Methods for Improving Software Quality and Life Cycle Cost (1985)

Pages
124

Size
8.5 x 10

ISBN
0309323126

Committee on Methods for Improving Software Quality and Life Cycle Cost; Air Force Studies Board; Commission on Engineering and Technical Systems; National Research Council

Methods for Improving Software Quality and Life Cycle Cost

Committee on Methods for Improving Software Quality
and Life Cycle Cost
Air Force Studies Board
Commission on Engineering and Technical Systems
National Research Council

NTIS order number:
ADA 176 576

NATIONAL ACADEMY PRESS
Washington, D.C. 1985

75-0048
2.1

NOTICE

This report represents work under Contract No. F49620-83-C-0111 between the United States Air Force and the National Academy of Sciences.

Copies of this publication are available from:

Air Force Studies Board
National Research Council
2101 Constitution Avenue, N.W.
Washington, D.C. 20418

Participants in the 1983 Summer Study on Methods for Improving Software Quality and Life Cycle Cost

Charles R. Vick, Study Director, Auburn University
Mack Alford, TRW
Gordon Bate, Auburn University
Barry Boehm, TRW, DSSG
David Burlingame, University of Southwestern Louisiana
Judith A. Clapp, The MITRE Corporation
Balakrishnan Dasarathy, GTE Laboratories, Inc.
Julian Davidson, Ex-Officio Member, Booz-Allen & Hamilton, Inc.
Dennis D. Doe, Boeing Aerospace Company
Walter J. Ellis, IBM Corporation
Carl Engelman, The MITRE Corporation
Clarence Giese, AIRMICS, Georgia Institute of Technology
John Gioia, Robbins-Gioia, Inc.
Jack Goldberg, SRI International
Cordell Green, Kestrel Institute
Edward L. Lafferty, The MITRE Corporation
Robert Larson, Private Consultant
John J. Martin, Bendix Corporation
Brockway McMillan, Bell Telephone Laboratories (Retired)
Edward Miller, Software Research Associates
Stanley R. Mohler, Wright State University School of Medicine
Larry Moore, McDonnell Douglas Astronautics Company
Carl Murphy, Science Applications, Inc.
Motasim Najeeb, University of Southwestern Louisiana
F. Robert Naka, GTE Sylvania
Pat Nanartowich, University of Southwestern Louisiana
C. V. Ramamoorthy, University of California - Berkeley
Paul B. Schneck, National Aeronautics and Space Administration
Wei-Tek Tsai, University of California at Berkeley
Joseph Urban, University of Southwestern Louisiana
Woodie Vandever, Higher Order Software, Inc.
Willis H. Ware, Rand Corporation
Raymond Yeh, University of Maryland

AIR FORCE LIAISON REPRESENTATIVES

Samuel Dinitto, RADC/COE, Griffiss Air Force Base, NY
Maj William R. Price, AFOSR/NM, Bolling Air Force Base, D.C.
Col James Riley, HQ, AFSC/DL, Andrews Air Force Base, D.C.
Maj Edward Stevens, HQ, AFSC/ALR, Andrews Air Force Base, D.C.

METHODS FOR IMPROVING SOFTWARE QUALITY AND LIFE CYCLE COST
1983 SUMMER STUDY

STATEMENT OF TASK

In spite of numerous past research and development endeavors related to software engineering tools and methodologies, the Air Force continues to experience problems in obtaining required performance, quality, and productivity while controlling cost and schedules of large software based systems. The solution to this problem does not reside in continued proliferation of fragmented software engineering elements but rather in the development of an integrated and comprehensive standardized software engineering environment that will support the software life cycle from user requirements through operations and maintenance.

Many of the tools and techniques required to construct such an environment now exist in a suitable form. Others will have to be modified or updated and still others must be developed. A comprehensive and consistent integrated support system is necessary to improve performance, quality, cost, and schedule to control O&M costs for error correction and product improvement.

A software engineering environment consisting of automated tools for design, development, test and validation; specification, design and implementation languages; predictive models for cost, schedule and reliability estimation; standards for documentation and quality control; and management tools and procedures derived from the software engineering structure as opposed to a force fit should have a dramatic positive impact on the "continuing software crises."

The Air Force Studies Board proposes to undertake a study that will update, in part, the AFSB 1976 Summer Study on operational software management and development. The study will characterize a comprehensive and integrated standardized software engineering environment. Identification of tools, techniques, standards, models, and metrics will be accomplished for each and every phase of the total software life cycle. An assessment of the state of the art of all elements of the environment will identify research and development requirements for the Air Force. The result of the study will be (1) the identification of a near-term software engineering environment consisting of off-the-shelf tools and techniques, which may be implemented as is or with minor modification and (2) a time-phased R&D plan for major modification, development, and integration of all identified elements leading to a longer-term comprehensive environment.

GLOSSARY

Ada	The Department of Defense common language
AF	Air Force
AFSC/IG	Air Force Systems Command/Inspector General
AI	Artificial Intelligence
CAI	Computer Aided Instruction
CDR	Critical Design Review
CONCAP	Concept Evaluation Capability
CPDP	Computer Program Development Plan
CRISP	Computer Resources Integration Support Plan
DSMC	Defense Systems Management College
DSS	Decision Support Systems
ECP	Engineering Change Proposal
ECR	Embedded Computer Resources
ESD	Electronic Systems Division
FSED	Full Scale Engineering Development
GFE	Government Furnished Equipment
HIPO	Hierarchical Input Processing Output
HOL	Higher Order Language
IOC	Initial Operating Capability
IUS	Inertial Upper Stage
IV&V	Independent Verification and Validation
KBS	Knowledge Based System
KBSA	Knowledge Based System Assistant
LISP	Artificial Intelligence Programming Language
NORAD	North American Air Defense Command
OBJ	Programming Language (as in LISP above)
O&M	Operation and Maintenance
PDR	Preliminary Design Reviews
PM	Program Manager
PMC	Program Management Center
PMD	Program Management Directive

PMDSS	Project Management Decision Support System
PMRT	Program Management Responsibility Transfer
P^3I	Pre-planned Product Improvement
PSL/PSA	Problem Statement Language/Problem Statement Analyser
RFP	Request for Proposal
RRDC	Rapid Requirements Definition Capability
RSL	Requirements Statement Language
SAMC	Software Acquisition Management Center
SEE	Software Engineering Environment
SPADOC	Space Defense Operation Center
SPO	Special Project Office
USAF	United States Air Force
VLSI	Very large-scale integration
WIS	WWMCCS Information System
WWMCCS	World Wide Military Command Control System. A network of dozens of HIS 6000 computers providing command and control of the U.S. defense forces.

1.0 INTRODUCTION

The amount of computer software in "mission critical" military systems and its importance in achieving the desired functionality of those systems have increased dramatically in the past two decades. Further, while projections for such increases in the next decade and beyond vary widely, all forecast even more impact for software on future military systems. The current and planned systems for the U.S. Air Force are among the most, if not the most, complex and sophisticated in the world, and to a large degree the requirements on the systems have to be satisfied by the software "embedded" within them. For example, in some C^3I applications, the software amounts to 80 percent of the total development effort, and new fighter aircraft are "flying" software measured in terms of hundreds of thousands of lines of code.

While hopes for the system capabilities to be realized through software have been high, very often the delivered product has grossly missed the mark in functionality. To make matters worse, more often than not, the software portion of the system is responsible for system delays and enormous cost growth. On occasion, these delays have been measured in years and the cost growth in multiples of the original software budget. Unfortunately, recent history has not shown a significant improvement in this situation.

Within the last 10 years, as more and more software-intensive systems have been fielded, the Department of Defense (DoD) has come to realize that software development costs are often dwarfed by support and modification (often called "maintenance") costs associated with fielded software. Depending on the useful life and the number of modifications to be made to the software, these costs can average three to five times the original development budget. For example, data exist to show that a change in the functionality made <u>after</u> rather than <u>before</u> a system is fielded can cost one or two orders of magnitude more. Thus, the DoD must be just as concerned, if not more so, with software life cycle costs as it is with those of the hardware life cycle.

The situation as described thus far motivated the Commander, Headquarters, U.S. Air Force Systems Command, to commission the Air Force Studies Board to conduct a summer study on methods for improving software quality and life cycle cost. This study was held from 11-29 July 1983 at the National Academy of Sciences' Woods Hole Study Center at Woods Hole, Massachusetts.

Since the late 1960s, expert opinion, university, industry, and military studies have placed a good measure of the blame for the poor state of software on the lack of engineering discipline applied to the development and support of software, and on the lack of suitable "environments" for enforcing and supporting a software engineering discipline. Such an environment should embody a single or possibly a set of software engineering disciplines and provide tools and techniques that improve the productivity of development and management teams alike in following a particular discipline. By providing such environments, experts and numerous studies agree that software development and support can be brought under control and become less expensive.

There is no exact definition for a "software engineering environment." For purposes of this report, the scope of such an "environment" will include:

1. Automated tools for specification, design, development, test and validation, operation, and maintenance;
2. Specification, design, and implementation languages;
3. Predictive models for cost, schedule, and reliability;
4. Standards for documentation and quality control; and
5. Management tools and procedures directed to/from the software engineering environment structure.

1.1 Steering Committee Meetings

Several meetings of the steering committee were held from January through June 1983 to:

1. Define and scope the task statement;
2. Gather and review background information on the state-of-the-art, promising but untried technology -- The U.S. Air Force, Army, and Navy plans and programs to improve their state-of-the-practice, and the Office of Secretary of Defense (OSD) plans for the $300 million program "Software Technology for Adaptable Reliable Systems (STARS)";
3. Determine the organization of the summer study; and
4. Determine the final membership of the actual study group.

It was decided that the study group would address three basic areas with regard to the goals of software life-cycle cost reduction and software quality improvement:

1. The identification of technology and management recommendations to be pursued;
2. The identification of a software engineering environment that could be fielded in the near term. The environment would consist of off-the-shelf tools and techniques that may be implemented as is or with minor modification; and
3. The identification of time-phased recommendations for major modification, development, and integration of tools and techniques leading to a longer-term, more comprehensive software engineering environment.

The committee was briefed on numerous topics, including:

1. The results of a survey on perceived needs in embedded software and technology solutions by Mr. Joseph Batz, OUSDR&E (R&AT).
2. A state-of-the-art survey of the software industry in the U.S. and Japan by Dr. Raymond Yeh, University of Maryland.
3. A strategy for a knowledged-based software assistant by Dr. C. Cordell Green, Kestrel Institute.
4. Several briefings on the background and status of the OSD STARS program, by Mr. Samuel DiNitto, Rome Air Development Center.
5. A survey of problem indicators gathered from actual U.S. Air Force Systems Acquisitions by Col. Edward T. Akerlund, HQ, Air Force Systems Command.
6. Actual experience with state-of-the-art software engineering techniques and tools on the U.S. Air Force's Data Systems Modernization Program by Mr. Anthony Jordano, IBM.
7. The development, specifics, and use of a common, modern, corporate software engineering environment by Dr. Samuel Steppel and Mr. Peter Belford, CSC.
8. An integrated software support system in use at the U.S. Air Force Wright Aeronautical Laboratories (AFWAL) by Maj. Israel Caro, AFWAL.
9. The U.S. Army's plans for Post Deployment Software Support (PDSS) Centers, including Ada* and the MIL-STD 1862 computer architecture, by Mr. James Hess, USA/DARCOM.
10. The U.S. Navy's experience with the support of the CMS-2 programming language and their plans for promulgating and supporting Ada by Mr. Owen McOmber, NAVMAT 08Y.
11. The state-of-the-practice in software engineering at an air logistics center by Mr. Alton Patterson, Sacramento Air Logistics Center.
12. The state-of-the-practice within the U.S. Air Force Systems Command's Product Divisions by those Product Divisions Computer Resource focal points.
13. Recommendations on future directions that the U.S. Air Force should take with regard to software by Dr. Edwin Stear, former Chief Scientist of the U.S. Air Force.

*Ada -- The Department of Defense common language. During the 1983 Summer Study several of the briefers discussed Ada. This is a modern high-order computer programming language which will become the standard language for writing software for DoD embedded computer applications.

1.2 Study Organization and Membership

It was decided that the study would be organized as three components concerned with (1) the management of software development and support, (2) techniques to be employed to improve the productivity, quality, and management visibility with regard to software, and (3) tools that would automate those techniques to help institutionalize them and further enhance productivity, quality, and visibility. The responsibility for addressing these components was given to three subcommittees with the same names, i.e., the Management, Techniques, and Tools subcommittees.

The subcommittees would be charged to study their respective components with a view toward implementing any recommendations within a software environment(s) that would be realizable and properly engineered within three time frames. Each subcommittees was to focus on a near-term environment which could be fielded in two to four years; a mid-term environment to be fielded in four to six years; and a far-term environment geared to future as well as present needs, to be fielded and continually enhanced within six to fifteen years. In addition, the subcommitteess were asked to update the Air Force Studies Board's 1976 Summer Study on Operational Software Management and Development[1].

The final membership of the 1983 Summer Study on Methods for Improving Software Quality and Life Cycle Cost is depicted on page iii. In rounding out their respective subcommittees (page v), the subcommittee chairmen attempted to achieve a proper balance with respect to academia, the software and defense systems industry, and the government. The Techniques Subcommittee membership was heavy in academic personnel and software industry personnel concerned with advancing the state-of-the-art. The Tools Subcommittee generally was composed of individuals with a history of developing and/or using software tools. Finally, the Management Subcommittee was heavily populated with senior managers with extensive experience in the acquisition and development of large, software-intensive military or military-related systems.

In addition to the "permanent" subcommittee members, each group was augmented with "part-time" individuals who served for a few hours up to a full week. These people provided formal and informal presentations on the state-of-the-practice, state-of-the art, and proposals for future systems. Often, they served as sounding boards for ideas and as representatives of the U.S. Air Force and the military industrial community. In all those roles, they were invaluable and indispensable. These individuals and their affiliations are on page iv.

2.0 SUMMARY OF RECOMMENDATIONS

Detailed recommendations are presented in succeeding sections. However, the major recommendations and some of the supporting rationale are provided at this point to serve as an executive summary.

It is the contention of the study group that gains in software productivity and quality, achievable with off-the-shelf tools and techniques, are not ensured. In other words, in many, if not most cases, the U.S. Air Force is not benefiting to the maximum extent possible from existing technology. The reasons for this situation are unawareness of the technology on the part of the U.S. Air Force and its contractors, poor engineering (usually "human" engineering) of the tools, and a lack of the kind of vendor support needed to use the tools in serious engineering of systems software. To correct the situation, the study group recommended that these actions be undertaken:

1. The U.S. Air Force "engineer" several first generation environments that utilize compatible, proven techniques and tools.
2. Training, distribution, and maintenance schemes be devised for the environments.
3. Generic, functional specifications for the environments be developed to serve as a "minimum" standard, and enforced as such in all software acquisitions.

To move beyond what the current state-of-the-art will provide in terms of productivity and quality, the group recommended that the U.S. Air Force create a second generation of software engineering environments. The second generation should be designed from the top down, with the complete integration of the methodology, tools, and techniques as a primary objective, rather than a forced fit as in the first generation environment(s). The environment must be comprehensive in that it must support the entire life cycle model. The first generation environment(s) could be hampered by a scarcity of proven tools in all phases except the code and unit test. This lack of a comprehensive and integrated set of tools will likely hinder communication and management oversight across life cycle phases.

To achieve the desired completeness in the tool set and support the integration of the tools across the entire lifecycle, the following must be undertaken:

1. An intense concentration on the software requirements and specification phases through better languages and analysis and validation tools.
2. Increased R&D in computer-aided test case design and analysis.

3. The use of intelligent data base management systems (DBMS) within the environment to help manage and integrate the huge amount of software engineering data that will be generated by the new tools.
4. Employment of a concept known as the "distributed software engineering work station" to allow acquisition and project managers, systems analysts, and software engineers to communicate and carry out their own tasks from a common baseline.

The study group generally agreed that the ever increasing complexity (in terms of demands on performance, reliability, flexibility, and deployment schemes) will eventually render even the second generation environments inadequate to manage, develop, and support the software needed to handle future complexity. At least an order of magnitude improvement over current productivity and quality will be required of a third generation environment. Significant advances and in some cases breakthroughs in the technology base will be necessary to cover such ground.

The application of artificial intelligence techniques, specifically knowledge-based expert systems, to increase the automation of the specification, design, implementation, and testing and validation processes is seen as a strong contender to achieve at least an order of magnitude improvement. The study group advocates the pursuit of a knowledge-based software assistant (KBSA) by the U.S. Air Force.

The study group also explored alternate development life cycle models to help deal with the forecast complexity. One alternative proposed is evolutionary system development with working, useful incremental system deliveries. This is in contrast to the "everything at once" philosophy that pervades the current systems acquisition process. Such an alternative requires more "up-front" work by the customer, acquisition element, and systems contractor, but its aim is to break the whole into simpler deliveries. This would give the U.S. Air Force a better understanding of the contractor's progress, the U.S. Air Force and contractor both a better understanding of the ultimate system, and serve to improve the chances of some operational capability being delivered to the government even if program difficulties are encountered. Prototyping and rapid prototyping are technologies worthy of investigation in support of this concept.

The management of software in the acquisition, development, and operations and maintenance (O&M) aspects of a system leaves much to be desired. This stems primarily from a lack of knowledgeable people as well as technical and policy support for software systems management. To overcome these problems, the group recommended the following:

1. Center(s) of expertise in software acquisition and management should be created. The number of such centers was left unspecified, but two possibilities are (a) for the U.S. Air Force Systems Command (AFSC), or (b) per product division and/or logistics center.
2. The U.S. Air Force should develop, prepare for and use alternate acquisition strategies. Prototyping is a strategy to establish system requirements that has proven its worth in industry, and it should be the rule for acquiring systems rather than the exception.
3. The U.S. Air Force must provide for the immediate development and follow-up support for software management tools to assist in costing and scheduling, user/System Project-Office/contractor communication, and quality prediction and assessment.
4. Better training and incentives must be provided for both management and software personnel. One bad example of the present situation, which the group was assured was already under correction, was the omission of bonuses for extending service time in computer-related U.S. Air Force military occupations while those in other engineering fields were rewarded for extending their service time.
5. There should be stronger and better selection procedures for software managers within the U.S. Air Force, as well as the military-industrial community. The U.S. Air Force may spend hundreds of thousands of dollars training and screening a pilot or a specialized mechanic, but may select software acquisition managers for multimillion-dollar systems with no related software experience.

With regard to system architectures employed or planned to be employed in U.S. Air Force mission critical systems, the group came to the conclusions that:

1. Present computer architectures, including the MIL-STD 1750-A, are marginal or inadequate for many applications, especially future ones in C^3I and even avionics; and
2. Due to a lack of tools and techniques to support software engineering for nontraditional architectures such as distributed systems, multilevel secure systems, direct-execution machines, highly parallel architecture, and artificial intelligence oriented architectures, the U.S. Air Force will not be able to incorporate the full advantages of the enhanced performance of these architectures into its systems.

For the near term, the group recommends that the U.S. Air Force enhance its evaluation of the NEBULA (MIL-STD 1862) instruction set architecture. Compared with the MIL-STD 1750A, the MIL-STD 1862 computer has a larger word size (32 bits), a larger address space (2^{32} vs. 2^{16} addressable bytes), and is much more supportive of Ada.

The cost of the development and support for this architecture is being borne by the U.S. Army, so this is a relatively inexpensive alternative.

The U.S. Air Force should prepare for the future by initiating and, in some cases, increasing its R&D in support of the use of the nontraditional architectures. Most immediately, it needs to augment the existing tool and technique set with aids for analyzing, designing, testing and evaluating distributed systems. Such systems are already being built or are proposed to be built to meet current performance requirements.

It is generally agreed that one way to improve software productivity is to reuse as much software as possible. However, there is disagreement as to the proper strategy for reusability. For instance, some experts say the concentration should be on reusability at the lowest, or module level, while others feel that reusability should be considered at the higher levels of specification and design. Still others claim that in the near-term a more certain payoff is to be achieved by reusing generic and application-specific software support tools such as source language level debuggers and simulation tools.

After much debate, the group came to recommend the following order of priority for concentration of resources for software reusability.

1. Reusable support tools, <u>within</u> all three generations of environments;
2. Reusable "models" for specifications and design of application-specific functions;
3. Reusable major subsystems such as message handlers, communicator protocol processors, etc; and
4. Reusable Ada packages for "mission software," where possible.

2.1 Review of the 1976 Summer Study

In 1976, the Air Force Studies Board (AFSB) devoted a summer study to "Operational Software Management and Development for U.S. Air Force Computer Systems." That study made seven major recommendations. This section addresses the relationships of the 1983 Summer Study on Methods for Improving Software Quality and Life Cycle Cost to the 1976 recommendations.

1. 1976 Recommendation: "The development contract should be preceded by a separate contract for detailed system design requirements and no commitment be made to development until the design requirements are completed. The detailed system design requirements

work may often be performed usefully by competing contractors on a level-of-effort contract." This recommendation resulted from the finding that often the U.S. Air Force committed to dollars and schedules in a system development before the interrelationships and functionality of the system were fully understood.

The 1983 Summer Study group supported this recommendation by once again advocating more frequent use of prototyping as an acquisition strategy; the advancement of technologies for rapid prototyping and communication of requirements among the customer, program office, and contractor(s); and supported competitive contract definition phase acquisition.

2. 1976 Recommendation: "Whenever possible, the development should proceed through a series of deliveries. Each should stand by itself, not require a rework of its predecessors, and each should be of tangible use. During this period, strong feedback is needed from the using organization and the maintaining organization to the development activity, in order to correct any recognized problems." This recommendation was motivated by the experience that during development too much was often attempted at once.

The 1983 Summer Study group fully endorses this acquisition philosophy and encourages the U.S. Air Force to employ it. The study group was not aware of many cases where it had been tried as an initial strategy but many where it was forced due to schedule slips.

3. 1976 Recommendation: "The System Program Office, especially during the detailed system design requirements phase, should be (a) responsible also for system engineering, (b) empowered to make tradeoffs within the system to optimize it, and (c) charged with balanced consideration of the systems development and life cycle. To improve the life cycle aspects, the using and the maintaining organizations should participate formally and continually in both the detailed system design requirements and the development phases as part of the SPO." This recommendation reflected the always potential conflict between the optimization of the systems development and total life cycle considerations.

The 1983 Summer Study group had similar concerns and made several recommendations dealing with designing for ease of modification: provision of the same software engineering environment for operations and maintenance as for development; involvement of eventual users and systems supporters in all life cycle considerations; development of comprehensive program data bases and their transfer to the using and supporting commands; and participation of using and support as well as developing organizations in establishing milestone success criteria.

4. 1976 Recommendation: "The USAF continues (SIC) to advance the state of software tools through continual research, development contracts, and support of and coordination with the DoD Software Management Steering Committee in its efforts toward standardization of high order languages." This recommendation recognized the contribution tools had made available to software productivity and quality and the potential of proposed tools and those already under development to do the same.

The 1983 Summer Study group concluded the U.S. Air Force was not taking full advantage of existing proven tools and suggested a near-term software engineering environment for achieving and ensuring the state-of-the-art in productivity and quality. Further, it recommended enhanced R&D in advanced tools and techniques, and their proper integration into second and third generation environments if the U.S. Air Force is to meet mid- and far-term requirements for productivity and quality. The group also recommended the creation of software centers at the product division.

5. 1976 Recommendation: "To establish continuity and accountability, the period of assignment of SPO officers should be related to SPO activities. For junior officers, the assignment should span the entire job, regardless of interim promotion. To help such officers, the USAF needs to develop short courses on management procedures, on how large computer systems work, and on management case histories." The point here concerned a lack of knowledgeable software managers and a lack of coordination of assignments and reassignments with the SPO's mission requirements.

The 1983 Summer Study group discussed the SPO assignment issue but did not make a specific recommendation. However, there are extensive recommendations on the identification, training, and retention of knowledgeable software managers. The summer study group fully recognizes the contribution personnel make to the success or failure of a mission critical system. It further recommended the creation of a special organization(s) to assist in software acquisition and support.

6. 1976 Recommendation: "The USAF maintains a vigorous program to improve the technology base. Among the topics needing attention:

a. The structure of, and quantitative measurement of, the development process.
b. Extending present software development tools and adding new ones, such as validation of requirements and design.
c. Computer hardware and software architectures to improve system performance and to extend life-cycle utility.
d. The applications of mini- and micro-computers, including problems of logistics, reliability, and maintenance, as well as function and performance.

e. Data communications, input and output methods, and means for dealing with interfaces between systems."

This set of recommendations is related to the rapidly changing computer technology and the U.S. Air Force's need to cope with and effectively utilize current as well as future technology.

The 1983 Summer Study group endorsed and expanded on the first three of these recommendations by:

a. Advocating the development of technology to provide better quantitative measures of software quality throughout the life cycle, and recommending the investigation of alternate life cycle models to deal with the application of emerging technology such as artificial intelligence;
b. Fostering increased R&D in technology for software requirements, specification, and design, as well as tool design and analysis and the proper integration of these technologies into the life cycle; and
c. Strongly emphasizing the need for R&D in technologies to allow the U.S. Air Force to use existing and near-future capabilities of distributed architectures, highly parallel architectures, artificial intelligence oriented architectures, etc. as well as the need for investigating the use of the new MIL-STD 1862 computer architecture in U.S. Air Force embedded applications.

7. 1976 Recommendation (This final major recommendation is related to U.S. Air Force involvement in the DoD Software Management Committee.): "The U.S. Air Force Systems Command strengthens (SIC) its support of the participation in this committee by the USAF, providing a stronger focal point for its own activities to promote such support. The present organization, the Directorate of Computer Resources Development Policy and Planning, in AFSC, does not call for the rank, staff size, or charter that such a focal point organization requires."

Since the Summer Study group was in communication with the planners of a USAF Scientific Advisory Board study that was to address such management issues, it did not review this specific recommendation.

3.0 IMPROVEMENT OF CURRENT MANAGEMENT PRACTICES

One of the more important areas in which the U.S. Air Force needs to improve the expertise of its management personnel and the use of management tools and techniques is in the acquisition of software-intensive systems. Software systems must be more intensively managed than comparable hardware systems since these typically undergo significant changes over their life cycles. Support systems to maintain requirements, specification, and design baselines create a broad definition of the delivered system for the entire life cycle. Manufacturing support facilities for hardware generally remain with the vendor.

To acquire higher quality, more efficiently produced software, the U.S. Air Force must improve its ability to estimate and manage cost and schedules as well as assess development status. Case histories relating to cost profiles, effectiveness of cost-estimating tools, use of management metrics, and project status reporting are required. Furthermore, a uniform acquisition policy and expertise in its application are required to properly control the software project throughout its life cycle. Finally, software acquisition management, data reporting, and tools must be a prominent part of government requirements for acquisition of software-intensive systems. Evaluation of these factors in contractors' proposals must be part of contract awards for software-intensive systems.

Three principal areas are addressed to improve the acquisition of software systems. First, the process of acquisition management is examined for improvements in life cycle considerations, acquisition strategies, and the appropriate use of standard terminology and metrics. Second, a Software Acquisition Management Center is recommended to serve as a pool of expertise from which the SPO may seek advice, and to provide a mechanism by which new technology may be transferred to the acquisition process. Finally, personnel problems are discussed, with recommendations as to how to identify, train, and keep skilled SPO personnel.

3.1 Acquisition Management

3.1.1. Introduction

The requirements definition process is, in the subcommittee's judgment, inadequate for USAF system acquisition needs. Studies on the economics of software and surveys of management indicate that the highest economic leverage point and the largest management problem area is the definition of user-command requirements. Previous developments of tools and techniques for support in this area have concentrated on design and not on requirements-refinement by the using command and SPO. The present process results in misunderstanding among using command, SPO, and the contract; inconsistencies in requirements; untestable and untraceable requirements; inaccurate cost and schedule estimates; and costly pre-allocation of function, which unnecessarily constrains design and provides inadequate support for budget/schedule tradeoffs.

Inadequate attention is being given to software life cycle considerations during systems definition, development, and operation. This often results in significant delays in achieving a U.S. Air Force software-support capability, costly interim contractor operational software support, and the introduction of expensive inefficiencies into the operational software-change process.

As a result of new technology providing hardware that is smaller, more compact, and faster, the requirements for U.S. Air Force weapon systems are more complex and costly. As a consequence, systems are becoming increasingly difficult to develop and support. The same sophistication that enhances performance often creates problems with system reliability and nightmares for logistical support personnel. These systems, characterized by distributed multiprocessor architectures, depend on complex interactive software to integrate numerous hardware devices. System complexity can reasonably be expected to increase, with the demands on software engineering certain to follow.

3.1.2 Issues

The present SPO requirements definition process is inadequate to support clear and consistent communications of requirements, accurate and complete statements of needs, useful identification of priorities, and quickly available data for system analysis in response to changing budget and schedule demands. Such support is required in assessing the continued impact of changes in requirements throughout development and life cycle maintenance. Without it, original functionality, schedule, cost, and quality might be compromised.

Insufficient planning -- particularly with respect to system engineering -- is accomplished prior to undertaking software-intensive system acquisitions. This shortcoming leads to system-level decisions that present severe difficulties to software acquisition, and often results in costly overruns, scheduling delays, and unsatisfactory system capabilities.

One aspect of poor system planning is improper allocation of computer hardware. Reviews of specific projects by the panel indicated that planned hardware reserves (usually 50 percent) are as a matter of practice not provided. The typical growth as software requirements become better understood forces the undesirable consequences of assembly language instead of approved HOL's, and a large amount of optimization, which results in the compromise of the structural integrity of the software.

Good strategies for acquiring complex software systems are often overlooked due to misconceptions concerning the regulations applicable to development of a pure hardware system.

Intense management of software product baselines is necessary. One factor that has a significant effect on software development and life cycle costs is the evolutionary change that each major system undergoes to upgrade its capabilities and extend its useful life. The characteristic flexibility of software makes it convenient to accomplish changes of capability and tactical doctrine. Making changes of similar functional capability in hardware would require numerous modifications of computer hardware, sensors, or actuators in aircraft or other computer-based equipment -- at considerable labor and expense -- with an unacceptable amount of system down-time.

System evolution is not the only reason for software modifications. Throughout the development process, changing requirements -- whether essential, misunderstood initial requirements, or indisputable deficiencies -- result in numerous changes to the software. During the operational phase, changes might result from detected errors, tactical changes, or a requirement for performance improvements.

For present U.S. Air Force software-intensive systems, a poor job is done of developing software requirements and designing application software to accommodate and expedite the numerous changes that an operational system must be expected to undergo. Good planning for the software life cycle, design of systems and software to accommodate life cycle requirements, and early acquisition of all necessary support resources during program development are essential to reduce software modification costs and inefficiencies as well as improve the life cycle management capability.

Inadequate standards hamper development. The software acquisition process of the U.S. Air Force is hampered by inadequate, ambiguous, and outdated standards and terminology. Largely because of this, communication within the software community -- and between managers of software development in particular -- suffers. In the course of the AFSB summer study, briefers invariably volunteered or agreed on the communication problem. The origin of the problem probably lies in the fact that software development began as a "cottage industry" at numerous places and times, when the jobs that were undertaken were sufficiently small such that a "family," or a few people, could accomplish the work. Driven by the expanding capabilities of computer hardware, software designs have grown larger and more complex, demanding cooperation between members of extended "families." Lacking a common language and shared measuring tools, these individuals -- like the builders of the Tower of Babel -- found themselves in chaos. Software development has entered an era in which standard terminology and metrics must be developed, promulgated, and enforced.

The life cycle of software is in some ways quite similar to that of hardware. System requirements must be allocated to one just as to the other, and although the technology and methodologies differ, both must go through a definition, design, and development phase that requires similar analysis and test considerations. A distinguishing factor for software is that its maintenance phase during systems operation is essentially continuous, repeating the steps used in initial system development, but with the additional experience of an operational requirements baseline.

This reiteration of development actually allows application of a consistent discipline for software changes. Although such a discipline is rarely seen for operational system evolution, it is attempted by USAF Air Logistics Centers once they finally acquire their software support systems. The Joint Logistics Commanders are also actively pursuing a standard software life cycle.

Given that, throughout a software life cycle, a system undergoes numerous changes through reiterative development, two major aspects of software need to be considered in improving software life cycle management and costs. First, the design of the software must be such that expected modifications can be implemented efficiently and effectively. This means that requirements for the anticipated system and its software growth, with expected changes, must be included in the initial system definition and software design. Concerns such as documentation, software modularity, built-in module testability, interface control, and global data bases make software easy to understand and easier to modify by people other than the development contractor. Second, the resources necessary to modify the software and to test changes must be available when needed. This capability is essential when transferring a developed system to a supporting organization, such as U.S. Air Force Logistics Command, that is to assume total responsibility for software support.

Over the past decade significant strides have been made in structuring software design. These are: use of structured programming, top-down design, higher-order languages, modularity, information hiding designed in memory, and processing reserve. The use of these tools, however, should be enforced more strictly. Too often, under threat of delays or promises of lower costs, a program office allows a contractor to use existing unstructured code or simply waives good software design practices. These alleged avoided costs and/or delays are rarely weighed against possible adverse impact on the total life cycle.

First of all, contractors must be pressed to follow good software development practices. This can be done by explicitly specifying the techniques desired in the Request For Proposal (RFP) and providing incentives for excelling in the human engineering of software designs. Software design and development requirements must be specified by the government, rigidly adhered to, and consistently applied to system development. Once the government goals for software development techniques are identified, the contractor must identify the specific methodology he will use to satisfy them. This will be accomplished by the Computer Program Development Plan (CPDP). The proposed CPDP, the assessment of contractor internal software development standards if they differ from the CPDP, and the contractor's past performance in similar software development efforts should be considered when assigning contracts.

One U.S. Air Force requirement that is often identified after development begins is the operational software support concept. Operational support varies from few changes for stable software systems to large changes on a cyclic basis for critical defense systems. Such systems might be supported by a U.S. Air Force organization, the original development contractor, the winner of a competition among contractors, or a contractor/government solution. Each of these support methods might require different levels of documentation and resources (manpower, facilities, equipment, etc.). Software design can particularly enhance supportability through any of these methods if addressed early enough in planning. The operational support concept must be identified for contractor RFP evaluation.

Requirement changes adversely affect life cycle. Many requirement changes that are imposed on a system do not address total life cycle impact. For example, a typical engineering change proposal (ECP) submitted during the development phase will have a cost associated with design as well as coding times for the actual software change and required descriptive documentation changes. But seldom does assessment of an ECP include its effects on future support factors such as added complexity, hindered future modifiability, needed special knowledge, additional required verification, or effects on other software components. Except to correct mission-critical deficiencies, changes should only be allowed in phased blocks, and all changes should be assessed in depth for possible life cycle effect.

Finally, for those instances in which sufficient detail is available to specify software support requirements but a conscious decision is made not to implement -- or delay implementation of -- those requirements, the decision must include a software, as well as a system, life cycle impact assessment. Further, the decision should be made jointly by SPO oversight management and the using and support organizations, not by the SPO management alone.

Software support resources are the first to be cut back. Program documentation and resources for support are the first victims of dollar shortages during development. Delayed planning, design, and acquisition of the resources required to support the operational software can result only in higher support cost and delays in U.S. Air Force operational support. It is essential that the U.S. Air Force software support requirements be included as part of the contractor's software development environment. This serves two purposes: (1) the contractor will be required to assess and build his internal capability in light of his understanding of the U.S. Air Force operational support requirement, and (2) the government will not receive a poor collection of tools that were not used by the contractor's software development engineers.

Finally, the software support environment should have fully verified and documented tools. This facility may be used by a variety of contractors or government personnel; therefore, it must be self-contained and efficient. Assessment of this delivered support environment by the supporting agency should be included as a part of the development program. Although the U.S. Air Force Operational Test and Evaluation Center has developed a questionnaire and methodology for evaluating software support environments, such environments are rarely available for evaluation on new developments until well after initial operational capability of the system.

<u>Standard acquisition models do not fit all software molds</u>. A system acquisition strategy is defined by the phases, milestones, and decision points established; the schedule associated with them; and the contractual roles and relationships among the users, buyers, developers, and support organizations. For many acquisitions that don't fit the standard, inappropriate acquisition strategies for software cause unnecessary cost and risks to the entire system, with resulting costs many times that of the software. Several U.S. Air Force and DoD studies have concluded that alternate procurement strategies are needed for the software.

The traditional U.S. Air Force system acquistion follows the life cycle shown in Figure 1. At the initiation of the Full Scale Engineering Development (FSED) phase, functional requirements are assumed to be completely defined. The bulk of the software development effort is in the FSED phase. The software life cycle is defined in Figure 2. It depends on a fixed set of requirements that are then implemented through system design, code, and test activities. Initial software cost and schedule estimates as well as funding commitments for FSED are often based on one-time performance of these activities in the sequence shown.

There have been many instances of cost and schedule overruns for software development in U.S. Air Force system acquisitions. One cause of these overruns has been the use of an acquisition strategy that initiates the FSED phase with inadequate information about system requirements. This leads to poor estimates of cost and schedule prior to FSED, and changes in requirements during system development that necessitate the repetition of software development activities.

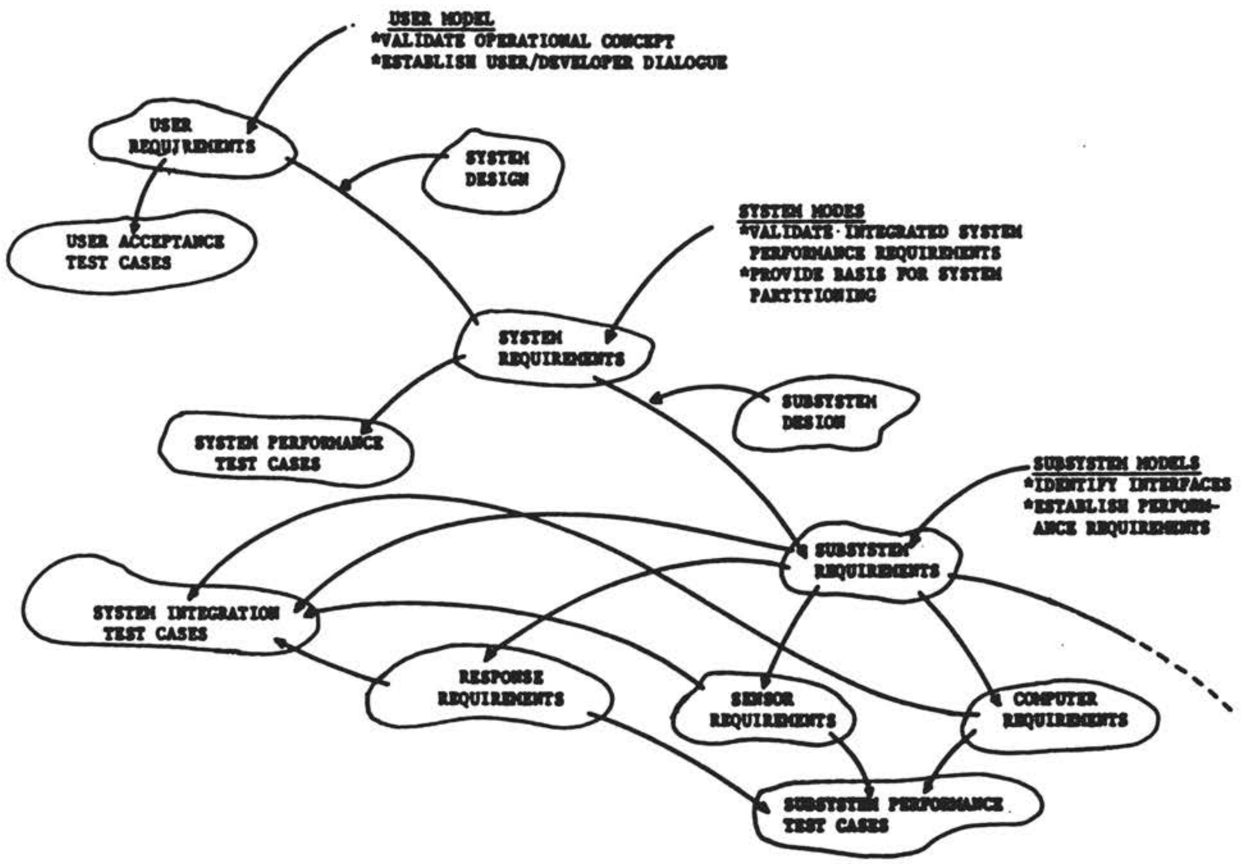
USER MODEL
*VALIDATE OPERATIONAL CONCEPT
*ESTABLISH USER/DEVELOPER DIALOGUE
USER REQUIREMENTS
SYSTEM DESIGN
USER ACCEPTANCE TEST CASES
SYSTEM MODES
*VALIDATE INTEGRATED SYSTEM PERFORMANCE REQUIREMENTS
*PROVIDE BASIS FOR SYSTEM PARTITIONING
SYSTEM REQUIREMENTS
SUBSYSTEM DESIGN
SYSTEM PERFORMANCE TEST CASES
SUBSYSTEM MODELS
*IDENTIFY INTERFACES
*ESTABLISH PERFORMANCE REQUIREMENTS
SUBSYSTEM REQUIREMENTS
SYSTEM INTEGRATION TEST CASES
RESPONSE REQUIREMENTS
SENSOR REQUIREMENTS
COMPUTER REQUIREMENTS
SUBSYSTEM PERFORMANCE TEST CASES

FIGURE 1 System acquisition life cycle for embedded systems.

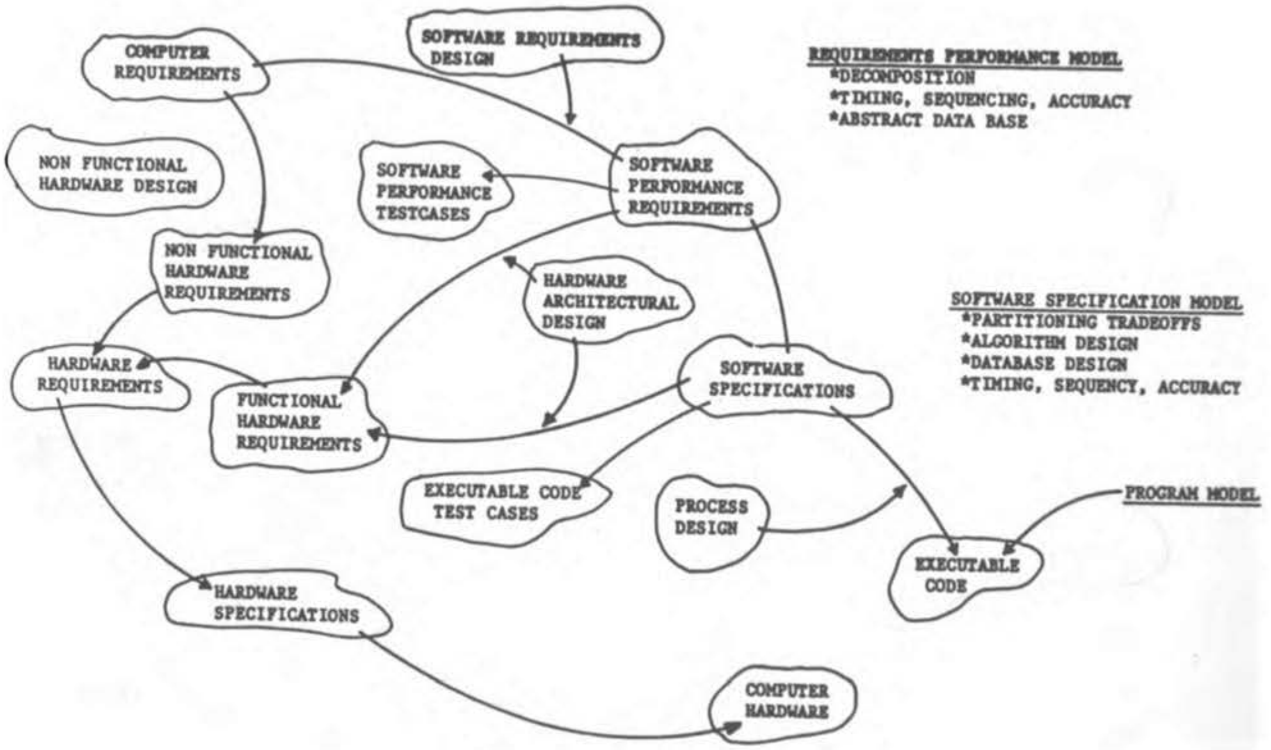
COMPUTER REQUIREMENTS
SOFTWARE REQUIREMENTS DESIGN
REQUIREMENTS PERFORMANCE MODEL
*DECOMPOSITION
*TIMING, SEQUENCING, ACCURACY
*ABSTRACT DATA BASE
NON FUNCTIONAL HARDWARE DESIGN
SOFTWARE PERFORMANCE TESTCASES
SOFTWARE PERFORMANCE REQUIREMENTS
NON FUNCTIONAL HARDWARE REQUIREMENTS
HARDWARE ARCHITECTURAL DESIGN
SOFTWARE SPECIFICATION MODEL
*PARTITIONING TRADEOFFS
*ALGORITHM DESIGN
*DATABASE DESIGN
*TIMING, SEQUENCY, ACCURACY
HARDWARE REQUIREMENTS
FUNCTIONAL HARDWARE REQUIREMENTS
SOFTWARE SPECIFICATIONS
EXECUTABLE CODE TEST CASES
PROCESS DESIGN
PROGRAM MODEL
EXECUTABLE CODE
HARDWARE SPECIFICATIONS
COMPUTER HARDWARE

Figure 2 Software life cycle for embedded systems.

Unanticipated and unnecessary cost can be avoided if the uncertainty in requirements is recognized and planned for in the acquisition strategy.

Inadequate definition of requirements can be the result of initiation of the FSED phase without completing the Concept Definition and Validation phases. For some types of systems, the difficulty in defining requirements is not one of insufficient time and preparation. Several studies, including a recent one by AFCEA[2], have noted that the requirements for Command, Control, and Communications (C^3) systems, in particular, may be difficult to determine. These systems support human decision makers with automated aids. The requirements of a C^3 system might be affected by current military policy, the preferences of particular commanders, and the functions of many other interacting systems. There might be no prior experience upon which to base requirements. Paper studies are not always adequate to determine system performance specifications; some operational experience and feedback is essential.

Even when requirements are understood, they can be large and complex, difficult to communicate to system developers and to achieve technically. A recent survey of managers involved in C^3 systems acquisitions at the Electronic Systems Division (ESD) cited such difficulties with requirements as the most frequent cause of risk in software acquisition. Choosing an appropriate acquisition strategy can considerably lessen the risks described above while providing users with better and earlier operational capabilities.

It should be noted that software is both a source of cost and schedule risk in system acquisition as well as a means for reducing risk. There is a limit to the size and level of complexity of a software effort beyond which it becomes unmanageable. Acquisition strategies that bound a system development effort are beneficial in reducing software problems.

3.1.3. Recommendations

In many situations reviewed by the Summer Study group, improving software life cycle management involved only closer adherence to established military standards and enforcement of good engineering management discipline. Many of the required policies and disciplines already exist but are just not enforced or are loosely interpreted by SPO personnel. Often these policies and disciplines are ignored for expediency or to meet cost and schedule constraints.

The following principles of sound software engineering are common to well-managed projects:

- A systematic methodology into the planning and decision process should be installed that thoroughly assesses decision impacts, documents those assessments, coordinates the effects with organizations needing that information, and then makes a conscious decision to proceed or delay knowing the full risks involved.

- Software engineering practices should be identified in the government's requirements and strictly enforced. Staffing requirements should be included in the RFP, and evaluation of the contractor's response in the CPDP should be included in contractor selection.

- Changes to system or software requirements should require a detailed software life-cycle effects assessment. This assessment should be performed by the Computer Resources Working Group that might task the supporting organization or Independent Verification and Validation (IV&V) contractor.

- Program decisions affecting life cycle considerations should not be made unilaterally by the SPO but should require close coordination with user and maintenance organizations.

Furthermore, well-managed projects require:

- Development of the Computer Resources Integrated Support Plan (CRISP) as part of the requirements definition process and included with the RFP package.

- Program Management Directive (PMD) identification of the U.S. Air Force software support concept, required IV&V, and the U.S. Air Force software support organization. The PMD will identify mandatory U.S. Air Force Logistics Command and using-command early planning support.

- Contractor delivery of a production quality support environment as a part of the contract. Near-term activity should require development and delivery no later than Program Management Responsibility Transfer (PMRT). Earlier delivery should be encouraged and rewarded. Mid-term activity should be directed toward making the contractor's development environment also the eventual operational support environment. Far-term activity should be directed toward a common or standard environment for development and support of all systems. Strong support agency involvement in planning the support environment is essential.

- Progress reports by SPO management on software development activities (e.g., assessment of program), CRISP status, IV&V status, etc., at program management reviews.

• SPO development of a comprehensive program database that will be transferred to the using and supporting commands where required and will also support development-phase management decisions. Initial definition of this program database should be made by the AFSC focal point.

• Basing of program development milestones, such as system requirements review, software requirements review (part of the new Joint Logistics Commander software development standard), and the preliminary and critical design reviews, on quantitative criteria that must be satisfactorily met before moving to the next milestone. Milestone success criteria should be developed among participating government organizations (not AFSC unilaterally) and well understood and agreed to by the contractor. Criteria must be specific and measurable, failure to meet criteria should have assigned penalties, and exceeding the metrics should have associated rewards.

The Air Force should continue leadership through the Joint Logistic Commanders in improved software military standards. The U.S. Air Force must develop, or cause to be developed, metrics or measures of effectiveness for specifying quality, costs, schedules, performance, and the like. The U.S. Air Force could make this recommendation to the Joint Logistics Commanders for standard terminology and metrics, and take the lead in their development and promulgation. In the near term, the U.S. Air Force ought to emphasize the collection of data on the measures cited, using the database to develop norms of effectiveness. Continuous collection of data would allow periodic updating of the norms.

The U.S. Air Force should make an exhaustive survey of the procedures used in industry and elsewhere for software development control. Using the information thus compiled, it should devise SPO procedures to guide the director and his software assistant in software acquisition. It is important that modern management information systems be used to implement the revised SPO procedures.

<u>Conduct high level reviews of software acquisition strategies</u>. The U.S. Air Force must consider system and software risks early in a software intensive program and choose appropriate acquisition strategies to reduce any high risks that are identified. To ensure that this happens, a high-level review, by knowledgeable people, of major software-intensive programs should be conducted early in the life of the system. The proposed Software Acquisition Management Center should have oversight of acquisition strategies.

<u>Encourage use of long lead prototypes for difficult software</u>. In system acquisitions, it has been customary to identify critical long-lead items, the production of which must be started early enough to avoid delaying delivery of the entire system. In some systems, soft-

ware might be the long-lead item. While the current policies and regulations do not preclude use of prototypes and early initiation of software development, this practice is not encouraged and is used infrequently. U.S. Air Force policies and procedures should encourage these strategies. Sufficient time should be allocated at the outset to carry out such strategies prior to initiating FSED. The up-front additional cost could average less than 5 percent of total acquisition costs, while the potential offsetting savings are enormous.

Provide system definition and evaluation facilities. If the USAF encourages the use of prototyping, there should be facilities readily available for prototyping and evaluating alternative system concepts in terms of operational capability, technical risk, and cost. Facilities may exist in several locations so that facility capabilities are accessible to users and planners during the Concept Definition and Validation phases. A common set of capabilities might be defined and implemented for a specific application area. The AFCEA report[2] identifies a Rapid Requirements Definition Capability (RRDC) for this purpose. Rome Air Development Center (RADC) has initiated a study of rapid prototyping, and is developing a C^3 Simulation Capability. Similar facilities might be used to evaluate a system during FSED and to study the feasibility of proposed modifications. The WIS program (WWMCCS Information System) establishing a Development and Evaluation Facility, the initial role of which is to permit the test and evaluation of software. It could evolve into a prototyping capability as well.

Encourage the reusability of software. The U.S. Air Force should exploit the potential for reusing software in mission-critical systems. Major application areas should be studied for commonly used, separable functional capabilities, and generic specifications should be developed for these functions. Experiments should be performed to apply current techniques, such as application generators, for generating software from specifications. Such techniques are used in some commercial applications with success.[3] The WWMCCS Information System (WIS) is using this approach for a common user subsystem. Program offices should be given incentives to encourage the use of existing military and commercial software for major portions of system components. Incentives should be provided to software designers as well as program managers to modify requirements, where possible, in order to make greater use of off-the-shelf commercial or military software or to permit joint development of software for more than one system.

Alternative acquisition models can reduce risk. Acquisition risk of software can be reduced by providing tools such as prototypes and simulations that allow preliminary investigations of system requirements issues before commitments are made to unrealistic or unachievable capabilities, schedules, and costs. Properly designed software can help to adapt systems performance as requirements change or become better understood.

A number of innovative risk-reducing system acquisition approaches have been recommended in recent studies and reports. Some of these approaches are being used in U.S. Air Force system acquisitions with increasing regularity. Roberts[4] describes acquisition strategies appropriate to C^3 systems.

The following are brief summaries of acquisition strategies that should be considered for risk reduction in software acquisition:

Prototyping. Rapid prototyping, mockups and simulations can be used prior to FSED to experiment with human interfaces, processing algorithms, performance prediction, software sizing, and other aspects of a system. Working prototypes can be evaluated in operational exercises. These approaches provide information to improve the decisions about system requirements and cost.

Incremental funding and delivery of software. In order to specify, develop, and deliver increments of system capability, budgeting and acquisition policies and procedures should be modified so that this approach is recognized and encouraged. It is now possible but not always easy to fund partial developments without some assurance that there will be funding for the total system. This strategy plans for but does not deliver a complete system capability all at once. A series of system increments are defined, acquired, delivered, and used. Each increment should be smaller and more manageable than the "all-up" system and deliverable in considerably less time. Such an approach can provide users with an earlier operational capability, allow for growth of experience by both users and developers, and accommodate changes in requirements based on user feedback as well as changes in the operational environment. This approach is particularly suitable when there is uncertainty about initial requirements. It permits containment of costs while a limited operational capability is tested and evaluated under realistic conditions. Unlike a prototype, the delivered system is expected to be used and maintained. Pre-Planned Product Improvement (P^3I) and Evolutionary Acquisition are two forms of this strategy. The AFCEA study[1] details the evolutionary acquisition approach.

Multi-phase acquisition. Separate and sometimes competitive contracts can be given for portions of the system acquisition, such as the requirements definition phase or system design, independent of the remaining implementation. Such an approach places more emphasis on the early phases and provides better information for cost estimation and software system sizing. It also provides intermediate decision points where the cost of the system and its capabilities can be renegotiated more easily, based on the added knowledge, than in a single-step FSED. ESD's SPADOC program recently completed a competitive Requirements Segment in which there were demonstrations, performance modeling, and trade studies leading to an A Specification, B Specifications, management plans, and an operational concept.

3.1.4 Cost/Benefit

The cost of implementing a standards review would be minimal. It is estimated that a DoD-wide team of perhaps eight people, working part time for a year could accomplish the task. The team should be sponsored by the Joint Logistics Commanders. When its task is complete, its recommendations should be adopted by the AFSC and its SPOs and appropriate contractors.

Adequate study of industry practices in software development control should take about one year and cost about $1 million, including service charges by industry. Conclusions reached by the study group could be implemented experimentally in a medium-sized program for about $500,000, the one-time cost of a stand-alone computer and peripherals. The SPO manning table should not be reduced in expectation of economics. The computer, with its procedures and data base, should become dedicated and pass on after IOC to the O&M agency. When these procedures are generally used in U.S. Air Force programs, the cost of the experiment should apply for each application.

The benefits of establishing standard terminology and metrics is difficult to estimate. But to the degree that miscommunication and inadequate management metrics cause cost and schedule overruns in the tens of percent, there is potential for savings. Standard terminology and metrics coupled with robust management procedures could save hundreds of millions of dollars yearly until the acquisition system has been overhauled.

The benefits of automated program control were mentioned in the discussion of standard terminology and metrics. A rough estimate was hundreds of millions of dollars per year until the acquisition process has been overhauled. In addition, incalculable benefits would accrue in dealing with Congress and in terms of the confidence level of top U.S. Air Force commanders.

3.1.5 Risk

Implementation of standard terminology and metrics has a low risk. It might be difficult to induce contractors to adopt standards different from their own, but this problem is not new, has been solved before, and can be solved again. It should be emphasized that the terminology and metrics used should be reviewed to keep pace with changing technology.

3.2 Software Acquisition Management Center

3.2.1 Introduction

To acquire software that is higher in quality and more efficiently produced, the U.S. Air Force must improve its ability to manage and estimate costs and schedules and to assess development status. Further, to use the tools effectively and interpret software project status, personnel management expertise and a uniform acquisition policy are required. Finally, software acquisition management, data reporting, and tools must be evaluated continuously and changed as necessary to keep pace with advancing technology. The issues contained herein and the recommendations to AFSC conclude that in addition to significant upgrading of acquisition management tools, a Software Acquisition Management Center must be established to properly address the technology integration of existing commercial products and also to assist SPO managers in the use of such tools and methods.

The mission functions assigned to this center will also support the development of software standards and their promulgation to SPO and other software managers, and the technical oversight and expertise that such a group would provide.

The Summer Study group concluded that software management personnel, although managing a highly complex software-intensive product, are poor users of computer automation to support their own decision making. Most notable is the limited use of automated tools such as software cost and schedule models, risk assessment, manpower loading assessment trade-off models, specification generation systems from project databases, and a general type of management support called decision support systems. A Project Management Decision Support System is recommended to integrate toolsets into a single database and provide a human-machine interface to a database management system, a dialogue graphics system, and an integrated set of models that can be used by the software manager.

3.2.2 Issues

Inadequate project planning prior to development of the Computer Resources Interface Support Plan (CRISP) results in severe project difficulties during development and fielding of software-dominant systems. The U.S. Air Force must increase the use of automation in supporting project management decision making for complex, software-intensive systems. This issue was stated in Section 3.1, Acquisition Management, but is restated here for emphasis. Managers with knowledge of the technical details of software-intensive systems who also have experience in management of such systems are in short supply. This seriously aggravates the problems of supporting the senior SPO manager during the formulation phase of a project and also in oversight and status assessment during contract performance. At critical junctions such as key design reviews, greater expertise and specialized tools and techniques are sorely needed to properly manage the levels of detail and to resolve problems that arise from in-depth reviews. This issue will also be treated in the next section, Personnel. It has been stated in this section because it supports the need for a Project Manager Decision Support System as a vehicle to gain productivity increases. It also supports the need for establishment of a Software Acquisition Management Center to provide expertise to SPO and other software managers.

Inadequate tools, procedures and data are chronic circumstances inhibiting project management personnel from performing competent management of software-intensive systems. Specific areas in which inadequate tools are used in projects reviewed by the Summer Study group include the following:

- Requirements collection and analysis tools for the "front-end" portion of acquisition management. Such tools and databases serve as an excellent vehicle for maintaining a baseline, as well as disseminating requirements information to other agencies and users;
- Cost, schedule, and risk assessment tools for improved analysis of important project parameters;
- Cost and schedule estimation tools and methods to provide more accurate assessments of completion costs and schedule; and
- Project Management Decision Support Systems that use available technology of interactive computers, on-line databases with query languages, and models for performing the analysis and generation of information on a graphic medium for use by project personnel.

The requirements definition process and use of automated requirements collection and analysis tools, in the opinion of members of the Summer Study, are inadequate to support the U.S. Air Forces' system acquisition needs. Studies on the economics of software and surveys of management indicate that the highest economic leverage point and the largest management problem are related to poor definition of requirements at the user and SPO interface. This results from an inadequate systems engineering activity that takes the definition of using command requirements and translates it into a system which incorporates as many of the user's needs as possible while at the same time satisfying other acquisition constraints such as available technology, cost, and schedules. Previous development of tools and techniques for support in this area have concentrated on interfaces between existing requirements specifications and design activities and not on the requirements collection phase by the using command and SPO. This process results in misunderstanding among using command, SPO, and contractor personnel; inconsistencies in requirements; untestable and untraceable requirements; inaccurate cost and schedule estimates; and costly premature selection of computer hardware or inappropriate allocation of functions to computer hardware. This constrains design and implementation unnecessarily and provides inadequate support for budget/schedule/risk tradeoffs. That is, the systems engineering process, which selects major system components and develops the system requirements, must be the prime focal point for improving the acquisition process. At present this process is nonautomated and labor intensive; errors often remain undiscovered well into the design and system testing phase.

3.2.3 Recommendations

The Summer Study group recommends that the AFSC establish a Software Acquisition Management Center (SAMC) to:

- Develop, review, and impose software acquisition policies, strategies, and regulations and standards necessary in software-intensive systems;
- Direct the development and use of improved project management support tools;
- Assist the U.S. Air Force in the application of tools and management policies applicable to software systems;
- Define and collect project management historical data and design decisions and modify management models to more accurately estimate such parameters in future acquisitions; and
- Develop a Project Management Decision Support System (PMDSS), distribute this technology to user organizations, advocate the use of such automated tools, and support software managers. This PMDSS would contain models for:
    - Cost and schedule estimating, and risk assessment; cost and schedule control during contract execution; requirements collection and analysis;
    - Automatic specification generation from requirements and other project information in a project database.

Members of the Summer Study group also felt that AFSC policy should consider establishment of SPO software engineers resident at the contractor's facility to monitor and assess development of large, complex or high-risk programs using indicators that may not be quantifiable or contractually reportable.[5] The Summer Study group believes that more resident, qualified SPO software engineers and fewer formal on-site visits would give a substantially better view of software status and improve cooperation between the SPO and the software contractor.

<u>Systems requirements definition tool</u>. As part of improving the system engineering process, the Summer Study group recommends the use of a system requirements definition tool at the U.S. Air Force program management level to facilitate project management personnel and user organization communications, to assist in the requirements analysis function, and to help control changing requirements throughout the entire life cycle. The present SPO requirements definition process is inadequate to support clear and consistent communication of requirements, accurate and complete statements of needs, useful

identification of priorities, and readily available data for system analysis in response to changing budget and schedule demands. Such support is required in assessing the continued impact of changes in requirements throughout development and logistics phases. Without it, original functionality, schedule, cost, and quality are always compromised as a consequence of the inability to manage the large amount of interrelated information in specifications and other documentation. Functional capability of the system requirements definition tool includes the following:

a. Static and dynamic analysis of the system requirements elements;

b. Traceability of requirements to system, subsystem, and software requirements through A, B, and C levels of specifications; and

c. Traceability of cost model data to requirements elements to assist estimating cost and schedule impacts of changes.

Cost estimating and scheduling. Initially, the Software Acquisition Management Center (SAMC) would provide experts in software cost and scheduling to assist SPO and other management personnel. These experts would select, improve, and apply existing software cost and scheduling models and tools based on their experience from participating in a number of system developments. An effort would be initiated to standardize work breakdown structures and the use of several life cycle models that would allow a common framework for gathering historical cost and schedule information. Software cost history, schedule performance data, and reporting requirements would then be recommended for use by SPO personnel.

The software cost and scheduling experts within SAMC would develop software costing and scheduling policy guidance and provide assistance to the SPO on individual programs. They would also develop and evolve new cost and schedule tools that are responsive to U.S. Air Force needs in the face of technological advances. In particular, one objective would be the development of software cost models not totally dependent upon estimated lines of code. This would permit estimation at a much earlier point in the project when lines of code are not available, a critical need for software managers.

Software status and quality assessment. The SAMC would also provide expertise to assist SPOs in the evaluation of software status. Initially, status data collection requirements could be adapted to and recommended for application on a specific project. In addition, metrics and indicators that best serve U.S. Air Force software status assessment would be defined and adapted to the particular application domain and project constraints. These would serve for evaluating performance and software quality status much earlier in the project history, as well as providing more quantitative measures of desired quality factors such as modularity, portability, and reliability.

Tools and techniques that apply empirical indicators or measures would also be developed by SAMC and they would be provided to the SPO for use in software status assessment. For example, the Summer Study group was briefed by Wolverton[6] and a user, where the following 10 leading indicators were used for management visibility.

Ten Leading Indicators

Reported	Derived
Code Production	Schedule Index
Defect Removal	Problem Index
Test Achievement	Development Hours Index
Defect Rates	Development Cost Index
Test Effectiveness	Work Performed Index

3.2.4 Costs/Benefits

Listed below are the preliminary estimates for the establishment of a SAMC, development and maintenance of a decision support system, and the key tools that integrate the system into a "manager's workbench" concept. Tools include those for software cost and schedule estimating, status assessment, cost and schedule control, and specification generation. It is necessary that the U.S. Air Force perform a more detailed cost analysis of implementing this recommendation based on the number and extent of centralization/decentralization of software experts, the specific tasks to be undertaken, and the extent of the tools and components to be developed and maintained.

Software Acquisition Management Center

Item	Cost
Staffing Project Support	$10 M/year
Project Management Decision Support System[2]	1 M/year
Software Cost and Scheduling Tool Development/Maintenance	2 M/year
Software Status Metrics	2 M/year
Requirements Definition[3] (SPO/User Level)	5 M/3 years

NOTES:
1. Total costs from the items listed in the table above are $15 million per year plus a $5 million effort over 3 years.
2. The cost of developing a Project Management Decision Support System is mainly the technology transfer of current research that is being performed in the field.
3. It should also be noted that the requirements definition tool at each installation will require approximately $400,000 for software, hardware, training, and SPO feedback. Enhancement programs should begin two years later and should be funded at a level of $2 million per year.

The benefits of a Software Acquisition Management Center are summarized as follows:

- Acquisition policies will be kept more current with rapidly evolving technology;
- A source of acquisition expertise will be available to the SPO and other software managers;
- Software cost and schedule models as well as the expertise to apply them to a specific project will be greatly improved; and
- Far better software development status information will be provided.

It has been clearly established that substantial leverage can be obtained by discovering software requirement problems early, as contrasted to correcting such problems much later during the fielding of software-intensive military systems. The benefit of early quality estimate in cost and schedule is immeasurable when compared with the present uncertainty that exists when manual methods are used. This situation demands correction through substantially improved management procedures based on quality information and valid estimates for completion.

The benefits of an automated PMDSS could be very significant compared to the manual, nonautomated methods in use at the present time. The substantial amount of information and the dynamic nature of change of this information require the use of a database system and query language. Linking the database elements to management models and providing a graphic presentation and hard copy output device in the hands of project managers will greatly accelerate the use of this automated management technology. One important advantage would be the improvement in response time required to obtain analysis of program changes compared to manual methods. This is highly advantageous for complex decision making, where answers must be generated for numerous "what if" drills. More sophisticated schedule and risk assessment models will also provide understanding when important fund and schedule changes are contemplated, or when accommodations for requirements changes that always occur in large projects are necessary.

3.2.5 Risks

It is believed that the risks associated with the Software Acquisition Management Center are low because no new technology is initially required. However, the SAMC would be expected to keep pace with evolving technology and use appropriate opportunities that become available in support of software costing, scheduling, and status assessment. Establishment of these management centers would complement current technological centers of research and would provide SPO personnel with access to sorely needed expertise during formative phases of a project. Risks inherent in these centers and the transfer of this technology to SPO personnel center primarily around organization and willingness of either SPO or SAMC personnel to pool expertise and operational know-how to formulate good management practices.

The risk of using cost estimation, schedule, and risk assessment tools is low because these tools are well established and shaken down by widespread use in industry. Cost and schedule-estimating tools are a must for managers of large or complex software projects. In industry, proposal preparers use cost-estimating tools for communication and arbitration between managers responsible for the entire system (Company PMs) and the lower work package managers who have to agree on capability, cost, and schedule for their part of the software project.

The risk of a requirements system at the project management level depends on the level of sophistication of models used to represent requirements elements and their interrelationships. The use of established methods such as data flow diagrams, Hierarchical Input Processing Output diagrams (HIPO), structured design techniques, or even the use of current requirements languages such as PSL/PSA or RSL have been shown to have considerable utility in the early phases of system design. Integration of existing tools such as data dictionary systems, structural design concepts, and the above-cited requirements languages will certainly provide considerable improvement over the use of manual techniques.

The risk in a Project Management Decision Support System depends upon the level of technology and how great a change the use of this technology represents to the user. It is demonstrably clear that personal computers with interactive Basic, Spread Sheets, simple database management systems, and other inexpensive commercial software have already given the manager a much higher level of computer-aided decision support in recent years.

The training a PM receives before a major SPO assignment will contribute to the acceptance of PMDSS, providing it addresses the needs of the manager and it is reasonably free of defects. It would be highly advantageous to work with a PMDSS in the Defense Systems Management College (DSMC), in the training that senior SPO personnel receive in automated management support. It would also be advantageous for product divisions to use the same PMDSS as that used at DSMC for management training.

3.2.6 Summary

SAMC is recommended as one solution to the severe shortage of acquisition expertise and to provide infusion of management technology into the SPO. It would be a center to assist the SPO and software managers by developing useful military standards and helping adapt the standards to suit special funding, schedule, and other system constraints. The next logical step in utilizing existing tools and techniques is their integration into a Project Management Decision Support System. Some of the existing tool sets that are available commercially are: cost and schedule estimation, risk assessment tools, and database management systems with a query language. All tool sets require calibration against historical databases of relevant project information to obtain good estimation accuracy. Interpretation for specifics of each project is also necessary since models are only guidelines and are subject to error if the wrong parameters -- number of instructions for lower-tier modules, development factors, complexity estimates, etc. -- are used.

A requirements system will greatly improve the "front-end" systems engineering process of software-intensive systems. Adoption of existing requirements languages by extending and enriching the descriptions for early phases of project definition will increase the use of these tools.

The Project Management Decision Support System will integrate tool sets into a single database and provide a human-machine interface to a database management system, a dialogue graphics system, and an integrated set of models that can be used by the software managers.

3.3 PERSONNEL

3.3.1 Introduction

We address the personnel issue in the restricted sense of software acquisition managers, not in the broad definition of "all" AFSC personnel needs. It is widely acknowledged, both within and outside the U.S. Air Force, that practitioners and managers skilled in software are in chronically short supply. Yet the U.S. Air Force and AFSC must have access to such skills. Software problems directly relate to the lack of experienced personnel in the SPO, where critical decisions are made on software-deliverable items and design reviews are conducted. Management shortfalls here lead to cost overruns, poor system performance, or difficulties in supporting the system when operationally deployed.

3.3.2 Issues

SPO management personnel -- including the director -- are not sufficiently knowledgeable in software procurement practices and technology to adequately manage the life cycle of computer dominant systems. Projects reviewed by the Summer Study group and personnel interviewed indicated that seemingly sound decisions in software management are in fact often erroneous, inappropriate, or shortsighted.

A second major issue prevalent in U.S. Air Force organizations is the extreme shortage of software experts available for managing software acquisitions. This shortage exists among both military officers and government civilian employees.

A contributing factor to inadequate and improperly motivated software managers is the poor image reflected by Air Force management toward officers and government civilian employees whose expertise is a key factor to managing a successful software project, or failure typical of examples cited below. The importance of software management skills is not uniformly reflected throughout the U.S. Air Force organization.

Difficulties often arise because proprietary software products or tools were used by the software developer or the proper life cycle support tools had not been acquired by the SPO. One distinguishing feature that separates hardware from software is the extensive changes of software over the expected life cycle of the product. This, under ideal circumstances, requires that the Air Force procure not only the fielded object (software) but the design and manufacturing support system that produced the software.

The most pervasive problems of the NORAD 427-program have been the lack of software management and poor allocation of hardware. Hardware was provided because it existed on a contract and was compatible with other hardware, not because it was judged suitable for system requirements. Specifications for the communications subsystem were too ambitious for the technical and management environments, and software did not come under positive configuration control until the operational baseline had been achieved. This situation resulted in severe problems for other operational software.

Similarly, the Joint Surveillance System program failed because of poor allocation of hardware and software. Visibility into the software development process did not exist, nor was there significant configuration management of software. The software developer was a tiered subcontractor, but the SPO had no conception of the difficulty involved in implementing software for its task in the system. Consequently, even when made aware that problems existed, the SPO could not evaluate the causes. Only when the Data Systems Evaluation center finally reviewed the program was it understood by the U.S. Air Force that software had been the problem. Regrettably, this insight came three years too late; the project was terminated.

In the short term, the AFSC can take little more than temporary measures to address the present critical personnel problem of computer resource management -- a variety of small moves that collectively, however, can make a significant improvement. In general, the approach must be to <u>find them</u>, <u>train them</u>, and <u>use them</u> to the best advantage, and <u>keep them</u>. AFSC must take the steps to identify all the software personnel assets that it holds, be they military officers, enlisted personnel, or civilian employees -- all are needed to staff project offices. Whether such identification takes the form of individual interviews, review of records for certain specialty codes, or some other techniques, the thrust is to identify the cadre of AFSC people who have the requisite software and management experience for SPO participation.

Software is acknowledged to be, in a PERT sense, "in line" to IOC availability and maintainability of weapons systems, yet it is often regarded with awe, fear, and distrust. This inhibits career opportunities and makes personnel wary of being identified as software specialists. Particularly, it becomes evident that career opportunities as a software specialist are not as promising as the other specialists.

Many of the irritants that trouble officers also affect civilian employees and drive them to other locations in government or to private industry. As a result, valuable expertise is lost, particularly concerning the application domain or project history. A stable group of civilian employees can provide valuable institutional memory.

The first imperative -- "find them" -- can be satisfied much more efficiently than at present. For example, AFSC holds a large population of 28XX engineers (Developmental Engineering), many with significant software experience. Individuals in the group should be evaluated as potential managers of software projects. Since the 51XX (Computer Systems) career field is nominally the source of computer-trained individuals, it too should be screened for individuals having management skills. In either instance, supplementing the formal education of promising individuals via the PMC course at DSMC will satisfy some of AFSC's need for software managers.

The 30XX career field (Communications and Electronics) is another population to be screened for management resources. Since both the 30XX and 51XX fields are sponsored by organizations external to AFSC, personnel transfers might involve some negotiation. However, broadened experience afforded to the persons transferred might prove beneficial to the U.S. Air Force in the long run.

Once promising individuals are found, there might be a need for supplemental training, especially for those at the rank of major and above. SPO candidates are commonly routed through the DSMC, but the present curriculum does not give an adequate foundation in software literacy. Additional computer resource training must be added to the curriculum for all SPO managers, as well as options for more specialized classes for those scheduled for assignment to project offices managing a software intensive effort. Material such as the Computer

Resource Acquisition course -- nicknamed the "Software Survival School" and presently taught as an AFSC unique initiative -- is needed on a continuing basis. Consideration should be given to requesting that DSMC offer such a class to DoD software-related personnel, military and civilian. For situations in which it might be inconvenient to be a student for three weeks, a tele-teaching class similar to the present General Acquisition Management class (now offered by the USAF Institute of Technology) would be appropriate.

The "Software Survival School" attempts to give the student some idea of how computer resources are acquired and the many pitfalls that can appear in even the best-managed project. Computer resource management requires extensive management of an abstract product. SPO people, especially the director, must be able to comprehend discussions about software and perceive the implications of decisions about software just as well as issues of cost or scheduling.

Regrettably, the career and promotion opportunities in 51XX, like its assignments, seem unattractive to officers. It may well be that it suffers from the poor image of software in general, an image held by the U.S. Air Force civilian community as well. The U.S. Air Force gives its people the impression that software and computers are not in the mainstream of action. The situation is clearly improving as evidenced by the recent establishment of the Assistant Chief of Staff/Information Systems -- a symbol of computer resource visibility at the Air Staff. But there are still many subtle indicators that not all of the old negative signals are gone. For example, the U.S. Air Force Communications Command, because of its label, is thought to be involved primarily with communications, whereas in truth it holds some of the largest computer responsibilities in the U.S. Air Force.

However well AFSC manages to marshal its software personnel or to enlarge the group through training, a shortage is still a likelihood. Thus, it is incumbent on AFSC and its product divisions to utilize such personnel effectively. For example, they can be time-shared among SPOs if no one needs a full-time resident software manager. Above all, their skills should not be wasted on incidental chores. SPO directors who have software literacy should not be diverted to projects that ignore their special talent. Software personnel must be seen as "scarce resources" to be carefully allocated. It does not follow that centralized control from AFSC is required, but it probably does follow that at a minimum the matter be focused in each product division; a Center of Expertise in each is warranted.

Finally, the skilled software engineer must be retained. Inevitably there is the "industrial pull," and there will be resignations and reassignments to nonsoftware jobs. This can be offset to some extent by assuring individuals of good assignments, by assuring opportunities for them to help improve and contribute to the U.S. Air

Force software community, by exploiting special bonus arrangements where possible, by special recognition if not special awards, and by demonstrating clearly at all levels of the Command that software specialists are respected, needed, mainstream citizens of the U.S. Air Force family. The highest motivational factor to a military officer is promotion. A clearly defined career and promotion path to at least O-8 must be made available, must be utilized, and must be advertised to attract individuals who have demonstrated management ability for software projects. The prime consideration in the promotion path should be performance as a software manager, not the holding of a specific U.S. Air Force Specialty Code.

In view of the above discussion, and the fact that the various prefixes and suffixes to identify computer-related experience have evolved over many years, it is unlikely that the current specialty within existing career fields (e.g., 51XX, 28XX, and 30XX) will be appropriate to present U.S. Air Force needs for identifying officers with computer and, in particular, software experience. Likewise, the codes can probably not assure that assignments and career progressions are properly managed.

3.3.3 Recommendations

AFSC, with the cooperation of other sponsors of specialty codes, should initiate a review of all specialty codes plus prefixes and suffixes that are used to identify individuals with computer-related experience. These should be restructured or new ones initiated as may be warranted by contemporary circumstances and the U.S. Air Force need for experienced computer acquisition and software managers.

Another recommendation to retain qualified personnel in software management is to change the view from software management expertise as a career handicap to the view that this specialty, both officer and civilian, offers good promotional opportunities, as well as a highly appreciated job position.

Specifically, the gap must be narrowed between senior SPO management and software technical personnel by:

- Establishing a career ladder up to management for technical software personnel;
- Providing software introduction and training for management;
- Creating in the short term a liaison to foster communication between management and junior technical personnel; and
- Establishing a special software manager awareness program.

There are other near-term actions that the AFSC can take to improve the effectiveness with which SPOs manage software projects. In particular, various oversight management techniques have been developed -- mostly by industrial contractors -- to monitor status and progress of software efforts. This visibility into contractor or software subcontractor organizations performing design, implementation, and testing is a critical task of the software manager. In the large, oversight techniques are pragmatic adaptations of schemes that have been used in large engineering projects. Though primitive compared to the highly automated tools that one can envision, they nonetheless can give the oversight management process significant clues about the health of a project.

While catalogs of tools for the software developer exist[7], corresponding catalogs of tools and techniques for the oversight manager do not. The needs of the oversight manager have been little addressed in the literature and by the research community. Some of the developer's tools can provide useful output data for oversight, and occasionally one will be directly useful; but for the most part, a SPO-level kit must contain different tools from that of the developer.

To this end, it is recommended that AFSC institute an effort to:

- Identify the types and details of information generally desired by and useful to the oversight manager.
- Survey experienced oversight managers, especially in industry, to ascertain what types and kinds of information each has found valuable -- or not valuable -- for oversight management.
- Survey past and present SPOs to determine what kinds of questions are normally asked at oversight reviews and identify the source(s) of data to answer them.
- Undertake a similar effort among industrial managers as well.
- Survey past/present SPOs and industrial software managers to identify tools, techniques, or tricks that each has found valuable.
- Evaluate all such information for U.S. Air Force application and finally assemble it into a "toolkit" for the SPO manager.

In addition to capitalizing on such useful schemes as may have been invented or adapted by SPOs or industrial managers, there is a major problem of infusing such knowledge into present and future SPO practices and into the attitudes, habits, and mind sets of the individuals who will be part of them. In the long run, enhanced training courses such as at DSMC, or the effect of the Software Acquisition Management Center (previously discussed) will make the essential transfer of knowledge and experience, but in the near term special steps will have to be taken.

One option is a special series of lectures and presentations addressed to relatively small groups of people. There are probably industrial lecturers who could prepare such short courses. Alternately, teaching staff might be found among the faculty of the DoD Computer Institute, the National Defense University, the Air University, or DSMC. In some way the AFSC will have to organize near-term short courses to provide the best toolkit that can be assembled from the wisdom, knowledge, and experience of industrial managers and U.S. Air Force officers.

One example of an U.S. Air Force innovation is the project database created by the Inertial Upper Stage (IUS) SPO on a small computer.[8] All SPO personnel had terminal access during the project, but as the launch date neared, terminals were provided to the contractor at his facilities and also at the launch location. In effect, the computerized database became the "authority" file that facilitated unambiguous communication, understanding, and project status and details among all participants.

3.3.4 Cost/Benefit

The additional costs for implementing recommendations for improved personnel training, retention, and motivation are very minimal compared to the direct costs for inexperienced and untrained personnel. One additional cost for expanded SPO management personnel at DSMC is estimated at $250,000 per year. Personnel training time must also be set aside for new personnel in transit to SPO offices. These costs have not been identified.

4.0 SOFTWARE ENGINEERING TOOLS

Software engineering disciplines will become increasingly critical to the accomplishment of U.S. Air Force missions as more computers of greater complexity are utilized in U.S. Air Force systems. As a result, the U.S. Air Force must concern itself with the overall life cycle cost and quality issue for computer software, seeking significant gains in the overall productivity while at the same time attaining significant increases in the delivered quality.

Software quality, in simple and direct terms, can be measured by the number of defects per line of source code. While this is a crude measure, it is an effective and revealing one. Current software is produced with defect content, when measured throughout the entire life cycle, of some 50-70 defects per thousand lines of code, KLOC. There are wide variations in this figure, as might be expected.

Productivity is measured by the average number of final product statements produced per programmer-day of effort. Productivity in this metric is currently low, lying in the range of a few tens of statements (lines) of newly produced code per programmer-day.

Overall technology gain is measured by the product of productivity gains (production rate increase) combined with quality gains (defect rate decreases). The goal is of course to expand the "productivity quality" product.

The opportunity exists today to collect existing, reasonably mature, technology from key software engineering technology areas to form an intermediate software engineering environment -- a software toolkit -- that can "fill the gap" between what is available now and what will be available in several years.

Techniques exist for keeping software engineering projects within pre-established productivity and/or quality limits, but not at acceptable costs. Less complex software may be attained with high productivity, but typically exhibits relatively low quality. Specialized software -- for example, that of the NASA Space Shuttle -- exhibits high quality, but was attained with very low productivity.

Most software engineering is currently supported by some type of software development "environment," or toolkit, that contains specialized software tools that support the production process, (i.e., requirements analysis, specification preliminary design, final design, coding, debugging, and maintenance). The product assurance activities include: requirements analysis, specification analysis, design and code inspection, static analysis, dynamic analysis, symbolic analysis, formal certification, and configuration control.

Both the production activities and product assurance activities are highly dependent on automated aids in the form of tools that support each stage of the life cycle. Although many of these tools exist, they are not available in a comprehensive integrated layout for a particular language, computer, or operating system. Therefore, it is often necessary to apply alternate techniques in the absense of an integrated toolkit.

The productivity quality product may be increased by enhancing the toolkit (i.e., the integrated software engineering environment) used in the production and testing of the software. This environment requires the assembly and integration of currently existing and yet-to-be-developed pieces.

The known and reasonably proven methods of software engineering are adequate for a near-term integrated environment. However, this does not eliminate the necessity for substantial improvements in the methods themselves. Research in applications of Artificial Intelligence (AI) methods could eventually lead to introduction and integration of Knowledge-Based (KB) techniques into the Software Engineering Environment, possibly producing a quantum increase in capability.

If such an enhanced production environment existed, what would some reasonable expectations for productivity and quality improvements be?

In the "near-term," this new environment could be applied to selected U.S. Air Force projects with predictable beneficial effects. After some improvements to the environment, requiring several years of development, an intermediate system could find much wider application and could provide even greater gains.

In the long run, fielding and using this kind of integrated software engineering system is essential to building the engineering base for a KB system.

If the U.S. Air Force acts soon and decisively, software environments can be developed that will achieve these goals. The advantages of a relatively short-term program to produce, distribute, and support one or more software development environments are clear. The risks involved are also clear.

The opportunity exists today to take advantage of the best of current software engineering methods and products to construct a software development and maintenance environment that could greatly assist the U.S. Air Force in meeting the needs of its mission.

To do this will mean constructing specialized "software toolkits," or Software Engineering Environments (SEEs), and making them available to users both inside the U.S. Air Force and in the contractor community.

The composition of a software toolkit of the kind envisioned consists of an integrated, consolidated collection of automated software engineering tools "packaged" in a way that presents a uniform user interface in four major areas:

- Requirements Specification/Design Area. These tools help users establish, analyze, and codify requirements descriptions of software systems to be built. They also provide certain kinds of support in constructing high-level software designs. Technical names for these tools include "requirement statement analyzers" and "program design language" tools, among others. Several of these exist already. Others, currently at the advanced development stage, need only minimal work to be practical.

- Production Area. These tools support production of code from detailed designs. They may include tools assisting in pseudocoding, a precoding stage. A good production environment -- as evidenced by practical experience -- involves such tools as a hierarchical file system, interactive screen editors, modern-design compilers and debugging systems, electronic mail, and office-quality word processing software and related facilities. These features are included in many modern operating systems, especially those developed for the newest minicomputers and super-microcomputers (micro-mainframe machines).

- Testing/Verification Area. Such tools address the quality question directly, and deal with such issues as the comprehensiveness of tests, methods to retest systems already tested and experiencing only slight changes, and direct proofs of complicated logic entities. These tools are currently available in only fragmented forms. To aggregate them for chosen language/machine combinations will require additional work, but no significant technological breakthroughs are needed.

- Management/Support Area. These tools assist in controlling, monitoring, and quantifying the various stages of the life cycle. They involve functions of scheduling and planning, tracking human activities, modeling current error rates to infer amounts of remaining work, as well as systems to keep precise control of a software configuration. Such tools are in wide use on many computer systems but are not integrated well with the other kinds of tools (with the possible exception of production tools).

All of the above must cooperate on one machine so that all of the facilities are within immediate grasp of a user. Many candidate machines exist, as do the operating systems needed to underlie the toolkit. Examples are: DEC's VAX with VMS, DG's MV8000 with AOS, and Prime's V70 with PRIMOS.

For a SEE to be successful (able to attract and satisfy its users), more functional capability supporting software development is required than current systems have been able to deliver. Each of these additional functional areas must appear to the users as complete in its own right. Perhaps equally important, tools in one area should largely "appear" the same as tools in another area to the user. They should present a "uniform interface," which once learned can be applied again.

Requirements Tools and Methodologies

In earlier sections a requirements definition capability for SPO usage was highlighted as part of a decision support system. This capability is geared to improving a SPO's external control over a project by improving milestone definition, problem statement, product verification, and visibility into the software development process.

Improving a developer's internal control is also necessary for both U.S. Air Force in-house development and contractors. Although the needs of developers are similar to the SPO in some respects, developers also require tools that aid in actual code production.

Well-stated requirements are a cornerstone to successful projects. Requirements that are complete, consistent, traceable, and testable will drastically improve quality and productivity and aid in avoiding the development of elegant solutions to wrong problems.

Currently, the U.S. Air Force develops systems in the following way: A user will specify his needs; a system engineer will specify software system capabilities designed to meet user needs; a software engineer will specify functions to provide system capabilities; and eventually a programmer will code a software module that together with the hardware fulfills the user's need. These specifications at higher levels are collectively called requirements, which represent the recording of the staged translations of user needs to system capabilities.

At any time in this process, the user may modify his recorded need and with it the intermediate specification to accommodate either a changing environment or an inadequate original need-specification. This necessitates flexibility in the system design, which is presently provided by modifying the lowest level code without the accompanying use of tools to assist in documenting updates to intermediate specifications. Software is often the chosen method of system implementation because it theoretically provides this flexibility. Experience has shown, however, that without requirements tools, software often provides only the illusion of flexibility. It only appears to be easy to change a line of code.

This situation points out the need for three types of requirements tools:

1. Tools oriented to assist the user in specifying needs in order to minimize post delivery system modifications.
2. Tools oriented to assist the developers in translating the user needs into system and software products, specifically the following: actual system capabilities, system specification, functional specifications, software specifications, software design, and finally, programs capable of operating on an embedded computer.
3. Tools oriented to trace and assess the impact of the changing need on the system as well as the impact of system limitations on user-needed capabilities.

Before the actual incorporation of tools, there should exist a methodology to specify user need and system capability. This methodology should foster completeness of specifications and consistency in language.

Requirement tools should be built to support this methodology by (1) assisting in the specification of interfaces (user to developer hardware to software, software to software), (2) requirements traceability through intermediate software life cycle products, and (3) language consistency checking. These tools should be further developed to assist the user and developer without negatively affecting productivity.

For the near-term, existing requirements tools should be selected to support projects with near-compatible methodologies and their front ends should be human-engineered to improve user and developer, acceptance. Their acceptance and effectiveness in the development environment should be studied and evaluated.

For the mid-to-long term, a general study on requirements methodologies specification languages should be established and integrated in at least four different U.S. Air Force development environments (e.g., avionics, satellite systems, management systems, and support systems). Their effectiveness should be examined and knowledge of these results should be used for tool enhancement, support and further dissemination throughout the U.S. Air Force.

Verification and Testing Tools

Software reliability is a crucial element in U.S. Air Force systems. In order to provide increased flexibility and capability, the U.S. Air Force is turning to software in virtually every major system. Current systems use software to affect communication, for tracking in radar systems, for weapons control, target acquisition, etc. The Space Shuttle is flown mainly by software and soon such is expected with advanced fighter aircraft. Program errors in these systems could mean missed engagement opportunities, improper aircraft identification, lost equipment (as with satellites), and even loss of life (in aircraft software failure).

Current practices in verification and testing provide no assurances as to the quality of the software. Even the Space Shuttle program, at a code production cost of over $3,000 per line of code, failed to provide error-free operation. SAC's 465L system, after operation for 12 years, averaged one failure per day attributable to software.[9]

Existing and advanced prototype software testing tools can successfully be integrated into an initial near-term software environment and can be considerably enhanced thereafter for 200-400 percent gains in quality productivity. Existing verification tools can be applied to selected modules with useful benefit.

Even though the technology, and in some cases the implementation, is known, test and verification tools are rarely applied in the software life cycle, with a resultant impact on software quality. Test tools attempt to identify defects after software production through a process of systematically characterizing the behavior to a measured degree of thoroughness. Verification tools show the mathematical consistency between a formal or semi-formal specification and the actual implemented code.

Some of the deficiencies known to exist are the following:

- Test planning is not coordinated with stated requirements;
- Test completeness criteria are not specified;
- Test plans, procedures, and results are not achieved;
- Verification is not often considered for critical software units; and
- Integration testing support tools need to be developed.

Most of these methods have known technical solutions, and in many cases prototype tools have been built to demonstrate the viability of the solutions. However, due to the relatively high cost and inconvenience of current implementations, these tools and techniques are not as widely used as they should be.

Verification technology, on the other hand, appears to be only barely applicable to practical problems, and only then in restricted circumstances. The payoff in terms of quality for formal verification, however, makes it imperative as a method for selected (critical) software units (modules).

In the near term, the recommendation is to implement versions of tools based on known techniques for test case design and test data generation to support functional and structural testing.

Some of the specific features that should be considered are the following:

- Cause effect graphing for specification-based testing. This technique, shown to be useful in practice for relatively smaller examples (upwards of 75 causes + effects) will organize specifications and also systematize the generation of test data.
- Structural coverage analysis for completeness assessment. Such "test coverage" tools abound for most languages, and await only centralized installation and checkout.
- Assessment of reliability of test data. This kind of analysis would indicate whether test data actually used was "reliable."
- Test file generators. Given the format of input files, this tool would generate sample files matching the format and containing user-selected values, possibly chosen at random.
- Interface testing completeness measures. Currently, interface testing is done in an ad hoc manner. The formalization of counting of variables exercised and values passed would considerably enhance the development of more reliable software.

In addition to this high-confidence work, there should be a program that begins immediate development of simple technical systems to address less-well-understood problem areas such as:

- Integrated support tools. These tools would assist in the complicated multi-component integration process.
- Automatic test harness systems. These tools generate the test environment for a subsystem or an individual component (i.e., for an entry point).
- System testing tools. Such tools would provide services at the system level similar to those which are now available in prototype form at the unit level only. Simulation facilities would be included at system level as they are in the unit level.
- Test-case archiving and regression systems. In this case, the problem is to store, compare, retrieve, and otherwise manage volumes of test ouput data generated under automated control. It is likely that a backing store will be needed, due to the overall volume, and so the possibility of use in a batch (background) environment should be investigated.

Finally, the careful use of verification methods should be considered for some "experimental" situations in which the software system to which the technique is applied is deemed critical enough to justify the extra care and attention paid to quality.

To meet the requirements of the intermediate-term "toolkit" it will be necessary to handle problems of dissemination, support, assembly, and distribution of the toolkit's testing and verification components.

The first step is to obtain appropriate suites of test tools, with user documentation, for widely used HOLs and integrate them within the near-term toolkit. This will also require training courses for specialized tools.

Consideration should also be given to experimental use of verification for critical modules, as described above.

These recommendations should result in an immediate increase in short-term quality of systems (25-50 percent defect reduction) and reduced effort in retesting (or capability to do this work that is now done). The overall risk for these recommendations is low. Many of the types of test tools mentioned above already exist and are in the U.S. Air Force or government inventory. Other test tools are ready for commercial use (with minor restrictions on host environment).

Tools to Maintain Operational Systems

It is evident that as software systems become larger and more complex, logistics and support organizations charged with the responsibility of maintaining and upgrading systems will require expanded capabilities. In addition to detecting and correcting software faults, these support organizations must respond quickly to user problems, maintain configuration control, and manage product improvements.

Advanced research into the areas of maintenance and support is necessary if the U.S. Air Force is to maintain control of the ever-increasing software inventory. Limitations in skilled personnel and the magnitude of the task will require moves toward automated tool support.

Maintenance is a difficult proposition. The existing software inventory contains code developed by nonuniform methods. Differences in documentation style, implementing languages, and machines are prevalent. Operational life of U.S. Air Force systems can be in excess of 20 years. Therefore, the corresponding software must be frequently changed, corrected, and enhanced. Conservative estimates show that over 60 percent of costs in the life of software is in maintenance and support.

Existing software can be classified as:

1. Well documented, using specifications and modern software methodologies with formal requirements and designs and test/plan cases.
2. Poorly documented, did not use modern software methodologies. The documents are incomplete, ambiguous or out of date.

For (1), software development tools should be available. Since modification to the system should proceed from the top down, it should start by analyzing the system requirements as well as proposals for modification, then analyze the specification and design before actually changing the codes. The usual development tools, such as requirement analysis tools, specification tools, design tools and testing tools, are all useful for maintenance work. Thus, with the availability of the software development tools as well as formal specification of the system the task of maintenance will be easier.

For (2), one needs to decide if the affected part is worth modifying or if a new version should be developed using the development tools. If the affected portion is modifiable, then the extent and nature of the changes must be specified. Tools most effective here are specification and documentation generation tools. These tools should aid in reconstructing the requirements and specifications from the implemented code. Since the current state-of-the-art technology is rather weak, new research is required.

Costs for this activity are estimated to be $1 million. Benefits include better-documented and maintained software, less system down time and a greater product enhancement capability for supporting organizations at minimal risk.

Tools for Human Factors Enhancements

Tools that minimize the "technology shock" of incorporating new tools and techniques into the software life cycle are needed. These tools can effectively fit into the user's everyday routine by careful consideration of the human factors and the user's requirements.

Careful consideration of the user's environment is essential to successful technology transfer. Well-engineered tools and environments improve user productivity, reduce time spent in the learning curve, and meet with less resistance of acceptance into the user community.

The lack of human factor considerations in the development of automated tools for use in the development, management, and maintenance of embedded software systems is a problem because tools that are difficult to use are either used ineffectively or not at all.

The majority of the automated tools in use today in the software community must be operated by highly skilled software technicians using standard keyboards and CRTs as the primary interface to the computer.

As software development management and maintenance tools become more sophisticated, the ability of the humans to use these tools effectively must be enhanced. The tools must be usable by humans with management skills, systems skills, testing skills, and maintenance skills. These individuals should not be required to have detailed software skills in order to use the tools; therefore, the human interface to these tools must be significantly improved.

In the near term, standardized methodologies and tool kits using existing software tools have been recommended. The human interface to the tools in the toolkits should be improved to simplify their use. Currently there are several tools in the market place that are designed to be used during the requirement phase of the software life cycle. Existing requirements tools as well as test tools are difficult to use. Once the requirement and testing tools have been selected for the near-term (one-four years) toolkits, a program should be initiated to improve the ability for humans (users) to interface and work with the requirements and test tools. Improvement of the human interface to the design and development tools is not recommended in the near term, because these tools are currently being used by software specialists who have learned to employ them effectively.

In the intermediate term, integrated software development, management, and maintenance environments have been recommended. The environments consist of tools in a local area network environment, including management, development, and clerical work stations connected to file servers, target computers, and environmental simulators as defined.

Human factor considerations must become an initial part of the integrated software tools environment. The specifications must include human factors requirements, and a man-machine interface design document that provides detailed design requirements for human interactions should be developed and maintained throughout the life of the tool environment. Demonstrations of the completed tools environment must clearly show that the system is easy to use.

These systems must employ easy-to-use graphics, color, summary reports, forms completion, etc., to accomplish the simplified user interface.

In the near term, contracts should be awarded for the improvement of human interface to selected specification and test tools in the toolkits.

For the integrated software management and development environment which will be implemented in this intermediate term, the human factor consideration must be an initial part of the requirements, specification, and design. Considerations for the human factor element must be included in the Request for Proposal (RFP) and must be part of the requirements for the system.

Implementation for improving human factors would add 25 percent to tool environment development costs. The potential benefit is an improved productivity of 50-100 percent over systems without a good, simplified human interface, at moderate risk.

Tools for Special Software Systems

Software development tools and software engineering environments are currently designed to provide great generality and wide applicability. However, when these tools and environments are applied to specific systems, there may be inefficiency and difficulty in carrying out the implementation. Indeed, the initial cost of adapting general tools and environments to a specific application constitutes a significant barrier to their use.

Specialized tools can drastically improve software productivity by meeting the particular requirements of U.S. Air Force systems. A careful balance must be maintained between specialization and generalization to minimize the number of special tools, and thus their cost, and good coverage of U.S. Air Force needs.

There are many characteristics of specific systems that make it difficult to apply generalized tools and environments. One characteristic is the applications software itself. Clearly, there are great differences in the nature of and requirements for algorithms in intelligence-gathering systems, onboard avionics systems, and integrated logistics systems. Another characteristic is the major function of the system; again, a system whose major function is the storage and retrieval of large amounts of data will differ considerably from a real-time guidance and control system.

Most current tools and environments are oriented toward a single embedded computer; however, it is clear that many future U.S. Air Force systems will utilize distributed computer networks, highly parallel machines, and other advanced architectures. Still another area is the level of security of the system. Multi-level secure software command systems, where trustworthiness of the outputs is paramount, present particularly difficult challenges. Finally, even the programming language may be an important factor; tools and environments that efficiently support a procedural language like Ada may be significantly different from those for a functional language like LISP. Obviously, any specific U.S. Air Force system will encompass a particular combination of these and other characteristics, and it is to be expected that the adaptation of generalized tools and environments will encounter difficulties and inefficiencies.

The above discussion highlights an important dilemma in the development of software tools and environments. On the one hand, if the tools and environments are made completely general, there will be inefficiencies and difficulties in application to a specific system. On the other hand, if the tools and environments are specialized to each individual system, the costs of developing and maintaining an ever-proliferating set of tools and environments will become prohibitive. The best compromise probably lies between these extremes. The Summer Study group recommended a three-part program to resolve this issue.

The first part of the program is an ongoing study of the feasibility of developing specialized tools and/or environments for specific classes of systems. One aspect of the study would be periodic surveys of U.S. Air Force system developers, users, and contractors to ascertain the perceived need for such specialized tools and environments. The other aspect would stress theoretical studies on the state-of-the-art and state-of-the-practice of tools and environments. The goal of the study would be to examine specific candidate areas for the development of specialized tools and environments, to estimate the cost-effectiveness of such development efforts, and to specify the requirements for incremental development in those cases where the cost-effectiveness justification is strong.

The second part of the program would fund the incremental development efforts for those systems where cost-effectiveness justification is high and where other conditions for the incremental development work are favorable. There are several reasons why this development work should be supported by a central U.S. Air Force organization rather than by the SPO for the individual system. In the first place, the specialized tools and/or environment will apply to several of the systems, and it is not equitable for the first project to pay all the costs. In the second place, unless a central organization controls the work, the resulting product will be tailored only to the first system and will not be transportable to other systems of the same class. Finally, by centrally funding the incremental development work, it is far easier to control the documentation of the work and to ensure that it will be supported and maintained to benefit future systems.

The third part of the program is concerned with the maintenance and support of the specialized environment and the transportability to appropriate systems of the same class as the original system. This support will be critical in acceptance of these specialized tools by other system developers, users, and contractors.

It is interesting to note that the concept of specialized environments is compatible with the notion of knowledge-based systems. In fact, the way in which the tools and/or environments are specialized provide important clues as to which system characteristics and data are important to maintain in the knowledge base. As will be noted later, knowledge-based systems is probably the most promising single technology for providing a breakthrough in software tools and environments area. It follows, then, that the work on specialized environments is an important contributor to the long-range introduction of knowledge based systems into this field.

The three-part program should continue for at least the next seven years. The ongoing surveys and studies will require between $300,000 and $500,000 per year, with a total investment of approximately $3 million. The incremental development cost will vary considerably from system to system. The average incremental development cost is about $2 million. Finally, the support of such systems will cost approximately $500,000 per year per system. If three systems are introduced more or less uniformly over the next seven years, the total support cost would amount to about $6 million. The total cost, then, for the ongoing studies, the incremental development for three systems, and the support for these systems is approximately $15 million.

The benefits for these efforts are difficult to quantify. However, because the ongoing study will allow the U.S. Air Force to choose carefully the systems on which to apply this approach, it is expected that the benefits will be considerable. In some cases, the use of specialized tools and environments will make the difference between success and failure of the system. Certainly, the ability of the U.S. Air Force to make <u>timely</u> application of new advances in hardware, architecture, and languages will depend greatly on providing <u>efficient</u> tools and environments for these technologies. Increases in software development productivity by 200-500 percent for systems where the specialized tools and environments apply are to be expected.

By carefully controlling the areas in which specialized tools and environments are applied, the proliferation of tools and environments can be controlled. The resulting cost savings to the U.S. Air Force could amount to $100 million over the seven-year period.

The risk of the approach is controllable because of the information developed in the ongoing study. Only high-payoff, low-risk projects, where the resulting incremental development work can be applied to numerous other U.S. Air Force systems, will be undertaken.

4.1 Requirements for Near- to Far-Term Environments

4.1.1 Requirements for a Near-Term Environment

Tools exist to comprise the needed software toolkit, but are essentially unavailable (for a variety of reasons, only some of which are technical) to U.S. Air Force labs and projects. Strong and effective leadership will be required to resolve the nontechnical problems.

The technical problems are definable and hence achievable though possibly requiring some compromise. Two particularly important areas needing emphasis for the nearterm are:

- Requirements Analysis Tools, which are not as easy to use as they should and could be. They need better human interfaces and more automated output forms. The principles of requirements analysis tools are well known, but the practice needs to be improved. Deficiencies exist in clarifying design interfaces, supporting test case generation from specifications, handling performance questions, and making incremental changes to requirements statements. Refinements to existing specifications tools, combined with enhancements targeted for the "intermediate" generation, should be adequate to meet the needs defined earlier.

- Program Analysis Tools, which are not used as much as they could be in the operational (maintenance) phases. Better improved end-user designs and better education in each tool's use will aid this problem. Tools for static analysis, coverage analysis, and test data or test file generation are available. Less available -- but not unknown -- are system testbeds, regression analysis systems, and formal verification tools.

A preliminary SEE must be developed by joining existing operating systems, commercially obtained software tools, and minimal but essential augmentations of the user interfaces.

With little technical risk, provided some problem areas are addressed quickly and effectively, it should be possible to build an integrated tool environment that has some very attractive capabilities. This could be done well in advance of commercial developments and could, in fact, lead commercial developments in many ways. The environment could achieve wide U.S. Air Force and U.S. Air Force contractor use, at least for the two-four year period until commercial versions "catch up."

Productivity gains of 25-50 percent should be immediately apparent; and, quality gains of 50 percent or more should be easy to achieve. While not order of magnitude, these are far from minor improvements. The combined effect would probably be a fourfold increase in the productivity quality product.

4.1.2 Requirements for an Intermediate-Term Environment

Once the initial near-term software engineering environment is assembled, immediate improvements must be undertaken to ensure user interest and longevity. Perhaps more important, the U.S. Air Force will probably have to bear the burden of control and distribution.

Improvements to the near-term SEE that will enable the next levels of productivity quality gain include:

- Organization of the system into "workstations" that communicate with each other.
- Creation of a uniform interface language that standardizes the language as seen by the user, e.g., a shell in Unix.
- Addition of special-purpose tools to aid in handling some of the more specialized problem areas, particularly to support operational systems.

Such an intermediate SEE would provide for collections of workstations at which software development, project communications, and most of the heavy clerical burden of software development would be undertaken. Attractive as it may be, a local area network is unnecessary for this task, but should be considered as an alternative.

There are many ways users can perceive a system as easy to use, and a number of facilities may soon be found to be essential in daily operations. Reliable machine-to-machine and intra-machine person-to-person electronic mail is an example that could be a most important feature.

Next in importance is the program that the user interacts with most in commanding subsystems. This is crucially important because minor variances and capabilities make enormous differences in productivity and user satisfaction. Therefore, care must be taken to assure existence of a high-quality interface.

Once a suitable initial SEE exists, re-hosting relevant tools should be a straightforward task. Problems with licensing and royalties would need resolution, but if commercial use fees and charge patterns can prevail, the short-term costs to the U.S. Air Force are quite manageable.

Use of the environment and its payload of tools requires that users have it available nearly all the time as the "standard" interface on machines. This means that the SEE must be distributed and well supported without delay.

Finally, recognizing that the SEE is a complex software package as prone to defect as any software system, provisions must be made to handle incremental modification and redistribution. Commercial patterns exist that would allow this task to be contracted.

Special attention was given by the Summer Study group to the problems of operational support of existing software. Currently, software in the field is maintained with a wide range of tools and techniques, some much more effective than others. Frequent changes, corrections, and enhancements, each small but adding up to a high volume of modification, characterize the practice in the operating commands.

Both the lifetime and the invested value in current operational software dictate special care -- and special support provisions by the SEE.

After initial operating SEEs are available, it should be possible to directly target a number of specialized tools to the problems of field maintenance. Such tools would provide previously unavailable program analysis, modification control, configuration definition, record keeping, and archival capabilities. Moreover, these needed functions would be available in a SEE context that should be expected not to change radically.

Currently available tools that are candidates for the SEE toolkit suffer most from specialist mentality. That is, they are designed by specialists and thus usually intended to be used by the same people. This implies a critical need for the enhancement of human factors. The result is highly capable tools that are inaccessible to users without special training.

Efficient interfaces allow the adoption of fairly proven tools without excessive learning burdens. Techniques using menus and command prompting are highly effective for this. It is important, however, to listen to users' complaints and respond to the problems that system designers might not necessarily notice. This suggests that the users work as a kind of user group to generate the necessary feedback.

Some specialized system areas need particular emphasis and certain kinds of more specialized tools. Applications areas included here are C^3I, avionics, and logistics. The needs are for database systems, security features, and certain kinds of networking.

As before, most of this capability is available but is not integrated in an environment. The special criticality of these missions can both test the SEE and contribute to it.

The intermediate term payoff for a well-engineered and effectively distributed SEE is substantial:

- An order of magnitude (a factor of 10) increase in final-product quality exhibited as significantly reduced error content; and
- A factor of two to four increase in programmer productivity over 1983 figures.

4.1.3 Requirements for a Far-Term Environment

In the long run, it seems clear that the Knowledge-Based (KB) system approach will be effectively applied to just the production of other software systems. A KB system for software production would collect sets of software engineering rules and integrate them in an environment that would permit application of the rules to embryo software systems.

One can imagine this done to the full range of needed disciplines, from requirements analysis and specification through design and coding, including verification and validation steps. The reduction in labor costs through high levels of intelligent automation of function is an attractive incentive.

The key to minimizing disruption as the intermediate term meshes with the future is to lay the groundwork now for the transition. Early KB systems can be built from rules based on experience gained from the measurement and feedback processes built into the near-term and intermediate SEE.

Specific methodologies can be compared, and the essential and effective elements of each can be incorporated in the KB systems that are built. In fact, it seems likely that some of the intermediate SEEs would provide the proof of principle basis for KB systems.

Most important, the structures set up for control and distribution of the SEEs would serve to assure that only mature technology would be accepted in any KB system.

The key to maximizing benefits in the AI/KB area is to control technical risk by managing the long-term evolutionary growth. This means that the near-term SEE must be used as the basis for experiments and activities that assure collecting the most useful type of knowledge base. Early feasibility demonstrations, which would be part of the regular program of development, would help accomplish this. Special data collection experiments and other rule- and data-gathering efforts could be made part of the R&D effort.

Overall, modest investments in the present will be of enormous value in the long run in developing systems.

4.1.4 Summary of Actions for Phased Environment

The previous sections have outlined a strategy for possible U.S. Air Force technical development aimed at the simultaneous goals of:

- Rapid development of near-term software engineering environment support systems;
- Growth of this system into an effective intermediate system;

● Assuring inter-operability with prospective knowledge-based implementations.

To take advantage of the opportunity that now exists and to meet these goals will require leadership and action.

Near-Term Actions

● A task group should be commissioned to determine the means for accomplishing the near-term tasks and to lay the foundation for the intermediate term.

● Existing and working software tools could be collected in uniform Software Engineering Environments (SEEs), and a standard methodology for the SEEs' application and use should be defined.

● The human interface for the SEEs has to be enhanced to minimize system interference while meeting defined quality and productivity goals.

Intermediate-Term Improvements

● Based on initial results with the SEEs, the "standard methodology" can be refined and redistributed to a wider U.S. Air Force user community.

● The SEE can be adapted to smaller computers, e.g., engineering workstations, as a means to further propogate the capability at lowered cost.

● Human factors for the SEE -- at both the lower level of direct interface style and the higher level of interface to subsystems of specialized nature -- should be enhanced and refined.

● Provisions for SEE system maintenance and upgrade should be built, and rapid and effective feedback capability should be introduced.

● Special features aimed at supporting operational maintenance should be developed and refined.

● Assessments of effectiveness in a set of controlled experiments and demonstrations should be made.

Research for Future Environments

Without question, application of AI techniques to software engineering support systems should be supported, and the knowledge base necessary for a KB system should be accumulated.

Research aimed at matching prototype KB systems with advanced versions of the SEEs should be performed. This may result in pilot systems ready for the 1990s.

4.2 Recommendations

In order to verify that the integrated environments accomplish their intended requirements, demonstrations on actual pilot projects must be performed. The tool environments which meet the objectives of the demonstration should be supported by the U.S. Air Force and provided as Government Furnished Equipment (GFE) to contractors on projects which include development of embedded software. These environments should also be maintained and used throughout the life cycle of each system by the government or industry agency responsible for maintaining and enhancing the system.

Each contractor selected by the U.S. Air Force to develop software tool environment systems should also select a minimum of three pilot projects upon which the demonstrations will be performed. The selected projects should meet the requirements stated above. The contractors and U.S. Air Force should coordinate and reach agreement on the selected pilot projects. Once the pilot projects have been selected, the impact of the demonstrations upon the cost and schedule of the pilot projects should be determined. This impact should be accounted for by both the contractor personnel and the U.S. Air Force as part of the system tool environment contract. Demonstrations should be performed by contractor personnel in accordance with the test plan agreed upon by the contractor and the U.S. Air Force.

A total of three tool environments should be built. Each environment should be used to develop, manage, and maintain the software on a minimum of three pilot projects: (1) a large project with an application such as C^2, C^3, or C^3I; (2) a medium-size project such as navigation or flight control system of a missile; and (3) a medium-to-small ground support system such as an automatic test set for a weapon system.

In order to demonstrate these environments on three different pilot projects, it is necessary that the companies that build the environments be in the business of building weapon systems for the U.S. Air Force. This is important, since the demonstrations must be performed on pilot projects being developed by the same company that is developing the integrated tool environment.

The demonstration will confirm that each tool environment meets all of the requirements specified in the tool environment specification. If a tool environment fails to satisfy the requirements, it should not be selected by the U.S. Air Force for full-scale deployment and support.

4.2.1 Demonstrate Integrated Environments

Demonstration of the Software Engineering Environments is the first step in ensuring their usage. Introduction of new technology often meets with resistance, both from within the U.S. Air Force and by developers. It is rarely the case that a tool itself will be sufficient to change the methodology of a software development community. Demonstrations should be effective in allaying developer's fears by showing that the environments meet performance objectives.

As previously mentioned, it is proposed that a minimum of three integrated tool environments be built for the development, management, and maintenance of embedded software. Each of these tool environments must be subjected to a specified verification process to confirm that they satisfy their objectives for all intended applications. The primary objectives for these environments will be to improve the quality and decrease the cost to the delivered mission software. Several sub-objectives of the tools environments must also be verified. These include a simplistic human interface, room to expand the tools environment as new tools are developed, ease in maintenance of the tools environment, ability to support large, medium, and small projects with different applications such as C^3I, space stations, ground support systems, missile systems, automatic test systems, trainees, and simulators.

4.2.2 Ensure Developer Use

Although demonstrations of the software engineering environments will help, it is doubtful that full utilization by developers will occur without other efforts. Developers and U.S. Air Force support organizations must be encouraged to utilize toolkits. An organization's resistance to such toolkits might be based on:

1. A substantial commitment to their own tool set and/or methodology;
2. Lack of appreciation of the life cycle cost-effectiveness of the proposed toolkits;
3. The learning curve required with the introduction of new toolkit technology;
4. Fear of change; or
5. Unknown impact on precise cost and schedule of toolkit introduction on development.

The issues above represent an open problem requiring further examination. It is recommended that the U.S. Air Force charter an organization(s) to examine the available alternatives:

1. Offer a GFE toolkit.
2. Set U.S. Air Force requirements as to minimal automated life cycle support (list of acceptable tools). Developers would then have the option of using a GFE toolkit or their own, provided it met with the requirements.
3. Provide a weighted award process whereby contractors using approved tools score higher than those that do not.

4. Jointly define toolkit composition with users, developers, and researchers.

5. Specify as delivery requirements the form of software products (i.e. formalized requirements document compatible with U.S. Air Force support organization tools).

Costs of the study are negligible (about $0.5 million). However, costs of implementing their recommendations have not been determined. The DoD STARs program is currently investigating how to implement Ada environments. The results of these efforts merit further examination.

4.2.3 Maintain and Support SEEs

Once toolkits and tool networks are put into operation on pilot projects and throughout their remaining life cycle, users will uncover shortcomings, operational requirements will dictate changes, and the systems must be maintained. In addition, users must be trained to operate and use the systems effectively.

Environments acquired from a variety of vendors must be maintained, have technical support, training support, and adequate configuration management. No single organization may have the expertise required to maintain the toolkits (including the vendors). For that matter, no support may be available at all for some tools, particularly if the tools are research environment products. Likewise, user support and training may be impaired. Finally, changes must be controlled to prevent entropy of toolkit configurations.

It is the recommendation of this committee that the U.S. Air Force set up a single internal organization responsible for support and training for the embedded software toolkits and integrated environments. It is likely that this organization will initially subcontract the major portion of this work to the contractors who developed the system; however, it is recommended that the U.S. Air Force gradually move into a strong role in this area.

This organization should be responsible for the following as a minimum:

- Rapid response to user needs, both within the U.S. Air Force and to contractors using the toolkits;
- Operation of the computer program library for (1) configuration control of software tools, (2) configuration control of reusable software design modules, and (3) distribution of tools, design modules, and documentation;
- Maintenance of system hardware/firmware;
- Releases and configuration control of versions of tools;
- Definition and implementation of major changes (enhancements);
- Re-hosting and reconfiguration as required; and
- Training and technical consultation.

The cost of maintenance and support of the toolkits over a two-year period is estimated at $10 million. The cost of maintaining the integrated networks which replace the near-term toolkits after four years is estimated at $20 million between year four and year seven. Maintenance and support costs beyond the seventh year have not been determined.

The benefits of maintenance and support for these tool systems, including the training, are fairly straightforward. As in any system deployed for full-scale operation, the maintenance and training must be provided over the entire operational life span in order for the system to remain effective.

This is a low-risk item with no new technology required to support the task.

4.2.4 Acquire Near-Term Toolkit

Near-term toolkits comprised largely of off-the-shelf components have been recommended. Composition and acquisition issues of the toolkits must be resolved prior to U.S. Air Force deployment.

Related to acquisition are several obstacles blocking the composition of the toolkits. The plethora of tools, methodologies, and host machines must be examined to find the most complete, effective, and usable tool kit. Evaluation criteria must be established and reporting procedures defined. A tool must be well engineered with human factors in mind. The tools should also aid developers on other commercial applications. Applicability and use of tools on commercial development activities assure that tools are providing desired results. Gaps in life cycle coverage, interface difficulties, methodology incompatibility, etc., make the determination of toolkit composition difficult. Further difficulties arise in that some tools are not widely disseminated or are products of a research environment. This requires refinement, improved documentation, and some modifications to make them production quality. Toolkit composition is further impaired by the proprietary nature of some of the available tools. Tools required by the U.S. Air Force must be in the public domain. This involves adequate compensation for tool developers and a range of complex legal issues.

The U.S. Air Force should charter a task force to define toolkit composition and provide an outline of an implementation plan detailing the required improvements in tools and hardware constraints. The task force should rate the tools for applicability to the U.S. Air Force situation and be comprised of representatives from industry, university research communities, and U.S. Air Force projects.

At a cost of $55 million over a four-year period, initial software toolkits should be produced (and their associated methodologies delineated) for U.S. Air Force-supported distribution to effect a 35-50 percent increase in productivity quality, with very low risk.

There are sufficient software tool alternatives operating on selected host computers to support immediate creation of 3-5 U.S. Air Force software production toolkits, provided that they are augmented in the requirements analysis/traceability and software quality management (program analysis) areas.

4.2.5 Acquire Intermediate-Term Networks

Improvements to the near-term toolkits should be fielded in the four-six year time frame. Improvements will be based on continuing evaluations of tool effectiveness, development of unifying methodologies enabling tools to interface with one another, and overall improvements.

Developing integrated networks requires expertise in all aspects of the software life cycle. Expertise in requirements statement, verification, and testing, along with expertise in specific U.S. Air Force systems and advanced human interface techniques, may be lacking in most potential tool developers.

Those organizations with the capabilities may be reluctant to share this expertise insofar as this could result in improved productivity of competitors.

The issues above represent an open problem requiring further study. Two alternatives are offered, however. The recommendation of this committee is that multi-contracts (three or more) be awarded to large system houses that are currently producing embedded computer systems for the U.S. Air Force. Each of these contracts should include provisions for both near-term toolkits and intermediate-term integrated networks of workstations, file servers, target computers, and environmental simulation.

The cost of these integrated environments is estimated at $44 million for the development of three networked systems. A jointly funded consortium approach could reduce this figure considerably. Demonstrations, system support, dissemination, and training are not included in this estimate.

Parallel development or joint development by consortium minimizes risk by reducing corporate bids, maximizing commercial utility, and providing the best available environment.

4.2.6 Integrate Tools Across Life Cycle

As we enter the intermediate term (years four-six), it will become necessary to integrate the tools to form standard, integrated tool sets (environments) that span the entire life cycle. In order to have these tool environments available in four years, the work must begin immediately. The integrated networks should have (1) software engineering development tools, (2) management tools, and (3) office and administration tools. Typical environments will contain a network of development workstations, management workstations, clerical workstations, file server(s), environmental simulators, target computers, and related tools.

Integration may be difficult, since a comprehensive environment for improved software quality and life cycle costs affects virtually all levels and organizations involved with software development and support.

Each workstation will have specialized tools as well as general user-support software. The entire network will have a software kernel common to all computers in the system and will include a standard operating system, networking software, and distributed database management system.

Typical management tools will include tools for configuration management, project scheduling, cost management, document creation and reference, information storage/retrieval, and other management tools that support the entire life cycle. The clerical workstation will include word processors, interoffice mail, and tools for report generation. The software development workstations will have a set of tools that are integrated across the development life cycle and allow system engineers to interact with requirements analysis tools. Once the system requirements have been adequately defined, design and code will be automatically generated and integrated. Easy-to-use test tools will support automatic testing of the integrated software modules. These workstations should be connected by local area networks to achieve maximum cooperation and sharing of information among all the workstations. In the near-term, tools can be introduced without integration and without respect to other tools and the software methodologies.

A time-phased specification for networked environments should be produced during this near-term. The time-phased specification will chart the tool environment at each time frame. The tools will be added and integrated into the environment incrementally.

The integrated networked environments should utilize Ada, but other language support will be retained in order to provide incremental support based on time-phased specification. Existing software written in other languages must be supported.

Provisions must be made to add knowledge based and advanced architecture technology in the future.

The principal benefits are the production of higher-quality, more reliable software for U.S. Air Force systems, reduced life cycle costs, improved schedules for software development and maintenance, and more effective utilization of skilled software personnel within the U.S. Air Force.

A 40 percent (for consistency) improvement in productivity is expected by the end of the year 1990, if the contracts are awarded in 1984. Incremental improvements in productivity will be achieved between 1984 and 1990 through the toolkits and a preliminary network of tools.

Even greater improvements (order of magnitude) are possible as artificial intelligence and advanced computer architecture technologies mature and are applied to the software development tool environments.

Several system houses have already demonstrated that the approach is feasible; therefore, the risks associated with deployment of these integrated networks of tools and workstations is moderate. Most of these techniques are presently being successfully applied to other engineering disciplines such as electronics design, structure design, and mechanical design.

The risk of inserting Knowledge-Based Systems Technology and Advanced Architecture Technology into these systems is high at this time; therefore, these technologies should be pursued as research tasks until the technologies mature sufficiently for insertion into the tool environments.

4.2.7 Transition to Software Engineering Environments

How and in what style the near-term and intermediate-term environments are developed, distributed, and supported is crucial to achieving the goals previously outlined in an effective manner. Such nontechnical problems may lie at the core of U.S. Air Force adaption and use patterns.

The near-term SEE(s) would have to be based largely on known operating systems and environments with fairly well understood productivity characteristics.

The proper proprietary concerns of creators of candidate software have to be addressed: if candidate systems' owners see no overall economic benefit, commensurate with what is perceived as necessary to recoup capital investment, the owners can be expected not to cooperate. If fair and equitable royalties and other incentives are used, the cooperation will be easier.

Before including a proprietary system in any SEE it would have to pass acceptance through some kind of committee and a rigorous acceptance scheme. In the short term, certain systems could be demonstrated on some existing U.S. Air Force program that would be specially chartered for this role. The demonstration would qualify the system for inclusion in the SEE. Such projects as the IV and V portion of some software procurements would be the ideal testing ground needed for this kind of preselection.

4.2.8 USAF Support and Maintain Environments

It is known that keeping an environment working is a nontrivial maintenance task. Even the private sector at its best has difficulty with the configuration control (change control) issue for a software system that might have millions of lines of source.

No matter how many pieces are involved, or how many suppliers are participating, there needs to be one central location at which changes are integrated and from which current system versions are distributed. Just where this should be -- logically and physically -- is a tough question; but the need should not be questioned.

Version Releases. Different versions of each SEE have to be well checked before distribution to the using community.

Rapid "fixes" to immediate problems (Hot Line). There must be user support to deal with emergency questions.

Contracting for major enhancements. Major changes in the SEE have to be handled carefully. It may be appropriate to contract much of this kind of work.

Re-hosting to new computers as needed. Moving a SEE from one computer to another must, by design, be a task of known and predictable dimensions. True portability, being impossible, has to be enhanced.

Training and consulting. System use training, including training in the application of the standard methodology, should be coordinated from a central source.

The group mentioned above by the Summer Study would have to oversee this set of activities.

Collecting data about software engineering productivity is a tough issue, in part because it touches on management issues that can affect employee relations. There has not been much success in making measurements in the past.

Assessing the effectiveness of competing SEEs requires that some kind of comparative data be collected. Now is the opportunity to instrument the environment so that good information can be recorded, and so that biasing, because data is being collected, does not occur.

The committee proposes that the U.S. Air Force provide an incrementally expanding capability for automating the development of embedded computer systems software. Automation of many of the labor-intensive tasks over the entire software development cycle with emphasis on the requirements and preliminary design phases should be accomplished by:

1. Implementing three-five integrated networks of software engineering workstations, software tools, test support systems and documentation;
2. Acquiring and installing "off-the-shelf" software tools or "toolkits" on selected host computers with improved requirements analysis and traceability tools to provide near-term benefits;
3. Developing and supporting a library of reusable software;
4. Producing a U.S. Air Force catalog of software tools and reusable software that will be available to all U.S. Air Force projects; and
5. Providing training and support to U.S. Air Force personnel in the use of the software-automated workstation environments and toolkits.

5.0 ADVANCED TECHNOLOGY OPPORTUNITIES

5.1 Advanced Acquisition Management

In the far-term, software acquisition management is expected to become more automated because of the array of tools that will have been perfected. However, the far-term capability will be realized only if the research and development of these tools is initiated in the near-term. In addition, as new technology, such as artificial intelligence, is incorporated into Mission Critical Systems, the life cycle and hence the management of that life cycle must be modified or altered.

This section concerns itself with the acquisition of Knowledge Based Systems and the development and use of Advanced Software Engineering Environments, including System Project Office Decision Support Tools and a Requirements Definition System for software systems acquisition.

5.1.1 Acquisition of Knowledge Based Systems

Today's acquisition of mission critical software by the U.S. Air Force is geared to a separation of the various phases of the life cycle, and reviews and inspections geared to those separate phases. The new MIL-STD-SDS further reinforces this model of distinct phases and adds further inspection/review points and guidance to help management across the software life cycle.

In contrast, the acquisition of Knowledge Based Systems to date has not followed the accepted pattern laid out in MIL-STD-SDS and its predecessors. These systems have evolved as numerous refinements of an initial prototype. Documents such as requirements (B) specs, design (C) specs, reviews such as preliminary design reviews (PDR) and critical design reviews (CDR), and other forms of documentation and policy are not applicable. If the U.S. Air Force is to acquire Knowledge Based Systems in support of, or as part of, its weapons and defense systems, it must know how to go about those acquisitions.

In the instance of acquiring knowledge-based mission systems the focus will shift from today's concept of software to a refinement of specifications. The issue then becomes: (1) specifying requirements and (2) cataloging inference rules. There are not yet adequate methods for measuring the quality of these processes.

Planning should begin now for the introduction of Knowledge Based Systems. Techniques are needed for specifying a Knowledge Based System, natural-language input and output, and other capabilities derived from artificial intelligence technology. The acquisition process must include knowledge acquisition and knowledge verification. A task force of experts in AI technology and people familiar with system acquisition should propose interim (near-term) procedures. This will permit the insertion of this technology without unnecessary risk or delay.

Eventually, based on the management procedures proposed, evaluation experiments should be performed and drafts prepared for revised regulations, standards, policies, and SPO guidelines, if necessary.

The risks in this area are considered moderate to high. The field of knowledge engineering is in its infancy and there are few guidelines available for specifying requirements and acquiring and verifying knowledge or expertise. If we have learned anything from the production of conventional software systems, it is the need for a discipline to assist in these activities.

In one class of Knowledge Based Systems, namely, expert systems, the verification problem may be simpler than for conventional software. This is because a self-explanation capability for results generated is usually automatically included in all prototypes produced. However, this thesis remains to be proven.

While the Summer Study group could not arrive at a detailed cost estimate to attack the wide class of problems associated with the acquisition of Knowledge Based Systems, it recognized that the expenditure of funds in this area is limited by the shortage of talent to address the issues. It is therefore recommended that the following level of effort be supported:

Year 1 -	$500K
Year 2 -	$500K
Year 3 -	$1,000K
Year 4 -	$1,000K
Year 5 -	$1,000K

The earlier years should be devoted to analyzing existing successful and unsuccessful Knowledge Based Systems and interviewing developers and users. The products should be guidelines to be followed in the acquisition of such systems and suggested approaches to measuring the qualitative and quantitative performance of Knowledge Based Systems. The later years should be devoted to implementing, evaluating, and modifying the guidelines and measuring the processes developed.

Beyond five years, activities in this area should be melded with the acquisition, program management, and measurement activities of the STARS program, the Joint Logistics Commanders, and others.

If experience with conventional software is any indication, attempting to provide a common discipline into the acquisition process of Knowledge Based Systems would be a wise move on the part of the U.S. Air Force. The sooner it is accomplished, the fewer the problems to be encountered as the number of acquisitions of Knowledge Based Systems dramatically increases in the 1990s.

The need for handling the acquisition of Knowledge Based Systems as part of the U.S. Air Force mission critical systems is imminent. The U.S. Air Force must put whatever discipline it can, as soon as it can, into the specification, requirements, verification, and performance aspects of these systems. This discipline must be augmented or otherwise modified as more knowledge and experience are gathered.

5.1.2 Advanced Tool Environments

We are witnessing a trend toward distributed architectures, incorporation of database management systems, incorporation of intelligent processors, and networking with other systems in the development of Embedded Computer Resources (ECR) systems. The software elements of these systems are likely to be developed by different software contractors with expertise in narrow and differing technical areas and with differing software development methodology, productivity, and quality.

The availability of Knowledge Based Systems (KBS) and application of artificial intelligence (AI) for software development will bring about a major change in the way software is acquired. The impact will be felt not because this will improve the highest level of expertise available, but because all communications will be at the knowledge-based "expert" level. Today's management style is often defensive -- its aim is to avoid or recover from failure. With KBS, management tools and techniques can be tuned to productivity issues.

The growing size and complexity of the software content in these future ECR systems will make it increasingly difficult to estimate accurately software development costs and to schedule realistically the software development effort. The tasks of accumulating and processing project status data and providing accurate software development status indicators, including warnings or crisis notifications, will become increasingly difficult. Also, the ability to reach management decisions by selecting and projecting the outcome of all reasonable alternatives will likely exceed human capacity for reasoning and intuition. Therefore, the development of new advanced software acquisition management systems is becoming increasingly important.

The U.S. Air Force should initiate development of Decision Support Systems (DSS) to support software cost estimating, project scheduling, and project status assessments and communication. The DSS will provide the basic functions depicted in Table 1 and will draw upon three important methodologies: Operations Research/Management Science, Computer Science, and Behavior Science. An example of an experimental DSS to support project scheduling is the Graphical Interactive Technique for Project Analysis, Scheduling, and Evaluation (GITPASE) System.

Recently, considerable effort in Computer Science has been directed toward development of Knowledge Based Systems (KBS) and artificial intelligence (AI). Both KBS and AI obviously offer considerable promise for DSS.

We recommend that the U.S. Air Force fund research to improve software productivity, quality, and cost metrics based on improved data collection and technology. The U.S. Air Force should also fund research to develop a Knowledge-Based Program Management Assistant.

The above research and development approach should be structured to provide mid-term as well as far-term useful products. These products are currently envisioned to include the following:

1. Mid-Term: Develop formal management models
   a. Performance, quality, and cost metrics enhanced
   b. Improved management performance and indicators
   c. Improved integrated decision aiding techniques
   d. Cost and schedule planning models based on realistic development life cycles for various applications
   e. Advanced Graphics

2. Far Term: Advanced automated management systems
   a. Through exploratory development
   b. Knowledge Based Management Assistant
   c. Advanced Manager Workstation

Table 1. Five Basic Decision Support System Functions

Interrogation	Interaction between the human and the DSS. Either the human or the DSS may initiate an information request or respond to the other.
Modeling	The process of employing mathematical or computer models to develop strategies and information.
Filtering	Reducing, summarizing, aggregating, and combining data into information.
Monitoring	Continuously observing the data and providing informational comments to the decision maker on an automatic basis, e.g., Software Schedule Alarm.
Gathering	The process of obtaining data from sources external to the DSS.

Cost/Benefit/Risk An initial five year cost estimate to develop a Knowledge Based Management Assistant with an Advanced Manager Workstation is $3 million.

Development of a Knowledge Based Program Management Assistant will provide direct benefit to the development of a "Knowledge Based Software Assistant" (KBSA), discussed later, that is envisioned to derive implementations from specifications for the development, evolution, and maintenance of large software projects. The complete KBSA development is projected to culminate in about 10 to 15 years.

The Knowledge Based Management Assistant is considered to be a medium-risk objective since precedent KBS developments and DSS are evolving from the research environment, with some innovation required to extend benefit to the Knowledge Based Program Management Assistant. The other objectives are considered to be low risk.

Summary The amount of information that acquisition managers and project managers must deal with to manage mid- to large-size software developments and maintenance activity is too much responsibility for one person. Even when managed as a team, there is a lack of communication among team members. Automated decision support systems, incorporating formal management models that can later form the basis of knowledge-based program management tools, are seen as a solution.

The long-range concept for a requirements tool is to extend its functions to ensure that project risks are minimized and that there is a favorable schedule impact on the acquisition of systems with significant software elements. A comprehensive requirements tool of the future should allow the user command and the SPO to define alternative weapon system concepts that can be quickly described and analyzed to ensure that technical, schedule, and budget risks are understood. Supporting tools would provide multiple evaluation levels that include requirements development, cost estimation, schedule estimation, technical evaluation, performance modeling, simulation, life cycle definition and analyses, and overall system optimization analysis.

The near-term tool environment will be lacking in several respects with regard to the above mode. Specifically, it will be lacking in modeling and prototype support; integrated tools for estimating project budget and schedule; tools for estimating technical risk; and tools for performance modeling. In addition, there will be little support for developing a total life cycle plan, and the entire set of tools will not be properly oriented to help acquisition managers and users make trade studies among technical, project and functional issues.

The near-term actions necessary to make a comprehensive system requirements and project definition tool possible in the long term are:

1. Define general requirements for a comprehensive requirements definition support system with the following recommended attributes.
   a. Natural-language interactive dialog to better articulate requirements;
   b. Machine interface to developer tools (e.g., specification tools, unit folders);
   c. Rapid prototyping demonstration capability for functional system;
   d. Automatic assists in Testing End Product Against Requirements (e.g., automated test generation); and
   e. Interfaces with cost/schedule tools.

2. Begin collection of the knowledge base requirements for operating the tool.
   a. Establish a team that interviews a subset of user command and SPO personnel to determine the knowledge base requirements in project budget, project schedule, performance estimation, system modeling, system simulators, and system analysis. The team should develop a refined plan for collection of knowledge for a comprehensive requirements and system definition tool.
   b. Have the team develop a knowledge collection format that is used to expand the interview process in quality and quantity.
   c. Identify the experts within the U.S. Air Force supporting contractors that have demonstrated successful concept stage formulation.
   d. Use the interview, identified experts, and the team to begin a collection of decision knowledge, techniques and tools that should be integrated into a comprehensive requirements tool.

3. Fund research in system definition methods that include the project budget and schedule, the life cycle plan, the technical risk factors, and the performance modeling -- the system prototyping that will support a comprehensive system requirements and project definition tool.

4. Track research on Knowledge Based Systems to establish the feasibility of development of a comprehensive requirements tool. Prototype subsystems should be developed when feasible. Present identified candidates for subsystems of a comprehensive requirements tool are:

a. Requirements (near-term recommendation);
b. Project Budget or Cost Estimate;
c. Project Schedule Estimation;
d. Technical Risk Evaluation;
e. Performance Modeling;
f. Concept Proof Prototyping;
g. Simulation;
h System Optimization Analysis; and
i. Life Cycle Analysis.

5. Use test site cases to prove the value of subsystems.

6. Integrate developed subsystems as usefulness is proven to build the comprehensive requirements tool.

5.2 Knowledge Based Software Technology

Knowledge Based Software Assistant (KBSA) is a concept for a methodology and a set of techniques for the automation and integration of software development and maintenance. This concept if proven feasible has the potential for bringing about a radical improvement in the software development process. A general plan has been developed, with U.S. Air Force support, for realizing this high-potential gain through a program of medium-risk, high-payoff product developments.

The Knowledge Based Software Assistant concept consists of a set of complementary principles and techniques. These principles and techniques are motivated by several key perceptions, relating to:

1. Use of prototyping and incremental development to assure that a system design is relevant to user needs;
2. Maintenance of design knowledge, e.g., objectives and design decisions, in explicit form to assure that the many facets of the development and maintenance of a software system will remain integrated over its life cycle;
3. Use of formal specifications, to assure unambiguous definition of other system developments; and
4. Use of automated program refinement and optimization techniques to generate working systems initially, and to regenerate them to meet new objectives, in order to reduce development costs and to assure design integrity over the life cycle.

Specific approaches have been proposed for tools and procedures that support the activities of developers and maintainers, managers, and users. A key organizing concept is that the tools and procedures be realized within a computer-based programming environment as a machine-intelligent assistant, with the capabilities for initiative, explanation, planning and execution, maintenance of system knowledge, and coordination of the activities of multiple participants.

A strategy has been proposed for developing the KBSA concept incrementally, through a sequence of low- and medium-risk efforts. For example, techniques for using design knowledge would be developed initially using informal representations, techniques for prototyping would be developed experimentally in specific application domains, and techniques for system generation would be developed initially with a high degree of programmer participation. Concurrently, research would be undertaken to develop techniques for formalizing and automating these functions. The research would exploit ideas being investigated in current research in computer science and artificial intelligence.

Current software practice is a highly fragmented and labor-intensive activity. For the complex software used in U.S. Air Force systems, these deficiencies lead to several serious shortcomings in quality and cost, including (1) inadequate responsiveness of systems to user needs, (2) high cost of initial development, (3) excessive cost of repair and modification, (4) lack of reusability for new applications and portability to new environments, and (5) high sensitivity of cost and performance to the level of skill of the developers. A significantly higher level of automation in the software development is therefore essential, not only to increase programmer productivity, but to assure that future systems will perform as needed, remain relevant and maintainable over their lifetimes, and be acquirable quickly and at acceptable cost.

Establishment of new software practices based on KBSA will require 7-10 years of intensive research and development. Propagation of the new technology into practice will require wide participation in the research and development community. Initial effort should be directed toward defining a set of specific mid-term and long-term products, funded at an increasing level, over a period of seven years. Participation by other services and agencies may be expected due to the generality and importance of the end goal.

The U.S. Air Force needs to justify fully the immediate establishment of the KBSA research program. Although support by other services and agencies may be expected, specific U.S. Air Force applications will benefit directly from KBSA techniques, and continuing support at a high level will be justified. An appropriate level of support in the initial seven years would be (in millions of dollars): 2.0, 3.0, 4.0, 4.0, 5.0, 4.0, and 2.0.

This schedule assumes technique development in years one-five, and initial implementations and demonstrations in years three-seven.

The benefit of the program may be measured by the difference between a natural growth in software development power (estimated by the STARs program as being on the order of a fourfold increase in 10 years), and a radical advance (on the order of at least ten fold in 10 years). Benefits in terms of correctness, timeliness, relevance, and adaptability of software will be multiplied greatly in terms of U.S. Air Force system effectiveness, if proven feasible.

The proposed program has high-risk and low-risk components. Individual products proposed, e.g., an intelligent design database manager, will have benefits independent of the KBSA context. Development of an integrated technology will require diligent effort, and cannot be guaranteed. There is little doubt that ideas developed during the research will find useful expression in the evolving software practice.

A major research and development program towards Knowledge-Based Software Assistant technology is recommended. The benefits will be a radical improvement in capability for producing software for U.S. Air Force systems, economically and quickly, that is responsive to initial and evolving user needs, easily maintained, and composed of elements that can be reused in other applications. Participation by other services and agencies can be expected.

5.3 Advanced Requirements Specification and Design Technology

This section concerns itself with advanced tools and techniques for establishing, specifying, and tracing requirements and with a new concept for creating, maintaining, and using libraries of specifications and designs.

Requirements and Specification Technology

In all software development projects the initial phase is the definition and communication of the requirements of the software system. The formal specifications for the software must then be correctly and accurately defined. This process has often resulted in the generation of many errors in the software due to the inadequacy of the current requirements/specification techniques and tools.

The requirements definition and the tracing of the requirements throughout the entire software life cycle is inadequately performed within the U.S. Air Force. The current requirements/specification languages, tools, and techniques used by the U.S. Air Force are inadequate due to the lack of generality and formality, and the inability to verify the successful implementation of the requirements.

Often the requirements for a software system will change during the life cycle of the system. The current requirements/specification tools and techniques do not support this critical need.

The following recommendation is made for the near term: perform an evaluation of the current requirements and specification technology in order to determine the relative successes and failures of the current techniques. The results of this survey can be used to determine the inadequacies that need to be corrected. In addition, with this information criteria can be developed for future advanced requirements/specification techniques such as those discussed below.

For the mid-term the following recommendations have been developed:

1. Develop requirements techniques that can support the analysis of system requirements and provide cost/benefit analysis of changes made in requirements.
2. Develop requirements/specification techniques with the following characteristics:
   a. The requirements/specification techniques should be integrated with other tools used in subsequent software life cycle phases (design, coding, testing, and maintenance);
   b. The techniques should support the traceability and verification of the user requirements;
   c. The techniques should provide automated consistency checking;
   d. The techniques should provide improved human interfaces (ease of input, understandability, natural-language based);
   e. The techniques should support testability of requirements; and
   f. The techniques should support the definition of nonfunctional requirements.

The successful implementation of the first mid-term recommendation requires that the U.S. Air Force establish a program that will develop application-specific libraries in which requirements/specification tools and techniques are a significant component. These libraries should provide a basic set of tools to be used in the analysis of system requirements. The capability to perform cost/benefit analysis of changes to the requirements should be included in the tools developed.

Research and development work should be funded to design tools with the characteristics stated in the second mid-term recommendation. The success of this work will have a profound effect on the successful adoption of the far-term recommendations, which are:

1. Develop specification languages that can be executed to provide early prototyping and modeling of software systems;
2. Develop techniques that can be used in advanced applications (distributed systems, multilevel security, concurrent processing, database applications, and Knowledge Based Systems).

The successful implementation of the first far-term recommendation requires that funding for research and development be provided in the area of executable specification languages. Automated specification languages must be developed in addition to the required language processors. With the aid of executable specifications, a significant portion of the software being developed can be demonstrated at an early stage of system development.

The implementation of the final recommendation requires that the previous recommendations be adopted and successfully completed. As applications become more complex and sophisticated, adequate requirements/specification tools and techniques must be developed for these applications. Based on evaluation of the success of implementing the previous four recommendations, the final recommendation is to provide funding for the development of advanced requirements and specification technology.

The total cost for the area of requirements and specification technology is $40 million. The allocation of these funds to the five recommendations is:

1. Survey of requirements/specification tools and techniques: $2 million;
2. Development of requirements techniques to support the analysis of system requirements: $8 million;
3. Development of enhanced requirements/specification techniques: $8 million;
4. Development of executable specification languages: $10 million;
5. Development of requirements/specification technology for advanced applications: $12 million.

The use of improved requirements and specification technology will result in lower software development cost and increased productivity, reliability, and correctness of software. The benefit that can be gained from the survey of existing requirements/specification tools and techniques is that criteria for evaluating future tools and techniques can be established.

The use of better requirements and specification technology will result in software systems that can be verified. The use of executable specifications will allow users to more accurately define and view the requirements of the software system being developed.

The overall risk of the five recommendations is medium. The degree of risk is affected by the fact that the risk involved in the survey is low, but the risk associated with the tool development is medium to high. This risk is primarily due to the complexity involved in developing new techniques for requirements and specifications. The area of Knowledge-Based Systems is clearly the radical area with the highest risk. Overall, however, the development of requirements and specification technology represents a conservative advance with the potential for significant gain.

Requirements and specification technology is an area that has been inadequately addressed by the U.S. Air Force. This lack of concern has resulted in software systems that are over budget, inaccurate, and incomplete. The development of improved and more sophisticated techology can provide solutions to these problems.

Specification and Design Libraries

The trend of increasing software development costs can in part be attributed to the practice of not taking advantage of the knowledge, experience, and products resulting from previous software development efforts.

Investigators have performed research related to the area of software specification and design libraries. Many have advocated the designing of software that is more contractable and expandable. Some of the techniques required for specification and design libraries have been investigated. Also, there are currently research efforts in application specific domains in the Knowledge-Based Systems area.

Facilities and techniques currently are minimal in the U.S. Air Force to support the reuse of software specifications and designs. Several research efforts have been conducted that are relevant to the establishment of specification and design libraries and the reusability of software products.

For the near term, the following recommendations provide a possible strategy for the investigation of specification and design libraries:

1. The feasibility of specification and design libraries should be determined by the U.S. Air Force. The tools and techniques required to support these libraries must also be considered.
2. The U.S. Air Force needs to determine the feasibility of specification and design libraries that would serve as generic units for application-specific domains.

3. The U.S. Air Force needs to determine the successes and failures of existing software libraries, and to develop a strategy for populating, using, and maintaining specification and design libraries.

The implementation of the three near-term recommendations requires that the U.S. Air Force establish and fund a survey that will investigate each recommendation. The survey should be performed in academia, industry, and government. The successful implementation of the mid- and far-term recommendations requires that each of the near-term recommendations be successfully accomplished.

The recommendations developed for the mid-term include the following:

1. Support environments for the development, configuration, and maintenance of the tools and techniques required for the software specification and design libraries must be developed.
2. Based on the survey of application domains, the U.S. Air Force must populate the support environments with specifications and designs for generic software units. It is recommended that at least two such libraries for different domains be populated.
3. The U.S. Air Force should select several system development programs to use the relevant libraries in the development of significant portions of the software. Data should be collected on the productivity of the software team and the quality of the software developed for those portions, and then compared with productivity and quality figures for software of equivalent function and complexity.

The successful implementation of the mid-term recommendations requires initially that the U.S. Air Force provide funding to develop support environments for the tools and techniques required by software specification and design libraries. The enhanced support environments can then be used in the development/population of the libraries and the further evaluation of the feasibility of the libraries through actual use.

If the initial phases are successful, the far-term goal should be to populate and use as many application development libraries of specifications and designs as are feasible. In a truly competitive atmosphere, if successful, the libraries should pay for themselves through heavy contractor use.

The cost breakdown (in dollars) for the near, mid- and far-term is as follows:

1. Research to determine the feasibility of specification and design libraries: $1 million;
2. Funding for a survey of existing software libraries: $1 million;

3. Funding for the investigation of the successes and failures of existing software libraries: $0.5 million;
4. Development of enhanced support environments for software libraries: $4 million;
5. Development of initial libraries and evaluation of merits through actual application: $4 million; and
6. Development of additional libraries: $10 million.

The primary benefit to be gained from the use of software specification and design libraries is that reusable software modules will begin to be developed and incorporated in future software systems. This will also encourage the development of software families. The reuse of specifications and designs can result in a dramatic decrease in the cost of software system development.

Overall, there exists a medium degree of risk in the realization of the six recommendations. The degree of risk for the first four recommendations is low but there exists a higher degree of risk for the successful accomplishment of both the mid- and far-term recommendations.

Techniques and tools that support the reusablility of application software for embedded systems do not currently exist. The benefits to be gained from the development and use of specification and design libraries are significant. The use of existing database technology, coupled with techniques for designing software families, provides an excellent area for future research by the U.S. Air Force.

5.4 Advanced Verification and Validation Techniques

Despite many years of research and practice, software system verification and validation continues to be responsible for a major component of the cost of system development and maintenance. The relatively low technological level of these activities makes it difficult and costly to achieve desired system quality, to assess system feasibility, and to predict development costs. For critical functions such as system safety, integrity and security, present verification techniques do not provide adequate assurance for the extremely high-level of correctness required. New technologies for software development offer prospects for radical changes in techniques for verification and validation, but these will not be available in practice for many years. Research is needed on how present practice can be substantially upgraded, both for large software systems and for critical system components.

Little progress has been made in the art of software system verification and validation. The primary methods continue to be testing and informal design review. The cost of testing remains a high fraction of system development and maintenance cost, e.g., from 40-60 percent, depending on system criticality. The cost of validating system requirements and design (to determine completeness and to assess feasibility of implementation) is a much smaller fraction, but the poor state of practice is responsible for much hidden cost. For example, many design and coding errors are found only after installation in the field, at which time the cost of repair is extremely high. For large systems of low and medium criticality, much expensive testing is needed in order to reasonably ensure that the system will perform correctly, while for small systems (or small components of large systems) with high criticality (e.g., controlling system safety or security), the use of testing or verification invariably leaves great uncertainty that required levels of protection will be reached.

This poor state of affairs is partly a consequence of the inherent limitations of testing (i.e., the response of a system to a finite behavior for the infinite set of inputs that may be experienced in actual operation). The use of informal descriptions for requirements and specifications also contributes harmful ambiguity to the validation and verification process.

Pending the radical improvement in system generation that may result from long-range research, several steps can be taken that would have a significant impact in the mid-term, e.g., five years. These would include systematic methods for structuring designs for testability and for instructing design document to provide clear and unambiguous definitions of intended behavior. These methods should be implemented by appropriate tools integrated into modern work environments.

More significant advances will require development of more fundamental methods for system definition and analysis. Areas for which recent, promising research results exist include (1) formal modeling of specifications, (2) formal verification, and (3) symbolic testing. In formal modeling, an abstract statement of specifications is constructed using mathematically well-defined notation. Such models can provide a succinct statement that can be evaluated for relevance to user needs by manual inspection. Some notations also allow direct machine interpretation, using test cases (recent introduction of efficiently interpreted very-high-level languages, e.g., Prolog, has expanded the possibilities for abstract definition in interpretable form). This method appears to have significant value for large systems provided that modular construction is employed. Work needed to develop this technique includes development of (1) languages for modeling the behavior and properties of complex real-time systems, and (2) tools for simulating system behavior directly from requirements models.

Formal specifications also can provide a reference for proving the correctness of designs with respect to the modeled requirement, for all possible input data conditions. Several sets of tools have been developed to support such verification, and have been applied experimentally in the areas of computer security, reliability, and communication protocols. For the foreseeable future, formal verification will be limited to small systems or limited aspects of large systems. Advances in this technology are nonetheless of great importance because of the prevalence of critical functions in U.S. Air Force systems. Work needed to advance this technique includes development of (1) more powerful and convenient-to-use tools and (2) libraries of proven (and parameterized) specifications and designs, for use in system construction.

Use of formal methods also has good potential for upgrading testing practice. Examples include (1) use of formal specification to define proper behavior of a software module for all allowed test inputs, and (2) symbolic analysis of program texts to simulate large sets of test inputs. Both of these approaches are at an early stage of development. Work is needed on basic techniques and on techniques that may be particularly effective in specific domains. Test methods are especially needed for systems that have complex behavior in the time domain.

A program of research is recommended in advanced techniques for the verification and validation of software systems. The primary components of the research should be:

1. Specifications aimed at the problems of large systems;
2. Formal verification, aimed at the problems of high-criticality systems; and
3. Testing, aimed at the problems of design implementation. Technique development should be both general and domain-specific. Promising techniques should be implemented in tools and tested in practical applications.

The following schedule of funding is recommended (in millions of dollars per year): 2.5, 3.0, 4.0, 4.0, 5.0, 4.0, and 3.0. This plan assumes theoretical studies for years one-four, and implementation and experimentation in years four-seven.

Expected benefits are (1) a substantial cut in the present 40-60 percent testing budget for large software system developments, and (2) a radical increase in the level of confidence in the correctness of critical systems.

Proposed work on formal modeling techniques for specifications including the use of interpretable, very-high-level languages, is certain to find some direct applications in system description, and is likely to have a beneficial, indirect impact on software practice by increasing the rigor of informal specifications. Formal verification has already been introduced into practice (in application to secure systems). The major uncertainty pertains to the degree to which advanced tools can ease the difficulty of formal verification for professional system programmers. Prospects for direct application of techniques based on formal testing are uncertain. Basic research in this subject will have a beneficial impact on the rigor of testing practice.

Research is recommended in advanced techniques for verification and validation of software systems. Use of formal methods is suggested in the area of requirements modeling for large systems, proof of correctness for critical systems and testing for correctness of program modules. Major benefits are expected in the areas of testing cost for large systems and confidence level for critical systems. Work on specifications and proof can be based on techniques that have been demonstrated in current research.

5.5 Distributed Software Engineering Environments

Support Systems

Currently, software developed for the U.S. Air Force may be produced on a variety of machines in several phases by different developers. Software standards are lacking, thus resulting in extremely high maintenance costs or in systems that do not meet desired criteria. Analysis of problems in the area points to communication as a major bottleneck in development. A system development technique, such as prototyping, has been suggested as improving developer-user communication. Integrated tool sets creating lines of communication between phases of the life cycle improve system development continuity, and independent user workstations evolving to an intelligent, powerful workstation should enhance development. The network connecting these stations is imperative in maintaining system integrity and in successful project development and use.

A serious problem faced by the U.S. Air Force in system development is the poor interorganizational transfer of delivered software systems. These poor transfer links result in increased error correction costs, poorly met/unmet requirements, and time delays in meeting changing requirements. Already highlighted are the poor communication links between the acquisition manager, developer, maintainer, and user. Techniques/tools successfully used by the developer may be unknown or unavailable to the eventual maintainer, with an obvious impact on maintenance cost. Anticipated requirement changes by the U.S. Air Force may be unknown to the developer until, due to earlier design decisions, a massive software effort is required to effect the changes.

A common distributed software support environment should be developed to serve as a common medium for the acquisition manager, developer, tester, maintainer, trainer, and end user. This environment should support all the earlier discussed technology and management technologies -- including metrics to aid in life cycle support, and the decision support to allow the acquisition manager/developer/maintainer to direct and control system development -- and provide an integrated tool base for development and maintenance continuity. The environment must implement security mechanisms to protect sensitive U.S. Air Force and contractor software and data.

Initially the feasibility of a distributed software support environment should be demonstrated. This environment itself may be physically distributed. A minimal network connecting the acquisition manager, developer, tester, maintainer, trainer, and user should be established and effectively demonstrated. During the feasibility study, standards to ensure the communication between component parts developed by separate individuals should be established, since integration problems may be severe without the use of simple, well-defined standards.

Upon success of the feasibility study, the environment should evolve through the addition of powerful workstations for both software engineers and managers. These individual workstations provide the software engineer with better time response and speed when compared with one centralized system. Incorporated into the design of the workstations must be the results of the human interface thrusts outlined elsewhere in this report, thus enhancing the acceptable and improving the exploitation of the new workstations. The far-term goal is a fully integrated, reasonably common distributed software support environment for all new projects. This environment should be integrated into MIL-NET, so the responsibility for security must rest with the local network(s) implanting the various portions of the environment itself.

Immediate cost for the design and demonstration of the common distributed software support system environment feasibility study would be approximately \$3 million. This initial study should provide the prototype for the individual workstations to be developed. The most expensive and difficult aspect will be the network development, which must assure communication and maintain security and product integrity (cost approximately \$5 million).

The environment produced would result in a minimization of the problems connected with passing software management, development, and maintenance responsibility from organization to organization. This common environment would also improve communication between management and software engineers.

The major problems with system security should be solved by an ongoing DoD thrust into multilevel security, and those remaining to be solved in implementing this recommendation are considered to be of medium difficulty and hence medium risk.

An environment that provides each system developer, acquisition manager, tester, maintainer, trainer, and end user with a common medium through which communication is possible will have a high position impact on the cost of all system life cycles. Decisions made at one phase will be better understood by other phases. This understanding will ease the modifications needed throughout the life of the software, thus improving productivity and decreasing labor and time for system maintenance.

Database Technology Applied to Environments

The software life cycle is defined as a series of phases through which a software project passes as it evolves from a concept into a completed system. The software life cycle is currently recognized to be an interactive process both between and among the individual phases. Each successive iteration further refines the developing system.

A large amount of information about the software, generated in each phase of the life cycle, must be recorded. A computerized database can record this information in addition to the documentation and other information concerning the evolution of the software system.

The lack of database technology applied to environments during the software development process has resulted in the following inadequacies:

1. Minimal visibility into software projects;
2. Minimal project history maintained;
3. Poor cost estimation and scheduling; and
4. Poor data collection techniques.

Relative maturity has been gained in the discipline of database technology. This technology has already been applied to several tools used in the various phases of the software life cycle.

The following near-term recommendations have been suggested in the area of database technology applied to environments:

1. The U.S. Air Force should develop a common baseline and strategy for the collection of critical software information. This strategy should be applied to U.S. Air Force software development projects.

2. The U.S. Air Force should establish a common database where this information can be collected and organized. The data should be available to government, academia, and industry so that the information can be used to develop improved information techniques and models.

The mid-term recommendation is as follows:

Develop, maintain, and require the use of tools that will accumulate and report project management data and other quantifiable measurements. The data and measurements can be used to evaluate project status and provide for project visibility.

The success of the following far-term recommendations is dependent upon the successful completion of the near- and mid-term recommendations. The primary far-term recommendations are as follows:

1. Database technology should be included in software development environments. The function of this technology will be to provide support for the collection and organization of any software critical data.
2. Knowledge-Based Systems, database technology, and high-level design languages must be integrated into a software development environment.

The following implementation recommendations develop the strategy for including database technology in software development environments:

1. The successful implementation of the first near-term recommendation requires that existing database management systems be used to develop a general purpose scheme for collecting life cycle information of developing software systems. This scheme includes information about design decisions, cost information, and personnel requirements.
2. The successful implementation of the first mid-term recommendation requires that research be funded for the development of project management tools. The project management tools can be used to support the manipulation of the information in the database.
3. The second mid-term recommendation can be successfully implemented if research is funded that can determine how database management systems and software development tools can be integrated and enhanced.
4. The implementation of the final recommendation requires that research be funded to encourage the use of database management systems, Knowledge-Based Systems, and high-level design languages. Research should be funded that will investigate the

coupling of database management systems and Knowledge-Based Systems. These two areas can provide a significant amount of support for software development environments.

The following cost breakdowns have been suggested for the implementation of the above recommendations:

1. The collection and organization of life cycle information using existing database management systems: $2.5 million;
2. Funding for the development of project management tools: $2.0 million;
3. Integration and enhancement of existing database management systems and software development tools: $2.0 million;
4. Coupling of database management systems, Knowledge-Based Systems and high-level design languages: $3.5 million.

The following benefits are envisioned as a result of the application of database technology to software environments:

1. An enhanced visibility of software projects will result;
2. The development of a project history database will allow for the analysis of mistakes and will prevent the subsequent repetition of the same mistakes;
3. Life cycle models of cost estimation and scheduling will result in decreased cost and reduction in the delay of software projects.

The overall element of risk involved in the successful implementation of the previous recommendations is considered to be medium. The individual risks involved for the successful completion of each recommendation follows:

1. The risk involved in using existing database management systems is low because of the wide body of knowledge and experience with these systems.
2. The risk involved in successful completion of the second and third recommendations is medium because this is a relatively new area of software research.
3. The risk involved in the successful implementation of the final recommendation is one of medium to high because of the higher element of risk involved in the areas of Knowledge-Based Systems and high-level design languages.

The use of database technology applied to software environments can have a positive impact on the acquisition of software development knowledge and experience. The use of database technology can contribute to the organization, analysis, and dissemination of the information collected. This practice will allow software developers to learn from previous development efforts. This information will also allow management to direct and track software projects.

5.6 Advanced Computer Architectures

Standard computer architecture is based on concepts, such as single, serial flow of control, and location-based data identification, that reflect hardware and programming technology of the late 1940s. Dramatic advances in circuit speed and the development of operating systems and compilers have helped extend the utility of the standard architectural model to meet accelerating demands for performance and service, but these techniques are unable to keep up with recent growth in computational requirements. For example, in defense systems, new levels of computational performance of at least one thousand fold are required in applications such as signal processing, autonomous weapon control, and battle management support. New kinds of logical support for computer programs and operations are also needed to reduce the cost of software development and to improve system reliability. These include (1) support for high-level languages (e.g., Ada) and very-high-level languages (e.g., Prolog and various so-called functional languages); (2) support for system design to assure multilevel data security and reliability (e.g., testability and fault tolerance); and (3) support for new or specialized modes of computing, e.g., distributed processing.

The de facto standard computer architecture, based on the 35-year-old von Neumann model, is unable to accommodate new computational requirements, such as a thousandfold increase in speed, data security, and ultrareliability, and new developments in hardware and software technology, such as VLSI arrays and very-high-level programming languages. Many new architectural models have been proposed, but these models are at an early stage of development. The lack of well-established models for advanced computer architectures seriously impedes development of future U.S. Air Force systems. Current industry- and government-supported research in this area is inadequate and not significantly motivated by U.S. Air Force requirements.

Requirements for high performance and new functionality employ increases in hardware complexity that cannot be realized effectively within the framework of the von Neumann architectural model, even with projected advances in device technology. For example, replication of components to achieve higher performance through parallelism is poorly accommodated in the present mode. Such replication, together with increased logical complexity, is made feasible both technically and economically by VLSI technology, but the present architectural model is poorly suited to exploit such advances.

In recent years, many new architectural concepts have been proposed that address these issues. These include (1) object-oriented machines to accommodate high-level languages such as Ada; (2) functionally oriented machines, such as dataflow and reduction machines, to accommodate very-high-level languages and to provide efficient organization for highly parallel processing; (3) array machines oriented toward highly structured computational problems such as image processing and physical simulations; and (4) network machines, oriented toward objectives such as distributed processing, security and fault tolerance. No single approach has yet been proven to have universal qualities, and it may be that future systems will consist of combinations of machine types, but it is too early to be confident about such a judgment. Considerable research is needed to evolve a small set of architectural schemes that can provide broad domain applicability and effective utilization of device technology.

A program of research should be undertaken to develop advanced computer architectural concepts in support of future U.S. Air Force requirements for performance and system development. The research should include both basic studies of architectural design principles and evaluation of particular machine concepts over a wide range of applications. The work would include analytic studies and construction of experimental models. Construction of general purpose testbeds to facilitate research is recommended. The work should be coordinated with similar work sponsored by other services.

The following seven-year schedule of effort is recommended (in millions of dollars per year): 2.0, 3.0, 4.0, 5.0, 5.0, 5.0, and 3.0.

This assumes a level of $1.0 million per year for basic studies for years one-four and the remainder for tool development and experimental studies.

Achievement of a small set of architectural schemes that exhibit many of the desirable objectives for speed and efficiency of support of very-high-level languages cannot be assured. The alternative may be a wide variety of techniques that would be employed as needed in particular architectures. Significant improvement in architectural design can be expected with confidence.

Potential benefits are extremely high, including radical expansion in maximum machine performance, adequate support of the next generation of software methodologies, and high levels of security and reliability.

A program of research is recommended in advanced computer architecture. The research is expected to make possible the radical improvements in performance and functionality needed to meet the computational requirements of future U.S. Air Force systems. The program will help accelerate current research and direct it toward meeting U.S. Air Force needs.

5.7 Logic Programming Technology

The software in many present defense systems, e.g., command and control, is strongly characterized by the manipulation of logical relations among data. This characterization will be increasingly apt in future systems. For example, there will be many kinds of autonomous systems that will need to conduct analysis and problem-solving activities normally performed by human operators (e.g., goal-pursuit and diagnosis), and there will be increasing demand for powerful logical support in human-interactive systems (e.g., natural language understanding, information dissemination and retrieval, and decision support). Present general-purpose programming languages can be used for such functions, but they involve the programmer in enormous amounts of detail and require repeated reinvention of fundamental operations required for symbolic and logical operations. Use of present methods is consequently costly and error-prone, with results that are highly dependent on the skills of programming experts.

The trend described is not limited to defense systems. Logic-intensive programming has been recognized as a basic characteristic of the next generation of computing systems, and the development of logic programming techniques and supportive computer architecture is a major objective in national programs such as Japan's Fifth Generation Computer Program and the DARPA Strategic Computing Program. Objectives of such programs include natural-language and image understanding, decision support, expert systems for design and diagnosis, intelligent database systems, and smart weapons.

A new generation of programming languages that are extremely well suited for logic programming has emerged from research in computer science and artificial intelligence. Examples include LISP, Prolog, LogLISP, and OBJ. These languages have been implemented in practical form and are being tried experimentally in a wide variety of applications. A commitment to Prolog was made at the beginning of the Japanese program, but it appears likely that some extension of the language will occur as the program matures. The merits and limitations of various languages (e.g., flexibility, efficiency, and amenability to parallel processing) are currently being debated by researchers. It is clear that these languages greatly simplify the programming process in a wide domain of applications, with little loss in computational efficiency compared with conventional implementations. Several languages also appear to be easy to learn for persons with little technical training. These qualities represent a major positive step toward increasing the conceptual and expressive level of programming, which has enormous potential benefits for program reliability, maintainability, and economy.

The various languages are still at an early stage of development. The full power of the languages for general applications remains to be determined. Since the conceptualization of problems for these languages is different from that traditionally practiced for current languages, good programming support tools and a large body of examples will be needed to establish this mode of programming in general practice.

Software in defense systems is increasingly characterized by complex logical, non-numerical relations among data. This trend will be greatly accentuated in future systems for both autonomous and human-interactive behavior. Use of present programming languages for such applications is awkward and error-prone, and discourages direct programming by user experts. Languages and techniques that will simplify the task of creating logic-intensive programs for defense applications are and will be sorely needed.

A substantial program of research and development in the technology of logic programming is recommended. This research program should include development of (1) advanced languages, with consideration of generality, suitability for parallel processing, and efficiency of implementation; (2) programming methods, with emphasis on development of problem-solving approaches over a wide range of applications and system levels; and (3) computational support for programming (i.e., tool environments) and run-time execution (i.e., hardware support for special operations).

The following schedule of effort is recommended (in millions of dollars per year, over a period of seven years): 1.5, 2.0, 2.0, 4.0, 4.0, 4.0, and 2.0.

This plan assumes techniques development in years one-three, and a range of implementation and experimental application efforts in years four-seven. The program should be coordinated with other DoD efforts in the area.

Potential benefits to the U.S. Air Force are great. There is a possibility for a radical improvement in productivity, program quality and maintainability for future, complex systems.

The main uncertainty in success is whether new software methods will emerge from research laboratories to be accepted by defense contractors as widely applicable in defense software. If this does not occur, the investment will still have a positive though indirect impact in the ways systems are conceived and specified. The commitment by the Japanese and DARPA programs suggests that the risk in not pursuing this new methodology will be greater than in pursuing it.

Research is recommended in technology for logic programming. The goal is a major advance in ability to realize logically powerful autonomous and human-interactive functions in future defense systems. The effort would build upon recent successful results of research in computer science and artificial intelligence, and would benefit from impending national and international efforts in the subject.

5.8 Advanced Human Interfaces

The importance of human interface and the need to support the software engineering cognitive processes have often been ignored in software environments. Poor human interfaces and poorly designed (thus unused or misused) techniques and tools have contributed to the inherent difficulty in understanding complex software and resulted in excessive logical errors and attendant costs in software development and maintenance. Research is beginning to address the cognitive process involved in how humans program. The use of graphic interfaces and displays during system development and for debugging/testing is progressing rapidly. The use and misuse of stimuli, for example, color, in presenting visual displays is also currently being addressed. The use of natural, English-like language at the front end of the life cycle has been discussed widely and is covered under the Requirements and Specification Technology area. Of interest in this area is the progress being made in the area of Knowledge-Based Systems and logic programming. These areas have considerable potential for developing the intelligent companion for the requirements writer, specifier, implementer, tester, and maintainer, as well as for training, thus having impact on all phases of the life cycle.

Productivity of software developers and reliability of software systems are limited by the difficulty in understanding complex relationships in software. Text-based software representations in present software development systems place a heavy cognitive load on software developers and maintainers.

Technological recommendations that are designed to alleviate these poor human interfaces include the following:

1. Cognitive interface analysis;
2. Graphic display, color, and verbal analysis/synthesis;
3. Research on the use of natural (English-like) languages as input/ output media;

4. Monitoring, detection, and provision of response to human errors; and
5. Development of intelligent systems to serve as user co-worker.

In the near term, it is neccessary to determine appropriate use of graphic displays, color, and in the mid-term, verbal input/output. Graphic displays may be used to aid the software engineer in error-detection analysis, data flow, module interface, and for training purposes. Detection and monitoring of common human errors to allow the forgiving of these errors and an associated friendly environment would improve user adaptation to a new technique/tool and enable the user to experiment more with the same. Future system design and development should incorporate an intelligent co-worker to aid in decision making and improve the human/machine interface. Studies on the use of a natural, English-type language as a medium for input on the early phases of the life cycle to provide a natural interface to tools should begin in the mid-term. A far-term goal is to develop the intelligent co-worker, which accepts as input a well-defined, English-type language and interacts with the user in that language.

In order to implement and improve the human interface, a research effort should be initiated to address human response to visual stimuli (e.g., graphics and color) for assisting in system development and maintenance. Data should be collected on common human errors in order to establish a users forgiving environment (an environment in which human errors can be easily and quickly recovered from without loss of work). Software specification, design, debugging, and testing could be enhanced through the use of a system tracing interface allowing friendly, easy developer/tester interaction and feedback. As new methodologies and interfaces are developed, they should be used in the development of computer-aided instruction materials for these techniques. As a long term goal, these techniques can be applied to aid the user and system developer in the entire system development, from the initial conception of the need to final product. The entire process may draw on such items as a natural-language interface, consistency/error checking, and logical analysis following a user model established from early data collection.

There is an initial need for data collection to determine the physical (graphic, color, and verbal) media and cognitive factors that serve an important role in human interface design. This data collection and research is expected to cost approximately $2 million. Research is needed on a natural-language interface and intelligent co-worker for the user-system designer to aid in the transfer of user requirements into system specifications and the eventual product. Cost for this research is expected to be approximately $3 million. Mid- and far-term research is needed in the area of cognitive patterns that users possess in order to predict possible user errors and allow for correction. Intelligent error prediction and correction coupled with the development of an intelligent co-worker would cost on the order of $5 million. From the beginning, a benefit will be gained as techniques and tools can be designed that are easier, faster to learn, and allow easier recovery from errors. Ease of use plus error recovery capabilities provide user confidence in the technique/tool, which allows for faster and more enhanced technique/tool use. Understanding of complex software will be aided by automated, intelligent co-worker ability to detect and prevent user errors. Software maintenance costs will then decrease dramatically as these intelligent workstations are networked together under the model presented for common distributed software support.

The risk involved in the initial stages of human interface research is low in comparison to the gain from human interface enhancement. Risk over the long-term increases as the system evolves to an automated intelligent co-worker system in which some ability to determine needs, errors, and thought expectations is required.

The importance of human factors has been overlooked in the field of computer science and is just beginning to receive its due attention. Interface design to enhance understanding and assist in cognitive processes will speed system development and aid in its correctness. Techniques and tools, the design which is based on knowledge of cognitive thought processes and other human factors, will be easier and faster to learn, reducing cost and increasing productivity.

5.9 Upgrading Software Engineering and Management Education/Training

The odds are that capable software personnel will remain in short supply for the foreseeable future. While the absolute supply may well increase, the continuing growth of computing in its broad aspects is likely to absorb the supply, no matter how rapidly it increases. In addition, there is a danger that new techniques and tools included in project plans will result in time delays and poor technique/tool utilization due to a lack of knowledge of the new technology.

There are technological opportunities to improve training environments for software managers. Corporations and, to some extent, academic institutions already use game-playing situations and simulation-based training aids to develop executive managers. Such techniques can obviously be used, particularly in conjunction with Computer Aid Instruction (CAI) systems, to facilitate training of software acquisition managers.

Currently the advanced technology developed by industry and academia is not being exploited to its fullest by the U.S. Air Force, with the resultant loss of information and outdated technology used in U.S. Air Force systems development and maintenance. Likewise, U.S. Air Force needs are not being transmitted to software tools and technique developers, hence the lack of match of training systems to the needs and expectations of the U.S. Air Force. Current techniques and technological advances in the area of cognitive human interface are not incorporated into training material to speed and improve technique/tool training. Curricula development, teaching techniques, and educational training path are not being adequately explored.

Implementation of these recommendations in the near term will require selection of several locations to enhance industry, government and academia information exchange, with a mid-term goal of actual personnel exchange as an objective. Case studies should be collected to serve as training examples and improve academia and industry understanding of U.S Air Force needs. Metric development must begin to aid in estimating time for training on the use of new techniques/tools. Use of available media for training is needed as training programs should not be limited to written material but include videotape, cassette, and computer training.

The mid-term training programs should include available current technology, including computer-aided instruction programs and other media available. These programs should be designed using the results from the advanced human interface analysis. Research at this point in advancing the technology involved in training is needed. Interactive CAI programs with an intelligent base to guide the trainee in learning new concepts is desired. Metering tools to evaluate trainee progress should also be incorporated.

With regard to training software managers: While the general principles of automated approaches to training are understood, and while necessary physical equipment exists, it will take an R&D effort to assemble a complete system tailored to the software-management context. It is therefore recommended that an R&D effort be undertaken that will eventuate in an automated simulation-based training complex for software acquisition managers.

Such a facility can have broader application. By changing the knowledge base within the system, it can be used to train, for example, managers for other specialties or levels of management.

Such a research and development effort need not start completely from zero. There are automated training aids -- for planning and budgeting, for example -- that might be exploited as part of an overall system. Moreover, knowledge-based components are already functioning in gaming environments, and such features could well be incorporated into the facility here suggested.

Educational cost to implement these recommendations is expected to be approximately $20 million. There is an initial near-term cost in training the trainers to present each new technique/method and in material preparation. This effort could be expected to cost approximately $5 million. The mid-term will require similar funding for trainee programs and will include development funding in order to begin the design of intelligent computer aided instruction programs and interactive training sessions. The mid-term cost could be expected to increase, therefore, to $7.5 million. The same amount will be needed in the far term as training materials are developed and reused. Continued research should improve and enhance the systems requiring extensions to the existing system. With such training a new method, technique, or tool may be efficently used, and rapidly incorporated into project development. Acceptance of the new technique or tool will occur much more rapidly with appropriate instruction and training. Development of intelligent computer-aided instruction programs will reduce cost in training time and enhance users' understanding of the new technique/tool due to 'hands on' experience during training.

The tools developed for CAI designers will aid in design/production time (hence reduce cost) and assist in incorporation of new features such as new human interface ideas and stimuli analysis. The tools will also allow trainee progress analysis and evaluation. Near-term risk involved in training is exceptionally low, as without such training a technique/tool may not be globally accepted or appropriately used by U.S. Air Force personnel.

Techniques and tools designed to aid the manager, developer, designer, tester, trainer, or user of a system will be ineffective if not taught in such a way that the technique/tool use and importance is understood. It is imperative that advanced training techniques be developed to allow self-paced instruction with intelligent feedback on progress and errors.

6.0 REFERENCES

1 National Academy of Sciences. Operational Software Management and Development of U.S. Air Force Computer Systems. Committee on Operational Software Management and Development of U.S. Air Force Computer Systems. Washington, D.C.: 1977.

2 The Armed Forces Communications and Electronics Association (AFCEA) Command and Control Systems Acquisition Study, Final Report. 1 September 1982.

3 Martin, J., Application Development Without Programmers. Prentice Hall: Englewood Cliffs, N.J. 1982.

4 Roberts, A. J. "Some Software Implications of C^2 System Acquisition," Signal, 19-25. July 1982.

5 Ackerlund, Col. E. T. "Wrong Stuff," (Briefing presentation to the Summer Study.) 1983.

6 Wolverton, R. "Management Impact of Programming Measurement Systems," (Briefing presentation to the Summer Study).

7 "Software Engineering Automated Tools Index." Software Research Associates, P.O. Box 2432, San Francisco, CA 94215. (Loose-leaf catalog periodically updated.)

8 Personal discussion with Col. Edward J. Simmons, Jr., formerly the software specialist with the IUS SPO.

9 Thayer, R.H. "Rome Air Development Center R&D Program in Computer Language Controls and Software Engineering Techniques," RADCTR-74-80, Griffiss Air Force Base, Rome, New York. 1974.

APPENDIX A: BRIEFERS to the COMMITTEE

Edward T. Akerlund, AFSC/ALM, Air Force Systems Command
Peter Belford, Computer Sciences Corporation (CSC)
Wayne Bodle, U.S. Army Computer Systems Command, Ft. Belvior
Israel Caro, AFWAL/AAAF, Wright-Patterson Air Force Base, Ohio
Jack Clemons, IBM
George Cox, Intel Corporation
Alan Davis, GTE
Jack Dennis, MIT Laboratory for Computer Science
Samuel DiNitto, RADC, Griffiss Air Force Base, New York
Dennis Doe, Boeing Aerospace Company
Loraine Duvall, IITRI
Don Dwiggins, LOGICON
Carl Engelman, The MITRE Corporation
Joseph Fox, Software A&E, Inc.
Carolyn Gannon, General Research Corporation
John Garman, NASA
C. Cordell Green, Kestrel Institute
James Hess, USA/DARCOM
Kenneth Johansen, Boeing Computer Services Company
Anthony Jordano, IBM Federal Systems Division
Jan Komorowski, Harvard University, AIKEN Computation Laboratory
Frank LaMonica, RADC/COEA, Griffiss Air Force Base, New York
Robert Larson, Optimization Technology Incorporated (OTI)
Mark Linton, University of California-Berkeley
Owen Macomber, USN/NAVMAT/08Y
James Miller, Computer * Thought
Donald Mnichowicz, GTE AE R&D
Leon Osterweil, University of Colorado
Alton Patterson, SMALC
Ben Prince, BCS Richland, Inc.
Larry Putnam, Quantitative Software Management, Inc.
Art Pyster, Private Consultant
Fred Reagor, General Dynamics
Debra Richardson, University of Massachusetts
William Riddle, Software Design and Analysis
Nick Roussopoulos, University of Maryland
William Rzepka, RADC, Griffiss Air Force Base, New York
Elliot Soloway, Yale University
Michael Smith, University of Texas
Edwin Stear, University of Washington
Sam Steppel, Computer Sciences Corporation
William Sweet, GTE Sylvania Systems Group
Warren Teitelman, Xerox Parc
Ken Thompson, Bell Telephone Laboratories
Woodie Vandever, Higher Order Software, Inc.
Richard Waters, MIT

Herbert Weber, Universitat Bremen, Federal Republic of Germany
Gio Wiederhold, Stanford University
Roy Williams, IBM
Ray Wolverton, ITT Programming and Applied Technology
Stephen Yau, Northwestern University
Nicholas Zvegintzov, private consultant

APPENDIX B: BIBLIOGRAPHY

A Directory of Computer Software and Related Technical Reports (PB80-110232) Springfield, Va.: Computer Products Support Group, National Technical Information Service (1980).

Alford, M. Distributed Computing Design System, Final Report. TRW Electronics and Defense Sector. 21 June 1983.

Alford, M. Distributed Computing Design System Description. Technical Report, DCDS Description. TRW Defense and Space Systems Group. 7 August 1981.

Ball, J. Eugene, et al. "The Spice Project," CMU Computer Science Research Review. 1980/81.

Bonus for Computer Engineers. Paper brief by AFSC/MP to AFSC/CV.

Booch, Grady. Software Engineering with Ada. Menlo Park, CA: Benjamin Cummings Co., 1983.

Calvery, A.L. and G.M. Lawrence. Software Standards System (S^3) Characterization of SM-ALC ECS Support. SM-ALC/MME, McClellan Air Force Base, CA: Revised 3 May 1983 [Charts].

Committee on Human Factors. Research Needs for Human Factors. National Research Council. National Academy Press: Washington, D.C. 1983.

"Compiler and Tool Set for ADA Design and Implementation," ADA Compiler Circle Reader Service. January 1983, pp. 90-94.

Computer Science Technical Report Services. University of Maryland, Technical Report 1161, February 1983.

Defense Electronics (ISSN0194-7885). Volume 15, Number 1. E. W. Communications. Palo Alto, CA: January 1983.

Defense Science Board. Task Force on Embedded Computer Resources (ECR) Acquisition and Management. Final Report. Washington, D.C. 1982.

Department of the Air Force. Letter reply on "New Speciality Codes for Computer Resource Related Disciplines." No date.

Department of the Air Force. "Management of Computer Resources in Systems." AFSC Supplement 1, dated 14 December 1982.

Department of Defense. Candidate R&D Thrusts for the Software Technology Initiative. May 1981.

Department of Defense. Software Technology for Adaptable, Reliable Systems (STARS) Functional Task Area Strategy for Acquisition. 30 March 1983.

Department of Defense. Software Technology for Adaptable, Reliable Systems (STARS) Functional Task Area Strategy for Application Specific. 30 March 1983.

Department of Defense. Software Technology for Adaptable, Reliable Systems (STARS) Functional Task Area Strategy for Human Engineering. 30 March 1983.

Department of Defense. Software Technology for Adaptable, Reliable Systems (STARS) Functional Task Area Strategy for Human Resources. 30 March 1983.

Department of Defense. Software Technology for Adaptable, Reliable Systems (STARS) Functional Task Area Strategy for Measurement. 30 March 1983.

Department of Defense. Software Technology for Adaptable, Reliable Systems (STARS) Functional Task Area Strategy for Project Management. 30 March 1983.

Department of Defense. Software Technology for Adaptable, Reliable Systems (STARS) Functional Task Area Strategy for Support Systems. 30 March 1983.

Department of Defense. Software Technology for Adaptable, Reliable Systems (STARS) Joint Task Force Report. 15 March 1983.

Department of Defense. Software Technology for Adaptable, Reliable Systems (STARS) Implementation Approach. 15 March 1983.

Department of Defense. Software Technology for Adaptable, Reliable Systems (STARS) Program Management Plan. 15 March 1983.

Department of Defense. Software Technology for Adaptable, Reliable Systems (STARS) Program Strategy. 15 March 1983.

Doane, Robert B. Lessons Learned Program. Letter Paper dated August 1982.

Druffel, Larry. Software Engineering Notes. ACM SIGSOFT, Vol. 8, No. 2, April 1983.

Electronic Systems Division. AFSC Lessons Learned. Volume II. RCS: SYS-SDD.

Financial Systems Redesign - Development Guide. 31 August 1982. Revised 15 December 1982.

Fox, Joseph M. Software Acquisition. 15 July 1983.

Giese, Clarence. GITPASE: An Interactive Planning Aid for Project Scheduling with Time-Resource Tradeoffs. Donovan Young & Ronald L. Rardin.

Giese, Dr. Clarence. Personnel Management for Mission Critical Software Systems. Unpublished paper. July 13, 1983.

Green, Cordell et al. Report on a Knowledge-Based Software Assistant. Kestrel Institute, dated 15 June 1983.

Groves, Bill. "Getting VHSIC into Real World Systems". Defense Electronics, January 1983, pp. 102-111.

Gypsy Verification Environment and Boyer/Moore Theorem Prover. Institute for Computing Science, The University of Texas at Austin.

Hibbard, Peter. Document Presentation Facilities for Spice. Carnegie-Mellon University, Computer Science Departments, 1983.

JLC Position on the Software Initiative. (A Combined Position of the JPCG-CRM and the JDL).

Johansen, Ken. The Boeing Embedded Software Standard. July 13, 1983.

Lax, Peter D. Large Scale Computing in Science and Engineering, National Science Foundation: Washington, D.C. 26 December 1982.

Lessons Learned Notebook. Computer Hardware/Software Acquisition. 1 April 1980.

Martin, Edith W. Stars (Software Initiative Program Implementation. Letter dated 31 March 1983.

Miller, Edward F., Jr. Requirements for the Test Specification Language for DCDS. Technical Report. San Francisco, Cal.: 20 May 1983.

Moore, Wayne Robert, et al. Concepts, The Journal of Defense Systems Acquisition Management, Autumn 1982. Volume 5, Number 4.

Patterson, Alton E. Air Force Integration Support Facilities: Their Total Utility. Presented at "NAECON 1980," Sacramento Air Logistics Center, McClellan Air Force Base, Cal.

Redwine, S. T. Jr., E. D. Siegel, and G. R., Berglass. Candidate R & D Thrusts for the Software Technology Initiative. Department of Defense: Washington, D.C. May 1981. MTIS/DTIC A102180.

Richardson, D. J. Symbolic Evaluation and Applications to Testing. Computer and Information Science, University of Massachusetts at Amherst.

Roussopoulos, N. Software Engineering Environment Support. Department of Computer Science, University of Maryland, College Park, MD 20742.

Siegel, E. D. Summary of Responses to the Software Technology Initiative Questionaire. The MITRE Corporation: May 1982. MTR - 82W00085.

SRO PSL/PSA Guide. PSL 17 Sept 1982, PSA 01 July 1982.

Stevens, M. Managing the Total Life Cycle. Doc 0012C Disk 0086A. July 14, 1983.

TRW. Distributed Computing Design System. Final Report, 21 June 1983.

TRW. Distributed Computing Design System. Technical Report, DCDS Description. 7 August 1981.

Ware, W. H. Avionics Software: Where are We? The Rand Corporation: September 1982.

Ware, W. H. Perspectives on Life-Cycle Support of Software. The Rand Corporation: 1 July 1983.

Ware, W. H., and R. L. Patrick. Perspectives on Oversight Management of Software Development Projects (N-2027-AF). The Rand Corporation: July 1983.

Wiederhold, G. The Database Design Process. Stanford University.

Wiederhold, G. Knowledge-Based Management Systems. A Method for the Design of Multi-Objective Database. July 1982.

Williams, R. H. Software Cost Parametrics: IBM/FSD User Requirements.

Williams, R. H. Owego Software Cost Engineering Overview Parametrics. 30 September 1982.

Yeh, R. T. et al. A Report on DoD's Software Technology Initiative.
Computer Science Technical Report Series, University of Maryland.

Zelkowitz, M. V.; R. Yeh; R. G. Hamlet; J. D. Gannon; V. R. Basili. The Software Industry: A State of the Art Survey. University of Maryland .

Zvegintzov, N. Maintenance Technology. 1983.

APPENDIX C: OUTBRIEFING OF THE 1983 SUMMER STUDY

The following briefing slides were used by the Committee Chairman to present the results of this study to General Thomas Marsh, Commander, Air Force Systems Command, and other dignitaries at the outbriefing on 29 July 1983 held at the National Academy of Sciences' Woods Hole Study Center, Woods Hole, Massachusetts.

TASK STATEMENT

- PROVIDE TECHNOLOGY BASE AND MANAGEMENT RECOMMENDATIONS TO THE AIR FORCE WITH REGARD TO:
  - SOFTWARE LIFE CYCLE COST REDUCTION
  - SOFTWARE QUALITY IMPROVEMENT
- IDENTIFY A NEAR TERM SOFTWARE ENGINEERING ENVIRONMENT CONSISTING OF OFF-THE-SHELF TOOLS AND TECHNIQUES WHICH MAY BE IMPLEMENTED, AS IS, OR WITH MINOR MODIFICATION
- PROVIDE TIME PHASED RECOMMENDATIONS FOR MAJOR MODIFICATION, DEVELOPMENT AND INTEGRATION OF IDENTIFIED ELEMENTS LEADING TO A LONGER TERM COMPREHENSIVE ENVIRONMENT

SOFTWARE ENGINEERING ENVIRONMENT

INCLUDES

- AUTOMATED TOOLS FOR SPECIFICATION, DESIGN, DEVELOPMENT, TEST AND VALIDATION, OPERATION AND MAINTENANCE
- SPECIFICATION, DESIGN AND IMPLEMENTATION LANGUAGES
- PREDICTIVE MODELS FOR COST, SCHEDULE, RELIABILITY
- STANDARDS FOR DOCUMENTATION AND QUALITY CONTROL
- MANAGEMENT TOOLS AND PROCEDURES DIRECTED TO/FROM THE SOFTWARE ENGINEERING ENVIRONMENT STRUCTURE

NEAR TERM ENVIRONMENT
(2-4 YEARS)

ISSUE:

- ACHIEVABLE GAINS IN PRODUCTIVITY AND QUALITY FROM OFF THE SHELF TOOLS AND TECHNIQUES ARE NOT INSURED

RECOMMENDATIONS:

- 'ENGINEER' SEVERAL 1ST GENERATION ENVIRONMENTS WHICH UTILIZE COMPATIBLE, PROVEN TECHNIQUES AND TOOLS
  - INITIATE TRAINING, DISTRIBUTION AND MAINTENANCE
  - ENFORCE AS THE MINIMUM FUNCTIONAL STANDARD

MID TERM ENVIRONMENT
(4-6 YEARS)

ISSUE:

- SUBSTANTIAL INCREASES IN PRODUCTIVITY AND QUALITY DEPEND ON 2ND GENERATION, COMPREHENSIVE AND INTEGRATED ENVIRONMENT(S) BASED ON THE CURRENT LIFE CYCLE MODEL

RECOMMENDATIONS:

- CONCENTRATE ON REQUIREMENTS AND SPECIFICATION PHASES
  - REQUIREMENTS EXPRESSION LANGUAGES
  - REQUIREMENTS ANALYSIS AND VALIDATION TOOLS
- INCREASE COMPUTER AIDED TEST DESIGN AND ANALYSIS
- INTEGRATE INTELLIGENT DBMS INTO ENVIRONMENT(S)
- INTRODUCE DISTRIBUTED SOFTWARE ENGINEERING WORK STATION CONCEPT

FAR TERM ENVIRONMENT
(6-15 YEARS)

ISSUE:

- INCREASED COMPLEXITY OF AIR FORCE MISSION (PERFORMANCE, RELIABILITY, FLEXIBILITY, DEPLOYMENT) WILL RENDER NEAR/MID TERM ENVIRONMENTS INADEQUATE
- ADVANCED TECH BASE REQUIRED TO ACHIEVE AN ORDER OF MAGNITUDE IMPROVEMENT OVER CURRENT PRODUCTIVITY AND QUALITY

RECOMMENDATION:

- INCREASE AUTOMATION OF THE SPECIFICATION, DESIGN, IMPLEMENTATION, TESTING AND VALIDATION PROCESS THROUGH INTEGRATION OF KNOWLEDGE BASED EXPERT SYSTEMS AND AI TECHNIQUES
- EXPLORE ALTERNATE DEVELOPMENT LIFE CYCLE MODELS