Alex Groce (agroce@gmail.com), Northern Arizona University

Federico Biancuzzi and Shane Warden's *Masterminds of Programming: Conversations with the Creators of Major Programming Languages* is a treasure. The book consists of interviews with the creators of, in order, C++, Python, APL, Forth, BASIC, AWK, Lua, Haskell, ML, SQL, Objective C, Java, C#, UML, Perl, PostScript, and Eiffel. Each chapter asks similar, but not identical questions, and the above-mentioned masterminds, including Larry Wall, James Gosling, Brian Kernighan, Bertrand Meyer, Robin Milner, Simon Peyton-Jones, Guido van Rossum, and Bjarne Stroustrop give a wide variety of answers. Some of the masterminds are charming, and many are contentious, even cranky; they are also, almost all, full of deep insights into the deepest problems of software engineering. This insight comes in two forms; first, programming languages are the mechanisms by which software engineering solutions are almost always produced. Second, perhaps even more importantly, creating and evolving a widely-used programming language is a heroic, Herculean, critical software engineering task. All of these masterminds have succeeded in a massive software engineering task; they are not mere ivory tower thinkers about software engineering, but have, in some cases, entire lives shaped by a single, extremely complex, software project. More on that key point below.

Rather than talk about this book, I'd like to whet your appetite by quoting it at some length. Almost all of what follows comes from a single chapter, not to say *which* chapter, as Woody Guthrie might put it.

> "Instead of learning from experience, they heedlessly start with something they believe is fundamentally new. In reality, very little has changed. As in the fashion world, there is much ado about next to nothing. In something as trivial as clothing this may be acceptable, but with the size of our investment in software this is wasteful, expensive, and absurd."
>
> "Developing software is not rocket science. Look at the 5–10 million people who call themselves software developers. Very few of them really do anything creative or fundamentally new. Unfortunately the outside world thinks that programmers are creative and brilliant people, and that's far from reality."
>
> "The computing field has a lot of people that think very well of themselves and seem to forget that there is any past to build upon. A lot of people keep reinventing things that have already been discovered."
>
> "Reusability is good, but it's not really the main goal in most systems. It's really hard to build good reusable libraries — most programmers can't do it very well and shouldn't be encouraged to try. It's a separate task from system building."
>
> "To quote Elrond, 'My memory reaches back to the Elder days, and I've seen many defeats and many fruitless victories.'"

> “Or the Star Trek movie effect.<REDACTED>: Now that one certainly is worth lots of formal studies.”
>
> “In general, people that work with software don’t read books or manuals. It’s only at universities that people really read, if anywhere. Saying that people use books and manuals while they work is just a myth.
>
> I have written a couple of books, and I am very happy that people buy my books, but as for all other books, they don’t read them.”
>
> “Implementation, to paraphrase a famous saying, is just design carried out by other means.”
>
> “The price to pay is that you have to renounce all the pleasures of life.”

The book has a lot more pithy bits where those come from. Last month we looked at a book by John McPhee, and I used a somewhat strained metaphor, to be honest, to motivate including it in *Passages*. But there is a common point here. When you deal with a John McPhee book, what you're paying for (the cost of the book, the greater cost of your time and attention) is a perception. A floating eye, and a mind that receives the world, and then presents it back to you; moreover, that eye and that mind are of a very high quality. We care about writers and artists, and scientists, because their eyes and minds are different, in a good way. That difference is common to all the masterminds interviewed in this month’s book. Like Boswell’s *Life of Johnson* this is a book that rewards picking it up and flipping to a random page and reading until you find something that enlightens you, amuses you, inspires you, or even that infuriates you (in this book; if something in Boswell infuriates you, you must be an odd kind of character).

The masterminds don’t agree with each other, in many instances, and you may not agree with them. Some of the masterminds take (cheap) shots at other masterminds, even. But, on some points, the masterminds agree. There *are* common threads of understanding, expressed differently, that run through the book, and these ideas amount to a view of The Problem(s) of Software Engineering shared by some of the most creative and battle-tested software engineers in history. What are those insights? I won’t tell you, any more than I’ll tell you which chapter(s) the quotes above come from. In this case, the *way* the underlying agreements (and points of departure) emerge is as important to grasping the ideas in question as any bald-faced presentation of the “point.” The masterminds express the core ideas differently, but I think if you read the whole book, you will agree that they do share a common understanding. They offer different solutions, but they mostly agree on certain key problems. Their eyes and minds are not aligned, not identical, not even perhaps compatible, but there are some things they almost all seem to see, and this book will, I hope, help you see as well.

This bring me back to McPhee, and the simple fact that the interviewees here are all creators of major, well-known, historically important programming languages. There are two ways to know things: one is by direct experience and perception, “muscle memory” so to speak. I know parts of software engineering in this way, because I have built moderately complex programs, and found nasty bugs in seriously complex programs, myself. I know these things by *doing*. But that method of knowledge has serious limits; some things you need to know before you do them, for common sense reasons (open heart surgery, nuclear reactor software development) and others you have no real hope of ever doing yourself. Few of us will be geologists prospecting for leftover silver in Nevada, but we might want to have knowledge of that; McPhee does that, and in other books gives us the knowledge of controlling the flow of the Mississippi River, crewing a merchant marine ship, or being in the Swiss army. This book gives you a bottle distillation of part of the knowledge obtained by creating and maintaining and changing one of the Great Programming Languages. Unlike the experiences McPhee offers, which are aesthetically pleasing but of little practical value to most of us, software engineers can bring the deep lessons of being a Mastermind to bear on their own work, but only by somehow doing what cannot, in fact, be done.