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

Charles Petzold's *Code: The Hidden Language of Computer Hardware and Software* may not be for you. Are you an embedded systems hacker of the first rank, or an electronics and homebrew retro-computing hobbyist with an Apple I in the basement (or, better yet, in the living room)? If so, *Code* will be old hat, and you can skip this column, unless you want a good book to recommend to the next person who is baffled by your cabinet full of antiquated motherboards. You know what Petzold is going to say. The rest of us, however, even if we technically have some training in computer architecture or have done systems programming, can enjoy the tour of computing's hardware basis.

I say hardware basis because, while the book's title suggests it covers the hidden language of hardware and software, this is mostly a hardware book. That, for *Passages*, is unusual. We are a software people. *Code* is also unusual in that while *Passages* aims at books that the interested "lay" reader can enjoy, the point is often stretched. Is anyone not somehow paid for making (or at least thinking about) software ever going to read *The Pragmatic Programmer*, or any of Knuth's selected papers volumes? Probably not. *Code* is a hardware book, and it is written for a general audience, not for the software engineering, or even computing, specialist.

What makes *Code* ideal for software engineers is that it is a useful counter to the ease with which we can forget that, under all our layers of abstraction and the software that masks the beast, there is a machine. *Code*, in a friendly, appealing, way, reminds the software engineer of the beast. Moreover, unlike a less popularly conceived book, focusing on modern hardware systems, *Code* invites us to get to know the beast, perhaps pet it, and maybe even adopt it as our own dog. The method of *Code*, you see, is 1950s Boys Life' adventure, Popular Science and Danny Dunn. It starts like this:

> "You're 10 years old. Your best friend lives across the street. In fact, the windows of your bedrooms face each other."

You and your best friend want to talk after dark, and so you turn to that object at the heart of Steven Spielberg nostalgic childhood, the flashlight. The flashlight leads us, naturally, to Morse code, and to Braille. *Code* lightly covers the history, but not too much for the 12 year old inside, and it soon turns to something even more essentially designed to invoke the golden age of science fiction: building a computer.

The heart of the book, chapters 12 to 17, is focused on building working, if simple, computers using relays. You can play along at home. In chapter 12, you build a simple binary adding machine. By chapter 17, you have a machine with a simple assembly language, and enough coding power to have moderately interesting bugs in its software. Chapter 18 runs through "computer pre-history" from (as the book puts it) "abaci to chips" in a mad rush, then chapter 19 discusses two classic microprocessors (you can probably guess which ones, if you're one of the people who might enjoy *Code* but don't really *need* to read it). No book on "code" would be

complete without a chapter on ASCII, and there are requisite and interesting chapters covering the bus, the operating system, and (deliver us from) floating point, fixed-point, and other numeric formats. The final chapter, which touches on GUIs and the web, is the one "cutting edge" (at the time it was written) aspect of the book, and so, naturally, the only part that feels significantly dated, somewhat to its detriment.

I started to write that the simple but clear walkthrough of computer basics here is useful to every software engineer who lacks familiarity with the hardware basis of computing. On reflection, however, I decided this is not true. Yes, if you were unaware of even the basics of assembly language, or the idea of a bus, this book is going to give you new tools for thought. But in practice, knowing how (primitive) memory works or what the carry bit means really isn't going to make you write better code. No one toiling in a PHP mine will be liberated by the discovery that DEADBEEF is a number.

No, the practical value of this book is quite limited; what goes on inside a modern computer is not infinitely removed from a relay machine with no operating system, JNZ, LOD, ADD, SUB, STO, and 64KB of RAM. But it isn't very close, either. ASCII is still around, but feels somewhat quaint (the book does mention Unicode). There are no helpful tips on coding, other than the useful note that as soon as you can write programs of any interest on the simplest machine, you have to start worrying about subtle bugs. If you're like me, you will likely read the book and remember the parts you already somewhat knew of the underlying electronics, and forget any details of flip-flops or edge-triggered vs. level triggered latches that you didn't already possess. What you will likely carry away is a refreshed appreciation, even if it is not the level on which you usually work, for the joy, charm, and physicality of computer hardware. Yes, you should give this book to every intelligent 12 year old you run into, and it might be a good book to hand your interested relatives who ask "how does a computer work?" (if they really want to know), but you can read it yourself, to be reminded that computers are both AM (actual machines, the most interesting kind of machines) *and* magic, in the way every project in a 1948 issue of Popular Mechanics (in September, we built our own arc welder!) was magic. The magic is magic precisely because you can make the magic yourself, and see how it is done, take it apart and put it back together, and yet it remains magic all the same.