

**Comparative Review:
Martin and Odell,
Object-Oriented Analysis and Design (1992),
Coad and Yourdon,
Object-Oriented Analysis (1991),
Embley et al.,
Object-Oriented Systems Analysis (1992),
Booch, *Object-Oriented Design
with Applications* (1991),
and Rumbaugh et al.,
Object-Oriented Modeling and Design (1991).**

T.H. Tse
Department of Computer Science
The University of Hong Kong
Pokfulam, Hong Kong

The most popular software engineering methodologies in the 1980s were entity modeling and structured analysis and design. A few problems have been uncovered over the years, however. These methodologies concentrate only on one or another aspect of the target system. In entity modeling, for example, the emphasis is on the static relationships of data, whereas in structured analysis, the emphasis is on the relationships among processes. Besides, the average software system has increased so much in size and complexity that it is no longer practical to develop a system as a single entity, even with a top-down approach. Furthermore, every component of the system is assumed to be designed and implemented from scratch. The reuse of the same model for different purposes is explicitly discouraged.

Object-oriented analysis and design have emerged quickly in the last few years and are considered to be the most influential software engineering methodologies for the 1990s. They originated from the object-oriented programming paradigm, which models the real world by a set of objects with their own attributes, behavior, and relationships with one another. Like their programming counterpart, object-oriented analysis and design support:

- classes, which are collections of objects that have common structures and behavior,
- data abstraction, which allows us to visualize objects with varying degrees of detail depending on specific needs,
- encapsulation, which hides the implementation details of objects,
- inheritance, which allows subclasses to reuse properties of parent classes, and

- polymorphism, which allows operations to have the same name but varying effects on different classes of objects.

In addition to the properties of object-oriented programming, object-oriented analysis and design help to specify systems in a graphical manner, enabling software engineers to visualize the requirements and communicate with users easily. The graphical tools are mostly based on the popular notations in entity modeling and structured analysis, but are more comprehensive and have an integrated context. The transition from analysis to design to implementation is less drastic than in structured methodology. The results of object-oriented analysis and design can be implemented using object-oriented programming languages and conventional programming languages as well as database management systems.

Table 1: Quantitative Data

	Martin and Odell	Coad and Yourdon	Embley et al.	Booch	Rumbaugh et al.
Number of chapters	29	10	8	12	20
Number of pages	513	233	302	580	500
Pages in bibliography	*	8	4	46	*
Number of references	*	97	68	Over 900	*
Pages of appendices	56	23	36	23	12
Exercises?	No	No	Yes	No	Yes
Answers to exercises?	No	No	No	No	Yes
Case studies?	No	Yes	Yes	Yes	Yes
Glossary?	Yes	No	No	Yes	Yes
Audience	Intermediate	Intermediate	Advanced	Advanced	Advanced
Analysis / design	Both	Analysis	Analysis	Design	Both

* *Distributed throughout the book.*

Martin and Odell

This book is an excellent text for professionals who would like to enter the world of object-oriented analysis and design. A major portion of the book (Parts 1 and 2) introduces the fundamental ideas in object-oriented analysis, design, and programming. Graphical techniques using object-relationship diagrams, event schemas, process-dependency diagrams, object-flow diagrams, and state-transition

diagrams are discussed. Part 3 presents methods to deal with object structures in terms of object relationships. Part 4 covers object behavior in terms of state transitions. Finally, Part 5 presents ideas on object-oriented design in terms of the conversion of the results of the analysis into programs. Emphasis is placed on independent techniques rather than on a complete methodology.

Like other works by Martin, this book tries to impress readers through hard sell, such as “This book [also by Martin] has been a ‘bible’ for many CASE vendors” (p.121), authoritative views, such as quoting from Professor Tony Hoare (p. 30), and jargon such as “parametric polymorphism” (p. 165). Such attempts may backfire, however, if the authors are not careful about the facts. For example, they introduce Tony Hoare as “Christopher Hoare,” and “parametric polymorphism” should read “inclusion polymorphism.”

In any case, the book is outstanding as an intermediate text for professionals. It is very readable, and nontrivial ideas are presented in an easy-to-understand way. It dares to introduce some of the more theoretical ideas on object-oriented analysis and design in a commercial package.

Coad and Yourdon

This book approaches object-oriented analysis from a commonsense point of view. The definitions are drawn from major dictionaries and encyclopedias. The section on “principles of managing complexity” introduces the fundamental object-oriented concepts and the reasons why they are necessary. The authors then provide a historical perspective on object-oriented analysis via a review of functional decomposition, data flow, and information modeling. The book introduces a complete object-oriented analysis (OOA) methodology in five major steps: finding class-and-objects, identifying structures, identifying subjects, defining attributes, and defining services. Finally, it discusses follow-up issues, such as CASE support, and object-oriented design and management issues.

The OOA model is divided into five layers: subjects (subsystems), class-and-objects, structures (which include inheritance and whole/part relationships), attributes, and services (which define the methods). All the layers are conglomerated into an “object diagram,” since the authors consider it easier for users to learn only one type of diagram. The object diagram is in fact an overlay of dataflow diagrams and extended entity-relationship diagrams, however, and may lead users into unnecessary and confusing details in large systems.

Given the popularity of the authors, this book is a little disappointing. Having said that, it is only fair to point out that its first edition was published at least a year before the other books on the subject. It proposes a complete OOA methodology, but advanced readers may find insufficient details for implementing the proposals. Furthermore, the text and figures are rather large, and some of the material is duplicated too many times for a book of this size. For example, the principles of managing complexity appear on pages 12, 18, 33, and 181, and the multilayer model appears on pages 35, 54, 64, and 152.

Embley, Kurtz, and Woodfield

Like the other books reviewed, this work proposes object-oriented systems analysis (OSA) in terms of three basic models: an object-relationship model using extended entity-relationship diagrams, an object-behavior model using state-transition diagrams, and an object-interaction model (chapters 2, 3, and 6). Chapters 4, 5, 7, and 8 then discuss the application of these models to complex analysis problems.

This book differs from most other texts in several respects. First, it makes an interesting attempt to integrate the three views of a system, providing a systematic means of detecting errors in the specification. Second, an interesting formal theoretical framework lies behind the graphical notations, which is expressed in terms of first-order predicate calculus and model theory in Appendix A. Its application to formal methods, not shown in this book, is worth pursuing. On the other hand, the concept of attributes is not supported by the models. As a result, attributes have to be shown on the same level as objects in object-relationship models, so that the complexity of objects will not be hidden from users. Moreover, further expansion of the techniques of OSA modeling will be required before it becomes a complete methodology.

Booch

This book introduces fundamental concepts of “object modeling” using examples in object-oriented programming. It then presents the notation for Booch diagrams and a methodology for systems development, with an emphasis on design. Five case studies are presented, complete with details of analysis, design, and implementation in Ada, C++, Common Lisp Object System (CLOS), Object Pascal, and Smalltalk.

Booch has remained the most-quoted author on object-oriented design. Booch diagrams are the earliest attempt to model object-oriented development graphically. The diagrams go into less detail than those of other authors, but this is made up for by detailed recommendations about program implementation.

Booch introduces object-oriented concepts using a bottom-up approach. He expresses them in terms of object-oriented programming. This choice is understandable given the historical development of Booch diagrams. The book tries to give equal time to object-oriented programming languages from the beginning. For example, abstraction is explained using Ada and C++ (on pages 42 to 44), encapsulation using Smalltalk (on pages 47 to 48), modularity using Object Pascal (on pages 53 to 54), inheritance using C++ (on page 56), and multiple inheritance using CLOS (on pages 57 to 58). Thus, readers of the book should be familiar with object-oriented programming.

The book is unbeatable as a reference. Complete details of case studies take up 249 pages. A classified bibliography with over 900 entries is presented, in addition to 17 pages of reference notes. A summary is provided at the end of each chapter, with annotated suggestions for further reading.

Rumbaugh, Blaha, Premerlani, Eddi, and Lorensen

I regard this as the best book on the subject. An object modeling technique (OMT) is proposed. The authors present a complete methodology, from analysis to design to implementation. Part 1 of the book discusses the OMT notation. Extended entity-relationship diagrams are used for object modeling, state-transition diagrams for dynamic modeling, and dataflow diagrams for functional modeling.

Part 2 deals with the OMT methodology, including analysis, system design, and object design. Practical recommendations for the analysis phase, such as identifying objects, classes, attributes, associations, and events, are given in detail. Design issues such as layering, partitioning, concurrency, control, optimization, boundary conditions, and tradeoffs are presented. The authors also compare OMT with other methodologies, such as structured analysis and structured design, Jackson structured development, information modeling, and other object-oriented methods.

Part 3 suggests ways of implementing the object-oriented design using various target environments, including object-oriented languages (such as C++, CLOS, Eiffel, and Smalltalk), non-object-oriented languages (such as C, Ada, and FORTRAN), and relational databases. Details of every mapping are given, including class definitions, object creation, methods, inheritance, and associations. Unlike readers of Booch, we need not be concerned about object-oriented programming in order to understand the conceptual and methodological parts of the book. Finally, three case studies are presented in Part 4.

OMT covers the whole spectrum of object-oriented diagramming notations, analysis, design, and implementation. It appears to have the most user-friendly notations and concepts among the various object-oriented methods. The notations are sufficiently general that they can be used easily by experienced users of structured methodologies.

Conclusion

The books reviewed here have more things in common than differences. Most recommend a collection of graphical notations for object-oriented analysis and design. Most of the diagrams used are extensions of those used in entity modeling and structured methodology. In simple terms, they consist of extended entity-relationship diagrams, dataflow diagrams, and state-transition diagrams, or variations on those themes. The amount of extension depends on individual authors, ranging from the almost direct adaptation of Rumbaugh et al. to Booch's self-contained object-oriented diagrams. All the approaches recommend the use of common English phrases, such as "is-a," "is-part-of," and "is-a-member-of," to identify relationships among objects. Unlike structured methodology, they recommend a gradual transition from the analysis to the design phase. Implementation can be done using various object-oriented or conventional programming languages, or using database management systems.

Martin and Odell, and Coad and Yourdon, are intermediate texts, suitable for professionals who are entering this interesting area. Booch, Embley et al., and Rumbaugh et al. are recommended for advanced users who would like more details.

If you would like a book on the fundamental concepts in object-oriented analysis and design, you should choose Martin and Odell. If you want a brief look at a complete object-oriented analysis methodology, you may want to consider Coad and Yourdon. For a more detailed look at object-oriented analysis techniques plus an integrated view of the results, Embley et al. is the book to choose. For a book on object-oriented design with a view to program implementation, Booch is the one. If, however, you would like all of the above, I strongly recommend Rumbaugh et al. It is probably the best buy on the subject of object-oriented analysis and design.