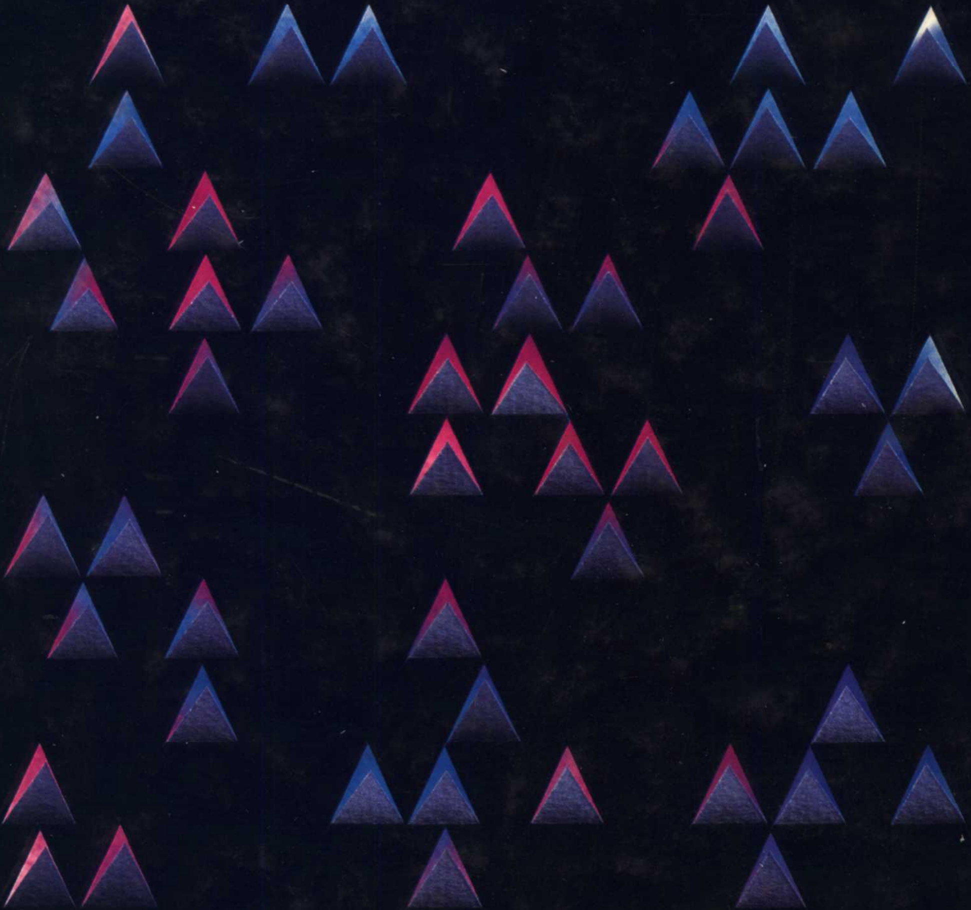
SOFTWARE ENGINEERING

WITH SYSTEMS ANALYSIS AND DESIGN



DONALD V. STEWARD

Software Engineering
with Systems Analysis and Design

Donald V. Steward

California State University, Sacramento

Brooks/Cole Publishing Company
Monterey, California

Brooks/Cole Publishing Company
A Division of Wadsworth, Inc.

Printed in the United States of America
10 9 8 7 6 5 4 3 2

Library of Congress Cataloging-in-Publication Data

Steward, Donald V.
Software engineering with systems analysis and design.

Includes index.
1. Computer software. 2. System analysis.
3. System design. I. Title.
QA76.754.S74 1987 005.1 86-26808
ISBN 0-534-07506-1

Sponsoring Editor: *Cynthia C. Stormer*
Project Development Editors: *John Bergez and Liz J. Currie*
Marketing Representative: *Jill McGillen*
Editorial Associate: *Corinne Kibbe*
Production Editor: *Ellen Brownstein*
Manuscript Editor: *Jonas Weisel*
Permissions Editor: *Carline Haga*
Interior and Cover Design: *Sharon L. Kinghan*
Cover Image: *Lisa Thompson*
Cover Photo: *Lee Hocker*
Art Coordinator: *Sue C. Howard*
Interior Illustration: *Carl Brown*
Typesetting: *Graphic Typesetting Service*
Printing/Binding: *The Maple-Vail Book Manufacturing Group*

Preface

Software Engineering with Systems Analysis and Design develops the natural integration between software engineering and systems analysis and design. The field of software engineering has become increasingly complex. A bewildering variety of methodologies are currently in use and new methodologies are continually being developed. This book presents the classical approaches to software engineering—data flow diagrams, structure charts, Warnier-Orr Diagrams—and explores many of the newer techniques, such as Trees and higher order software. Furthermore, it presents an integrated approach that serves to illustrate basic principles while solving many of the problems intrinsic to the classical methods. It can be used alone to develop systems or in conjunction with readily-available computer aids.

Software Engineering with Systems Analysis and Design is written with both the student and the practitioner in mind. The book is appropriate for classes in software engineering or systems analysis, as well as in the spectrum of computer science and management information systems classes. The software engineering practitioner will find the text a valuable tool for improving his or her current methods.

The impetus for this book comes from two sources, one in industry and one in academic life. My twenty years of experience in industry (General Electric and Burroughs Research Laboratories) and my observations of senior projects at California State University, Sacramento, convinced me that students are frequently ill-prepared for the real-world problem solving they encounter when they graduate. This realization was the main motivation for my developing the course in software engineering that is now required of all computer science majors at CSUS. It has also guided my approach in the text.

Organization and Features

Software Engineering with Systems Analysis and Design is divided into five parts:

Part 1 introduces the principles used throughout the rest of the book;
Part 2 develops techniques for describing the product and working with its requirements and design;
Part 3 describes how the project is managed;
Part 4 examines information-systems technologies; and
Part 5 applies the techniques developed in the previous parts to each phase of the project.

Part 1 should be read first. Parts 2, 3, and 4 may be read in any order. Part 5 builds on the principles developed in Parts 1 through 4 and should be read in conjunction with the case study in the appendix.

The integrated approach presented here is supported by such unifying themes as the tree, which is used as a hierarchical representation of systems or programs; matching requirements and means; and the role of expectations in project management. The reader is introduced to needs and feasibility analyses, Two-Entity Data Flow Diagrams, Trees, and structured design, and then learns how these techniques can be extended and integrated throughout the entire software development life cycle. This approach shows the reader how to simplify transitions between phases in the life cycle, improve traceability, provide progress reports to managers, and estimate, plan, schedule, and control projects. Principles and general methods are presented first, followed by step-by-step applications of these methods through the phases of the project. The need for up-front planning is emphasized and methods of representation and progress measurement are provided. Also included in this book are chapters on problem definition, cost, and feasibility, topics that are often omitted or underplayed in other software engineering books. A case study at the end of the book provides a practical model for the concepts presented in the text.

To the Software Engineering Practitioner

There are currently many software engineering tools in use (data flow diagrams, structure charts, Warnier-Orr Diagrams, entity relation diagrams, state transition diagrams), each with its own application to a specific aspect of the system. If we use structure charts, we do structured design but, if we use Warnier-Orr Diagrams, we probably don't. We may use data flow diagrams and structured English for requirements, structure charts for design, and pseudocode and a high-level programming language for implementation. With all these methods in use on the same project and different methods in use during each phase, it is very difficult to provide consistency of representation, to ensure that ideas and work are not lost when converting from one method to another, and to track the progress of the entire project throughout its life cycle. It is also difficult to ensure that the requirements developed at the beginning of the project have been preserved in the end product, harder still to collect data that will aid in estimating future projects, and very difficult to use computer aids.

This book provides a means of using both the conventional and newer, more powerful methods of software engineering more effectively. It presents an integrated approach, based on the fundamental concepts of the classical approach, that can greatly increase productivity. The integrated approach provides the following advantages:

1. a consistent measure of progress throughout the entire cycle, from requirements to code generation
2. a means of tracking requirements throughout the cycle, ensuring that no requirements are lost from one phase to the next
3. a method of representation (tree editors) that does not have to be modified from one phase to another
4. a means of calculating metrics using computers
5. a method of collecting data automatically as the system develops
6. a method that can be implemented with computer aids.

In short, the integrated approach presented here will help the software engineering practitioner incorporate the methods he or she already uses with those that are newly developed and those that undoubtedly will be developed as the field of software engineering continues to grow.

Acknowledgments

There are many people to whom I owe a debt of gratitude for their valuable suggestions and contributions along the way. On our faculty at CSUS, I wish to thank: Richard Hill, Robert Buckley, Martin Meyers, Ronald Ernst, Nancy Miner, and Joan Al-Kazily. At Brooks/Cole, Ellen Brownstein, Mike Needham, Cindy Stormer, and Neil Oatley have lent that vital assistance needed to turn these ideas into a book.

Thanks are also due to the reviewers of this book: Meledath Damodaran, University of Bridgeport; Richard Hill, California State University, Sacramento; Gregory Jones, Utah State University, Logan; William Junk, University of Idaho, Moscow; Richard LeBlanc, Georgia Institute of Technology, Atlanta; Christopher Pidgeon, University of California, Irvine; Vaclav Rajlich, Wayne State University, Detroit; K.V. K. Reddy, McNeese State University, Lake Charles, Louisiana; Paul Ross, Millersville State College, Millersville, Pennsylvania; and Stephen Thebaut, University of Florida, Gainesville.

Donald V. Steward

Contents

1 PART

Introducing Software Engineering

1

What Is Software Engineering?

- The field of software engineering is concerned with all of the activities involved in the solution of problems through the development of computer systems.
- Today careful planning and coordination are necessary to produce software because the problems they solve and the programs themselves are complex, the development must be done by teams, software is expensive, and mistakes are costly and difficult to correct.
- Software engineering is the management of expectations, computer technology, human skills, time, and money in order to create a software product that meets the expectations of the client with a satisfactory return to the producer.
- Software projects should be planned at the start rather than ad hoc as they proceed. They require communication between the two cultures of producer and client, require a commitment of resources before many important questions can be resolved, and culminate in an entirely new product.
- Software engineering continues to evolve, finding better methods for producing today's increasingly complex software.