SOFTWARE ENGINEERING: A PRACTITIONER'S APPROACH

ROGER S. PRESSMAN

SOFTWARE ENGINEERING
A Practitioner's Approach

Roger S. Pressman, Ph.D.

Adjunct Professor of Computer Engineering
University of Bridgeport
and
President, R. S. Pressman & Associates, Inc.

McGraw-Hill Book Company

New York St. Louis San Francisco Auckland Bogotá Hamburg
Johannesburg London Madrid Mexico Montreal New Delhi
Panama Paris São Paulo Singapore Sydney Tokyo Toronto

This book was set in Times Roman by World Composition Services, Inc.
The editors were Charles E. Stewart and Gail Gavert;
the production supervisor was Dominick Petrellese.
New drawings were done by Fine Line Illustrations, Inc.
R. R. Donnelley & Sons Company was printer and binder.

SOFTWARE ENGINEERING
A Practitioner's Approach

1234567890 DODO 898765432

ISBN 0-07-050781-3

Library of Congress Cataloging in Publication Data

Pressman, Roger S.
Software engineering.

(McGraw-Hill software engineering and technology series)
Includes bibliographies and index.
1. Electronic digital computers—Programming.
I. Title. II. Series.
QA76.6.P73 001.64'2 81-20718
ISBN 0-07-050781-3 AACR2

PREFACE

In the brief history of the electronic digital computer, the 1950s and 1960s were decades of hardware. The 1970s were a period of transition and a time of recognition of software. The decade of software is now upon us. In fact, advances in computing may become limited by our inability to produce quality software that can tap the enormous capacity of 1980-era processors.

During the past decade we have grown to recognize circumstances that are collectively called the *software crisis.* Software costs escalated dramatically, becoming the largest dollar item in many computer-based systems. Schedules and completion dates were set but rarely kept. As software systems grew larger, quality became suspect. Individuals responsible for software development projects had limited historical data to use as guides and less control over the course of a project.

A set of techniques, collectively called *software engineering,* has evolved as a response to the software crisis. These techniques deal with software as an engineered product that requires planning, analysis, design, implementation, testing, and maintenance. The goal of this text is to provide a concise presentation of each step in the software engineering process.

The contents of this book closely parallel the software life cycle. Early chapters present the planning phase, emphasizing system definition (computer systems engineering), software planning, and software requirements analysis. Specific techniques for software costs and schedule estimation should be of particular interest to project managers as well as to technical practitioners and students.

In subsequent chapters emphasis shifts to the software development phase. The fundamental principles of software design are introduced. In addition, descriptions of two important classes of software design methodology are presented in detail. A variety of software tools are discussed. Comparisons among techniques and among tools are provided to assist the practitioner and student alike. Coding style is also stressed in the context of the software engineering process.

The concluding chapters deal with software testing techniques, reliability, and software maintenance. Software engineering steps associated with testing are described and specific techniques for software testing are presented. The current status of software reliability prediction is discussed and an overview of reliability models and program correctness approaches is presented. The concluding chapter considers both management and technical aspects of software maintenance.

This book is an outgrowth of a senior-level/first-year-graduate course in software engineering offered at the University of Bridgeport. The course and this text cover both management and technical aspects of the software development process. The chapters of the text correspond roughly to major lecture topics. In fact, the text is derived in part from edited versions of transcribed notes of these lectures. Writing style is therefore purposely casual and figures are derived from viewgraphs used during the course.

Software Engineering: A Practitioner's Approach may be used in a number of ways for various audiences. The text can serve as a concise guide to software engineering for the practicing manager, analyst, or programmer. It can also serve as the basic text for an upper-level undergraduate or graduate course in software engineering. Lastly, the text can be used as a supplementary guide for software development early in computer science or computer engineering undergraduate curricula.

The software engineering literature has expanded rapidly during the past decade. I gratefully acknowledge the many authors who have helped this new discipline evolve. Their work has had an important influence on this book and my method of presentation. I also wish to acknowledge Pat Duran, Leo Lambert, Kyu Lee, John Musa, Claude Walston, Anthony Wasserman, Marvin Zelkowitz, and Nicholas Zvegintzov, the reviewers of this book, and Peter Freeman, the series editor. Their thoughtful insights and suggestions have been invaluable during the final stages of preparation. Special thanks go to Leo Lambert and his colleagues from the Computer Management Operation, General Electric Company, who have allowed me to tap their broad collective experience during my long association with them. In addition, to the students at the University of Bridgeport and the hundreds of software professionals and their managers who have attended short courses that I have taught, my thanks for the arguments, the ideas, and the challenges that are essential in a field such as ours.

Finally, to B rbara, Mathew, and Michael, my love and thanks for tolerating the genesis of book number two.

Roger S. Pressman

CONTENTS

CHAPTER

ONE

COMPUTER SYSTEM ENGINEERING

Four hundred and fifty years ago Machiavelli said:

> There is nothing more difficult to take in hand, more perilous to conduct or more uncertain in its success, than to take the lead in the introduction of a new order of things. . . .

In the decade of the 1980s computer-based systems will introduce a new order. Although technology has made great strides since Machiavelli spoke, his words continue to ring true.

Software engineering—the topic to which this book is dedicated—and hardware engineering are activities within the broader category that we shall call *computer system engineering.* Each of these disciplines represents an attempt to bring order to the development of computer-based systems.

Engineering techniques for computer hardware developed from electronic design and have reached a state of relative maturity in little more than three decades. Hardware design techniques are well established, manufacturing methods are continually improved, and reliability is a realistic expectation, rather than a modest hope.

Unfortunately, computer software still suffers from the Machiavellian description stated above. In computer-based systems software has replaced hardware as the system element that is most difficult to plan, least likely to succeed (on time and within cost), and most dangerous to manage. Yet the demand for software continues unabated as computer-based systems grow in number, complexity, and application.

Engineering techniques for ccmputer software have only recently gained widespread acceptance. During the 1950s and 1960s computer programming was viewed as an art. No engineering preredent existed, and no engineering approach was applied.

Times are changing!

1.1 COMPUTER SYSTEM EVOLUTION

The context in which software has been developed is closely coupled to three decades of computer system evolution. Better hardware performance, smaller size, and lower cost have precipitated more sophisticated systems. In three and one-half machine generations we've moved from vacuum tube processors to microelectronic devices. In recent popular books on "the computer revolution," Osborne [1] has characterized the 1980s as a "new industrial revolution" and Toffler [2] calls the advent of microelectronics part of "the third wave of change" in human history.

Figure 1.1 depicts the evolution of computer-based systems in terms of application area, rather than hardware characteristics. During the early years of computer system development, hardware underwent continual change while software was viewed by many as an afterthought. Computer programming was a "seat-of-the-pants" art for which few systematic methods existed. Software development was virtually unmanaged—until schedules slipped or costs began to escalate. During this period a batch orientation was used for most systems. Notable exceptions were interactive systems such as the early American Airlines reservation system and real-time defense-oriented systems such as SAGE. For the most part, however, hardware was dedicated to the execution of a single program that in turn was dedicated to a specific application.

During the early years general-purpose hardware became commonplace. Software, on the other hand, was custom designed for each application and had a relatively limited distribution.

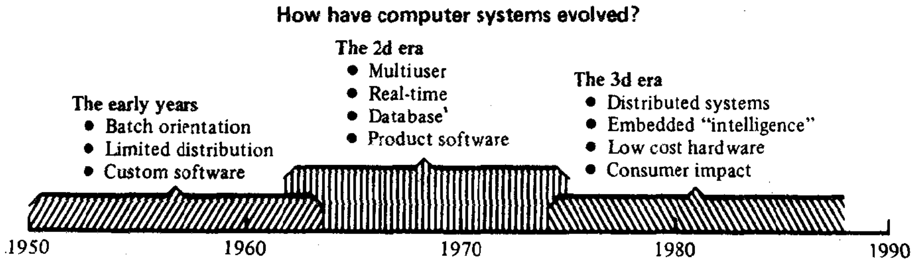

Figure 1.1

Product software (i.e., programs developed to be sold to one or more customers) was in its infancy. Most software was developed and ultimately used by the same person or organization. You wrote it, you got it running, and if it failed, you fixed it. Because job mobility was low, managers could rest assured that you'd be there when bugs were encountered. Because of this personalized software environment, design was an implicit process performed in one's head and documentation was often nonexistent.

During the early years we learned much about the implementation of computer-based systems, but relatively little about computer system engineering. In fairness, however, we must acknowledge the many outstanding computer-based systems that were developed during this era. Some of these systems remain in use today and provide landmark achievements that continue to justify admiration.

The second era of computer system evolution (Figure 1.1) spanned the decade from the mid-1960s to the mid-1970s. Multiprogramming, multiuser systems introduced new concepts of human-machine interaction. Interactive techniques opened a new world of applications and new levels of hardware and software sophistication. Real-time systems could collect, analyze, and transform data from multiple sources, thereby controlling processes and producing output in milliseconds rather than minutes. Advances in on-line secondary memory devices lead to the first generation of database management systems.

The second era was also characterized by the use of product software and the advent of "software houses." Software was developed for widespread distribution in a multidisciplinary market. Entrepreneurs from industry, government, and academia broke away to "develop the ultimate software package" and earn a bundle of money.

As the number of computer-based systems grew, libraries of computer software began to expand. In-house-developed projects produced tens of thousands of program source statements. Software products purchased from the outside added hundreds of thousands of new statements. A dark cloud appeared on the horizon. All of these programs—all of these source statements—had to be maintained when faults were detected, modified as user requirements changed, or adapted to new hardware that was purchased. Effort spent on software maintenance began to absorb resources at an alarming rate. Worse yet, the personalized nature of many programs made them virtually unmaintainable. A "software crisis" had begun.

The third era of computer system evolution began in the early 1970s and continues through the early 1980s. The distributed system—multiple computers, each performing functions concurrently and communicating with one another—greatly increased the complexity of computer-based systems. As microprocessors and related components became more powerful and less expensive, products with "embedded intelligence" replaced larger computers as the most common computer application area.

In addition, the advent of microprocessors has resulted in the availability of complex logical functions at exceptionally low cost. This technology is being integrated into products by technical staff who understand hardware but are frequently novices where software is considered.

Rapid advances in hardware have already begun to outpace our ability to provide supporting software. During the third era, the software crisis intensified. Software maintenance absorbed over 50 percent of data processing budgets, and software development productivity could not keep pace with demands for new systems. In response to a growing crisis, software engineering was taken seriously for the first time.

A transition to a fourth era of computer system evolution has already begun. Sixteen- and 32-bit microprocessors with one megabyte of primary memory will open as yet unforeseen application areas for computer-based systems. The transition from a technical to a consumer marketplace demands professionalism that can be accomplished only through computer system engineering.

1.2 COMPUTER SYSTEM ENGINEERING

Computer system engineering is a problem-solving activity. Desired system functions are uncovered, analyzed, and allocated to individual system elements. An overview of the computer system engineering process is illustrated in Figure 1.2. Techniques for system analysis and definition are discussed in detail in Chapter 3.

The genesis of most new systems begins with a rather nebulous concept of desired function. The objective of system analysis and definition is to uncover the scope of the project that lies ahead. This is accomplished by a systematic refinement of information to be processed, required functions, desired performance, design constraints, and validation criteria.

After scope has been established, the computer system engineer must consider a number of alternative configurations that could potentially satisfy scope. The following trade-off criteria govern the selection of a system configuration:

1. *Business considerations.* Does the configuration represent the most profitable solution? Can it be marketed successfully? Will ultimate payoff justify development risk?

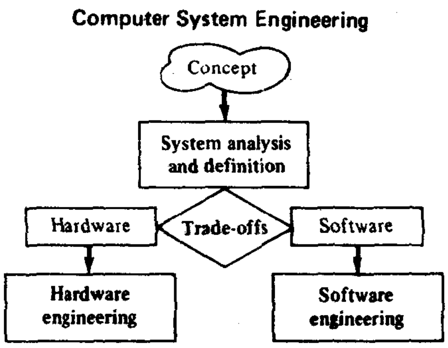

Figure 1.2

2. *Technical analysis.* Does the technology exist to develop all elements of the system? Are function and performance assured? Can the configuration be adequately mantained? Do technical resources exist? What is the risk associated with the technology?
3. *Manufacturing evaluation.* Are manufacturing facilities and equipment available? Is there a shortage of necessary components? Can quality assurance be adequately performed?
4. *Human problems.* Are trained personnel available for development and manufacture? Do political problems exist? Does the requester understand what the system is to accomplish?
5. *Environmental interfaces.* Does the proposed configuration properly interface with the system's external environment? Are machine-machine and human-machine communication handled in an intelligent manner?
6. *Legal considerations.* Does this configuration introduce undue liability risk? Can proprietary aspects be adequately protected? Is there potential infringement?

The weight of the above criteria vary with each system.

After trade-offs have been considered, a configuration is selected and functions allocated among potential system elements. For a computer-based system, hardware, firmware, and software are the elements most likely to be selected.

1.3 HARDWARE CONSIDERATIONS

Computer system engineering always allocates one or more system functions to computer hardware. In the following paragraphs basic hardware components and applications are discussed. In addition, an overview of hardware engineering is presented.

1.3.1 Hardware Components

The computer system engineer selects some combination of hardware components that comprise one element of the computer-based system. Hardware selection, although by no means simple, is aided by a number of characteristics: (1) components are packaged as individual building blocks; (2) interfaces among components are standardized; (3) numerous "off-the-shelf" alternatives are available; and (4) performance, cost, and availability are relatively easy to determine.

The hardware configuration evolves from the "building blocks" shown in Figure 1.3. Discrete components (i.e., integrated circuits and electronic components such as resistors and capacitors) are assembled as a printed circuit board that performs a specific set of operations. Boards are interconnected to form system components (e.g., processor and memory) that in turn are integrated to become the hardware subsystem or the hardware system element.

A complete discussion of the hardware configuration is beyond the scope of