Alex Groce (agroce@gmail.com), Oregon State University

Jon Bentley's *More Programming Pearls: Confessions of a Coder* is the second collection of Bentley's Communications of the ACM Programming Pearls columns. I wrote about the first such collection (*Programming Pearls*) in my March 2014 *Passages* column. I'm now writing about the second collection for two reasons. First, it is a worthy classic of software engineering. Second, I believe many fewer people read the second book than the first. For example, on Amazon.com, *Programming Pearls* is ranked 32,548 among Best Sellers, while *More Programming Pearls* is ranked in lowly 783,168th position. Admittedly, Amazon's rankings are not exactly a well-defined, easily comprehended, and reliable metric, but the fact remains: many programmers and computer scientists read (or at least buy) the first volume of Bentley's columns and then stop. Perhaps distractions intrude, and there is a plan to eventually get around to the second volume, and the time never comes -- sequels are always inferior, anyway, right? Perhaps the idea is that the Pearls columns are like very good split pea soup: one bowl is fantastic, but two bowls is a bit too much of the same good thing, even if it is just as good as the first volume. It is even (though hardly conceivable to me) possible that some people don't much care for the first volume, so avoid the second volume. I can't help the last group. The first two groups, however, I can advise: *More Programming Pearls* is just as good as the first volume, and it is sufficiently different that, especially for a software engineer, it offers rewards distinct from those in the first volume. If *Programming Pearls* is perfect split pea soup, *More Programming Pearls* is cold gazpacho on a hot summer day. We'll get to the differences between the books in a moment.

To start with, however, there may be those who have read neither volume, and need a different kind of convincing. Why should a programmer read this volume, whether or not that programmer has read the first volume? Bentley explains the topics covered in the Preface: "Computer programming is fun. Sometimes programming is elegant science. It's also building and using new software tools. Programming is about people, too: What problem does my customer really want to solve? How can I make it easy for users to communicate with my program? Programming has led me to learn about topics ranging from organic chemistry to Napoleon's campaigns. This book describes all these aspects of programming, and many more." If that doesn't sound like the summary of a software engineering classic, I don't know what does. Furthermore, Bentley not only covers these topics, he covers them well, with charm, insight, and humor. This is the reason to read *More Programming Pearls: Cold Gazpacho* that applies equally well to *Programming Pearls: Split Pea Soup*. What are the differences?

Where the first *Programming Pearls* is (largely) centered around the idea of performance, with some side-trips, the second volume ranges much more widely. The book is divided into four parts: Programming Techniques, Tricks of the Trade, I/O Fit for Humans, and Algorithms. In this book you will find an introduction to profilers (Bentley hasn't exactly abandoned performance as a topic), a discussion of "little languages" that pre-dates the current high interest in Domain-Specific-Languages, a return to Bentley's "back of the envelope" intuition building exercises, and a collection of Bentley's favorite "bumper stickers" -- pithy adages from CACM

readers and famous nuggets of general programming and software engineering wisdom. All of these diverse topics are handled concisely and energetically, and often, again, with a touch of humor. One adage in particular made me chuckle, a point made by the first computer scientist to publish a solid description of serious differential testing, Bill McKeeman: "Making a wrong program worse is no sin." This is definitely the motto of modern C compiler optimization writers.

The topics of *More Programming Pearls* take it further into the realm software engineering as-such than the first book, which was more focused on either algorithms or perhaps "programming languages" (in a certain sense) than on the larger questions of system design and creation. Chapter 10, Document Design, for example, is largely focused on nuts, bolts, and document appearance, but emphasizes that 1) (good) programmers spend a lot of time writing text for other humans and 2) that text exists for "the reader, for whom [it] exists in the first place." That the examples are "dated" to early 80s computing, in my opinion helps the actual ideas come across clearly. Sure, the "little language" Pic is no longer used, as far as I know, by anyone. That makes the point of a little language more obvious than using a little language to solve some "problem of the moment" where the unwise reader may think that the problem at hand is what is interesting, not how the problem is solved.

Taken together, the two *Programming Pearls* books are unique. They range over whatever Jon Bentley had in mind when he was writing, without a systematic programme of "this is what programming is" or "this is how you should learn computer science," yet they manage to construct a good picture of computer science as a whole, as rooted in the construction and understanding of computer programs. As with the work of Knuth, you can see a central idea, not always expressed in code, but often leading to code, that computer science is the study of what can be automated, *and how to automate it*. Because that concept is broad, but specific enough to describe a real discipline, the columns range far and wide. In that sense, one way to think of these books is as computer science equivalents of Lewis Thomas' *The Lives of a Cell*: on the one hand, Bentley is less lyrical than Thomas, and less interested in a view of life as a whole. On the other hand, *The Lives of a Cell* would be more a lot more fun if it included exercises and let the reader create some life (and perhaps profile it to see how well it is performing, or prove it correct).