

style over imperative programming is illustrated by comparing Standard ML code and ANSI C code for implementing a function which swaps two integers. For readers acquainted with ANSI C, this is a clear illustration of the variable-oriented approach of imperative programming; however, for readers who are unfamiliar with any programming language and are using the text as an introduction to programming, this comparison is likely to be meaningless. The profusion of footnotes also suggests difficulties in this dual-audience approach.

For those new to programming this book provides a good introduction to the discipline, using the functional style to illustrate sound design technique, supported by a good range of examples and some intelligent exercises. This text would also be useful for experienced programmers, perhaps already familiar with functional programming, who wish to learn to use Standard ML. However, for experienced programmers who wish to learn software engineering in the functional style, this is not an appropriate text.

P. MUKHERJEE

*University of Birmingham*

R. J. MITCHELL

*C++ Object-Oriented Programming*. Macmillan, 1993, £15.99, 255 pp softbound, ISBN 0-333-58937-8

The main aim of this book is to teach object-oriented programming methodology and to show how it can be used in the development of large programs. C++ is used as the language for all the example code. No previous knowledge of C++ is necessary, but the book assumes familiarity with either C, or a language similar to Pascal, Ada or Modula-2. C programmers could jump straight into this book, but for fans of other languages, Appendix 1 gives a concise introduction to C.

The introduction is a good description of the basics of object-oriented techniques, using examples in C++ and contrasting the object-oriented method with more traditional approaches. It sets the high standards of explanation and informal but clear tone found throughout the rest of the book.

The whole of the remaining text is centred around the development of a single example project, a drawing package. The choice of a graphically-based example does of course mean that a some of the code is compiler-specific. The compiler chosen is the Borland C++ compiler, available only to the IBM-compatible PC platform. For anyone without a PC, there are a few parts of the program that would have to be altered if the code is actually to work.

If all the code in the book were put together as the reader progressed it would take little extra work to produce a well-featured drawing package. I believe that this is one of the best ways to learn the ins and outs of object-oriented programming and C++, and where several other C++ books I have read fall down: isolated and incomplete examples lead to a lack of motivation

and understanding in many students. A disk is available containing the finished package and answers to exercises.

Language features are introduced a few at a time in a sensible order, all clearly explained. First come classes, then inheritance, objects and pointers and finally streams. Motivation for the use of all these features is well-provided, with examples from text and graphics window classes, through the traditional linked list and tree structures to the development of a context-sensitive help system. Virtual functions, operator overloading and input/output are dealt with thoroughly.

For some reason, possibly because the compiler chosen does not support it, templating is not covered. As templating is one of the most powerful tools available in C++ I found this to be an unfortunate omission.

For people who have experience in other languages and want a good introduction to both object-oriented programming and C++, especially as it is implemented on the PC, I would recommend this book. Its few drawbacks are outweighed by the clarity and thoroughness of its explanations.

M. B. M. GIBSON

*Warwick University*

P. J. PLAUGER

*Programming on Purpose II*. PTR Prentice Hall, 1993, £21.25, 204 pp, softbound, ISBN 0-13-328105-1

This is the middle of three volumes of essays written by the author for his column 'Programming on Purpose' in the *Computer Language* magazine over a period of six and a half years. Each volume has a theme, and the subtitle of this one is *Essays on Software People*. As you would expect, therefore, it is not a really technical book, and it contains less technical detail than either of the other two volumes, on *Software Design* and *Software Technology*. Plauger has been involved in the software industry over many years, first as a programmer, then as the founder and manager of a software company, and now as a consumer. This gives considerable authority to his comments on any aspect of the industry. There is a great deal of common sense and good humour, mixed in with some obviously heart-felt advice. The 26 essays make easy and entertaining reading, though I would recommend they be read one at a time for best effect. Each essay is followed by an afterword to complete what were ongoing stories at the time of writing the essay, or to report on the response to the original. Although they are often very brief, these afterwords definitely enhance the essays.

In spite of the subtitle, these are not essays about individuals. They cover a wide range of issues related to people, such as ethics, intellectual property, the duties of reviewers and how not to antagonize customers. In one or two of the essays the reader is reminded that Plauger writes in an American context, but in the software world no one can ignore the USA. His remarks in three essays on language standardization, the result

of his close involvement with the standard for C, are both entertaining and instructive—and should be required reading for all who argue about language standards.

If I have any complaint about this book, it is just that the price seems a little high. Maybe the intended audience has become accustomed to paying this sort of price for more technical books and the publisher does not expect large sales. Perhaps the strongest recommendation I can make, therefore, is to say that I went out and bought volumes I and III on the strength of this volume, and I look forward to further hours of educational entertainment.

C. C. KIRKHAM  
Manchester University

MARK NORRIS, PETER RIGBY AND MALCOLM PAYNE  
*The Healthy Software Project*. John Wiley & Sons, 1993, £24.95, 198 pp hardbound, ISBN 0-471-94042-9

The authors of *The Healthy Software Project* are employees of British Telecom and have extensive experience of project management using process quality standards, principally ISO9001. They present some valuable learning points, gleaned from their experience and scholarship. Readers that have worked in software development may suffer, as your reviewer did, the aching of old scars.

The book, as its title indicates, is structured around a medical analogy: the authors have, mercifully, resisted the temptation to call it a 'paradigm'. Its eight chapters introduce projects and the analogy; discuss the special characteristics of software projects; describe some projects (good, bad and average); and cover symptoms of project 'disease', therapy (including euthanasia), recovery and health maintenance. Three appendices treat of quality management systems, product metrics, and a process applicable to project recovery. The analogy is strained at times, even to the inadvertent black humour of: 'As with people, it is the quality of life ... that is important not mere survival. An invalid software project can cost a fortune to keep.'

However, the most important message of the book—often ignored in both theory and practice—is that projects can be, and sometimes should be, *recovered*. Project management is not merely about adequate planning and monitoring; it must also encompass recognition of when things go wrong (they usually do), and what to do to put them right. Starting with a clean sheet is as rare a privilege for a project manager as it is for a software designer.

The authors' attempt to present their learning points as an ordered collection is not entirely successful. We find a number of radical remarks that are made once and then ignored: 'A QMS ... provides a basic level of project control, but no more'. Yet the 'therapies' proposed all seem to be process ('management') therapies:

for 'sick' projects, we need 'the necessary control mechanisms to recover them'. Most radical (and very true): 'the production of software is a creative process and is usually done on the roll' (i.e. in a sustained, enthusiastic burst by skilled and committed software writers). Quite: the authors give lip service to 'people' (along with process and product): but in none of their case histories do they mention any of those extraordinary, talented, and often wild people that write the best software, and recover failing projects, technically. This is unquestionably a book by and for project managers: it won't fool your techies any more than you do.

On metrics, the authors quote Professor Dijkstra ('Metrics is crap.') but decline to engage with his arguments. However, they remark that 'many product measures ... are currently offered by the Snake Oil Salesmen ... the consultants and the tool vendors, with very little evidence that what can be measured is ... useful to measure'. Would that we could make the right judgement about metrics merely on the basis of who promotes them: depend on those recommended by employees of large corporations who get no commission on sales?

Finally, a plea to the publisher: proof reading is as important in your trade as testing is in ours. The reader should be spared, at least, poor spelling ('euthenasia', 'committment') and punctuation ('the customers aspirations'); although the correct placement of 'only' is nowadays perhaps more than we could hope ('that will only be achieved through projects ...'). Even more constructive editing might have clarified, for instance, the ambiguous 'key drivers': does 'key' mean 'major', 'important' or 'essential'? does 'drivers' mean 'resources', 'characteristics', or merely 'factors'? Perhaps the authors could advise on TAM (Total Quality Management—neither in the glossary; indeed—and for shame—there are no acronyms in the glossary, and not all are in the index).

A. LARNER  
De Montfort University

BONNIE A. NARDI  
*A Small Matter of Programming*. MIT Press, 1993, 162 pp, hardbound, £29.95 ISBN 0-262-14053-5

In 1979, Bonnie Nardi tells us, she spent several months in Papua New Guinea and came back with an account of how its villagers communicated with slit-gongs. This book is her account of what she has learned in several years of studying the communications of another exotic folk: users of application software packages. The burden of Nardi's account concerns the amazing ability of these people to master formal systems and notations, and the ways in which they co-operate with each other in their spreadsheeting and computer assisted design.

These topics are—or ought to be—of vital interest to the designers of software packages. However, plenty of advice on usability has already been proffered to the