelf

Dallas, Texas • Raleigh, North Carolina

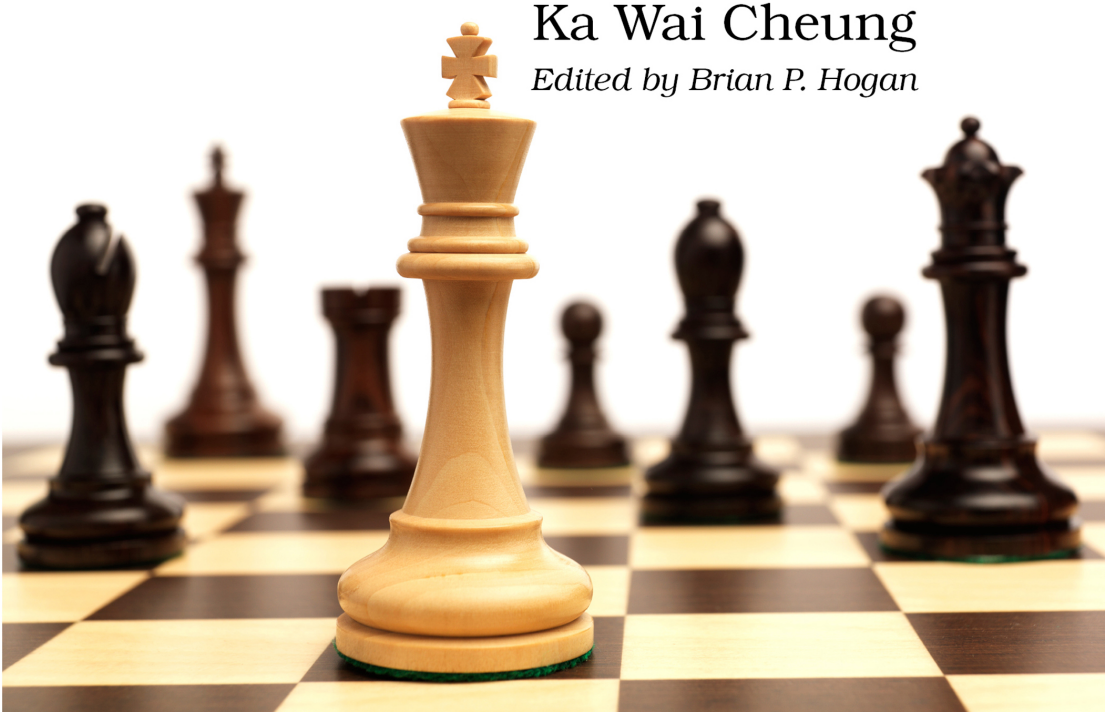
The Pragmatic Programmers

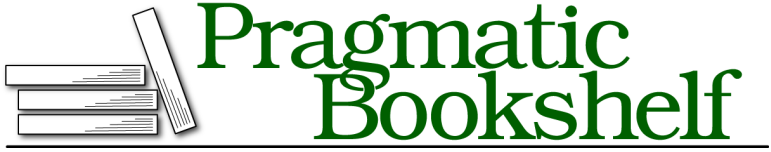
The Developer's Code

What Real Programmers Do

Ka Wai Cheung

Edited by Brian P. Hogan





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*.

Cartoons courtesy of Mark Anderson, reproduced with permission of the artist. *http://www.andertoons.com*

The team that produced this book includes:

Brian P. Hogan (editor)
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: 978-1-934356-79-1
Printed on acid-free paper.
Book version: P1.0—February 2012

Essay 27

The Simplicity Paradox

What makes complexity a strange phenomenon is this: *Everyone. Loves. Simple.* That's why people say "I just want things to be simple." Who says "I just want things to be complicated"...*ever*?

I decided to find out. So, I looked it up on Google.

As of this writing, if you Google the phrase "I want things to be simple," you'll get approximately 954,000 matching results. There is *one* unique matching results for the phrase "I want things to be complicated."

One. The only unique matching result? A blog post that I wrote about this very subject in October 2009. Extract myself from the annals of recorded human civilization, and apparently no one has ever wanted or even thought about the idea of voluntary complication.

Then, why do we run into this Jeffersonian problem of complexity when we're building our own stuff? Why do the things we produce often wind up festered in complication? How do so many well-intentioned pieces of software matriculate from simple idea to functional nightmare?

Simple Products Can Actually Be Hard to Build

Most ideas, simple at the surface, are viciously complicated when we get into the details. Ideas, at a high level, are always simple. Every business idea must be accompanied by the elevator pitch: sixty seconds to get the message across from beginning to end. We can't pack complexity into a sixty-second description.

When ideas start feeling complex, we leave the comforts of Idea Land and enter the naked reality of implementation. Once we dig into the details, we discover where all the holes in logic are. That's just the nature of *detail*. An idea that hasn't been thought through completely (read: most of them) has

little chance of surviving Complexityville at this point. Rather than rethinking the idea altogether, it's sometimes easier to plow through the problems with head down and blinders up. Half-baked decisions are made, and features are added all for the sake of preserving the sanctity of the "big idea." Then, complexity festers.

Simple Sometimes Seems Like *Not Enough*

If everyone likes simple software and most software isn't simple to build, it would appear that the sweet spot for good software would be *both* simple to use *and* simple to build. It's a win-win for both user and developer. But that kind of software rarely exists in our world. There has to be something more to this mystery.

The answer lies in our own fear of inadequacy. When we build something simply, it doesn't feel like...*enough*. We convince ourselves into believing our customer isn't getting his money's worth. A simple thing that's *also* simple to build feels valueless. An idea that's easily implemented is rarely considered a "big idea" at all.

Venture capitalists don't throw millions of dollars at simple ideas. They throw all that money at the Donald Trump-esque superlatives. Is it best-in-class? Is it innovative? Is it cutting-edge? Oftentimes, these are just other ways of saying an idea is *complex* enough to be worth its weight.

Herein lies the paradox. From a builder's point of view, we often equate the worth of software we build to its complexity, and more complexity equals more value.

The view from the other side of the mirror is different. The reality is 90 percent of our users use only 10 percent of the features built in the average enterprise-level software. When users can't find the few functions they need because they're buried among the many features they don't need, they either take it out on their own perceived shortcomings or blame it on the software itself. While builders and stakeholders see simplicity as the shortcoming, users see complexity as the shortcoming.

At my company, the natural urge to complicate is something we resist constantly. We have to re-sell and re-pitch simple to ourselves all the time.

Countless internal arguments about features in our own software end up with incredibly simple solutions. Sometimes the UI just needs a small tweak in text. Other times, it's just a re-organization of links. Sometimes we'll argue for hours about a new feature before ultimately deciding the feature just isn't worth the complexity it adds.

The lesson is this. You don't have to "merit" lengthy hours of feature discussion with an equally large amount of feature additions. It's natural to feel that the amount of time you spend on something should parallel the amount of measurable output you put into the product, regardless of the benefit of that new feature. But free yourself from that debilitating thought. Once you've let go of the vulnerable feeling that simplicity cheapens your worth, you can finally get on with building good software.

A simple solution shouldn't be thought of as "not enough" of anything. Sometimes it is exactly enough of everything.