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

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 89-37350

ISBN 0-471-61758-X 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 Chapters 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

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] t 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 1.2*a* 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 1.2*b* indicates this symbiotic embedding with respect to people, and Figure 1.2*c* 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

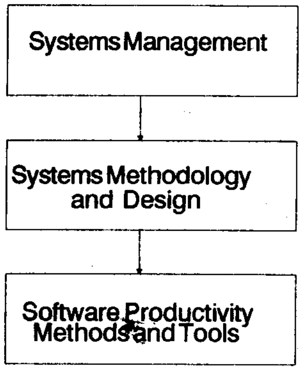
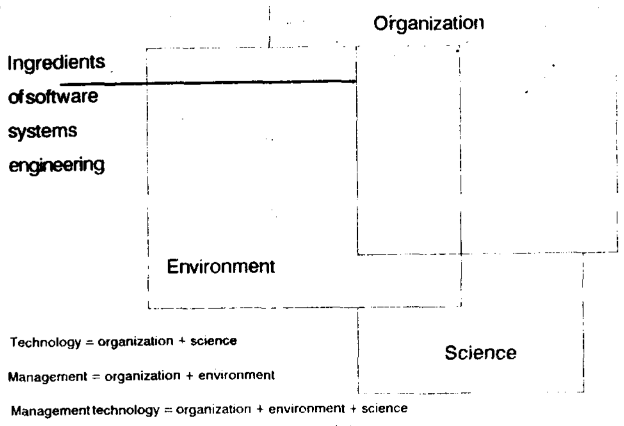
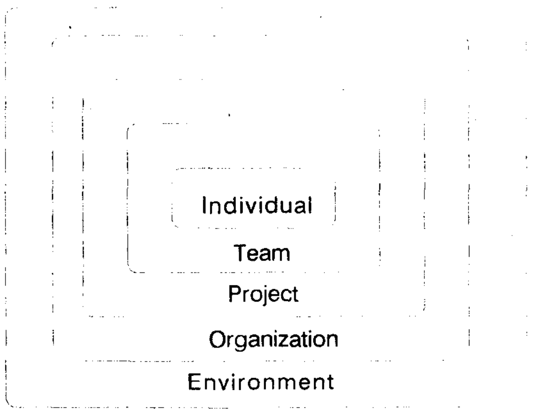

Systems Management
Systems Methodology and Design
Software Productivity Methods and Tools

FIGURE 1.1 The three levels of software systems engineering


Organization
Ingredients of software systems engineering
Environment
Science
Technology = organization + science
Management = organization + environment
Management technology = organization + environment + science

(a)

FIGURE 1.2a Software systems engineering as a management technology



(b)

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