Automated Software Engineering, 12, 139–141, 2005

Desert Island Column

SUSAN STEPNEY
Department of Computer Science, University of York

So, I'm to be marooned on a desert island with a handful of books relating to software engineering? I have two immediate, conflicting, reactions. One is to complete the punchline to a music-hall joke: "Second prize, marooned with *two* handfuls of software engineering books". The other is to think, I take a handful of books on a short train journey, because I'm terrified of being caught without reading material: how will I last on the island? This wins over the joke, and I start to compose my list.

I am rather enamoured of the concept of Patterns. A pattern is a codification of a well-known solution to a common problem, along with a description of where it is and is not applicable. Good software engineering patterns can increase your analysis and design vocabulary, making it easier to "chunk" complexity, and saving reinventing many kinds of wheels. The emphasis on *applicability* is a refreshing change: unlike many overhyped methods, a pattern is not given as *the* way to solve a problem, it is just one possible solution.

A simple increase in vocabulary is not enough, however: words need to be strung together in some structure to make sentences: we need a *language*. This is where many of the more simplistic software engineering patterns fall down: a random bunch of nouns does not a language make. The originator of patterns, architect Christopher Alexander, invented a whole Pattern Language for designing buildings: all the way from the smallest detailing of architraves, through placement of windows, to whole houses, and even whole towns. Reading his books is an aesthetically pleasing experience at several levels: the intriguing concepts, the smooth prose, even the physical feel of each book with its slightly flexible hard covers.

So, take Alexander's *A Pattern Language* and *The Timeless Way of Building* to the island? But I have already read these. I would more like to find out what he's been up to in the intervening quarter century. So what I propose to take with me is his latest magnum opus, the four volume *The Nature of Order*, which I have yet to read. As I understand it, this is a more philosophical look at structure, pattern, and design, both natural and artificial: all concepts very relevant to software engineering. If it is even half as thought-provoking and influential as his previous work, this will be time well spent.

I mentioned aesthetics. The aesthetic dimension is of utmost importance in design. The theoretical physicist Paul Dirac could say "it is more important to have beauty in one's equations that to have them fit experiment". Engineering is somewhat different: we do have to make our artefacts work in the often messy real world. And that real world is sometimes even messier for *software* engineers, because much of it comprises our own older, possibly not fully thought through, design decisions. But that unavoidable

messiness is no excuse for what often appears to be gratuitously ugly design! One of the many areas that such ugliness often manifests is in the diagrams used to express or explain designs. Diagrams should be primarily about *communication*: poor diagrams hinder this. To take just one example, the world is full of UML class diagrams that have boxes splattered seemingly at random over the page, with association links snaking around with dazzling Moiré effects. There seems to be some law of conservation of spaghetti at work: as we have managed to remove it from our code, it has popped up vividly in our diagrams.

One author pursuing a crusade against bad diagrams, bad charts, bad figures, is Edward Tufte, inventor of the gloriously pejorative term *chartjunk*, and of the important concepts of macro and micro readings: that a diagram has many levels of detail. He has three brilliant and beautiful books (it is always reassuring when a book about good aesthetics is itself aesthetically pleasing) on this subject: *The Visual Display of Quantitative Information*, *Envisioning Information*, and *Visual Explanations*. These are lavishly illustrated with examples of great, and appalling, diagrammatic designs, and should be required reading for anyone drawing any form of graphical representation of information. Since a new volume, *Beautiful Evidence*, is promised for 2005, I will choose to take it with me (or have it delivered) to the island.

One much-hyped software engineering paradigm is object orientation. OO has many good points, and I would not like to lose it. But it has its limitations, not the least of which is the inheritance hierarchy. The OO brigade would have one believe that the hierarchy is the one true way to structure relationships between similar things. Yet when we try, we often find ourselves shoehorning things that don't seem to fit properly. (Penguins seem to feature heavily in such arguments.) So several years ago I was pleased to read *Developing Object-Oriented Software for the Macintosh*, a very sensible book about OO (and not as restricted as its title might suggest), counteracting much of the OO hype. It includes a gorgeous little example of how (un)natural OO models actually are: a railway track object that "knows how to lay itself".

I was even more pleased to read a book from its bibliography (the bibliography of a really good book leads one on to many other good books): Lakoff's *Women, Fire, and Dangerous Things*. This is a book about linguistics, metaphor, and the way people actually categorise things in the world. For example, the Australian Aboriginal language Dyirbal has four classes of nouns; women, fire, and dangerous things are some of the nouns in class II. Lakoff chose this as his title because he finds it an amusingly arbitrary collection of things to be classed together. (But it looks fairly natural to me...) What the book shows is that we very much *don't* use hierarchies as our natural category structures. Over many painstakingly explained examples, including more about the word "over" than you might think possible, Lakoff shows that we instead form radial and metaphorical categories from prototypical exemplars. I wonder, what would a software engineering design philosophy based on this actually natural style of categorisation look like? (And before you say it, no, it's rather more sophisticated than object-based delegation.) It has been a while since I read the book, and I think it's about time I read it again: my island sojourn should be the ideal opportunity.

Books mentioned

Alexander, C. *et al*. 1977. *A Pattern Language: Towns, Buildings, Construction*. Oxford University Press.
Alexander, C. 1979. *The Timeless Way of Building*. Oxford University Press.
Alexander, C. 2002–2004. *The Nature of Order*, vols. 1–4. Center for Environmental Structure.
Dirac, P.A.M. 1963. *Scientific American*.
Goldstein, N. and Alger, J. 1992. *Developing Object-Oriented Software for the Macintosh: Analysis, Design and Programming*. Addison-Wesley.
Lakoff, G. 1987. *Women, Fire, and Dangerous Things: What Categories Reveal About the Mind*. University of Chicago Press.
Tufte, E.R. 1983. *The Visual Display of Quantitative Information*. Graphics Press.
Tufte, E.R. 1990. *Envisioning Information*. Graphics Press.
Tufte, E.R. 1997. *Visual Explanations*. Graphics Press.
Tufte, E.R. 2005. *Beautiful Evidence*. Graphics Press.