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

Donald E. Knuth's *Selected Papers on Computer Science* is not the obvious Knuth book for this column. What is? *The Art of Computer Programming* is Knuth's great work, and *Literate Programming* is (for good or for ill) his work that is most clearly concerned with software engineering. *Selected Papers on Computer Science* is not as well known, but I think it merits its place in the canon of software engineering classics, as defined in this column: a book more than ten years old, of special interest to software engineers (as opposed to computer scientists in general, however much they may also like the book), worth reading multiple times and of some interest even to the general educated reader who is not a computer scientist.

In fact, the last point is a defining concept for these papers chosen by Knuth:

"This book assembles under one roof all of the things I've written about computer science for people who aren't necessarily specialists in the subject -- for scientists and mathematicians in general, and for educated people in all fields."

The seventeen essays, numbered 0-16 (yes, of course, Knuth starts numbering at 0), can be divided into a few major groups. First, Knuth spends 6 chapters and just over 120 pages discussing algorithms, which turns out to require defining computer science and defining mathematics, with the aim of determining just what the difference between computer science and mathematics *is*, to his satisfaction. The briefest definition of computer science, given in Chapter 0 (a short letter), is the most interesting to software engineers, and we will return to it later: Knuth says "Computer science answers the question 'What can be automated?'"

Chapters 5-9 are titled "Theory and Practice, I-IV" and are likely the most interesting to software engineers. It is as a pre-eminent combiner of theory and practice that Knuth is perhaps best known. His precise mathematical analysis of algorithms is the embodiment of theory, in a sense, but he is also famous as the developer of one of the most famous software systems of all time, TeX. The 11 years Knuth "took off" from theoretical computer science to write software are in many ways at the heart of his life, and central to the story of *The Art of Computer Programming*. In the "Theory and Practice" chapters, Knuth explores that time period, and what it meant to him as a computer scientist.

Chapter 10 considers whether toy problems are useful, and the remainder of the book (6 chapters) is about the history, both ancient and modern, of computer science, starting with Babylonian tablet algorithms and continuing to von Neumann's first program, the IBM 650, and beyond. Many software engineers with a historical bent will find the details of early programming systems and their constraints (memory drum cycles!) exciting, if not particularly useful in day to day life.

This sounds like an interesting book, and Knuth is almost always entertaining and insightful, but what is the special appeal to software engineers here? Why not read his papers on

programming languages, or on toys and games for that matter?

The reason to read this book is that it offers a perspective on software engineering that is unusual and important. First, Knuth argues that computer science is *about* algorithms, at heart, and that algorithms are "concepts that have existence apart from any programming language." Given Knuth's work on (and the large portion of this book devoted to) careful, precise analysis of the runtime of algorithms, this claim at first glance this seems to put Knuth's view of computer science largely outside the ballpark of software engineering, in the theoretical camp -- this is a book for SODA people, not SEN or even ICSE people! However, the chapters on practice and theory show that Knuth does not believe "true" computer science is the analysis of algorithms in a vacuum, uninformed by the process of *building working software systems.* Knuth says, in a speech to a meeting of theorists in Greece, "... it's natural for me to be musing about what I've learned from my odyssey into the real world of software creation" and the first lesson is "*software is hard*; and it takes a long time." He concludes his remarks (again to the assembled body of hard-core theorists) with a remarkable recommendation: "I urge you to consider devoting more of your time to questions of actual implementation. I sincerely believe that all of you will benefit from more programming activity…" and explains how his excursion into building a real, complex, working software system helped improve his work as a theorist. It is not known how many of those present took the advice.

Some of Knuth's insights into software engineering are surprising. For example, he believes that developing software demands *more* attention than working out complex mathematical theory! He explains himself thus: "A great deal of technical information must be kept in one's head, all at once, in high-speed random access memory somewhere in the brain." Knuth notes that while he had often taught classes and worked on deep theoretical problems (and written volumes of *The Art of Computer Programming*) he was forced to take leaves of absence from teaching while working on TeX and METAFONT, due to the demands of concentration --- he couldn't "do justice to both activities simultaneously." He ascribes this problem to two things, one being the aforementioned amount of technical detail that must be kept readily at hand. The other aspect is interesting: "programming demands a significantly higher standard of accuracy."

This is a critical point. Prior to reading this book, I'd have said, without much thought, that complicated mathematical proofs demand a higher attention to detail than programming. This is, perhaps, true in some sense, but in another, equally important sense, it is false. A human being interprets a non-formal proof, and often even multiple errors are ignored so long as the "gist" of the argument is preserved. However, a compiler makes no such allowances. Perhaps this is the origin of Knuth's famous statement to "Beware of bugs in the above code; I have only proved it correct, not tried it."

The message for software engineers here is in four parts: (1) computer science is the study of what can be (efficiently) automated (the study of algorithms); (2) designing and

understanding actual automations (not algorithms but software systems) is fun but *very hard*; (3) the general study of what can be automated is (best?) inspired by the desire to automate some real task; and (4) the ability to automate any task depends on understanding of what can be automated, and how. Knuth offers an enjoyable argument that software practice should be informed by some theory, and computer theory should be rooted in engineering.