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

Jon Bentley's *Programming Pearls* (Second Edition, in this review) is, in John Bentley's own words, about "a more glamorous aspect" of programming than *The Mythical Man-Month*, which "paints the big picture" of software engineering. *Programming Pearls* is based around a series of perfect miniatures whose "origins lie beyond solid engineering, in the realm of insight and creativity." Bentley explains the title thus: "Just as natural pearls grow grains of sand that have irritated oysters, these programming pearls have grown from real problems that have irritated real programmers." From irritation, Bentley produces beauty.

This is not to say that *Programming Pearls* is just a book of algorithmic cleverness, or seductive offbeat code to thrill the hacker, with little relevance to real-world tasks. *Programming Pearls* is extremely relevant to real software engineers because it produces its beauty from real grit, without ignoring any of the major aspects of a software engineering task. In the fifteen chapters (mostly adapted from columns in *Communications of the ACM*) Bentley covers requirements and specification, design, implementation, optimization for performance, and testing, verification, and debugging. Embedded in the pearls are a wonderful series of little anecdotes and sidelines, sometimes central to the topic of a chapter, sometimes simply enlightening side-trips from Bentley's long and varied career.

Programming Pearls is about programming, and it is probably not a book for anyone who isn't a moderately capable programmer. The code included is never extremely complex, and much of it is pseudocode, with a small amount of real C and C++, but the substance of many chapters requires some sophistication in understanding how code works that will frustrate a more casual reader. While *Programming Pearls* is by no means a textbook, it does contain exercises, and much more can be taken from the book if a reader actually does these exercises. As Aristotle said of playing the harp, the way to learn to produce programming pearls is to produce programming pearls. One thing Bentley does remarkably well is to mix a strong author personality and set of insights (and your insights are probably not nearly as good as Bentley's insights, especially once he has incorporated all the insights of other people he has worked with over the years) with a collaborative effort in the exercises. Some of the immediacy of the "interaction" comes, no doubt, from the fact the chapters almost all first appeared as magazine columns. They were shaped in public, and appeared as a string of dispatches to the world rather than being slowly developed in isolation before being thrust on more than a handful of readers. The structure of the book, despite coming from a series of columns, is not haphazard and disjoint. There is a real flow here, though re-reading any favorite chapter is also pleasant.

Chapter 1 touches on what is arguably the core problem of software engineering, vs. other engineering fields: deciding what you *actually need to do*, given the extraordinary amount of freedom in software. The question "How do I sort a disk file?" leads to analyzing the real problem, which turns out to have better solutions than sorting a disk file. It's a useful lesson to remember these days that simply turning to *stackoverflow.com* to find the answer to a

narrow technical “How do I?” question may not be the best way to accomplish your real goal. Or, if you are more formal and less pragmatic, a specification should be careful what it specifies.

Chapter 2 introduces binary search, an algorithm that will form the focus of much of the first third of the book, from columns 2-5. The 3rd chapter/column discusses how programs can be made smaller, faster, and better by wise use of appropriate data structures. Chapter 4 turns to the means by which *correct* programs can be written, emphasizing close analysis and semi-formal proof, essentially briefly sketching a manual approach to program verification. It is worth noting that the proven-correct binary search in chapter 4 actually contains a “bug” in that is assigns *m = (l + u) / 2*, which can cause overflow. The bug was arguably irrelevant when the code was written, since arrays with more than 2^30 elements weren’t exactly common in those days, but times have changed. The final chapter in this first section of the book (a section Bentley calls “Preliminaries”) is a “column” introduced for the second edition rather than one of the original CACM-published chapters. It is a practical look at actually implementing, testing, and debugging the binary search code. While far from a complete look at testing (for example, there’s no real look at differential testing or random generation of data), this short chapter still manages to give some excellent advice and introduce automated testing in a compelling way.

The second section of the book, “Performance,” is less foundational, but includes what is possibly the most valuable piece in the book, Bentley’s superb examination of back-of-the-envelope-calculations in Column 7. Informal estimation is a technique essential to good software engineering but one seldom given the attention it merits. Recently, the use of these kinds of questions in job interviews has raised awareness of the technique, but Bentley’s combination of “the envelope” with informal experiments is something more than simply a tool for doing better at brain-teasers. The chapter also returns to the connection of software engineering and “real” engineering in a brief discussion of safety factors.

The last five columns, collected in a section called “The Product,” turn to more interesting complete programs tackling small but compelling problems. It concludes with a concise and very clever Markov-text generator that improves on the already clever method in Kernighan and Pike’s *The Practice of Programming*.

Discussing the contents of the columns gives an idea of the scope of Bentley’s subjects -- especially the attention to picking the right algorithm but going beyond a limited “big O looks good” approach -- Bentley is really thinking about both the problem at hand and the real behavior of the programs in question. A walk through the chapters doesn’t really capture the style of the book, however, which is part of what makes *Programming Pearls* a favorite of so many programmers. The fun of the book is perhaps best seen by reading the two epilogues, given in the form of two self-interviews. The self interview can be dull in the hands of a self-absorbed or humorless writer; it can be a minor literary masterpiece in the hands of a great writer. Vladimir Nabokov famously never published any interviews *other* than

self-interviews; Oscar Wilde, James Barrie, F. Scott Fitzgerald, Evelyn Waugh, Gore Vidal, Glenn Gould, Philip Roth, Walker Percy, and Gene Wolfe all used this unique form well. Bentley's interviews are short, but manage to justify the book, entertain the reader (all the jokes in this book are pretty good), and even produce a much-quoted set of ten suggestions for programmers (suggestions for all engineers, Bentley notes):

"Work on the right problem.
Explore the design space of solutions.
Look at the data.
Use the back of the envelope.
Exploit symmetry.
Design with components.
Build prototypes.
Made tradeoffs when you have to.
Keep it simple.
Strive for elegance."

That's a good ten-line summary of what *Programming Pearls* is about, and the way it manages to produce a string of pearls whose luster is these ten suggestions is why it is a classic of software engineering.