



SOFTWARE SYSTEMS ENGINEERING

SOFTWARE SYSTEMS ENGINEERING

**ANDREW P. SAGE
JAMES D. PALMER**

School of Information Technology and Engineering
George Mason University
Fairfax, Virginia 22030



WILEY

A Wiley-Interscience Publication

JOHN WILEY & SONS

New York / Chichester / Brisbane / Toronto / Singapore

Copyright © 1990 by John Wiley & Sons, Inc

All rights reserved. Published simultaneously in Canada.

Reproduction or translation of any part of this work beyond that permitted by Section 107 or 108 of the 1976 United States Copyright Act without the permission of the copyright owner is unlawful. Requests for permission or further information should be addressed to the Permissions Department, John Wiley & Sons, Inc

Library of Congress Cataloging-in-Publication Data:

Sage, Andrew P.

Software systems engineering / Andrew P Sage and James D Palmer
p. cm.— (Wiley series in system engineering)

"A Wiley-Interscience publication."

Includes bibliographical references.

1. Software engineering. 2. Systems engineering. I. Palmer, James D. II. Title. III. Series.

QA76.758.S24 1990

005.1—dc20

ISBN 0-471-61758-X

89-37350

CIP

Preface

This book is written for a first course in software engineering, particularly one that emphasizes a *systems engineering and systems management for software productivity* perspective. The book is reasonably self-contained. It is not a book specifically addressing programmer productivity concerns, although these are, in part, addressed in the book for the sake of completeness. It is focused primarily on a systems approach to lifecycle management of software production. The book discusses all the lifecycle phases of systems development. There is considerable discussion of such industrially relevant material as software quality, software reliability, development environments, integration, maintenance, management, and cost analysis.

We begin our efforts with an indication of why we necessarily associate the word "engineering" with software, as contrasted with the word "science." Then we indicate why the production of trustworthy software can be best accomplished through use of the approaches of "systems engineering." Following this, we present a brief discourse concerning various topics of interest and importance in software systems engineering. Throughout our presentations in this book, we are especially concerned with ways in which software productivity may be improved through use of the methods, design methodologies, and management approaches of systems engineering. The framework and outline that we develop in Chapter 1 provides a basis for the design of trustworthy software as well as a logical organization for this text.

Software engineering generally has given attention to the development of micro-level tools to address the growing needs to increase software productivity. The major thrust of this book is to outline a systems engineering approach to increasing software productivity that encompasses these micro-level tools. We also discuss the need for such macro productivity tools as

rapid prototyping, reusability constructs, knowledge-based systems for software development; and an interactive support system environment to aid in software development. Also, we are very concerned with systems management of all aspects of the software production process.

Thus, we are concerned with *software engineering in the small*, or program and programmer productivity; and *software engineering in the large*, or software systems engineering. We are concerned, in part, with the "tools" for software engineering that enable micro-enhancement and macro-enhancement of software quality. We are also concerned with an overarching *systems design methodology* that will enable selection of an appropriate set of software engineering tools. We are, in addition, interested in software engineering as a process, and thus we devote a considerable portion of our effort to the *systems management* of software.

Our effort in Chapter 2 begins with a discussion of lifecycle approaches to the systems engineering of software. We outline several variants that lead to phased development of software systems. Then we address the very important question of identification of the user or client requirements that a software system must satisfy. User requirements specification and software requirements specification will be the first phase of effort in our development of software, and we devote Chapter 3 to this topic. Following the initial determination of user requirements, these user or client requirements are transformed into computer software oriented requirements.

Micro-enhancement tools are important for productivity enhancement throughout the software development lifecycle. So, we next study micro-enhancement approaches for the various phases of a typical lifecycle for software development. Chapters 4 and 5 present a number of these approaches. We elaborate on the most widely used micro-enhancement approaches and, through a typical software acquisition lifecycle, establish the need for a taxonomy of methods in order to make productivity tools generally available and subject to greater use.

Chapters 6 and 7 discuss the latter portions of the software lifecycle. In particular, efforts that are concerned with reliability, maintainability, and quality assurance are studied in Chapter 6. Chapter 7 presents an overview of system integration, operational implementation, and software development environments. This is followed by a discussion (in Chapter 8) of macro-enhancement approaches to software productivity including prototyping, software reusability, and the use of expert system techniques to enhance the production of software.

The next two chapters of the book treat management, maintenance, and standards procedures for software productivity. Chapter 9 is concerned with systems management-related topics. Chapter 10 is concerned mainly with the development of models estimating cost and benefit for software development. The final chapter of the book presents a very carefully selected and annotated bibliography of pertinent references.

Thus, our book on software systems engineering provides an introductory,

but reasonably complete, treatment of all aspects of the development lifecycle for software production. It is, therefore, suited for an introductory course in software engineering that emphasizes systems management of software production. It is also very appropriate for those who manage these efforts and who wish to have an overview of the programmer productivity approaches that are needed for software development.

Most introductory books on software engineering concentrate on programmer productivity. While we do not ignore this, we focus more on the macro-level and systems management approaches that many believe offer much more promise for productivity enhancement than do approaches that rely only or primarily on enhancement of the efforts of individual programmers.

Many studies indicate that a very large percentage of system costs are expended on software. Usually, it is necessary to *maintain* new systems such that they are able to be continually responsive to changing user and environmental needs. In many systems, the larger part of maintenance monies are spent for software maintenance. A large number of difficulties both cause and emanate from the current lack of trustworthy and effective software that is produced at a reasonable price. These include: inconsistent, incomplete, and otherwise imperfect system requirements specifications; system requirements that do not provide for change as user needs evolve over time, and poorly defined management structures for product design and delivery. These lead to delivered products that are difficult to use, that do not solve the intended problem, that operate in an unreliable fashion, that are unmaintainable, and that—as a result—are not used. *And, the problem appears to be getting worse.*

These same studies generally reveal that the major problems associated with the production of trustworthy software are more concerned with the *organization and management of complexity* than with direct technological concerns that affect individual programmer productivity.

Since the critical areas associated with software productivity improvement are fundamentally systems engineering areas, we intentionally use the term “software systems engineering” to describe the general area of coverage for this book.

Individual chapters are devoted to the major efforts that need to be accomplished as part of the lifecycle of software development. A number of the major design methods are described in a stepwise, easy-to-understand fashion. References to the contemporary literature that provides more detailed discussions is a feature of the book. Many current-generation *computer-aided systems engineering* (CASE) tools are discussed throughout the book.

This is a textbook. It contains about 30% more material than can be covered in a rapidly paced three-semester-hour introductory graduate-level course. Through the use of a term paper and several projects, especially of a laboratory development nature; during the course, it provides sufficient material for a full-year course.

We have generally followed the sequenced pattern in the text from Chap-

ters 1 through 10 in our own teaching efforts. For use in software engineering curricula where there are a number of succeeding courses on specialized topics, it may be desirable to omit coverage of some of the specialized topics that are discussed later.

We have had some experience in using this material for industrial short courses where participants were already experienced programmers who were generally familiar with the programming productivity content of Chapters 4 and 5. Omitting these two chapters led to no loss in continuity, especially because of the detailed overview of the book that is presented in Chapter 1.

The book is intended for use in an introductory graduate-level course in *software systems engineering*. The course is generally taken by many master's-level students in systems engineering who do not intend to undertake detailed study in software but who wish an overview of developments in this area. The courses on which the book is based has also been taken by computer science students who intend to specialize in one of the programmer productivity areas. It has also been used for short courses offered for professional development.

Although there are no officially listed prerequisites for the course for which this text is written, it is by no means an introductory course. The students taking it are expected to be familiar either with computer programming and software design, or systems engineering, and preferably with both areas.

ANDREW P. SAGE
JAMES D. PALMER

Fairfax, Virginia
December 11, 1989

Acknowledgments

Many people contribute to a book other than the authors. Almost all of this book has been classroom tested at least twice. Many students have provided helpful suggestions. Ann Fields and Charlie Smith were very helpful with respect to their detailed reading of the final version of the text. Kathleen Johnson has assisted relative to some typing chores. George Telecki was always very supportive and understanding despite being told numerous times that the book was *in the mail*. The authors wives, LaVerne and Margret, were extraordinarily understanding of many late night and weekend sessions that the authors needed to spend with a word processor. We acknowledge the assistance and support of all these special people.

A.P.S.
J.D.P.

Contents

Preface	xiii
Acknowledgments	xvii
1. An Introduction to Software Systems Engineering	1
1.1 The Emergence of Systems Engineering	5
1.2 From Computer to Information and Knowledge Technologies	6
1.3 The Need for Software Systems Engineering	7
1.4 A Methodology for Software Systems Engineering Design	11
1.4.1 Design	11
1.4.2 Software Systems Design Methodology	12
1.4.3 The Nature of Design	25
1.4.4 Information Requirements for Software Production	29
1.4.5 Objectives for Software Systems Engineering	31
1.5 Methods for Enhancement of Software Productivity	32
1.6 Micro-Enhancement Aids	35
1.7 Macro-Enhancement Approaches	36
1.8 Perspectives on Software Systems Engineering	42
1.9 Problems	46

2. Models for the Software Development Lifecycle	48
2.1 Introduction	48
2.2 The "Waterfall" Software Development Lifecycle Model	50
2.3 Iterative Waterfall Software Development Lifecycle Models	57
2.3.1 The Structured Project Lifecycle	57
2.3.2 Software Development Lifecycle Models with Feedback	62
2.4 The Spiral Model	67
2.5 The Evolutionary-Prototyping Model	73
2.6 The Operational-Transformational Model	76
2.7 The Knowledge-Based Lifecycle Model	77
2.8 Summary	78
2.9 Problems	81
3. System Requirements Identification and Software Requirements Specification	84
3.1 Introduction	84
3.2 Definition of the User	88
3.3 System Requirements Specifications	90
3.3.1 An Activities Matrix for Preparation of System-Level Requirements Specifications	93
3.3.2 Knowledge Acquisition Approaches	95
3.3.3 System Description	102
3.3.4 Hardware Description	102
3.3.5 System Structure Model	103
3.3.6 Functional Requirements	103
3.3.7 Nonfunctional Requirements	104
3.3.8 Database Requirements	104
3.3.9 Installation and Maintenance	104
3.3.10 Interfaces and Partitioning	105
3.4 Software Requirements Specification	105
3.4.1 Software Requirements Specification Methods	109
3.4.2 Methods and Tools for Software Requirements Specifications Analysis	114
3.4.3 Specific Methods and Techniques	115
3.4.4 Impact of Errors in System and Software Requirements Specifications	124
3.5 System and Software Requirements Specification Documents	125

3.6	Activities and Products from Software Requirements Specifications	128
3.7	Summary and Conclusions	128
3.8	Problems	129
4.	Conceptual and Logical Design and Detailed Design	131
4.1	Introduction	131
4.2	Structured Logical Design for Functional Decomposition	134
4.2.1	Structured Design Tools	134
4.2.2	Flowcharts	134
4.2.3	Data Flow Diagrams	137
4.2.4	Data Structure-Oriented Methods	145
4.2.6	Entity-Relationship Diagrams	146
4.2.7	Data Structured Systems Development	149
4.2.8	Jackson System Development	150
4.3	Detailed Design and Coding	152
4.3.1	Data Flow Design Approach	161
4.3.2	Data Structure Design Approach	163
4.4	Object-Oriented Design Approaches	164
4.5	Automated Software Development Tools	175
4.6	Summary	177
4.7	Problems	177
5.	Programming Languages, and Testing	180
5.1	Introduction	180
5.2	Coding Practices	183
5.2.1	Introduction	183
5.2.2	Data Structure	184
5.2.3	Structured Programming Approaches	184
5.2.4	Module Coupling and Cohesion	187
5.3	Programming Languages	190
5.3.1	Control Structures	192
5.3.2	Exception Handling	193
5.3.3	Information Hiding	193
5.3.4	Data Types	194
5.3.5	Data Typing	195
5.3.6	Declarations	196
5.3.7	Initialization and Constants	196
5.3.8	Modules or Subprograms	196

5.3.9	Pseudo-Programming Languages	197
5.3.10	Fourth-Generation Language Automated Program Generators	199
5.4	Programming Environments	201
5.5	Software Testing	204
5.5.1	Function and Module Testing	209
5.5.2	System and Subsystem Testing	213
5.5.3	Program Verification and Validation	214
5.5.4	Configuration Management	218
5.6	Summary	219
5.7	Problems	219
6.	Software Reliability, Maintainability, and Quality Assurance	221
6.1	Introduction	221
6.2	A Taxonomy of Attributes and Associated Metrics for Software Quality Assurance	228
6.2.1	Definitions and Acronyms for Quality Assurance	229
6.2.2	Software Quality Assurance Attributes, Errors, and Plans	233
6.3	A Multiattribute Approach to Quality Assurance Evaluation Measurements	241
6.3.1	The MAUT Framework for Decision Analysis	242
6.3.2	Independence Concepts	243
6.3.3	Additive Representations	245
6.3.4	Dominance and the Efficient Frontier	247
6.3.5	Decision Analysis Methods	247
6.3.6	Assessment Methods	249
6.3.7	An Illustrative Software Quality Assurance Evaluation Using MAUT	250
6.4	Methods for Software Quality Assurance Measurement and Test	255
6.4.1	Structural Testing	257
6.4.2	Functional Testing	262
6.4.3	Integration Testing	263
6.4.4	System-Level Testing	265
6.5	Software Maintenance	267
6.6	Summary	274
6.7	Problems	274

7. Operational Implementation, System Integration, and Environments for Software Systems Engineering	279
7.1 Implementation and Integration	279
7.2 Management of Software System Implementation and Integration	286
7.3 Integration Needs in Software System Implementation	289
7.4 Software Engineering Environments	294
7.5 Environments for System Design	299
7.6 Requirements for an Integrated Software Systems Design Environment	304
7.7 Some Hypotheses Concerning a Software Environment Support Facility	311
7.8 CASE Tools for Software Systems Engineering Environments	315
7.8.1 The Ada Language Environment—APSE	318
7.8.2 Boeing Automated Software Engineering	319
7.8.3 Software through Pictures	320
7.8.4 NASTEC CASE Lifecycle Manager Toolkit	320
7.8.5 Analyst Designer Toolkit	321
7.8.6 DesignAid	322
7.8.7 Excelerator	322
7.8.8 PCSA	323
7.8.9 Current CASE Packages	324
7.9 Summary	325
7.10 Problems	326
8. Prototyping, Reusability, and Expert Systems	329
8.1 Introduction	329
8.2 Prototyping and Prototyping Environments	332
8.2.1 A Taxonomy of Prototyping Approaches	335
8.2.2 Explicit Phases Involved in Prototyping	340
8.2.3 Observations Concerning Prototyping	343
8.3 Reusability and Reuse	354
8.3.1 Approaches to Softproduct Reusability	362
8.3.2 Approaches to Softprocess Reusability	371
8.3.3 Characteristics of a Software Development Environment for Reusability	372
8.3.4 Reusability and Software Productivity	375

8.4	Knowledge-Based Systems for Software Productivity	377
8.4.1	Types of Knowledge Support Systems	381
8.4.2	Generic Software Development Support Systems	383
8.4.3	Software Development Support Systems	385
8.5	Summary	391
8.6	Problems	392
9.	Management of the Software Systems Engineering Process	395
9.1	Introduction	395
9.2	Organization Management—Philosophical Perspectives	396
9.3	Organizational Management—Pragmatic Perspectives	404
9.4	Organizational Management—Crisis Perspectives	405
9.5	Implications for Organizational Design <i>and</i> Software Design	407
9.6	Software Systems Management Planning	417
9.7	Software Engineering Project Management	419
9.8	The Role of Standards in Software Systems Engineering	432
9.9	Summary	434
9.10	Problems	434
10.	Software Cost and Value Models	436
10.1	Introduction to Software Cost and Value	436
10.2	Model Construction	443
10.3	Wholistic Expert-Judgment-Based Software Cost Models	451
10.4	Heuristics for Software Cost Estimation	459
10.4.1	The Bailey-Basili Model	461
10.4.2	The Walston-Felix Model	463
10.4.3	The Putnam Resource Allocation Model	466
10.4.4	The Constructive Cost Model (COCOMO)	473
10.4.5	Other Models	482
10.5	Validation of Software Effort and Cost Models	484
10.6	Summary	489
10.7	Problems	491
	Bibliography for Software Systems Engineering	495
	Index	513

Chapter 1

An Introduction to Software Systems Engineering

In this chapter we provide an overview of our efforts to follow in software systems engineering. We begin with an indication of why we necessarily associate the word “engineering” with software, as contrasted with the word science. Then we indicate why the production of trustworthy software can be best accomplished through use of the approaches of “systems engineering.” Following this, we present a brief discourse concerning various topics of interest and importance in software systems engineering. Throughout our presentations in this book, we are especially concerned with ways in which software productivity may be improved through use of the methods, design methodologies, and management approaches of systems engineering. The framework and outline that we develop in this chapter provide a basis for the design of trustworthy software as well as a logical organization for this text.

There are a number of reasons why software productivity improvement studies and methods are of much importance at this time. The primary one is that the annual expenditures for software development are very large and the productivity not very high.

Software engineering generally has given attention to the development of microlevel tools to address the growing needs to increase software productivity. The major thrust of this book is to outline a systems engineering approach to increasing software productivity that encompasses these microlevel tools. We also discuss the need for such macro-productivity tools as rapid prototyping, reusability constructs, and an interactive support system environment that involves the systems engineer, the user, and the software engineer. Also, we are very concerned with systems management of all aspects of the software production process.

Thus, we are concerned with software engineering in the small, or program

and programmer productivity; and software engineering in the large, or software systems engineering.

We are concerned, in part, with the “tools for software engineering” that enable micro-enhancement and macro-enhancement of software quality. We are also concerned with an overarching “systems design methodology” that will enable selection of an appropriate set of software engineering tools. We are, in addition, interested in software engineering as a process, and thus we devote a considerable portion of our effort to the “systems management” of software.

Our goal is to utilize this just described three-layer approach [Sage, 1982] to software systems engineering, as shown in Figure 1.1, in order to integrate together the technology for software production within an appropriate design approach that is matched to the organization and environment in which the software must function. From this perspective, software production becomes a systems engineering activity. It, like systems engineering, is then a management technology in that it involves technology, which is the organization and delivery science for the betterment of humankind, and management, which is the art and science of enabling an organization to function in an environment in such a way as to achieve objectives. Figure 1.2a illustrates this view of software systems engineering. Through use of this three-level approach to software engineering, we hope to provide and describe symbiotic relationships between individual members of a programming team to enable successful completion of projects that enable better performance of organizations in operational environments. Figure 1.2b indicates this symbiotic embedding with respect to people, and Figure 1.2c illustrates the embedding of software ingredients. Successful efforts in software systems engineering must be concerned with productivity across each of these entities; we will be much concerned with a systems management approach to software development in our efforts to follow

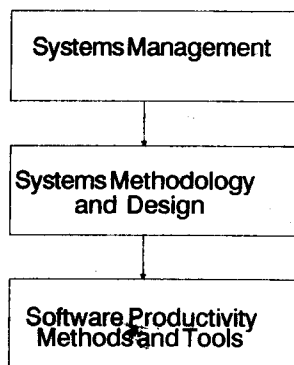


FIGURE 1.1 The three levels of software systems engineering

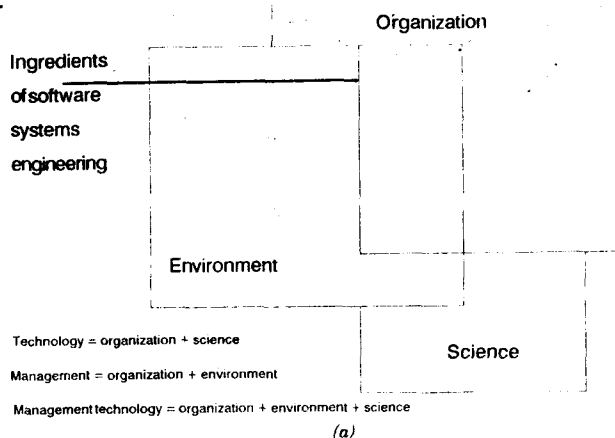


FIGURE 1.2a Software systems engineering as a management technology

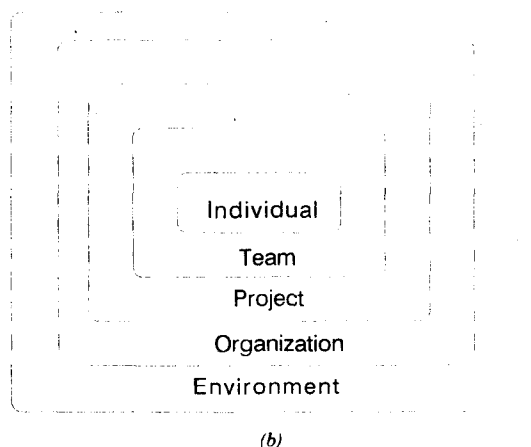


FIGURE 1.2b Interactions addressed through software systems engineering

Our effort in Chapter 2 begins with a discussion of lifecycle approaches to the systems engineering of software. We outline several variants that lead to phased development of software systems. Then we address the very important question of identification of the user or client requirements that a software system must satisfy. Requirements specification identification will be the first phase of effort in our development of software, and we devote Chapter 3 to this topic. Following the initial determination of user requirements, these user