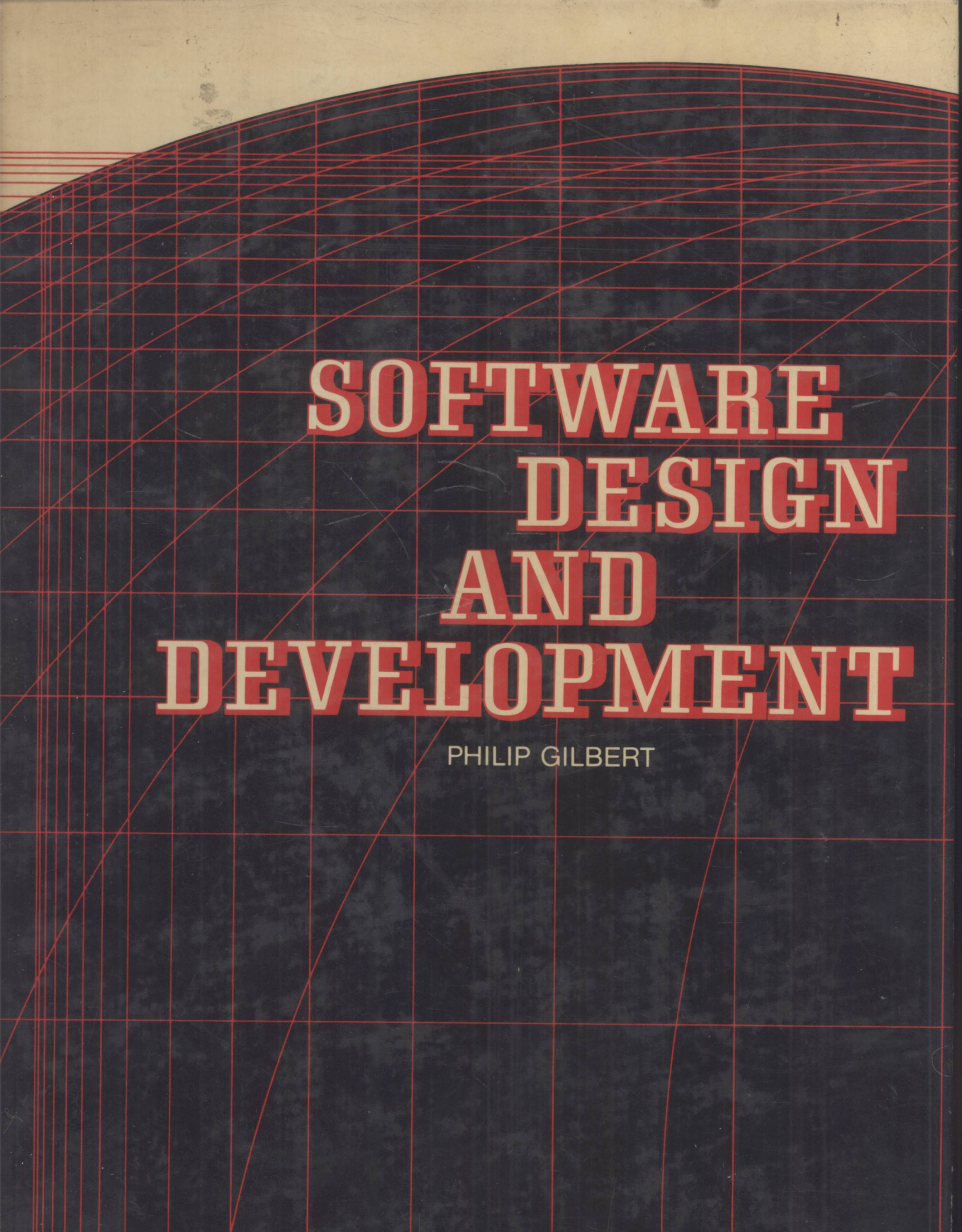
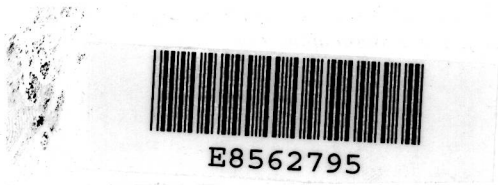

SOFTWARE
DESIGN
AND
DEVELOPMENT
PHILIP GILBERT

TP31
G26

8562795

SOFTWARE DESIGN AND DEVELOPMENT



Philip Gilbert

California State University, Northridge

SRA® SCIENCE RESEARCH ASSOCIATES, INC.
Chicago, Henley-on-Thames, Sydney, Toronto
A Subsidiary of IBM

The SRA Computer Science Series

William A. Barrett and John D. Couch, *Compiler Construction: Theory and Practice*
Marilyn Bohl and Arline Walter, *Introduction to PL/I Programming and PL/C*
Mark Elson, *Concepts of Programming Languages*
Mark Elson, *Data Structures*
Peter Freeman, *Software Systems Principles: A Survey*
C. W. Gear, *Introduction to Computer Science: Short Edition*
Philip Gilbert, *Software Design and Development*
A. N. Habermann, *Introduction to Operating System Design*
Harry Katzan, Jr., *Computer Systems Organization and Programming*
Henry Ledgard and Michael Marcotty, *The Programming Language Landscape*
James L. Parker and Marilyn Bohl, *FORTRAN Programming and WATFIV*
Stephen M. Pizer, *Numerical Computing and Mathematical Analysis*
Harold S. Stone, *Discrete Mathematical Structures and Their Applications*
Harold S. Stone, *Introduction to Computer Architecture, Second Edition*

Acquisition Editor	Alan W. Lowe
Project Editor	James C. Budd
Editor	Betty Berenson
Production	Bookman Productions
Compositor	Lehmann Graphics
Illustrator	Reese Thornton
Designer	Judith Olson

Library of Congress Cataloging in Publication Data

Gilbert, Philip, 1933–
Software design and development.

(SRA computer science series)
Bibliography: p.
Includes index.
1. Electronic digital computers—Programming.
2. System design. I. Title. II. Series.
QA76.6.G553 1983 001.64'25 82-16817
ISBN 0-574-21430-5

Copyright © Science Research Associates, Inc. 1983.
All rights reserved.

Printed in the United States of America.

10 9 8 7 6 5 4 3 2

SOFTWARE DESIGN AND DEVELOPMENT

To Dara and Josh

试读结束，需要全本PDF请购买 www.ertongbook.com

Preface

AUDIENCE AND OBJECTIVES

This book presents methods for developing small- to medium-scale software systems, with emphasis on the:

- Necessity of understanding the problem
- Importance of planning and design
- Integration of documentation into the development process
- Use of techniques to improve the quality of designs and programs

The increasing use of microcomputers will soon result in the widespread development of small- to medium-scale software systems. My intention is to prescribe the development of such a system in a way that is understandable to—and usable by—students whose previous experience in programming may be limited to one or two programming courses. The presentation is directed to the sophomore and junior undergraduate levels and presumes a knowledge of a structured programming language. Example programs are shown in PASCAL code or pseudocode and should be easily understandable by students who know a structured programming language.

The book is suitable for a second course in computing, or it can be used at a more advanced level. It covers the material of the undergraduate course in software engineering recently proposed as part of a model undergraduate electrical engineering curriculum by the Educational Activities Board of the Institute of Electrical and Electronic Engineers (IEEE). Such a course is also an appropriate elective for computer science students. At California State University, Northridge, all computer science majors take a junior design course, for which this text was developed. Successive versions of this text have been used in this class over the last five years.

Last but not least, the text is suitable for an Information Systems Curriculum, in a course on Systems Analysis and Design. It brings together many pertinent topics, including requirements analysis based on structured systems analysis techniques, data dictionaries, process specifications using

decision tables, specification-based testing, and specification and evaluation of desired system qualities—as well as quality assurance and other project management techniques. Also presented are the data structure-based design method of Jackson and Warnier, and Jackson's technique for system design via process simulation. This text covers almost all of the material in the IS8 Systems Design Process Course contained in the Information Systems Curriculum Recommendations just made (Communications of the ACM, November 1982) by the ACM Curriculum Committee on Information Systems. About half of the material in the IS5 Information Analysis Course in that curriculum is also covered.

System development involves topics such as module organization, testing strategies, and project management. On the other hand, the limited experience of expected readers has dictated careful discussions of program structuring (using top-down design), implementation techniques, and testing methods. The presentation is balanced between larger-scale system topics and smaller-scale program topics.

Each phase is treated in a unified manner. Where several different methods exist, the discussion shows the relation of each method to the others and integrates them into a coherent whole. The concern is not to survey or contrast different methods but instead to show how they may be used in concert.

Concept and application are equally stressed. The techniques presented, while strongly based in concept, work for real-life problems. Concepts are clearly stated, and problems that sharpen understanding of the concepts are included. Wherever possible, a procedure for applying the concepts is also given—as a comprehensive sequence of steps, as a checklist, or as a project outline. Problems in the appendixes serve as a focus for student projects based on the project outlines.

A course in software design is not complete without the experience of projects. To provide such experience, problems in design (Appendix D), in evaluation (Appendix E), and in system formulation (Appendix C) are furnished. Almost every chapter has a project outline and sets of problems, and thus a project pertaining to each chapter can be applied to problems in the appendixes.

We emphasize design. For each phase of design, the underlying problems, solution possibilities, and principles are discussed, and a comprehensive approach (usually in the form of a sequence of steps or procedures) is given to guide the application of the technique to real problems. Design projects and problems are included.

The relation between design and evaluation/testing is also emphasized. Critical evaluation of program specifications and of design solutions is stressed. Testing is included in the project outlines of Chapters 2 and 3 to help foster the student's ability to evaluate designs objectively. Problems in Appendix E deal with the evaluation and analysis of previously written programs.

USE AT ELEMENTARY AND ADVANCED LEVELS

The topics in this book have varying levels of difficulty. Most of the book presents techniques for getting the job done right—the basic idea in design and development. These topics vary from simple code evaluation (Section 8.4) and charting methods (Section 4.1) through structured programming (Section 7.1) to more complex design topics. (A sequence of topics for inexperienced students is suggested later in this preface.)

A second layer of topics is concerned with overview of software development: documenting, ensuring that high quality is attained, guiding the system evolution (via testing, development, and delivery strategies), and managing the project. For advanced students, the materials on documentation, guidance, management, and quality can be strongly emphasized, by the use of extensive projects and papers. The problems in Appendix C are suitable for this approach.

The combination of basic and advanced information may be suitable for graduate students whose undergraduate work is in other disciplines. For these students, a spiral approach might be used, in which the basic techniques are discussed and then the development cycle is reconsidered from an advanced viewpoint.

PLAN OF THE BOOK

Part One, "Introduction," consisting of Chapter 1, gives an overview of the software development process, motivates the study of design and stresses the need for quality assurance and documentation in the development process. In accordance with this view, almost every chapter in the rest of the book has a section discussing appropriate documentation and one that presents techniques for assuring quality.

The next three parts present software development steps, in the order that would occur in development of a medium-scale system. Part Two, "Initial Design Steps," consists of Chapter 2, "Discovering the Problem," and Chapter 3, "The Design Concept." Chapter 2 discusses requirements analysis, program specifications, and quality specificatios. Data flow diagrams (called requirements diagrams in Chapter 2) are presented to annotate a network of activities, and to show the data flows between activities. The use of decision tables to define or analyze an activity specification (equivalently, a program specification) is shown. The student's attention is directed to understanding user needs and the possibilities for variations in the problem. The quality specification, which precisely states desired characteristics of a software system, is introduced. Methods of defining and measuring qualities are discussed.

Chapter 3 discusses the overall design approach. The use of data flow techniques is shown for a simple problem and for a design problem involving complex data transformations. Design using simulation models is discussed, and a documenting technique that aids development is described. Evaluation of the initial design with respect to the quality specification is also discussed.

Part Three considers system design and development. Chapter 4 introduces organizations of modules. Since undergraduate students are usually not familiar with the annotation of organizations of modules or the complexities and possibilities of such systems, these points are systematically introduced. Section 4.1 introduces the organization diagram to annotate organizations of modules. Section 4.2 develops all possible organizations for a simple program (thereby giving practice with organization diagrams), gives details of the more complex organizations, and compares the organizations. Chapter 5 discusses strategies for deriving program organization, for testing, and for implementation. A principle technique in Chapter 5 is the derivation of program organization from a data flow diagram, by partitioning the diagram.

Part Four deals with module design and development. Chapter 6 presents several techniques for module design. The necessity for analyzing problem situations and algorithms is stressed, and hints about analysis are given. The finite state model, data-structure-based design, and table-directed design techniques are discussed. Then Chapter 7 discusses module implementation using the top-down design technique of stepwise refinement.

Chapter 8 deals with problems of program construction. The features of popular programming languages, points of programming style, and the use of measurement or estimation techniques to achieve greater program execution speed are some of the topics presented. Chapter 9 considers test methods. The discussions of Section 9.2 parallel those of Section 2.3, and the example of Section 9.2 clearly illustrates the concepts of Section 2.3.

Finally, Chapter 10, which comprises Part Five, is concerned with management issues. The topics discussed are key points in managing software projects, organizing and scheduling techniques, resource estimation, peer reviews, and constraints on software development.

SEQUENCE OF TOPICS

Chapters 2 through 9 carry the student through the production steps of a medium-scale program system, with system design presented in Chapters 4 and 5. As is appropriate for a medium-scale system, large-scale problems such as module organization and testing strategies are considered before small-scale problems of module design and development.

The chapters have been written to allow easy variation of the sequence of topics. Single-program or small-scale development can be treated first, and

medium-scale development can be treated later, by using Chapters 3, 6, and 7 in sequence. For example, the material in Section 6.1 relates back to discussions in Chapter 3. Module design material in the rest of Chapter 6 can be immediately treated or can be skipped initially and reconsidered later. Discussions in Chapter 7 relate back to Section 6.1 and also to Chapter 3.

Section 2.3, program specifications, and Section 9.2, on specification-based testing, are strongly related; the example test in Section 9.2 illustrates the concepts and techniques of Section 2.3. Accordingly, it may be desirable to discuss Chapter 2 and then Section 9.2 before proceeding to Chapter 3, to reinforce the notion of designing to specifications. Another possibility is to begin with a partial treatment of Chapter 9, to immediately introduce the notion of critical evaluation and test of designs, and then continue with Chapter 2. For this variation, an introductory project might involve writing a program for a simple problem (Problem 1 in Appendix D is a good choice) first, then writing a test to evaluate the solutions of other students, and finally cross-evaluating student solutions with student tests.

For students with very little programming experience, it may be useful to begin the course with documentation and evaluation, applying the techniques of Sections 4.1 and 8.4 to projects in Appendix E, in order to foster programming familiarity and to introduce immediately useful evaluation and documentation techniques. Structured programming and stepwise refinement (Sections 7.1 and 7.2) could then be presented, followed by simple applications of data flow concepts (Section 3.2). The course presentation might end in discussion of documentation and evaluation at a higher level.

ACKNOWLEDGMENTS

I am greatly indebted to the work of others. I have been much influenced by the works of Dijkstra, Parnas, Yourdon, and Kernighan and Plauger, to mention but a few. I am grateful to Robert Persig's *Zen and the Art of Motorcycle Maintenance* for suggesting the notion of quality and also ways to achieve it.

I would also like to thank the following for their help in reviewing the text: William W. Agresti, the University of Michigan—Dearborn; Dr. Anna Mae Walsh Burke, Director, Center for Science and Engineering, Nova University; John D. Gannon, Univerity of Maryland; Dr. A. F. Norcio, U.S. Naval Academy; Gruia-Catalin Roman, Washington University in Saint Louis; and David C. Rine, Computer Science Division, Western Illinois University.

It was Gary Hordemann who pushed me into working full-scale on this book; Jack Alanen read through a draft and pointed out errors of fact, style, and explication. Steven Stepanek and Elaine La Delfa examined and corrected all of the programs in the book (more precisely, all that I remembered to show them). Wendie Diane Edie provided word processing for many drafts.

Alan Lowe guided me through the publication process with great skill. And my wife Hadassa continually encouraged me through it all.

I must thank the many past students in my program design classes at California State University, Northridge. They suffered through the development of this manuscript, unerringly finding the inadequacies of each version.

试读结束，需要全本PDF请购买 www.ertongbook.com

8562795

Contents

Part One
INTRODUCTION

A cautionary tale about myth and reality . . . the basic development cycle . . . the right attitude . . . communication and documentation . . . the need for quality

chapter one
The Software Development Process

1.1 A CAUTIONARY TALE

The phone rang. Before I knew it, the voice on the other end was reeling out a tale of woe. The software never worked properly, he said, and they'd been patching it for over a year. "We really need the system; what can we do? We're desperate for help!"

"Hmm," I said, deciding against the tweed jacket because it was a warm night, "I can be over there in 20 minutes. Of course, you know my fee—payable in advance."

"We've got it in cash."

"Twenty minutes, then." On with the silk instead. I picked up my personal computer and was out the door.

It would be a tricky one, but I knew I could pull it off. . . .

Perhaps the next pulp novel superhero will be the dashing computer consultant armed with trusty personal computer, who rushes off into the night and saves the faltering software system. Because coding is the most visible activity in software development, and because programmers do rush about fixing the errors in their programs, it is easy to fall prey to the myth of the superprogrammer who can instantly fix all of the problems of an ailing program. Unfortunately, the real story almost always has a different last line.

I didn't tell him that it was already too late. . . .

It would be wonderful if the original story was usually true. About two-thirds of the overall cost of software is spent in software maintenance [Boe 79b], and the situation of the story—that errors continue to be found after the software development has supposedly been finished—is a sad but common one. Obviously, program maintenance is by far the most costly software activity. Successful maintenance is greatly desired, but all the important decisions that affect maintenance have been made long before maintenance begins [War 78].

Two important factors that could cause our hero's efforts to fail are program scale and complexity. Large scale, in and of itself, changes the nature of a

试读结束，需要全本PDF请购买 www.ertongbook.com