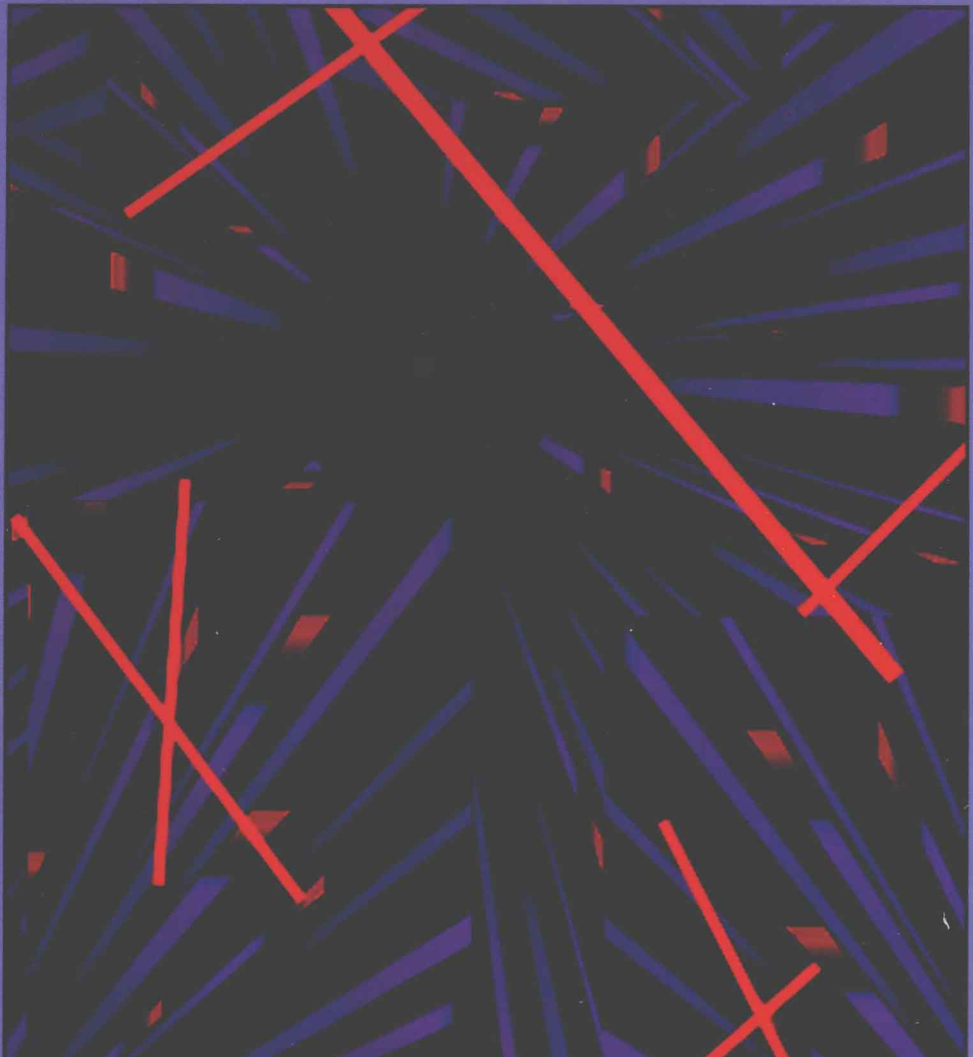
SOFTWARE ENGINEERING

THEORY AND PRACTICE

SECOND EDITION



SHARI LAWRENCE PFLEEGER

SOFTWARE ENGINEERING

THEORY AND PRACTICE

SECOND EDITION

Shari Lawrence Pfleeger

An Alan R. Apt Book

Prentice Hall
Upper Saddle River, New Jersey 07458

Library of Congress Cataloging-in-Publication Data

Pfleeger, Shari Lawrence.
Software Engineering : theory and practice / Shari Lawrence Pfleeger.--[2nd ed.].
p. cm.
Includes bibliographical references and index.
ISBN 0-13-029049-1
1. Software engineering. I. Title.

QA76.758.P49 2001
005.1--DC21

00-051674

Vice President and Editorial Director, ECS: Marcia J. Horton
Publisher: Alan R. Apt
Associate Editor: Toni D. Holm
Vice President and Director of Production and Manufacturing, ESM: David Riccardi
Executive Managing Editor: Vince O'Brien
Managing Editor: David A. George
Production Editor: Scott Disanno
Director of Creative Services: Paul Belfanti
Creative Director: Carole Anson
Art Director: Heather Scott
Art Editor: Adam Velthaus
Manufacturing Manager: Trudy Pisciotti
Manufacturing Buyer: Pat Brown
Senior Marketing Manager: Jennie Burger
Editorial Assistant: Amy K. Todd

Printed in the United States of America

10 9 8 7 6 5 4

ISBN 0-13-029049-1

Prentice-Hall International (UK) Limited, *London*
Prentice-Hall of Australia Pty. Limited, *Sydney*
Prentice-Hall Canada Inc., *Toronto*
Prentice-Hall Hispanoamericana, S.A., *Mexico*
Prentice-Hall of India Private Limited, *New Delhi*
Prentice-Hall of Japan, Inc., *Tokyo*
Pearson Education Asia Pte. Ltd., *Singapore*
Editora Prentice-Hall do Brasil, Ltda., *Rio de Janeiro*

From so much loving and journeying, books emerge.
Pablo Neruda

To Florence Rogart for lighting the flame;
to Norma Mertz for helping to keep it burning.

Preface

BRIDGING THE GAP BETWEEN RESEARCH AND PRACTICE

Software engineering has come a long way since 1968, when the term was first used at a NATO conference. And software itself has entered our lives in ways that few had anticipated, even a decade ago. So a firm grounding in software engineering theory and practice is essential for understanding how to build good software and for evaluating the risks and opportunities that software presents in our everyday lives. This text represents the blending of the two current software engineering worlds: that of the practitioner, whose main focus is to build high-quality products that perform useful functions, and that of the researcher, who strives to find ways to improve the quality of products and the productivity of those who build them.

Designed for an undergraduate software engineering curriculum, this book paints a pragmatic picture of software engineering research and practices. Examples speak to a student's limited experience but illustrate clearly how large software development projects progress from need to idea to reality.

The book is also suitable for a graduate course offering an introduction to software engineering concepts and practices, or for practitioners wishing to expand their knowledge of the subject. It includes examples that represent the many situations readers are likely to experience: large projects and small, object-oriented and procedural, real-time and transaction processing, development and maintenance. In particular, Chapters 12, 13, and 14 present thought-provoking material designed to interest graduate students in current research topics.

KEY FEATURES

This text has many key features that distinguish it from other books.

- Unlike other software engineering books that consider measurement a separate issue, this book blends measurement with software engineering. Measurement issues are considered as an integral part of software engineering strategy, rather than as a separate discipline. This approach shows students how to involve quantitative assessment and improvement in their daily activities. They can evaluate their progress on an individual, team, and project basis.
- Similarly, concepts such as reuse, risk management, and quality engineering are embedded in the software engineering activities that are affected by them, instead of treating them as separate issues.
- Each chapter applies its concepts to two common examples: one that represents a typical information system, and another that represents a real-time system. Both

examples are based on actual projects. The information system example describes the software needed to determine the price of advertising time for a large British television company. The real-time system is the control software for the Ariane-5 rocket; we look at the problems reported, and explore how software engineering techniques could have helped to locate and avoid some of them. Students can follow the progress of two typical projects, seeing how the various practices described in the book are merged into the technologies used to build systems.

- At the end of every chapter, the results are expressed in three ways: what the content of the chapter means for development teams, what it means for individual developers, and what it means for researchers. The student can easily review the highlights of each chapter and see the chapter's relevance to both research and practice.
- The book has an associated Web page, containing current examples from the literature, links to Web pages for relevant tool and method vendors, and a study guide for students. It is on the Web that students can find real requirements documents, designs, code, test plans, and more, so they can see real software engineering project artifacts. Students seeking additional in-depth information are pointed to reputable accessible publications and Web sites. The Web pages are updated regularly to keep the material in the textbook current and include a facility for feedback to the author and the publisher.
- The book is replete with case studies and examples from the literature. Many of the one-page case studies shown as sidebars in the book are expanded on the Web page. The student can see how the book's theoretical concepts are applied to real-life situations.
- Each chapter ends with thought-provoking questions about legal and ethical issues in software engineering. Students see software engineering in its social and political contexts. As with other sciences, software engineering decisions must be viewed in terms of the people their consequences will affect.
- Every chapter addresses both procedural and object-oriented development. In addition, a new chapter on object-oriented development explains the steps of an object-oriented development process. Using UML for common notation, each step is applied to a common example, from requirements specification through program design.
- The book has an annotated bibliography that points to many of the seminal papers in software engineering. In addition, the Web page points to annotated bibliographies and discussion groups for specialized areas, such as software reliability, fault tolerance, computer security, and more.
- The book has a solutions manual, available from Prentice Hall, as are PowerPoint slides with the figures, tables, and sample instructional slides.
- Each chapter includes a description of a term project, involving development of software for a mortgage processing system. The instructor may use this term project, or a variation of it, in class assignments.
- Each chapter ends with a list of key references for the concepts in the chapter, enabling students to find in-depth information about particular tools and methods discussed in the chapter.

CONTENTS AND ORGANIZATION

This text is organized in three parts. The first part (Chapters 1 to 3) motivates the reader, explaining why knowledge of software engineering is important to practitioners and researchers alike. Part I also discusses the need for understanding process issues and for doing careful project planning. Part II (Chapters 4 to 11) walks through the major steps of development and maintenance, regardless of the process model used to build the software: eliciting and checking the requirements, designing a solution to the problem, writing and testing the code, and turning it over to the customer. Part III (Chapters 12 to 14) focuses on evaluation and improvement. It looks at how we can assess the quality of our processes and products, and how to take steps to improve them.

Chapter 1: Why Software Engineering?

In this chapter we address our track record, motivating the reader and highlighting where in later chapters certain key issues are examined. In particular, we look at Wasserman's key factors that help define software engineering: abstraction, analysis and design methods and notations, modularity and architecture, software life cycle and press, reuse, measurement, tools and integrated environments, and user interface and prototyping. We discuss the difference between computer science and software engineering, explaining some of the major types of problems that can be encountered, and laying the groundwork for the rest of the book. We also explore the need to take a systems approach to building software, and we introduce the two common examples that will be used in every chapter. We also introduce the context for the term project.

Chapter 2: Modeling the Process and Life Cycle

In this chapter, we present an overview of different types of process and life-cycle models, including the waterfall model, the V-model, the spiral model, and various prototyping models. We also describe several modeling techniques and tools, including systems dynamics, SADT, and other commonly-used approaches. Each of the two common examples is modeled in part with some of the techniques introduced here.

Chapter 3: Planning and Managing the Project

Here, we look at project planning and scheduling. We introduce notions such as activities and milestones, work breakdown structure, activity graphs, risk management, and costs and cost estimation. Estimation models are used to estimate the cost and schedule of the two common examples. We focus on actual case studies, including management of software development for the F-16 airplane and for Digital's alpha AXP programs.

Chapter 4: Capturing the Requirements

In this chapter, we look at requirements analysis and specification. We explain the difference between functional and nonfunctional requirements, present several ways to describe different kinds of requirements, and discuss how to prototype requirements. We see how several types of formal methods can be used in specifying and evaluating requirements. Other topics discussed include requirements documentation, requirements reviews, requirements quality and how to measure it, requirements testability, and how to select a specification method. The chapter ends with application of some of the methods to the two common examples.

Chapter 5: Designing the System

This chapter focuses on architectural issues, and we begin by discussing Shaw and Garlan's framework for software architecture. Next, we describe the difference between the conceptual design and the technical design. We discuss the roles of the personnel who perform the design and describe two basic approaches to design: composition and decomposition. Then, we identify characteristics of good design, introduce several design strategies, and give examples of several system design techniques and tools. It is in this chapter that the reader learns about client-server architecture, reusable design components, human-computer interface design, design for secure and reliable systems (including error handling and fault tolerance), design patterns, formal design methods, and how to assess design trade-offs. After explaining how to evaluate and validate the quality of a design, and how to document the results, we turn to issues of program design.

Program design guidelines are explained, including top-down versus bottom-up, modularity and independence, and the difference between logical and physical design. We look at design for concurrency and for safety-critical systems, and we examine the design flaws that led to the Therac-25 malfunctions. We describe several design tools, and there is a thorough discussion of design quality and how to measure it. The chapter introduces design reuse, reviews, and inspections and explains the need to document design rationale. Finally, the chapter ends with examples of design for the information system and real-time examples.

Chapter 6: Concerning Objects

Chapter 6 takes a detour to consider the special properties of object-oriented development. We begin by describing use case scenarios, discussing how to capture objects and their characteristics from the natural language requirements. Next, we examine system design, to see how to generate the high-level information needed to find a solution. We then enrich the system design, adding nonfunctional requirements and details required in the program design. Employing UML and its constructs, we generate an object-oriented specification and design for a common example, the Royal Service Station.

Taking a careful look at object-oriented measurement, we apply some of the common object-oriented metrics to the service station example. We note how to use changes in the metrics to help us decide how to allocate resources and search for faults. Finally, we apply object-oriented concepts to our information systems and real-time examples.

Chapter 7: Writing the Programs

In this chapter, we address issues in implementing the design to produce high-quality code. We discuss standards and procedures and suggest some simple programming guidelines. Examples are provided in a variety of languages, including both object-oriented and procedural. There are thorough discussions of the need for program documentation and an error-handling strategy, and the chapter ends by applying some of the concepts to the two common examples.

Chapter 8: Testing the Programs

In this chapter, we explore several aspects of testing programs. We distinguish conventional testing approaches from the cleanroom method, and we look at how to test a variety of systems. We present definitions and categories of software problems, and we discuss how orthogonal defect classification can make data collection and analysis more effective. We then explain the difference between unit testing and integration testing. After introducing several automated test tools and techniques, we explain the need for a testing life cycle and how the tools can be integrated into it. Finally, the chapter applies these concepts to the two common examples.

Chapter 9: Testing the System

We begin with principles of system testing, including reuse of test suites and data, and the need for careful configuration management. Concepts introduced include function testing, performance testing, acceptance testing, and installation testing. We look at the special needs of testing object-oriented systems. Several test tools are described, and the roles of test team members are discussed. Next, we introduce the reader to software reliability modeling, and issues of reliability, maintainability, and availability are discussed. The reader learns how to use the results of testing to estimate the likely characteristics of the delivered product. The several types of test documentation are introduced, too, and the chapter ends by describing the test strategies of the two common examples.

Chapter 10: Delivering the System

This chapter discusses the need for training and documentation and presents several examples of training and documents that could accompany the information system and real-time examples.

Chapter 11: Maintaining the System

In this chapter, we address the results of system change. We explain how changes can occur during the system's life cycle, and how system design, code, test process, and documentation must accommodate them. Typical maintenance problems are discussed, as well as the need for careful configuration management. There is a thorough discussion of the use of measurement to predict likely changes, and to evaluate the effects of change. We look at reengineering and restructuring in the overall context of rejuvenating legacy systems. Finally, the two common examples are evaluated in terms of the likelihood of change.

Chapter 12: Evaluating Products, Processes, and Resources

Since many software engineering decisions involve the incorporation and integration of existing components, this chapter addresses ways to evaluate processes and products. It discusses the need for empirical evaluation and gives several examples to show how measurement can be used to establish a baseline for quality and productivity. We look at several quality models, how to evaluate systems for reusability, how to perform postmortems, and how to understand return on investment in information technology. These concepts are applied to the two common examples.

Chapter 13: Improving Predictions, Products, Processes, and Resources

This chapter builds on Chapter 11 by showing how prediction, product, process, and resource improvement can be accomplished. It contains several in-depth case studies to show how prediction models, inspection techniques, and other aspects of software engineering can be understood and improved using a variety of investigative techniques. This chapter ends with a set of guidelines for evaluating current situations and identifying opportunities for improvement.

Chapter 14: The Future of Software Engineering

In this final chapter, we look at several open issues in software engineering. We revisit Wasserman's concepts to see how well we are doing as a discipline. In addition, we examine several issues in technology transfer and decision-making to determine if we do a good job at moving important ideas from research to practice.

ACKNOWLEDGMENTS

Books are written as friends and family provide technical and emotional support. It is impossible to list here all those who helped to sustain me during the writing and revising, and I apologize in advance for any omissions. Carolyn Seaman (University of Maryland—Baltimore Campus) was a terrific reviewer, suggesting ways to clarify and simplify, and helping me to produce a tighter, more understandable text. She also prepared most of the solutions to the exercises and helped to set up the book's Web site. I am grateful for her friendship and assistance. Forrest Shull (Fraunhofer Center) has updated the solutions manual to reflect new exercises and material. Yiqing Liang has graciously scrubbed the Web site for bad links and has added new material. Carla Valle of the Federal University of Rio de Janeiro has also updated the Web site and will continue to add substantial new material.

I am particularly indebted to Guilherme Travassos (Federal University of Rio de Janeiro) for the use of material that we developed together at the University of Maryland—College Park, and that he enriched and expanded considerably for use in

subsequent classes. Likewise, I am grateful to Manny Lawrence, the manager of the real Royal Service Station, and to his bookkeeper Bea Lawrence not only for working with me and my students on the specification of the Royal system, but also for their affection and guidance in their other job: as my parents.

Helpful and thoughtful reviewers for both editions included Barbara Kitchenham (Keele University, UK), Bernard Woolfolk (Lucent Technologies), Ana Regina Cavalcanti da Rocha (Federal University of Rio de Janeiro), Frances Uku (University of California at Berkeley), Lee Scott Ehrhart (MITRE), Laurie Werth (University of Texas), Vickie Almstrum (University of Texas), Lionel Briand (Carleton University, Ottawa), Steve Thibaut (University of Florida), Lee Wittenberg (Kean College of New Jersey), and several anonymous reviewers provided by Prentice Hall. Discussions with Greg Hislop (Drexel University), John Favaro (Intecs Sistemi, Italy), Filippo Lanubile (Università di Bari, Italy), John d'Ambra (University of New South Wales, Australia), Chuck Howell (MITRE), and James and Suzanne Robertson (Atlantic Systems Guild, UK) led to many improvements and enhancements.

I owe a huge thank you to Forrest Shull (Fraunhofer Center—Maryland) and Roseanne Tesoriero (Catholic University of America), who developed the study guide for this book. And I salute John Gannon (University of Maryland—College Park) posthumously, for his convictions, encouragement, and the legacy he has left to all of us in software engineering.

Particular thanks go to Katharita Lamoza, Sondra Chavez, and Alan Apt, who made the first edition of the book's production interesting and relatively painless. Thanks too to James and Suzanne Robertson for the use of the Piccadilly example, and to Norman Fenton for the use of material from our software metrics book. Scott Disanno, Amy Todd, Alan Apt, Jake Warde and Toni Holm were wonderful in helping the second edition come to life.

Many thanks to the publishers of several of the figures and examples for granting permission to reproduce them here.

The material from *Complete Systems Analysis* (Robertson and Robertson 1994) is drawn from Dorset House Publishing, at www.dorsethouse.com. All rights reserved.

The article in exercise 1.1 is reproduced from the Washington Post with permission from the Associated Press. Figures 2.15 and 2.16 are reproduced from Barghouti et al. (1995) by permission of John Wiley and Sons Limited. Figures 12.14 and 12.15 are reproduced from Rout (1995) by permission of John Wiley and Sons Limited.

Figures and tables in Chapters 2, 3, 4, 5, 9, 11, and 12 that are noted with an IEEE copyright are reprinted with permission of the Institute of Electrical and Electronics Engineers. Table 2.1 and Figure 2.11 from Lai (1991) are reproduced with permission from the Software Productivity Consortium. Figures 8.16 and 8.17 from Graham (1996a) are reprinted with permission from Dorothy R. Graham. Figure 12.11 and Table 12.2 are adapted from Liebman (1994) with permission from the Centre for Science in the Public Interest, 1875 Connecticut Avenue NW, Washington DC. Tables 8.2, 8.3, 8.5, and 8.6 are reproduced with permission of the McGraw-Hill Companies. Figures and examples from Shaw and Garland (1996), Card and Glass (1990), Grady (1997), and Lee and Tepfenhart (1997) are reproduced with permission from Prentice Hall.

Tables 9.3, 9.4, 9.6, 9.7, 13.1, 13.2, 13.3, and 13.4, as well as Figures 1.15, 9.7, 9.8, 9.9, 9.14, 13.1, 13.2, 13.3, 13.4, 13.5,13.6, and 13.7 are reproduced or adapted from Fenton

and Pfleeger (1997) in whole or in part with permission from Norman Fenton. Figures 3.16, 5.19, and 5.20 are reproduced or adapted from Norman Fenton's course notes, with his kind permission

I am grateful to Sonya Smith, Deborah Cooper and Pat Ehlers for their understanding and patience. And, as always, Charles Pfleeger was a constant and much-appreciated source of support and encouragement.

Shari Lawrence Pfleeger
Washington, DC

About the Author

Shari Lawrence Pfleeger is president of Systems/Software, Inc., a consultancy specializing in software engineering and technology. In the past, she has been a member of the University of Maryland's Computer Science Department, founder and director of Howard University's Center for Research in Evaluating Software Technology (CREST), a visiting scientist at the City University (London) Centre for Software Reliability, principal scientist at MITRE Corporation's Software Engineering Center, and manager of the measurement program at the Contel Technology Center. Thus, she has experience both with the practical problems of software development and the theoretical underpinnings of software engineering and computer science. Pfleeger is well-known for her work in empirical studies of software engineering.

Dr. Pfleeger has been associate editor-in-chief of *IEEE Software*, where she edited the Quality Time column. She is currently associate editor of *IEEE Transactions on Software Engineering*. A member of IEEE, the IEEE Computer Society, and the Association for Computing Machinery, Pfleeger has twice been on the executive committee of the Technical Council on Software Engineering. She was the general chair of the Second International Symposium on Software Metrics (in London, England), the program co-chair of the Fourth International Symposium on software Metrics (in Albuquerque, New Mexico), and the co-chair of the Fifth Workshop on Empirical Studies of Software Engineering (Bethesda, Maryland).

Dr. Pfleeger is the author of many books and articles; she has been named repeatedly by the *Journal of Systems and Software* as one of the world's top software engineering researchers. Among her books are *Introduction to Discrete Structures* (with David Straight; Wiley, 1985), *Software Engineering: The Production of Quality Software* (Macmillan, 1987 and 1991), *Software Metrics: A Rigorous and Practical Approach* (with Norman Fenton; PWS Publishing, 1997) and *Applying Software Metrics* (with Paul Oman; IEEE Computer Society Press, 1997).

Contents