Unfortunately, I find the title of the book somewhat misleading since the number of applications to concrete problems is limited to a few toy-examples. Rather, application of temporal logic is here to be understood as its adaptation within different theoretical branches of computer science. Therefore, the book will probably not be of particular interest to the software engineer unless interested in the perspectives of formal methods.

Hans Henrik Løvengreen
Department of Computer Science
Technical University of Denmark
Lyngby, Denmark


This book takes a modern approach to data structures. Its content is that of a typical data structures course: it looks at stacks, queues, priority queues, lists, trees, files, and so on; it shows how data structures can be implemented by algorithms over programming language types such as arrays, records and pointers; and it examines the performance characteristics of various data structures and algorithms, particularly in relation to searching and sorting. Its approach is modern in that it uses abstract data type specifications to provide abstract views of data structures, free from representation details. Operations on abstract data types are mapped to procedures that manipulate data structures, with the goal of separating what a programmer needs to know to use a data structure from how the data structure is implemented. This separation is reinforced by examples that show the use of a data structure within a program as manipulations of the procedures that provide an abstract view of the data structure, before possible representations have been discussed.

As the title of the book suggests, all the implementations are in Pascal. This limits the book to using data abstraction as a basis for improving the process of designing and implementing the major data structures within a program, rather than as a basis for the harder task of designing the architecture of a complete program around some module or package construct. The use of a now old-fashioned programming language also seems to encourage various unwelcome practices. For instance, in different chapters we find different solutions to the problem of what to do when a procedure is called when its pre-condition is not satisfied, without proper discussion of the differences. And in Chapter 2 we find that a change to the representation of
the stack type is accompanied by a change to the abstract view of the procedures associated with the type. Overall, the book would have benefited from a more systematic way of moving from abstract data types to program components.

The book is written in a readable style and contains plenty of examples, diagrams and exercises, especially in the early chapters (like many data structures books, it does not show the reader how to implement the more advanced ways of organising external files in its chosen programming language).

The book can be recommended as a supporting text for an introductory data structures course which sets out to give students a first taste of data abstraction and which uses Pascal for the practical work.

R.J. Mitchell
Information Technology Research Institute
Brighton Polytechnic
Brighton, United Kingdom


Bertrand Meyer's view on object-oriented programming, as expressed in this book, is that it is an extremely useful tool for the systematic construction of high-quality software. This is in sharp contrast with the, apparently prevailing, view that object-oriented programming is best suited for rapid prototyping, where the speed of creating new applications and modifying existing ones is considered much more important than the quality of the resulting software. It is quite clear that these different approaches require different tools and techniques.

The first part of the book introduces the main aspects of software quality (correctness, robustness, extendibility, reusability, and compatibility) and indicates in broad terms how object-oriented programming and design can help to achieve these goals. Object-oriented design is contrasted with top-down functional design and it is explained why the latter is not very suitable for producing reusable and extendible software.

In the second and largest part, object-oriented programming is presented in more detail. The basic notions of objects, classes, fields (called instance variables in Smalltalk), routines (often called methods), and inheritance are carefully explained and illustrated with many programming examples. The question of how to design systems in an object-oriented way is given much attention. The programming notation used is the language Eiffel, which the author has designed himself. In this way, the book becomes an introduction to Eiffel as well as to object-oriented programming in general.

Part three of the book discusses how to apply these object-oriented techniques in environments different from Eiffel. The general conclusion here is that this cannot be recommended. Languages like Pascal and C do not provide the required information hiding facilities. Ada and Modula-2 provide mechanisms for modularity but