D. Software

D.2 SOFTWARE ENGINEERING

LISKOV, BARBARA; AND GUTTAG, JOHN 8709-0716
(Massachusetts Institute of Technology, Cambridge)
Abstraction and specification in program development.
MIT Press, Cambridge, MA, 1986, 469 pp., $39.95, ISBN 0-262-12112-3. [The MIT electrical engineering and computing science series.]

This is a splendid, up-to-date introduction to software engineering, suitable for either an undergraduate course or self-study. It has a wide range of topics: from techniques for designing large programs to purely theoretical aspects of formal verification.

Introductory texts on software engineering are a mixed lot. There are some excellent books on programming issues, such as Fred Brooks's *The mythical man month* [1], and some excellent books on stylistic and low-level software issues, such as Kernighan and Plauger's *The elements of programming style* [2] or Bentley's *Writing efficient programs* [3]. Liskov and Guttag's new book fits nicely between these two approaches by addressing the problems that face the individual programmer writing a large program or system.

As the title suggests, the unifying theme of this book is the use of abstraction and specification to produce clean, robust, highly modular, and efficient software. When they discuss abstraction, the authors present data abstraction (the use of high-level constructs to support object-oriented programming), procedural abstraction (the decomposition of problems into hierarchical units), and control abstraction (the use of iterators as a control structure to loop through objects known only by their data abstraction). With emphasis on informal methods, the authors use specifications not merely as a documentation technique but as an integral part of the software design and implementation process. The authors illustrate their techniques with numerous examples.

One important feature of this book is the use of the programming language CLU. By examining the structure of CLU, the book is able to give a detailed exposition of the abstraction mechanism. It would be difficult to give such a clean description if the authors had chosen C or Modula-2 as their base language. Ada certainly supports abstraction well, but a discussion of Ada would also bring in many extraneous complications that would clutter the text. CLU was an excellent choice of language since it provides good support for exactly those features needed for data and control abstraction. Since many readers will program in languages other than CLU, the authors are careful to explain at length exactly how abstraction and specification can be used in languages such as Pascal and Ada. For readers not familiar with CLU, the book includes a leisurely introduction to the language as well as a reference manual.

Along the way, the authors present a thorough survey of all of the pragmatic issues confronting a software engineer: the software life cycle, requirements analysis, design, documentation, evaluation, testing, validation, and verification. Of particular interest are the chapter on using exception handling methods to produce programs with graceful degradation and the chapter on testing and debugging. These chapters carefully cover material not usually found in introductory books on software engineering. For those of a more theoretical bent, the authors also cover formal methods for specification and verification (after placing these techniques in context). The theoretical material is well presented, but is somewhat disjointed and denser than the rest of the text.

The reviewer used this book as a basic text for an advanced undergraduate course in software engineering at Carnegie-Mellon University. The book is well suited for this purpose. The writing is crisp and holds the student's interest, and the examples clearly illustrate the points made by the authors. Although there is more material in the book than could be covered in a single semester, many of the later chapters are self-contained and can easily be omitted. Two extremely useful features for an instructor are an appendix that contains a sequence of student programming projects and an extended example of a simple text formatter that demonstrates how the concepts introduced in the book fit together. The example starts from design considerations and meticulously shows the steps a programmer works through to get the final program. This approach has two advantages. It gives an additional illustration of the individual methods presented earlier, and it provides students who learn by studying examples with a sample of conscientious programming. Without doubt, this book will be widely appreciated by both students and practicing software engineers.

—*J. D. Tygar,* Pittsburgh, PA

References

[1] Brooks, F. P., Jr. *The mythical man-month: essays on software engineering,* Addison-Wesley, Reading, MA, 1975. See *CR* **16**, 10 (Oct. 1975), Rev. 28,944.

[2] Kernighan, B. W.; and Plauger, P. J. *The elements of programming style,* McGraw-Hill, New York, 1978. See *CR* **20**, 2 (Feb. 1979), Rev. 34,059.

[3] Bentley, J. L. *Writing efficient programs,* Prentice-Hall, Englewood Cliffs, NJ, 1982. See *CR* **23**, 10 (Oct. 1982), Rev. 39,767.

GENERAL TERMS: DESIGN, LANGUAGES, VERIFICATION