vision applications and to those who suspect that computer vision might offer solutions to their problems but who have no experience in the field and require a fairly comprehensive and very comprehensible introduction.

SIMON CLIPPINGDALE
University of Warwick

JOSEPH BERGIN
Data Abstraction, the Object-Oriented Approach Using C++. McGraw Hill, 1994, £41.95, 666 pp., hardbound, ISBN 0 07 911691 4

This book aims to teach data abstraction, object-oriented programming and computer science in that order of priority. In these aims I believe the book largely succeeds, with two qualifications described below.

The first part of the book ('Fundamentals') introduces C++ with enough detail so that examples in the book can be understood. (The author makes clear that the book is not intended as a comprehensive primer for C++ and a basic knowledge of C is assumed.) Concepts of data abstraction, and object-oriented (OO) programming are then introduced followed by an introduction to 'high level' classes like 'object', 'magnitude' (characters, strings, fractions etc), 'collection' (sets, bags, lists, etc), and so on. The ideas are then applied to the design and implementation of a deterministic finite state automaton and to the translation of arithmetic expressions.

The second part of the book ('Implementations') gives a relatively detailed account of representational structures including lists, sets, stacks, queues, trees, characters, strings, heaps, graphs, with related concepts such as sorting and recursion.

The last part ('Applications') gives examples in areas such as program translation, recursive descent parsing, program verification and database normalization.

One reservation I have about the book is tht it puts what I feel is an unreasonable emphasis on data and data abstraction at the expense of procedures and the abstraction of procedural code. It is misleading to define software objects as 'data elements' since, in OO languages from Simula onwards, an object comprises either or both of data structures and procedures (methods or functions). Inheritance, which is the key abstraction mechanism in OO languages, applies to procedural code just as much as it does to data structures. And structures like lists and trees can be, and often are, used to represent procedures as well as data (witness Lisp and OO extensions of Lisp). 'Software Abstraction' would provide a more even-handed focus for the book.

I also have a reservation about the way the book concentrates almost exclusively on computer-oriented concepts (lists, arrays, finite-state automata, etc) and neglects what I believe is the great strength of the the OO paradigm: to facilitate the modelling of objects and classes in the world *outside* the computer. The author recognises that 'object-oriented programming ... offers the benefit of making it possible, and easy, to make the running program more nearly model the real situation in which the original problem arose ... every program is a simulation or model of some phenomenon' (p. 46). It is true, as he says, that computer-oriented concepts may be modelled in software. But most applications of computers are about 'real world' things like people, cars and aeroplanes, and about procedures, processes and rules in the world outside the computer.

OO design allows a 1:1 mapping from these kinds of objects (and classes of objects) to corresponding 'objects' (and 'classes') in software. Despite the attractive simplicity of this idea, subtle problems of analysis can arise. In my experience, students need plenty of examples to show how OO modelling can be applied in the design of software to assist the management of things like libraries, shops, clubs, warehouses, elections and so on. It would be good if the 'Applications' section of any future edition of this book could plug this gap.

Leaving these points aside, the book is clearly written and well presented. It is good to have the C++ examples supplied on a disk. For any course which aims to teach representational devices (lists, stacks, arrays, etc) and associated concepts within the object-oriented paradigm, this book will serve very well.

GERRY WOLFF
University of Wales, Bangor

KEVIN LANO and HOWARD HAUGHTON (Eds)
Object-oriented Specification Case Studies. Prentice Hall, 1994, £22.95, 236 pp., softbound, ISBN 0 13 097015 8

This book splits naturally into two main parts. These are discussed here in reverse order. The second part comprises of Chapters 4–10 inclusive and this contains the specification case studies which give the book its name. These are drawn from a number of areas. Included are specifications of the Unix filing system written in MooZ, a mobile phone system in Object-Z, a concept-recognition system—a kind of machine-learning program—in Z^{++}, a variety of bank accounts and a block-structured symbol table in OOZE, various quadrilaterals in VDM++ and a partial specification of a process scheduler in SmallVDM. Also included in this part of the book is an 18-page chapter entitled 'Refinement in Fresco'. Although this contains some small specifications of quadrilaterals, it is predominantly devoted to the topic of the refinement of specifications into Smalltalk and the proof system needed for this. In my opinion, Lano and Haughton should not have included this chapter in their book as it has a very different emphasis and focus from everything else that the book contains. There is a very pronounced emphasis on object-oriented specification languages based on Z in this part of the book. In fact, 70% of it is devoted to discussing case studies written in languages that are extensions of Z, whereas only 17% is devoted to languages derived from