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 4

The "Ivory Tower" Architect Is a Myth

I've never liked the idea that technical architects should stop coding.

In physical architecture, architects perch in an ivory tower, living in a world of only planning. They don't hammer in the nails or solder joints together. Requiring architects to do the physical work of drilling holes and laying concrete is simply impractical. Architecting and developing are two distinct career tracks.

The Corporate Ladder Leads to Less Code

In our industry, we work our way up to the role of a technical architect by actually developing—by doing the "physical" work of building applications. But in most organizations, as we move up the software development ladder, we write less code. We immerse ourselves more with planning than with discovering the problems on the front line. We concern ourselves more about an overall vision and less about the intimate details of code. As an industry, we've held on to the notion that architects should plan and developers should develop.

This creates the false perception that once we've reached a certain level, programming is no longer where we're most valuable. Just leave the dirty work to the junior developers! At the same time, it pushes lower-level programmers away from thinking about the overall goals and direction of the project. They're asked to concentrate just on the implementation. The architect-developer model makes both parties less accountable for the application as a whole.

When we split up roles into the somewhat arbitrary hierarchy of those who think about the technical "big picture" and those who think only in `if` statements, `for` loops, and markup, we fracture two disciplines that belong intimately together.

Pure technical architects can take a guess at the best architecture, the best design pattern, or the best practice. It's only when they're knee-deep in code that they discover where the real challenges exist. At the same time, the developer who isn't given the reins to think at a high level also doesn't get the chance to voice a second opinion. Often, it's the guy who's doing the actual implementation who can see the bumps ahead most clearly.

We've taken the architect-developer analogy too far. The corporate ladder in the software industry needs a better analogy.

To build a building, architects architect and developers develop. Traditional architects know how to create elaborate plans and specs in fine detail. But they don't build. It's simply not reasonable. The separation between those who think at a high level and those who work in the trenches is largely for practical reasons.

In software, it doesn't have to be that way. Great developers can live both "in the trenches" and "at a high level" at the same time. Sure, an architect might spend most of her time thinking at a high level, but she should be involved in development a little to get the full picture.

Making Time for Code

In many technical organizations, what I've proposed thus far just isn't feasible. Most technical architects have a full-time job in meetings with other groups in the company. They're often brought in to client phone calls to discuss all kinds of technical challenges that face a software project. Where's the time to code, anyway?

A few months after I started one of my full-time web developer gigs, we hired a new senior architect. Adam came in and set a very different tone among our group of young web developers. Despite all the normal duties he took on in his role, it was clear his passion lived in code. Immediately, I felt like I was talking to just another programmer, albeit one a lot smarter and wiser than I. Our architect-developer relationship became my personal mentorship.

On our first project, an extranet for a major law firm, Adam mentioned something about code generation. To me, it sounded a bit sci-fi. However, as I would soon find out, the underlying server-side objects, queries, and methods I was going to write by hand were mainly algorithmic. We could largely deduce them from the extranet's database schema. Instead of plodding ahead with a brute-force approach to development, Adam suggested I focus on building out custom forms and screens while he began writing a code generator. He did this for a few weeks on his one-hour train ride to and from the office each day. In a few weeks' time, we had a rudimentary but powerful tool to generate a lot of the stuff I would've written manually.

Whatever time we lost with our little divergence into code generation, we quickly recouped as we began using the code generator. Each time we changed our database schema, he'd run his little magic app that would rewrite all the code I needed to keep building the application. It wasn't long before the effort Adam made in writing his tool more than paid back the cost in writing it. And this was a tool we could use again and again.

So, while I was still very much the lead developer on the project, Adam had a large stake in the development. If I needed some different bits of algorithmic code, he'd work on adding those features to the generator on train rides home. The next day, I'd have a fresh set of code that I wouldn't have to ever write by hand again. Much of what I learned in those first few months I keep with me today.

As you work your way up the programmer chain of command, from developer to architect, don't forget that code is the glue that binds each role. It may not be a train ride. Maybe it's an hour or two you can devote, at work, to only writing code. You'll see ways of committing yourself to code-only time in Essay 23, *Create "Off-Time" with Your Team*, on page ?. In the end, regardless of where you are in the development hierarchy, keep coding. It's where you're most valuable.