

# Desert Island Books

by John Knight

Growing up in England, I was fortunate to be able to listen to Desert Island Disks on BBC radio each week. That is the program which inspired this column, of course, and it remains very popular. I often made up my own list of records, and it is a great pleasure to be able to write this column and reflect upon the books that I would like to have with me on a desert island. It is quite an eclectic collection, but these are the books I would want with me--I have not used this opportunity to discuss books that I think others should read.

So I am to become a castaway on a desert island with a collection of books related to software engineering. The books will have to provide entertainment, guidance, insight, challenges and memories, so they had better be chosen carefully. To this end, I think there are a couple of strategies that need to be followed. First, a practical requirement is to include books that can serve dual purposes. Is there anything to be learned from software engineering that can be applied to the problems I will face on the island? How could I improve my CMM (Castaway Maturity Model) rating by studying software? The risks are not the same but there are risks. Second, there are a lot of papers and other documents that I would like to have with me so I have chosen collections of papers in three cases. Fortunately, there are some very useful collections available.

I find most textbooks on software engineering to be quite disappointing, and I would not have much use for them on the island. The important thing about software engineering is that it is engineering. This vital point is missing from a lot of practice and from most textbooks; the former is largely the result of the latter. Much of the commercial software practice that I see is far from good engineering. Good software results from the careful, methodical, and thorough application of a relatively few basic principles, yet most textbooks do not cover these principles. The books I have chosen document many of these basic principles, mostly in the words of the developers of the ideas.

Although it is very familiar, there are so many insights and so much useful guidance in the Mythical Man Month by Fred Brooks that I would want it with me on the island. It would have to be the 25th anniversary edition though, so that I could get chapter 16, a reprint of Brooks' "No Silver Bullet" article. It's easy to become complacent and think that, having read the book in the past, one has learned all its lessons and is applying them. I doubt that this is the case for most of us; it certainly is not for me. For example, do you remember what Brooks said about "time to market" and how to deal with rapid changes in technology? The insights in this book came from years of experience including the development of a massive software project, and the lessons are subtle and many. This is definitely a dual-use book with many fundamental ideas that will help me on the island. "Always a working system" clearly can be restated "Always a working shelter", and so on. However, it would not be necessary to worry about Brooks' law since the island would, I assume, be deserted!

The next book on my list is Software Fundamentals, a collection of papers by Dave Parnas edited by Dave Weiss and Dan Hoffman. This is a wonderful book because it presents many of Parnas' ideas in one package. The use of the word "Fundamentals" in the title is exactly right--the

material is far from simple and is certainly not elementary, but it is fundamental. Parnas' contributions have covered a wide variety of topics from specification to design to inspection. Information hiding, introduced by Parnas in 1972, is a basic principle that is completely misunderstood by most textbook authors. It is fairly simple to state but remarkably difficult to understand and to practice. It should be covered in depth by any book on software engineering but almost never is. The ramifications of information hiding, such as program families and design flexibility, have been explored by Parnas and others in a series of papers published over a period of many years. These concepts are of great practical utility in software development, but they are also intellectually challenging and so do not even get mentioned in most textbooks. The complete set of ideas is best seen as a unified collection, and that is now possible thanks to this book.

An unusual and to some an incongruous choice, perhaps, but my next book is the Ada Language Reference Manual. This is partly a sentimental choice and partly technical. The sentimental part derives from the fact that my wife's professional interest is in compiler construction, and she spent a lot of time on Ada compilers. This led to numerous conversations between us of the form "Do you know what this Ada fragment does?" I was a great supporter of the Ada project and the technical reason for wanting to have the manual with me is to be able to spend time on the island (when I am not fishing) trying to answer the question "Why did it fail?" Ada tackled many of the important practical problems in programming with creativity and care. It is possible, for example, to port an Ada program from one platform to another very quickly. The Ada developers took a stand (a stand with which you might disagree but at least they tried) on language subsets, compiler validation, and many other practical issues. Somehow, we have ended up in a situation where much of the world's software is written in C, an almost typeless structured assembly language with a confusing syntax. All kinds of software including operating systems, telecommunications systems, and even some safety-critical systems are written in C. There are circumstances that warrant the use of such a language, but they are very few and very far between. Deriving other languages from C seems equally unfortunate. I am convinced that the use of C is actually having a negative impact on national productivity.

Classics in Software Engineering is a collection of papers edited by Ed Yourdon that was published in 1979. Don't be put off by the date--with a total of twenty four chapters, many of which are significant papers, it provides a lot of important material even now. One of the reasons that I chose this collection was to be sure that I had a copy of Dijkstra's paper "The Humble Programmer", his Turing Award Lecture. Although it was written thirty years ago, it remains a valuable paper. The paper argues eloquently that developing correct programs is a tremendous intellectual challenge, and that the right way to build software is not to expect quality through testing, not to expect problems to be dispelled by new tools, and so on. It is, unfortunately, as relevant today as when it was written. Other important papers in the collection include three more by Dijkstra, the original paper by Bohm and Jacopini that presented the fundamental theorem of structured programming, and Baker and Mills' paper on the Chief Programmer Team.

One of the things that I find fascinating is the way in which electronic computation developed. We tend to be caught up in the achievements of modern microprocessors, disks, and so on, but a review of mechanical computation, relay computation, vacuum tube logic circuits, memories of just a few hundred words, and programming with such equipment is an instructive exercise. Many modern software concepts were discussed by Ada, Countess of Lovelace, in her writings about

the Analytical Engine in the 19th century. The programming environment used with the EDSAC at Cambridge around 1950 was heavily focused on software reuse. With this in mind, the next book on my list is *The Origins of Digital Computers: Selected Papers* edited by Brian Randell. This book is about computers, not just hardware, and so it provides a comprehensive account of how hardware developed, how designers thought such hardware would be used, and early ideas on software. The variety of ideas that were pursued by computer pioneers like Babbage, the Countess of Lovelace, Zuse, Atanasoff, and Ludgate is remarkable. Are there ideas within their papers that we could now pursue? I confess that the main reason I would like the book with me is pleasure, but it also occurred to me that the various ideas contained in Charles Babbage's papers might be a start to building a computer on the island.

With the texts listed above I have a good collection of materials that will remind me about and act as references for many of the basic concepts in software engineering. But what about the future? I will have time to think and contemplate new ideas. Is there a book that will provide me with a set of intellectual challenges to work on? The book I want to do that is *Security Engineering: A Guide to Building Dependable Distributed Systems* by Ross Anderson. Published in 2001, this book provides a summary of the computer security field. It contains a wealth of material about the spectrum of problems we face with modern systems and a number of details about the security technologies that we have at our disposal. With so many security attacks being based on buffer overflows more than a decade after the Morris worm, I am convinced that the fundamental security problem is poor software. Thus I view Anderson's book as a requirements statement for software research and technology transfer. A wide range of research topics come to mind as one studies the material after one gets over the feeling of panic about the magnitude of the problem that arises when reviewing the table of contents. We are building and planning systems of vast size and importance, and they have to be operated in what amounts to a hostile environment. The software should not be a source of concern in the future although it will be. There should be very little doubt that a security protocol or a cryptographic algorithm is implemented correctly nor that its interfaces will operate correctly in the broader system context. To achieve such assurance, however, will require a lot of progress in requirements analysis, systems modeling, system and software design, verification, and maintenance.

On *Desert Island Disks*, the radio program, the castaway is allowed to take one book along with the collection of ten records. Since this column is about books, I feel that a single record (in my case it's three CDs) should be allowed too. My choice is the great masterpiece by J.S. Bach, *The St. Matthew Passion* (BWV 244). Written almost 300 years ago (the version performed in modern times was completed in 1736), this work is just what one would need on a desert island to be reminded about the enduring genius of mankind outside of technology.

#### References:

- (1) F.P. Brooks, *The Mythical Man Month* (anniversary edition), Addison Wesley, 1995.
- (2) U.S. Department of Defense, *Reference Manual for the Ada Programming Language*
- (3) D.M. Hoffman and D.M. Weiss (Eds), *Software Fundamentals: Collected Papers by David L. Parnas*, Addison Wesley, 2001.

- (4) E.N. Yourdon (Ed), *Classics in Software Engineering*, Yourdon Press, 1979.
- (5) B. Randell (Ed), *The Origins of Digital Computers: Selected papers*, Springer-Verlag, 1973.
- (6) R. Anderson, *Security Engineering: A Guide to Building Dependable Distributed Systems*, Wiley, 2001.