PROCEEDINGS OF THE
SYMPOSIUM ON

COMPUTER SOFTWARE ENGINEERING

New York, N. Y., April 20–22, 1976

Microwave Research Institute Symposia Series
JEROME FOX, Editor

Volume XXIV

POLYTECHNIC PRESS
OF THE
POLYTECHNIC INSTITUTE OF NEW YORK, BROOKLYN, N.Y.

COMPUTER SOFTWARE ENGINEERING

POLYTECHNIC INSTITUTE OF NEW YORK

MICROWAVE RESEARCH INSTITUTE

SYMPOSIA SERIES

I	Modern Network Synthesis	April 1952
II	Nonlinear Circuit Analysis	April 1953
III	Information Networks	April 1954
IV	Modern Advances in Microwave Techniques	November 1954
V	Modern Network Synthesis II	April 1955
VI	Nonlinear Circuit Analysis II	April 1956
VII	The Role of Solid State Phenomena in Electric Circuits	April 1957
VIII	Electronic Waveguides	April 1958
IX	Millimeter Waves	March-April 1959
X	Active Networks and Feedback Systems	April 1960
XI	Electromagnetics and Fluid Dynamics of Gaseous Plasma	April 1961
XII	Mathematical Theory of Automata	April 1962
XIII	Optical Masers	April 1963
XIV	Quasi-Optics	June 1964
XV	System Theory	April 1965
XVI	Generalized Networks	April 1966
XVII	Modern Optics	March 1967
XVIII	Turbulence of Fluids and Plasmas	April 1968
XIX	Computer Processing in Communications	April 1969
XX	Submillimeter Waves	March-April 1970
XXI	Computers and Automata	April 1971
XXII	Computer-Communications Networks and Teletraffic	April 1972
XXIII	Optical and Acoustical Micro-Electronics	April 1974
XXIV	Computer Software Engineering	April 1976

Organized by

The Polytechnic Institute of New York
Microwave Research Institute

With the participation of

The Institute of Electrical and Electronics Engineers
Computer Society
Professional Group on Reliability
The Association for Computing Machinery

Co-Sponsored by

The Air Force Office of Scientific Research
The Office of Naval Research
The U.S. Army Research Office

Arrangements for this Symposium were supported in part under the Joint Services Electronics Program at the Polytechnic by the Air Force Office of Scientific Research under Grant Number AFOSR-75-28911

DISTRIBUTORS: **HALSTED PRESS, A DIVISION OF JOHN WILEY & SONS, INC. NEW YORK/LONDON/SYDNEY/TORONTO**

Edited by
Jerome Fox

Associate Editor
Mark Howard Schlam

Assistant Editor
Walter O. Peter

Library of Congress Catalog Card Number 76-26332

TABLE OF CONTENTS

FOREWORD

WELCOMING ADDRESS

Dr. George Bugliarello
President, Polytechnic Institute of New York

I am here to extend the warmest greetings of Polytechnic and the hope that this symposium—the twenty-fourth in the series sponsored by MRI—will be a successful and rewarding one.

On behalf of Polytechnic, I should like to express to all of you our thanks for your being willing to meet in this embattled city of New York. We can give you reasonable assurances that it will not sink under your feet in these three days—but should it sink, please remember—women and children first!

We are delighted at the most distinguished group of speakers and participants that the symposium has gathered. I know that I am expressing the gratitude, not only of Polytechnic, but of all of you, in thanking for their sponsorship the Joint Services Technical Advisory Committee, the Institute of Electrical and Electronics Engineers, and the Association for Computer Machinery. But above all our thanks go to Professor Oliner, Professor Shooman, Jerry Fox, and the other members of the Program Committee.

We hope that for some of you it may be possible, during or after the symposium, to take a closer look at what we are doing at Polytechnic, both in Brooklyn and Farmingdale. In general, the two years that have elapsed from the last MRI symposium have been a period of immense progress at Polytechnic. To give you just two figures, our freshman enrollment has increased by over 130%—to a considerable extent because of the opening of undergraduate programs at our Farmingdale Campus—and our research awards per faculty member have increased by some 40%. We have also opened graduate programs in Westchester—including programs in computer science.

Again, thank you very much for coming, and welcome.

GREETINGS FROM THE CO-SPONSORING AGENCIES

Lt. Col. Donald R. Lasher
Commander and Director, U.S. Army Communications/Automatic Data Processing Laboratory

It is a distinct honor and pleasure for me to represent the Department of Defense in this twenty-fourth annual symposium organized by the Microwave Research Institute of the Polytechnic Institute of New York. It is a particular pleasure for me to be here since the subject of Computer Software Engineering is one in which I have special interest, and I feel it is a topic most timely and appropriate for such a symposium. Therefore, on

behalf of the Department of Defense, the Army, the Navy, and the Air Force, I wish to extend our best wishes for a very successful and profitable meeting.

In reviewing the agenda, I was not only impressed with the breadth and depth of the coverage, but also reminded of the long way we have come in software development and engineering since I first wrote a computer program almost twenty years ago. In those days our *Software Engineering* was done by sitting at the operator's console and half-stepping our way through the programs, most often done in the wee hours of the night.

High-order languages and compilers (for which we are so indebted to the likes of Jean Sammet and Captain Grace Hopper) were in their infancy; and design, verification, and debug tools were essentially nonexistent. Programming was considered an *art*, and disciplines were also nonexistent. But it was fun!

Of course, computers were also much different then. The term nanosecond had not even been coined, much less comprehended; and the equivalent of central processors that then weighed tons can now be held in your hand! Even though I am presently in the business of developing even smaller and faster technology, such progress still boggles my mind.

Therein lies much of our current problem, which is that hardware technology has to a large degree far outstripped our abilities to use it efficiently. This assertion is supported by several studies that show that where software used to be 10 to 20% of total system cost, it is now often 75 to 80%, with predictions that it will reach 95% by the 1980's!

I recognize that much of this relative hardware-to-software cost ratio is due to the dramatic increases in hardware "bang for the buck," and to the substantial increase in the complexity of our computer based systems and their software. Nevertheless, it is fact that the costs of software development and maintenance is burgeoning—at least within the Department of Defense, where our annual cost is measured in billions each year.

As Mr. Gansler will no doubt amplify in his address, the Department of Defense has become seriously concerned about the sharply rising costs of computer software and the importance of computer software in the weapons systems needed to insure our national security.

Our software problems in the Army are probably representative of those that exist throughout the DoD. For our part we have come to realize that software too often is expensive, late, unreliable, inflexible, and fails to meet the user's needs. This has been especially true in the tactical or weapons systems environment, where there is a need to perform complex functions in so called *real-time* situations. Add to this the complication of concurrently developing and debugging the hardware and the executive and support software (run-time executives, compilers, etc.) and you can begin to appreciate the problem.

In sum, we haven't done very well in the past in developing or procuring software. Of the many causative factors contributing to this situation two of the most important are: (1) a lack of management understanding of and emphasis on the software development process (after all hardware is easier to kick and touch); and (2) the absence of DoD/Industry accepted software development disciplines. The first we are doing something about, the second we hope you will help us with.

In recognition of this situation several joint service efforts have culminated in a soon to be published DoD Directive called 5000.XX. This directive, among other things, requires the DoD components to "develop and implement a disciplined approach to the management of software design, engineering, and programming." This directive also requires that all developments use a high-order language.

And so our work is cut out, we need to:

(1) Achieve effective high-level language programming for our time, space, and weight-critical systems
(2) Develop an effective generalized approach to real-time tactical executive systems
(3) Develop an approach to take advantage of existing and emerging commercial hardware architectures and software, while also achieving an integrated approach to software development that will significantly reduce errors and life cycle costs.

We are working on these, but we need help. So you can appreciate why we are very interested in promoting symposia such as this one, which will, we hope, eventually lead to a software engineering discipline applicable to our software development efforts.

Thank you and good luck!

Dr. Robert F. Cotellessa

Executive Vice President, Institute of Electrical and Electronics Engineers

Mr. Chairman, President Bugliarello, Mr. Gansler, distinguished participants, and guests in this twenty-fourth Microwave Research Institute symposium, it is a unique pleasure for me to represent IEEE in this introductory session. IEEE has enjoyed an association with the MRI symposia beginning with the first one in 1952, and feels privileged to do so.

In view of IEEE's strong espousal of leadership in research and development, it is appropriate to applaud the support that the U.S. Army, Navy, and Air Force research offices have continued to provide. Harold Zahl, Arnold Shostak, and William Otting may be remembered as the persons who represented these offices for the early symposia.

The Microwave Research Institute has demonstrated great prescience, first, in not dedicating the symposia to a single subject area, and, second, in its choice of different topics. Each meeting is a gratifying surprise which, with the reputation for excellence that has been achieved, attracts attention and attendance from all parts of the world. I am sure that the focus that each symposium has given to a nascent area has influenced directions of research and development. Individual investigators have been stimulated to develop new ideas and to perceive new objectives.

The topic of this symposium is no exception to the historically established characteristics of the series. Computer Software Engineering is viewed by many as a bridge between the increasingly fuzzy boundaries of computer hardware and software. In this context, it is appropriate that the two technical societies, IEEE and ACM, are joint participants in this meeting. For the IEEE, the Computer Society and the Group on Reliability represent the Institute. The content of the technical program itself provides insight as to the identification of the bridge.

I cannot conclude without paying tribute to Arthur A. Oliner and the works he has wrought. Many here may not be aware that he was one of the first IEEE Microwave

Theory and Techniques Society National Lecturers and set the high standard by which that program has prospered; he has contributed in many other roles in IEEE as well. He and his colleagues at MRI are to be congratulated today on the excellent symposium that has been arranged.

To all participants, greetings and best wishes for realizing the opportunities that this symposium offers.

Ms. Jean E. Sammet
President, Association for Computing Machinery

It has been true throughout the history of the computer field that there have been a number of fads, both in terminology and in concept. This is not necessarily bad; sometimes these fads tend to point up in a very concrete and useful way various concepts and ideas which have been around for many years, albeit sometimes under different headings. Some years ago one of the fads was modular programming; today the fad terminology includes automatic programming, structured programming and of course the subject of this symposium, namely Software Engineering. I am sure that almost every speaker will provide his or her own definition of Software Engineering and therefore I am not going to add to the confusion by trying to create still another definition. I will simply say that in my view Software Engineering, as the term is being used today, is meant to encompass all of the concepts, ideas, and techniques which improve the likelihood of getting programs written on time, more cost effectively, reliably, and with more efficiency of both personnel and machine time. That of course is a very large scope of activity and does not leave out very much of anything!

I am very glad to say that using that terminology, in one way or another ACM has been involved with Software Engineering almost since ACM's beginning in 1947, even though the words have not been used that frequently. Throughout its 29 year history most of the emphasis and activity within ACM have been in the software area although we do have a number of members who are interested in such topics as computer architecture and microprogramming. The concerns of ACM have traditionally ranged from the very theoretical—such as automata theory—to the far more practical, namely how does one get better programs. It is certainly true that more of ACM's publications have been in the theoretical area than in the practical, and this is to be expected since more of that work is done at colleges and universities than in industry and furthermore the faulty people are often given incentives for publication whereas the industrial people unfortunately are not.

Since ACM is a very large organization—and in fact the largest in the computer field—having over 32,000 members as of April 1976, it is natural that some type of subgrouping is needed to allow people to concentrate on specialized interests. The mechanism for doing this within ACM is via our grass roots Special Interest Groups and Committees, of which we have almost 30 ranging in membership size from a few hundred to over 5000 for the Special Interest Groups on Programming Languages and Operating Systems. Directly related to the subject of this meeting, last year we formed a Special Interest Committee on Software Engineering which is just beginning to get off the ground. Since some of the issues within the field of Software Engineering specifically relate to documentation I should point out that we have a separate (and somewhat older) Special

Interest Committee (SICDOC) just on that subject and it puts out an excellent newsletter. (Our Special Interest Groups and Committees traditionally put out informal newsletters containing technical material as well as news and notices. The first issue of the SICSOFT newsletter is expected very soon.) Those of you who want more information can contact ACM.

Let me say just a few words on SICSOFT activities that are under way. This Special Interest Committee will be organizing technical sessions at the ACM annual conference in Houston in November 1976; we have joined with the IEEE Computer Society and the National Bureau of Standards in the Second International Conference on Software Engineering to be held in San Francisco in October 1976; and there is a specific ACM conference on Language Design for Reliable Software which will be held in Raleigh, North Carolina in March 1977.

I want to urge all of the attendees not to be entirely misled by the terminology *Software Engineering*. There are many cases in which material has been identified under that label but rather incorrectly in my judgement; similarly, there is quite a bit of activity going on which is quite properly within that framework but which uses other terms to describe it. Hence when you look at conferences and papers and publications, it is important to look below the title to see what the subject really is. As I looked at the abstracts for this conference, they appeared to range from theoretical work to fairly practical attempts to improve the quality of our software, which is after all what we are all interested in. On the other hand, some of the abstracts seem to deal with subjects which have shown up at numerous other apparently unrelated conferences. This just supports the contention I made at the beginning which is that the term Software Engineering is being used for almost any work people wish to report on. In that very broad framework, most of ACM's technical activities involve Software Engineering.

POLYTECHNIC INSTITUTE OF NEW YORK

G. Bugliarello, *President*
J. J. Conti, *Provost*
R. J. Cresci, *Associate Provost for Research*

MRI SYMPOSIUM COMMITTEE

General Chairman: A. A. Oliner, *Director, MRI*
Executive Secretary: J. Fox, *Assistant Director, MRI*

MRI PROGRAM COMMITTEE

M. L. Shooman, *Chairman*

M. Adamowicz	M. H. Halstead	M. G. Mesecher
L. Belady	H. Hecht	J. D. Musa
B. W. Boehm	A. E. Laemmel	P. D. Patent
R. Flynn	M. M. Lehman	H. Ruston
S. Habib	J. H. Manley	J. R. Suttle

THE JOINT SERVICES TECHNICAL ADVISORY COMMITTEE

OFFICE OF NAVAL RESEARCH
J. O. Dimmock, *Chairman* D. K. Ferry

AIR FORCE OFFICE OF SCIENTIFIC RESEARCH
J. W. Gregory, *Member* G. E. Knausenberger

THE U.S. ARMY RESEARCH OFFICE
H. K. Ziegler, *Member* H. Robl
J. E. Teti, *Executive Secretary*

PARTICIPATING PROFESSIONAL SOCIETIES

THE INSTITUTE OF ELECTRICAL AND ELECTRONICS ENGINEERS
J. K. Dillard, *President*

ASSOCIATION FOR COMPUTING MACHINERY
J. E. Sammet, *President*

THE *EVEREST* OF SOFTWARE

Lewis M. Branscomb
Vice President and Chief Scientist, IBM Corporation, Armonk, NY

Much has been written about the tremendous advances of computing over the past few decades. In fact, when we computerniks get together, we generally pat each other on the back and tell ourselves how great we are. I, myself, am quite guilty of quoting frequently the famous statement by John Pierce, who said: "After twenty-five years of extraordinary progress, the computer industry is ready to enter its infancy."

Before I comment on this statement in detail, let me correct an historical inaccuracy. It is not true that computers have been with us for only twenty-five years. In fact, the earliest recorded reference to computers was made by Jonathan Swift in his "A Voyage to Laputa," Chapter 5, where he describes a visit of Gulliver to the Academy of Laputa. This passage describes a device invented by the Professor of Speculative Learning, which can be described as nothing less than a computer, and, in fact, contains many of the elements of some computer science projects of our days.

The computer of Laputa was a "Frame" that contained "· · · all the words of their language in their several Moods, Tenses, and Declensions, but without any Order · · ·." These words were cranked out by the Professor's pupils, and any strings of them that could compose a piece of a sentence were put aside. Thus the Professor expected to create "a complete Body of all Arts and Sciences; which however might be still improved, and much expedited, if the Publick would raise a Fund for making and employing five hundred such Frames · · ·." As you can see, this passage contains not only a fairly technical discussion of a modern Monte Carlo text generation project, but, also, the usual appeal for government support.

Let me now analyze the elements of a computing process. According to Harlan Mills, a computing process is nothing more nor less than the operation of a multiprocessing system. Even the simplest computing operation involves at least two processors: a human being; and a computer. In order to make these two processors work together, we need two other elements: a users' language which interfaces with the human being; and a software package which interfaces with the computer.

Let us now go over these four elements of a computing process and see how well John Pierce's statement reflects reality; that is, how well we stand today with respect to the development of these four elements, and where we are going from here. Let's take the human being first. I think it is fair to say that there is no reason for great concern for the technological development of that element. It has been engineered for many centuries, and its manufacturing process has been perfected to a great extent. I would be perfectly willing to say more about this subject, but I understand it is outside the scope of this meeting.

The users' manual is a puzzling subject to me. As near as I can make it, there are two kinds of people in this world, the writers of users' manuals and the readers of users' manuals. The two groups work on completely different planes of understanding. Every

---

Address presented at the banquet following the opening day of the symposium on Computer Software Engineering, Polytechnic Institute of New York, April 20–22, 1976.

time I open a users' manual, I remember the following opening sentence in a book by A. A. Blasov of Moscow University: "The purpose of the present course is the deepening and development of difficulties underlying contemporary theory." Nevertheless, I feel that we are only a couple of mutations away from developing a breed of computer specialists who can understand users' manuals, so I will not worry about this particular topic.

The third element of the process is the computer itself. There we have absolutely nothing to worry about. As everyone knows, the advances in computer hardware have been so phenomenal that even the pioneers in the computing industry are amazed at how far we have gone during the past twenty-five years. This is, of course, exactly what John Pierce meant by his famous statement. Progress in computer hardware, with respect to cost, speed of computations, and decrease in size, is measured by several orders of magnitude, and the most dramatic is the 40% compound annual rate of reduction in the unit cost of memory and storage. This has permitted all kinds of shortcomings in software efficiency to be swept under the rug of expanding memory.

So now, let us take a look at the fourth element of the process, the software. There we seem to be in trouble. We all know that a science of software has yet to be developed. Software design is still largely an art, and you good people are the struggling practitioners of this noble art. I know how difficult your struggle is, and I want to tell you that I am very sympathetic with your problems. Of course, for me to display my sympathy with your problems is very much like the practice of certain Greek peasants of lying in bed in sympathy for their wives who are convalescing from childbirth.

But what is really wrong with the software process? One of the problems, of course, is the increasing complexity of software, particularly systems programs. The size of systems programs was measured in a few thousand instructions in the early days of computing, but it is measured in millions of instructions today, in some cases. What is more, we seem to be unable to develop building blocks of software, analogous to the building blocks that we have developed for hardware. In fact, it is worse than that. It has been said that every systems program is done by an amateur, because everyone who has prepared a systems program once doesn't want to do it ever again.

Everyone who has given any thought to the problem of software comes to the conclusion that we *must* develop a design methodology for software. I wish I could have stood before you today and given you a blueprint for the development of this design methodology. I cannot give you that, but I can try to identify some principles that we may use in scaling this seemingly unreachable height, the *Everest* of software. First of all, I think we must do away with our sometimes excessive preoccupation with efficient use of the hardware. This may sound like a self-serving statement, in view of the nature of my employment, but it is not. I assure you that I am speaking at this moment as a user of a computer, and not as a seller of hardware. There *must* be a balanced consideration of questions of efficiency.

Let me illustrate where efficiency may sometimes lead you by reading a passage from a book titled "A Random Walk In Science," which contains a report by an anonymous author of a visit by a team of efficiency experts to the Royal Festival Hall:

> "For considerable periods the four oboe players had nothing to do. Their numbers should be reduced, and the work spread more evenly over the whole of the concert, thus eliminating peaks of activity. · · · All the twelve first violins were playing identical notes. This seems unnecessary multiplication. The staff of this section should be drastically cut; if a large volume of sound is

> required, it could be obtained by means of electronic amplifiers. · · · Much effort was absorbed in the playing of demisemiquavers. This seems an excessive refinement. It is recommended that all notes should be rounded up to the nearest semiquaver. If this were done it would be possible to use trainees and lower grade operatives more extensively. · · · There seems to be too much repetition of some musical passages. Scores should be drastically pruned. No useful purpose is served by repeating on the horns a passage which has already been handled by the strings. It is estimated that if all redundant passages were eliminated the whole concert time of two hours could be reduced to twenty minutes, and there would be no need for an interval."

On a more serious vein, efficient use of hardware must be balanced against efficient use of the programmers who write software. Sometimes, I feel that we have been conditioned from the days when computers were very expensive tools that had to be extremely well utilized. This is a well-known economic principle, that is not only appropriate for the use of computers. A piece of machinery must be utilized a lot better in India where its relative value with respect to human labor is much higher than it is in this country. The same piece of machinery can be *burned* in this country in order to save a few man-hours which are expensive.

I think there is a certain lag in our awareness of the decreasing costs of hardware as a percentage of the total cost of the computing process. If we look at any data, we come to the conclusion that the relative cost of hardware, which started at something like 90% of the total cost of computing in the early days of computing, is now well below 50%, and slated to be less than 10% by the middle 1980s. When the software cost becomes nine times the hardware cost, simple arithmetic will tell you that you can afford to double the cost of hardware if you can save anything more than one-ninth of your software expenditures. If you want a more sophisticated economic statement, it must be: "The code efficiency must be at a level such that the cost for a marginal increase of efficiency must be equal to the corresponding marginal decrease of hardware costs." I have a nagging suspicion that we are not running our computing business according to this principle today. The problem has its roots in the lack of a methodology for performance specification and evaluation—a way to put together the appropriate mix of hardware and software to meet performance requirements.

I have stated an economic principle in balancing software development costs against hardware costs, but I am well aware that even the best economic principle does not by itself solve the software design problems. I have mentioned, earlier in my talk, the desire to have modularity in software, which would permit building a large system out of building blocks. But modularity implies stabilized interfaces. And the rate of evolution of data processing technology—the shift from batch processing to time-sharing, and from remote job entry to on-line interaction, plus networking and distributed processing, makes this stabilization very difficult at this stage of the computing industry.

Undoubtedly, as our industry matures, some stabilization in software design interfaces will be implemented, which will hopefully accomplish two things. First, it will make the design of software more tractable. And, second, it will decrease the software maintenance costs. This second objective is, of course, a very basic one. We all know that a major and increasing part of a data processing department's budget is spent in maintaining its software system. Thus, a major portion of our software manpower is spent in maintenance rather than development of software. In fact, if present trends were

to continue, one could see the day when our development effort would grind down to a halt because all our software manpower would be used in maintenance of existing software. A mathematical friend of mine proposed a nightmare scenario based on the hypothesis that this point of zero development is not necessarily a limiting one. He suggests that it is possible to have more than a hundred percent of the software manpower devoted to software maintenance, together with a negative software development effort. Astronomers would recognize this as the black hole of programming that follows the collapse of a giant software effort whose energy has burned out. Clearly, we must strive to avoid such a disastrous fate. And we must also avoid falling victim to "Conway's Law." For those of you who do not know, Conway is a Professor of Computer Science who observed that "the organization of an operating system resembles the organization chart of the group that created it."

I realize that all my remarks are elaborations on the central theme that software development is lagging behind hardware development, and we must correct the situation. I assure you that I didn't come here tonight just to *Rub It In*. Software development is lagging because it is a much more difficult job. If it is any consolation to you, there is an analogy between the computer software versus hardware situation and the world picture. If you look at where the world stands today, you see that we have been extremely successful in managing our *hard* discipline of production, but not so good in developing the *soft* disciplines of managing the output of our production capability.

Finally, let me explain why I selected the specific title of my talk. The *Everest* of software is not climbed without hazards. Bosses who are derisive about systems programming producitivity—which typically runs 16 instructions per programmer per week—forget how often you are swept away by avalanches, how easily one gets out of breath, and what it feels like to have someone else climb up your back wearing crampons.

Harlan Mills, the eloquent advocate of top-down, structured programming, says the way to climb the Everest is to work smarter, not harder. You don't start at the bottom—not even knowing whether there is a top; you start at the top and work your way down, the work getting easier and easier.

Thus, the moral of my talk is to improve on the old adage "you climb the mountain because it is there." Instead, you must specify, design, and implement the mountain yourselves—beginning with a specification of where you want to end up at the top. Then you can tell the rock, snow, and ice factories what you want them to build and how it should fit together. And all around the bottom of the mountain, you can attach lush valleys, bright rivers, and green forests, knowing that the people who enjoy them don't need to worry about avalanches and rock slides—or even need to know what the mountain looks like under its majestic shrouds of clouds.

KEYNOTE: SOFTWARE MANAGEMENT

J. S. Gansler
Deputy Assistant Secretary of Defense (Materiel Acquisition), The Pentagon, Washington, DC

The magnitude of software activity within the DoD is discussed from a cost and criticality point of view. Problem areas and their propagation through the life cycle are examined both with respect to observable manifestations and underlying causes. An overview of current Department of Defense actions to remove or diminish these problems is presented. The identified components of the solution are: (1) organizational focii with DoD and the Military Departments; (2) policy initiative; (3) practice and procedure initiatives; and (4) technology initiatives blended together around the theme of increased discipline and rigor in the software design, development, implementation, test, operation, and maintenance activities. Each of these components is discussed, and a prognosis given for ultimate success of the DoD Defense System Software Management program.

I. INTRODUCTION

Within the Department of Defense, we are presently spending over three billion dollars per year on Defense System Software (excluding Automatic Data Processing). In my opinion, we have been doing a poor job managing this increasingly important resource, and further we have been doing little research and development on the ways and means to improve it. Both of these shortcomings must change!

Today I'd like to discuss the problems, and their underlying causes which confront us in this area, to summarize the corrective actions we are now taking, and finally to solicit your help in carrying out the major new initiative we are undertaking. Neither the problems nor the proposed solutions that I will discuss are new. They have been studied at great lengths in technical meetings, and in industry for the last three years or more. The new thing that I want to convey here today is the sense, at all levels of DoD, of the need to act decisively and to act now!

Over the past few years we have had a series of studies, each of which highlighted this area of Defense System Software as one requiring change for reasons of cost and reliability. During the past year, we have put together (with much help from industry and the university communities) a preliminary plan of action, that is my topic for today.

Let me begin by saying that the main thrust of my remarks will not be directed at the traditional, general purpose Automatic Data Processing environment, the ADP community has been in existence for several decades now and there does exist an adequate organization and enough identifiable resources to attack these problems in an effective way. Instead, I will concern myself with those issues which provide the incentives for, and the barriers to good software engineering and management in the Defense System acquisition process. Recognizing, however, the need for full consistency with the ADP

Keynote address presented at the Symposium on Computer Software Engineering, Polytechnic Institute of New York, April 20–22, 1976.

community, close ties are, and must continue to be maintained to assure maximum transferability of ideas, tools, and techniques.

In general, we believe that because of their R&D nature, and close tie-in with other components of Defense Systems, the software practices most applicable to embedded computers are closely allied to those management practices which have been developed for hardware acquisition. I will elaborate on this later, but first let us consider the scope and the problems.

II. COST PERSPECTIVE

Software is big business within the Department of Defense. The current annual expenditure on Defense System software is now estimated in excess of three billion dollars; yet even this substantial sum is the tip of the iceberg. It includes direct costs only and represents a conservative estimate based on incomplete and nonuniform data. This uncertainty, as a matter of fact, is indicative of a clear problem in itself. The distribution of costs for a given year shows that 68% of the known costs is consumed in system development, while the remaining 32% of the known cost is classified as operation and maintenance. As we move further into the "age of computers," more systems now in development will transition to the operation/maintenance phase. This, coupled with high system longevity, may ultimately result in a five or ten to one ratio of operation/maintenance cost to development cost when viewed over the total life cycle (i.e., similar to the hardware ratio of life cycle to development costs).

III. MISSION CRITICALITY PERSPECTIVE

Over and above the cost picture, software is finding its way onto the critical path of more and more Defense Systems. Major Defense Systems currently exhibiting a critical software dependency number approximately 115, with 50% of these in the Research & Development phase and the remaining 50% in the Operations and Maintenance phase.

The functional applications of software within the DoD pervade almost every program. Software applications can be found in diverse systems from the large World-Wide Military Command and Control System and the B-1 Strategic Bomber down to miniaturized avionics packages and such ancillary operational equipment as trainers, simulators, automatic test equipment, and certain types of test ranges and test vehicles.

IV. NATURE OF THE PROBLEM

Several clearly observable manifestations such as excessive development and maintenance costs; schedule slippages and delays; excessive errors or faults; and duplication

and lack of standardization have emerged from our "lessons learned" base as characteristic problems of most major Defense Systems acquisitions. These manifestations, of course, are symptomatic of underlying problems which originate far earlier in the development cycle. These include: lack of early management visibility and discipline; lack of life cycle perspective; insufficient R&D; insufficient control over expenditures; lack of standardization; lack of transferability; lack of hardware/software trade-offs; and the treatment of software as data rather than a configuration item are some of the real leverage points for relieving the cost and quality pressures underlying the observable manifestations of the problem.

Over the next few days, you will be hearing many papers which deal with these basic problems. Lets briefly explore some of the issues which are of particular relevance to this conference.

A. Inadequate Cost and Schedule Estimates

Studies by industry have concluded that there are no simple universal rules for costing software accurately, and that to estimate it accurately it is necessary to understand the nature of the individual program and the individual routine within the program. I am sure that this will remain in the situation for some time to come but we must begin now to take action to reduce the amount of "individuality" in cost estimating. In this regard, it must be said that we currently suffer from a poor historical cost data base. Not only are we unaware of what we in the DoD are spending on software in the development, production and especially operational and maintenance phases of Defense System's life; but we are unsure of the proportions of dollars which we should be spending. As it currently stands we do not allow dollars or time for the likely problems and changes which occur in each of these phases.

This situation is aggravated by the lack of common definitions, procedures, and organization in planning and managing software development. While most approaches may have merit, and some are excellent, it is not practical for Government review officials to be well acquainted with all the approaches presented to them. As a result, they cannot develop broad applicable yardsticks--they cannot really understand what is proposed or in process for each program--they cannot apply sound judgment in their management responsibilities. In general, we *believe* the estimates which are provided to us--no matter how optimistic--and budget to them. That's how the problem got started!

B. Tracking User Requirements

This is one of the biggest contributors to the high cost of software as one of your sessions recognize. The problem here again has to do with the absence of a clear understanding on the part of those managing software as to what can be accomplished with software. Frequently, they either underestimate or overestimate the state-of-the-art. At the same time it is important that software specialists be able to anticipate the likely directions of change and design software and software tools so that it is fairly easy to accommodate changes when they come.

C. People as a Cost Factor

Each contractor has the problem of having highly trained individuals to develop and maintain the software. At a recent software conference, statistics were produced to show that the turn-over time for an average programmer was about three years. There is a motivation problem for the software "production" worker, just as with the hardware "assembly line" worker. To illustrate how highly-labor-intensive software production is, it has been estimated that increasing programmer productivity from an average of ten instructions per man-day to eleven could save as much as 45 million dollars per year.

D. People in Hardware/Software Trade Offs

Another part of the software environment impacting on the personnel issue is the lack of our ability to make the necessary trade-offs between hardware and software implementation. To assess the strengths, weaknesses, and ensuing implications of these trade-offs on life cycle cost and reliability, we need people with in-depth understanding of both disciplines, and who can objectively perform and integrate trade-off analyses to produce a balanced system. These people are in extremely short supply and their cultivation in the future remains a major educational problem.

E. Duplication of Applications Software Efforts

I have no way of knowing exactly how much we in Defense spend on "applications" software which had already been accomplished, for some other program or programs. People with whom I have talked in the Services, tell me that it is extensive and that our first efforts to control costs should begin in this area. Without clear software development standards and adequate software management in Defense Systems acquisition, the increase in costs owing to the duplication of efforts can only be expected to grow.

F. High Cost of Maintenance

A significant cost factor has been the software errors or problems discovered well after acquisition. One recent DoD study showed that Air Force avionics software costs something like 75 dollars per instruction to develop, but the maintenance of the software has shown costs in the range of 4,000 dollars per instruction. The purpose of quoting these figures is not to offer them as representative numbers but to demonstrate that the costs of maintenance are many times those for development. I might mention here that software maintenance, in addition to correction of problems, includes updating and revision of applications programs caused by changes or expansion of the operational mission. In the future the DoD will need to take a strong look at the life cycle approach to acquiring software. This will include the formulation of design and management principles to assure software life cycle cost models, and design for ease of maintenance and program update.

G. Insufficient Software R&D

The next issue which I would like to take up concerns the need for more directed research and development on software. I refer here to the need to convert software from an art into a technology. This can only be accomplished by giving increased attention to the Research and Development of software tools. We must get away from the notion that software advancements are the sole domain of "rugged individualists" or "artists." The best practitioners, individual stars, can produce exceptionally fine software on schedule at low cost. The general run of programmers and analysts are, however, far from this standard and increase the cost, schedules, and quality of our procurements by one or two orders of magnitude. It is not very useful to say "just hire the good guys." We have no measure of either the software or its practitioners. It is all "unknown," and the fact that software is "invisible" makes it that much harder. In this regard, I am greatly encouraged by many of the efforts which I have recently observed in both the Government and business sectors to increase software production capabilities. The "software factory" concept is fast becoming an important means for effecting the necessary changes that are needed in software development practices. It involves the employment of an integrated set of tools to provide a disciplined and repeatable approach to software development and to replace ad hoc agglomerations of developmental techniques and tools with a standardized methodology. One of the key objectives of this approach has been to introduce manufacturing methods and engineering principles into those software production processes which satisfy common design, implementation, and management requirements of projects.

H. Insufficient Software Management & Control

My final area of overall concern has to do with the subject of software management in the Department of Defense, and specifically as it pertains to Defense Systems. We have become somewhat expert at knowing how to divide the responsibilities of the "requirements" and the "procurement" people in the hardware acquisition process; and there is a fairly clear line of demarcation between what is hardware and what constitutes data. Software, however, creates something of a problem, for up until recently most managers and contracting personnel were content to treat it simply as data. As costs began to soar it became obvious that some management changes were in order. If we are to manage software in the same manner as we do hardware perhaps we must begin to think of software as "property" and not solely as "data" (fully recognizing the legal implications of the term "property"). Cost data which are submitted for data items in contracts are usually only estimates and do not provide for detailed cost breakdowns for each data item. Frequently, it is difficult to get a clear and distinct separation of data costs from engineering efforts tied to a deliverable contract schedule item. We must certainly take steps to clear up this matter for software estimates.

More to the point, however, we must recognize that from a functional standpoint computer software is equivalent to hardware, and must be delivered as an active system component. This means that technical and management control is required to insure a well engineered quality product. Management instruments and disciplines influencing computer software engineering, prototyping, configuration control, quality assurance,

production control, reliability and maintainability, standardization, modular partitioning, design reviews, and life cycle costing must be applied.

V. CURRENT ACTIONS

The problems and issues I have raised are real, and they require our immediate attention. We have begun to take action aimed at improving the management situation for both the short and the long term. I would like to elaborate further on specific actions now taking place within the DoD, and to highlight the areas in which further policy guidance will soon be forthcoming.

We have created the proper organizational focii within DoD both at the OSD and Service levels. A DoD-wide software management plan which addresses all of the problems I have spoken of this morning has been derived and developed. This plan has been released and is soon to be available through the Defense Documentation Center; a DoD Directive establishing policy for the management and control, by DoD Components of computer resources and software during development, acquisition, deployment, and support of Defense Systems, has been written, co-ordinated throughout all OSD and Service organizations, and submitted to the Deputy Secretary of Defense for signature. The theme pervading all of these steps is to elevate software policy, practices, procedure, and technology from an artistic enterprise to a true engineering discipline. Or to say it another way, to treat software more like hardware throughout its complete life cycle.

VI. SOFTWARE REQUIREMENTS AND RISK ANALYSIS

The first area of emphasis under our newly formed policy initiative concerns the requirements validation and risk analysis attendent to computer resources and software. This topic will be covered in one of your sessions later today. Briefly stated, computer resource requirements with particular emphasis on software, and on hardware/software trade-offs must be reviewed, analyzed, and validated during the Concept Formulation and Program Validation phases of Defense System development, prior to the full scale development decision point. This analysis must assure conformance of planned computer resources with stated operational requirements. Risk analysis, preliminary design, hardware/software integration methodology, use of existing software modules, standardization, external interface control, security features, and life cycle system planning will be included in the review. Correctness of software, reliability, integrity, maintainability, ease of modification, and transferability will be major considerations in the initial design. The risk areas, and a plan for their resolution shall be included in the Decision Coordinating Paper at the OSD level. In addition, computer resource requirements will be continuously co-ordinated and reconciled with system operational requirements throughout system development after the decision to enter full scale development.

The effect of this policy will be to emphasize the front end technical efforts which occur prior to a major management commitment, and to insure that it is given the same attention as hardware during the early phases of system development. In addition it will

provide top level Service and OSD visibility into cost, schedule, option, and risk parameters at a time when subsequent development can be meaningfully impacted.

VII. COMPUTER RESOURCE LIFE CYCLE PLANNING

A computer resource plan will be developed prior to the decision to enter full scale development, and will be maintained throughout the life cycle. The purpose of the plan is to identify important Defense System computer resources acquisition and life cycle planning factors, both direct and indirect; and to establish specific guidelines to ensure that these factors are adequately considered in the acquisition planning process. Resource planning is to include equipment, software, documentation, and personnel.

This policy will place economic trade-offs, acquisition strategy, maintenance and modification decisions on a life cycle basis, and eliminate the tendency to optimize development costs, schedules, and quality at the expense of the subsequent operations and support costs. We must stop mortgaging our future in exchange for fleeting benefits during development.

VIII. CONFIGURATION MANAGEMENT OF COMPUTER RESOURCES

The next policy area concerns the configuration management of computer resources in major Defense Systems. We can no longer afford to treat software as a data element to be acquired by a one-line entry on a Contract Data Requirements List. Instead it will be treated as a full-fledged configuration item, with all the attendant disciplines and control involved. The emphasis will be on product definition, requirements traceability interface definition and control, cost and equality traceability, and the corollary control discipline.

IX. SUPPORT SOFTWARE DELIVERABLES

When it is cost-effective to do so, unique support items required to develop and maintain the delivered computer resources over the system's life cycle will be specified as deliverable, with DoD acquiring rights to their design and/or use. Examples of such support items are compilers, environmental simulators, documentation aids, test case generators and analyzers, and training aids. The provisions of the Armed Service Procurement Regulations will govern the implementation of this policy.

This policy again emphasizes the life cycle cost-effectiveness aspect of the acquisition decision. It will remove the long term dependence on a single development contractor (thereby preserving DoD's maintenance and support options for the longest possible time), and it represents a necessary, although not sufficient step toward achieving true transferability of support software across mission and application lines.

X. MILESTONE DEFINITION AND ATTAINMENT CRITERIA

Specific milestones to manage the life cycle development of computer resources, including computer system and support software will be used to ensure the proper sequence of analysis, design, implementation, integration, test, documentation, operation, maintenance, and modification. These milestones will include specific criteria that measure their attainment.

This policy relates to the product definition and work accomplishment aspects of configuration management but the additional stress on quantitative demonstration criteria is significant to note. Also of particular significance is the rigorous treatment which must be accorded to test and evaluation, beginning in the earliest phases of system development, and culminating in a complete operational test and evaluation by the ultimate military users.

XI. SOFTWARE LANGUAGE STANDARDIZATION AND CONTROL

The next policy issue deals with programming languages, which is the subject of another of your sessions this week. DoD approved High Order Programming Languages (HOLs), *will be used* to develop Defense System software, unless it is conclusively demonstrated that none of the approved HOLs are cost-effective over the system life cycle, and this will not be easy. Each DoD approved HOL will be assigned to a designated control agent who will be responsible for issuing the stability of the language, validating compliance of compiler implementations with the standard language specifications, gathering data as to the use of the language, for disseminating information, compilers, and tools, and for improving it over time.

This policy impacts both the language selection and proliferation problems I have noted earlier. In general, high order languages do afford considerable life cycle benefits (particularly in the operation and support phases) even though some inefficiencies may be experienced in development. Exceptions to the use of high order languages must be justified over the life cycle, and not just for development advantages.

Our long range objective is to get down to a minimum number of DoD High Order Languages—and we are working in that direction; but our objective in achieving this standard is for cost reductions—so we must be flexible in how we apply it and how we allow it to improve over time.

XII. CO-ORDINATED RESEARCH AND DEVELOPMENT

In the area of research and development, a disciplined engineering approach to management of software design, engineering, and programming is essential as your conference indicates. We have already given funding guidance to the Military Departments to assure that this methodology is developed and is used. Specific actions underway within DoD are:

(1) Plan and execute a co-ordinated research and development program to identify and supply the technological base needed to support the policy, practice, and procedure initiatives contained in the Defense System Software Management Plan. Obviously all of the sessions to be covered in this conference are germane to this rather broad change to the research and development community
(2) Prepare and maintain appropriate guidance documents (e.g., guidelines, checklists, handbooks, and descriptive examples) covering requirements definition, development, acquisition, operation, and support issues attendant to computer software in Defense Systems. These documents will be available for use as necessary by program managers and their staffs as well as organizations tasked with specific responsibility for developing, acquiring, operating, and supporting the computer resource elements
(3) Establish and/or maintain appropriate education, training, and experience career paths with accompanying career incentives to foster the development and retention of professional computer resource engineers, managers, and technicians.

XIII. LOOKING AHEAD

I have summarized this morning some of the more important policy actions we are now taking within the Department of Defense. In varying degrees, these techniques are being applied to all current and new Defense System programs. I think we are now getting to the point where we can impact many areas which will change the way we do business within the Defense software community. We have great need of your help in achieving these objectives. I am certain that we will be able to rely on you, as we have in the past.