therefore recommend the book to someone with a background and interest in theorem proving techniques, especially related to algebraic specification.

Without detracting from the merits of this book, let me end with a tribute to Donald Knuth: we have got so used to seeing TeX-quality mathematical typesetting, that one wishes the publishers of this book had also typeset it in TeX.

K. LODAYA
Institute of Mathematical Sciences, Madras

RACHEL HARRISON
Abstract Data Types in Standard ML. Wiley, 1993, £19.95, 212 pp softbound, ISBN 0-471-93844-0

The notion of abstraction is a vital one in software engineering—addressing the different phases of software development at the appropriate level of detail is a precondition for the construction of high-quality, reusable software. This book tackles the issue of data abstraction in a functional framework, using the language Standard ML (SML) as the implementation language.

In keeping with conventional texts on abstract data types, the book looks at the normal range of data structures such as lists, stacks, queues and trees. Each data structure is specified informally in terms of pre- and post-conditions and implemented generically using SML's structures. Throughout the book the twin goals of correctness and modularity are emphasised.

The book is not intended as an introductory text to functional programming nor to SML. Herein lies its principle problem—the amount of knowledge assumed on the part of the reader is not clear. The preface states that some familiarity with functional programming is assumed; however on opposite pages we find the term 'head of list' defined whereas curried functions are used without explanation—it is difficult to imagine a reader who understands the latter without already understanding the former! Other minor quibbles are that the use of pre/post specifications is at times inconsistent (varying from vague type requirements to precise relationships between function arguments) and the inexplicable lack of use of SML's exception handling mechanism (using instead `hd` □ to raise *every* exception).

These points aside, this book has admirable objectives and, by and large, achieves them. The advantage of the functional approach is that the relationship between specifications and implementations is clear, unlike in conventional treatments where this relationship is easily lost in the mire of code. For those students of software engineering with a good grasp of functional programming, who wish to learn the rudiments of data abstraction, this is an excellent text.

P. MUKHERJEE
University of Birmingham

COLIN MYERS, CHRIS CLACK AND ELLEN POON
Programming with Standard ML. Prentice-Hall, 1993, £19.95, 301 pp softbound, ISBN 0-13-722075-8

This book is intended as an introduction to functional programming using a functional subset of Standard ML. It is aimed at those with little programming experience as well as more experienced programmers who wish to learn about functional programming. It intentionally differs from other texts on the same subject by avoiding mathematics and mathematical reasoning whenever possible, instead concentrating on more practical programming issues.

The book begins by introducing fundamental functional programming concepts such as expressions, types and referential transparency. From there, the notion of a function is introduced. Simple examples of functions are presented and the ideas of recursion and polymorphism are described. The principle of allotting a single purpose to each function when decomposing a problem is continually stressed, both explicitly and by example.

The third chapter is a long chapter on lists. The notion of a list is introduced, functions over lists are described and different kinds of recursion over lists are discussed. An extended example is presented—a function similar to the Unix *grep* command—to demonstrate that 'real' software can be written in a functional programming language. The remaining distinguishing property of functional programming, namely curried and higher order functions, is described in chapter 4, having been motivated by the introduction of lists. Standard functions such as *map*, *foldl* and *foldr* are introduced.

Chapters 5–7 give a detailed description of the features available in Standard ML, based on the ideas of local definitions, encapsulation, information hiding and modularity. Thus these chapters are concerned with more practical issues than the preceding chapters which attempt to teach the principles of sound program design in a functional style.

A consequence of the intentional absence of mathematics is that there is no mention of proving properties of programs. In particular, correctness is not treated at all. Related ideas such as validation of functions are similarly absent. These aspects of software construction are hardly dry theoretical issues; on the contrary, they are vital for software construction to be considered as an engineering discipline. Moreover, this is one area where functional programming languages score heavily over imperative languages because referential transparency greatly eases reasoning. Even if a detailed treatment of program proof and correctness was undesirable, a brief discussion of the merits of correctness by construction would have been sufficient.

The book also occasionally runs into problems due to its stated ambition of appealing to both those with little or no programming experience and experienced programmers wishing to learn a functional language. For instance, the relative simplicity of the functional

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 funtions, 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