AD-E301008
UNCLASSIFIED
Copy 17 of 54 copies
DTIC FILE COPY
(2)
AD-A197 504

IDA PAPER P-1926

SDI SOFTWARE TECHNOLOGY PROGRAM PLAN

Cathy Jo Linn
Samuel T. Redwine, Jr.
Michael I. Bloom
Bill Brykczynski
John Chludzinski
Jeff Grover
Deborah Heystek
Michael R. Kappel
J. Michael Lake
R. J. Martin
Sarah H. Nash
John Salasin

DTIC
ELECTE
AUG 3 0 1988
S D
E

June 1987

Prepared for
Strategic Defense Initiative Organization (SDIO)

This document has been approved for public release and sale; its distribution is unlimited.

INSTITUTE FOR DEFENSE ANALYSES
1801 N. Beauregard Street, Alexandria, Virginia 22311

UNCLASSIFIED
88 8 30 045
IDA Log No. HQ 86-030962

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE AD-A197504

1a REPORT SECURITY CLASSIFICATION Unclassified	**1b RESTRICTIVE MARKINGS**
2a SECURITY CLASSIFICATION AUTHORITY	**3 DISTRIBUTION/AVAILABILITY OF REPORT** Approved for public release: distribution unlimited.
2b DECLASSIFICATION/DOWNGRADING SCHEDULE	
4 PERFORMING ORGANIZATION REPORT NUMBER(S) IDA Paper P-1926	**5 MONITORING ORGANIZATION REPORT NUMBER(S)**
6a NAME OF PERFORMING ORGANIZATION Institute for Defense Analyses **6b OFFICE SYMBOL** IDA	**7a NAME OF MONITORING ORGANIZATION** OUSDA/DIMO
6c ADDRESS (City, State, and Zip Code) 1801 N. Beauregard St. Alexandria, VA 22311	**7b ADDRESS (City, State, and Zip Code)** 1801 N. Beauregard St. Alexandria, VA 22311
8a NAME OF FUNDING/SPONSORING ORGANIZATION Strategic Def. Init. Org. **8b OFFICE SYMBOL (if applicable)** SDIO	**9 PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER** MDA 903 84 C 0031
8c ADDRESS (City, State, and Zip Code) SDIO, Pentagon BM/C3, 1E149, Washington, D.C. 20301-7100	**10 SOURCE OF FUNDING NUMBERS** PROGRAM ELEMENT NO. / PROJECT NO. / TASK NO. T-R5-422 / WORK UNIT ACCESSION NO.
11 TITLE (Include Security Classification) SDI Software Technology Program Plan [redacted]	
12 PERSONAL AUTHOR(S) CLinn, Redwine, Bloom, Brykczynski, Chludzinski, Grover, Heystek, Kappel, Lake, Martin, Nash, Salasin	
13a TYPE OF REPORT Final **13b TIME COVERED** FROM _____ TO _____	**14 DATE OF REPORT (Year, Month, Day)** 1987 June **15 PAGE COUNT** 470
16 SUPPLEMENTARY NOTATION	
17 COSATI CODES FIELD / GROUP / SUB-GROUP	**18 SUBJECT TERMS (Continue on reverse if necessary and identify by block number)** Strategic Defense Initiative Organization (SDIO); Battle Management Command, Control and Communications (BM/C3); Software Research and Development (R&D); Software R&D Technology Management Plan;

19 ABSTRACT (Continue on reverse if necessary and identify by block number)

IDA Paper P-1926 was prepared in response to a request from the Battle Management C3 office within the Strategic Defense Initiative Organization (SDIO) of the Department of Defense (DoD). The request was for a software technology program plan to define software research and development (R&D) efforts required by the SDI, and to provide the basis for integrating the SDIO software technology program with ongoing non-SDIO programs. This Paper emphasizes reviewing the ongoing software programs and plans within the DoD and academia. The reviews identify critical software technology areas required to meet the unique SDI requirements, and indicate priorities among the software technologies to meet attainability, productivity and reliability goals, as well as funtional performance objectives.

20 DISTRIBUTION/AVAILABILITY OF ABSTRACT ☒ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT. ☐ DTIC USERS	**21 ABSTRACT SECURITY CLASSIFICATION** Unclassified
22a NAME OF RESPONSIBLE INDIVIDUAL Dr. Cathy Jo Linn	**22b TELEPHONE (Include area code)** (703) 824-5520 **22c OFFICE SYMBOL** IDA/CSED

DD FORM 1473, 84 MAR

83 APR edition may be used until exhausted
All other editions are obsolete

SECURITY CLASSIFICATION OF THIS PAGE

UNCLASSIFIED

18. SUBJECT TERMS Continued.

Office of Technology Assessment; Defensive Technologies Study Team (DTST); BM/C3 and Data Processing; Ballistic Missile Defense (BMD).

IDA PAPER P-1926

SDI SOFTWARE TECHNOLOGY PROGRAM PLAN [redacted]

Cathy Jo Linn
Samuel T. Redwine
Michael I. Bloom
Bill Brykczynski
John Chludzinski
Jeff Grover
Deborah Heystek
Michael R. Kappel
J. Michael Lake
R. J. Martin
Sarah H. Nash

Accession For	
NTIS GRA&I	☒
DTIC TAB	☐
Unannounced	☐
Justification	
By	
Distribution/	
Availability Codes	
Dist	Avail and/or Special
A-1	

June 1987

INSTITUTE FOR DEFENSE ANALYSES

Contract MDA 903 84 C 0031
Task T-R5-422

DRAFT

Preface

This document was prepared as the second response to a request from the Strategic Defense Initiative Organization (SDIO) for a software technology program plan that would provide guidance plus the means for integrating the SDIO software technology program with ongoing non-SDIO software programs. The first response was an SDI Preliminary Software Technology Program plan draft dated 2 July 1986. These plans are part of SDIO tasking the Institute for Defense Analyses with:

(1) Reviewing the ongoing software programs and plans within the Services, Department of Defense (DoD), industry, and academia;

(2) Performing a review and technical assessment of the SDIO software program;

(3) Identifying SDIO-unique software requirements and deficiencies within the SDIO software program;

(4) Identifying critical software technology areas required to meet the unique SDI requirements; and

(5) Prioritizing SDIO software technology required to meet attainability, productivity and reliability goals as well as functional performance objectives.

This plan emphasizes points one, four, and five. It identifies the software technology areas critical to SDI requirements, reviews the major U. S. government, industry, and university software projects in these areas, and presents a list of recommendations for each area. It provides a systematic plan, including dealing with priorities across the areas. Only Battle Management/Command, Control, & Communication (BM/C3) related technology is covered in this plan, although other elements of an SDI system such as weapons and sensors would have software components as well and should be included in a comprehensive plan.

This version reflects more feedback from the SDIO and other government reviewers and sets priorities across software technology areas. It also reflects the 1987 SDIO Software Technology Program Plan as it had been formed during the summer of 1986.

The final version of this Plan should serve as guidance for planning, internal DoD project proposals, and revisions of Work Program Descriptions.

Comments on this Plan should be sent to:

Dr. Cathy J. Linn
Institute for Defense Analyses
1801 N. Beauregard Street
Alexandria, VA 22311

or by phone:
(703) 824-5520

or via ARPANET:
CLINN @ADA20.ISI.EDU

Acknowledgments

The authors are indebted to a number of people for their assistance with the main body of this document. IDA reviewers who provided many helpful suggestions include Gil Berglass, Deborah Heystek, Michael Kappel, Ostop Kosovych, Mike Lake, Cathy McDonald, Terry Mayfield, Richard Morton, Al Perrella, Katydean Price, and Bob Turner. External reviewers who also made valuable contributions include Harlan Mills from IBM, Barry Boehm from TRW, Dick Martin from the Software Engineering Institute, Bill Riddle from Software Productivity Consortium, Sue Voigt from NASA, Sam DiNitto from Rome Air Development Center, Terry Courtwright from WIS JPMO, Larry Tubbs from the Strategic Defense Command, and Rich DeMillo from Georgia Institute of Technology. In addition, many thanks go to Tracy Poole, Joyce Walker, Jayne Rhiger, Michelle Null, and Elaine Watson for supporting the production and distribution of the document.

The cataloging of all the projects, their objective, and their funding in Appendix A was done by Sharon Null. This categorization of projects by recommendations could not have been done without her help.

The authors of Appendix B are indebted to a number of people for their help on this study. A list of the technical reviewers of each part of Appendix B follows. In addition, the authors of each section provided technical reviews for several other sections. John Salasin reviewed the sections in his role as Deputy Division Director and Samuel T. Redwine, Jr. in his role as principal author and task leader. Special thanks go to Elaine Watson and Katydean Price for their support in the preparation of the document.

BM/C3:

Vicki Huo
Bill Kilmer
Alex Levis
Terry Mayfield
Al Perrella
Bob Turner

Network Communications:

Ashok Agrawala
Clyde Roby
Don Towsley

Distributed Operating Systems:

Gil Berglass
Robert P. Cook
Clyde Roby
Alan Jay Smith

Distributed Data Management:

Arthur Keller
Terry Mayfield
Gio Wiederhold

DRAFT

Human Factors/Man Machine Interface:

Betsy Bailey
Gil Berglass
William Curtis
James Foley
Tom Kaczmark
Michael McGreevy

Parallel Processing

David Cheriton
Ken Kennedy
Joseph L. Linn

Hardware/Software Tradeoffs:

John Cuadrado
Geoffrey A. Frank
Daniel D. Gajski
Joseph L. Linn
Terry Mayfield
James Pennell
Marko Slusarczuk
Robert I. Winner

Software Engineering Environments:

Frank Lamonica
Joseph L. Linn
Richard Morton
Lee Osterweil
Bill Riddle
Christine Youngblut
Francoise Youssefi

Simulation and Evaluation:

Peter Dorato
Vicki Huo
Alexander Levis
Paul Roth
Bruce Schmeiser

Reliability and Survivability:

Joe Cavano
Richard DeMillo
Amrit Goel
John C. Knight
James McCall
Terry Mayfield
John D. Musa
A. Vito

DRAFT

People and Organizations:

Bob Aiken
John Bailey
Gil Berglass
Bill Curtis
Gary A. Ford
Norman E. Gibbs
Cathy McDonald
Terry Mayfield
James W. Murrell
Roger Sobkowiak
Jim Tomayko
Robert I. Winner

Technology Transition:

Joseph D. Morgan III
William E. Riddle
Susan Voigt

EXECUTIVE SUMMARY

Introduction

Software is projected to play an important part in the Strategic Defense Initiative (SDI) system. The Fletcher and Eastport study groups, among others, anticipate that the volume of this software will be greater than any to date, and its scope will be broad, spanning the spectrum of software technologies [Fletcher 84; Eastport 85]. Its requirements, in terms of size, reliability, testability, security, and difficulty, all push beyond the state of practice in software technology.

This document was prepared as a response to a request from the Strategic Defense Initiative Organization (SDIO) of the Department of Defense (DoD) for a Software Technology Program Plan to provide an overall plan for an SDIO software technology program and the means for integrating the SDIO software technology program with ongoing non-SDIO software programs.

Objectives

The primary objectives of a SDI software technology Research & Development (R&D) program are to provide input to the decision of whether to build the SDI system and to develop the capability to provide the software to meet SDI requirements. Improvements in application-specific software technology, generic foundation software technology, and in the software engineering process are necessary to meet both of these objectives.

This plan has several purposes, including:

a. Identifying and categorizing research areas important to the SDI.

b. Providing a framework for describing relevant research efforts.

c. Providing guidance to services/agencies preparing research proposals to SDIO.

d. Serving as the starting point in the process of proposal and program assessment.

While the plan is not overly detailed, it does identify needed activities at a sufficient level of detail to serve as guidance for developing an FY 88 program.

Scope

This document identifies and describes needed R&D efforts in 12 areas that are important to the SDI system and describes a continuing process for managing and coordinating software technology research efforts. While the report describes the current state of research in each area, time and resource constraints make it impossible to ensure that this description is always exhaustive.

Early Decisions

The requirements imposed by the need to support the early 1990s full-scale engineering development decision are still unclear. Therefore, over the next six months, the SDIO should rapidly establish preliminary versions of:

a. Threshold software technology feasibility criteria.

b. Believability/decision criteria of the decision makers and influential people and organizations likely to be involved in the decision.

c. Nature of the demonstration(s) needed to convince these decision makers.

SDI system characteristics that must be overcome for a positive full-scale engineering development decision include its size, required speed, reliability, adaptability, and the DoD procurement process. Many people and organizations must work together to build the software which is of unprecedented size and complexity. Without ever having been fully tested, this software must work reliably and quickly to discriminate targets, fire weapons, communicate between assets, and survive enemy countermeasures. The software must adapt to changing conditions over time including new technologies and changes in the threat. The DoD procurement process requires improvements and streamlining before products and services of the required quality and innovativeness can be acquired in a timely and efficient manner. Any of these characteristics left unconquered might make building the SDI system infeasible.

Strategy and Assumptions

The overall SDI software technology R&D strategy is one of try, evaluate, and improve. This fits with the prototyping approach planned for building the SDI experimental versions, large-scale demonstrations, and the SDI system(s). The strategy will exploit current technology, build on existing activities, mix evolutionary and revolutionary technologies, build several generations, follow an evolving plan, and coordinate the collected expertise of people in many organizations. The SDI effort will build on and coordinate with the efforts of existing large software programs such as the Software Technology for Adaptable, Reliable System (STARS), Software Engineering Institute (SEI), Federal Aviation Administration (FAA), NASA Space Station, and WWMCCS (World Wide Military Command and Control System) Information System (WIS).

The open systems approach (as advocated by the Eastport Group) is essential for evolutionary development of the SDI system. Prototyping supports evolutionary development by incrementally improving the system. An important side benefit is the learning experience provided to the people and organizations building the prototypes as a part of the critical upgrading required. Prototyping will also be used in software technology R&D. These technology prototypes will flow into the system prototypes.

It is assumed that SDI will not invest in the design and development of new programming languages/systems unless there is convincing evidence of advantages to be gained. The large time and cost involved in language design and validation, in building basic support tools and environments, and in establishing an infrastructure to support configuration control, education, training, etc., make such evidence unlikely.

This report therefore assumes that SDI will use Ada* (or Common-LISP for Artificial Intelligence applications). Since clear advantages exist for building prototypes in the same language as a deployed system, it is also assumed that prototyping efforts will use Ada (Common-Lisp).

*Ada is a registered trademark of the U.S. Government, Ada Joint Program Office

Software Technology Task Area Findings

Appendix B contains the software technology R&D task area assessments, major results from which are summarized below. Figure 1 lists the technology task areas.

- Application- Specific
  - Battle Management/C3
- Foundations
  - Network Communications
  - Distributed Operating Systems
  - Data Management Systems
  - Human Factors/Man-Machine Interface
  - Parallel Processing
- Software Engineering Process
  - Computer Systems Engineering Technology
  - Software Engineering Environments
  - Simulation
  - Software Dependability
  - People and Organizations
  - Technology Transition

Figure 1. Software Technology Areas

Two common areas of concern appear in a number of task areas. One area involves systems requirements and design issues, and the other area is concerned with measurement and evaluation. The systems-related concerns include requirements' and specifications' notations, methods, and tools; rapid prototyping; algorithms; degree of decentralization; the role of humans; the distributed real-time nature of the system; standard interfaces; and security. Another cluster of concerns includes prototyping, evaluation, modeling, testing, verification, and measurement. In addition, every task area is concerned with accelerating the rate of improvement in the area and bringing the technology and its use to the states needed for the SDI full scale engineering development decision and the building of the SDI system.

Application-Specific

Battle Management/Command, Control, and Communications (BM/C3) [Appendix B2]

a. The SDIO should investigate the proposed architectures and identify where each architecture stresses existing software technology.

b. A glossary of terminology for each organization participating in the Phase III BM/C3 architecture and system study should be provided by the SDIO.

c. A Parallel Algorithm Test Center for research and evaluation should be established.

d. The SDIO should research parallel algorithms for multi-sensor data fusion.

e. The role that humans will play during the battle needs to be studied and prototyped.

f. Possible applications of artificial intelligence for SDI need to be investigated.

g. The types of support systems required for SDI and the National Test Bed (NTB) need to be identified.

Foundations

The generic foundation software requirements of the SDI system are centered around three different environments: (1) the platform environment, (2) the platform management (or Command, Control, and Intelligence, (C2I)) environment, and (3) the software engineering environment. Within each of these environments, varying requirements regarding performance, security, integrity, distribution, etc., will be found.

Network Communications [Appendix B3]

A variety of techniques currently exists for performing the required basic functions of a network communications system. The ability of these techniques to meet the dynamic reconfigurability, reliability, security, and real-time performance requirements of the SDI system must be determined. Therefore:

a. A commitment must be made to a protocol standard to achieve the desired interoperability among distributed computer communication networks and provide an open and extensible framework.

b. Algorithms for naming, routing, topology updating, and flow control must be developed to meet the reliability, security, reconfigurability, and real-time performance requirements of the SDI.

c. Appropriate models for security of computer communication networks must be developed early.

d. Prototype computer communications networks must be built and integrated with the prototype distributed operating systems being developed for the SDI.

e. Research the automatic generation of protocol software.

Distributed Operating Systems [Appendix B4]

a. To demonstrate feasibility and to obtain the technology and experience to build a large and robust system, the SDIO should fund the development of up to four prototype secure, real-time, fault-tolerant, and reliable operating systems.

b. Additional research projects in (1) modeling and analysis of distributed systems, (2) secure kernels, (3) rapid recovery and approximate recovery, and (4) real-time and dynamic scheduling/resource allocation must be tightly integrated with the development of these prototypes.

c. While the prototype developments will yield short-term results and aid in the full-deployment decision, intermediate to long-term results require widespread and convenient means of experimentation. The SDIO should, therefore, fund the development of testbeds for distributed systems and widely distribute state-of-the-art software for distributed systems.

Data Management Systems [Appendix B5]

a. Several government agencies (WIS, STARS, NASA Space Station, etc.) may be able to provide leverage for several database areas critical to the SDI. In addition, they may provide good opportunities for performing some of the research and development that will be needed in developing the various prototypes outlined below.

b. The data management of the space-based platform environment will primarily be implemented as high performance algorithms operating on high speed memory. Research is needed to understand this function. Prototypes of these algorithms should be constructed soon to determine critical areas where necessary software and hardware technology is lacking so that these can be addressed.

c. Much of the current database research activity will be applicable within the platform management environment. However, prototype distributed database management systems need to be developed early to identify areas where technology is lacking (e.g., data models, error recovery, etc.).

d. Within the software engineering environment (SEE), some research is needed in developing a fully integrated object management system for storing objects such as test data, source code, output results, etc. Research is needed to define the role the SEE database will play, as well as the definition of interfaces, amount of data to store, the role of active components, the level of distribution, what types of security are necessary, etc.

Man-Machine Interface [Appendix B6]

a. SDIO should continue its support of the SDI Human Factors Program at the USAF Electronics System Division.

b. Because of the potential utility of a user interface that employs artificial intelligence techniques, SDIO should monitor its state of the art and, in particular, the work sponsored by Defense Advanced Research Projects Agency (DARPA).

c. SDIO should monitor government development efforts for workstation managers, window managers, graphics packages, and user interface management systems.

d. SDIO should monitor current standardization efforts for graphics packages.

e. SDIO should survey commercially developed and research-oriented User Interface Management System (UIMS) for adaptation and integration into the strategic defense system.

f. SDIO should continue to fund the U. S. Army Strategic Defense Command's program for a Command and Control Decision Aids Test Environment.

Parallel Processing [Appendix B7]

The future of this technology will require further research and development of parallel processing engines, programming languages, compilers, operating systems, and algorithms. More specifically this should include:

a. Research of techniques for expressing parallelism, both explicitly and implicitly.

b. Development of Ada compilers targeted to representative classes of parallel architectures.

c. Development of operating systems for different parallel engines,

d. Development of facilities for testing and evaluating algorithmic behavior on particular parallel engines.

Software Engineering Process

Computer Systems Engineering Technology [Appendix B8]

The system requirements of the SDI indicate the need for a design approach that integrates hardware and software. This will require the development of design support environments and integrated toolsets. Input to the design system should be a system-level, implementation-independent specification generated from and traceable to a set of system requirements. Therefore it is recommended that the SDIO evaluate existing work, and either provide continued funding for this work or initiate projects leading to the development of:

a. A method of system-level requirements capture.

b. A system specification language/notation that is neutral with respect to eventual design implementations.

c. Automatic verification (design meets specification) capabilities.

d. A design environment that provides support for maintaining multiple design alternatives, multiple function to implementation assignments, multiple levels of design representation, and propagation of design changes across representations.

e. Advanced design aids such as automated system decomposition tools, tools to automate the assignment of functions to levels within the implementation technology hierarchy, simulators to assist in tradeoff analysis, and tools to permit transformation of representations such as microcode compilers and gate array compilers.

f. Policy decisions on the appropriate level of standardization in the areas of requirement and design specification, and engineering environments.

Software Engineering Environments [Appendix B9]

a. The SDIO approach to Software Engineering Enviornments (SEE) must recognize that several generations of SEE technology will occur during its lifetime and that development SEE(s) can be less standardized than SEE for test, evaluation, or maintenance. Early emphasis on standardization of software

work product deliverable interfaces rather than numerous internal SEE interfaces fits these nicely.

b. The SDIO needs to determine the appropriate levels of standardization over time. This should include early work on a strongly typed object management system.

c. The SDIO effort should position itself to benefit from several existing SEE efforts and maintain a posture that will be consistent with SEI and DARPA efforts in the longer term. The SDI effort should mainly be an evaluator, accelerator, addresser of special SDI needs, interface standardizer, and integrator. Indeed, the issue of SEE integration and extensibility should possibly be SDI's highest SEE research priority.

d. SDI should consider funding tools in areas such as methodologies, requirements development, rapid prototyping, artificial intelligence, visual programming formal verification, code generation/compiling, testing and metrics. The state of practice in most of these areas is not suitable for an SDI SEE.

Simulation [Appendix B10]

a. SDIO should examine the operational roles of the NTB and Experimental Version (EV) programs for possible redirection to plug gaps and eliminate redundancy.

b. SDIO should choose Ada as the standard simulation programming language. All new simulation software should be coded in Ada; exceptions require government approval.

c. SDIO should standardize an Ada Simulation Support Environment for SDI simulation and evaluation.

d. SDIO should fund basic research into techniques for validating models of large-scale systems.

e. SDIO should standardize a testing methodology for SDI software and hardware components.

Software Dependability [Appendix B11]

Dependability is a general term encompassing the many qualities of a system that relate to the ability to justifiably rely on its services. Qualities covered include reliability, availability, safety and maintainability. Dependability issues involving security and the distributed nature of the system are covered in other sections.

a. A maximal, comprehensive approach combining many techniques into a should be used in prototyping multi-teered defense against software failures.

b. The systems requirements and engineering activities should have strong software awareness and result in formal software requirements specifications.

c. SDIO must ensure the proper selection, development, organization and use of technology and personnel and characterize their error patterns.

d. Particular development and support technologies must be imporved: explore multiple specifcation and vaidation techniques; identify the error detection power of verification techniques; support the enhancement for use on SDI BM/C3 of technology for software testing, simulating faults and recording error propagation, and formal verification with Ada; and monitor automatic programming research for potential investment.

e. Quality measurement and modeling research should be supported and used on all BM/C3 software efforts.

f. Fault-tolerance R&D is needed at both the software and systems levels resulting in stable levels of machine abstraction and fault-tolerance services that application software can use straightforwardly.

People and Organizations [Appendix B12]

The projected size, difficulty, and criticality of the SDI effort will require large numbers of people and organizations performing at high levels. To meet this requirement, the SDIO should identify specific mechanisms for encouraging and ensuring this performance through contracted mechanisms and should fund relevant efforts -- to the extent practical through the software contractors themselves -- in:

a. A Human Resources Plan and upgrading of SDI software personnel.

b. An Organization Development approach to improve organizational effectiveness and health through planned interventions at the individual, team, project, organization, and interorganizational levels.

c. Research that identifies characteristics of excellent software-related personnel and measures their relative performance.

d. Research in matching workers to jobs and should monitor the application of AI to the selection process.

e. Empirical studies of certification programs to find out how effective such certification is in predicting job success.

f. Research that focuses on productive and excellent software organizations for characteristic clues as to what motivates them to be productive and excellent, including behavioral differences of people within innovative organizations.

g. Human-machine and computer-mediated communications research.

Technology Transition [Appendix B13]

Technology transition is the planned overt actions taken to improve one's technology and covers the entire process of transforming research results -- usually through a series of stages -- into usable forms and accomplishing successful use of them. The Eastport Group recommended that SDI *not* rely on breakthroughs for proving the feasibility of doing the BM/C3 software. This, of course implies a reliance on technology transition for achieving this proof.

a. Immediately, SDIO should:

(1) Emphasize establishing relationships with sources of technology.

(2) Begin to augment the efforts of the DoD Software Engineering Institute (the Federally Funded Research & Development Center (FFRDC) designated with the role of software technology transition promotion) to improve its coverage of SDI concerns.

(3) Establish the requirements for being ready for the early 1990's full scale engineering development decision.

(4) Develop the provisions to place in contracts to encourage and ensure transition including requirements for contractor Technology Insertion Plans.

b. In the near term, SDI should also address:

(1) Its technical strategies for transitioning technology, particularly standards and compatibility.

(2) The communication of its requirements to researchers.

(3) The technology evaluation capabilities for the different stages in the transition process.

(4) Its SEE area strategy.

(6) Plans for transitioning the different technologies.

c. In both the near and longer terms SDI should pursue actions, and research and experiments aimed at improving the transition of technology, observe their effectiveness and proceed accordingly.

Priorities and Emphases Across Task Areas

The budget for BM/C3 software technology is currently insufficient to support all the efforts that could be beneficially undertaken and this is expected to continue to be the case for at least the next several years. This and other considerations imply the need to consider the comparitive merit of efforts across task areas in addition to the consideration already given within each area.

Figure 2 attempts to combine all the considerations and list aspects in order of priority or emphasis down the lefthand side and the task areas involved across the columns. They are divided into five major levels (numbered 1-5) and in order within each level (as indicated by letter). The differences within levels are smaller than between levels. A single task area may have aspects or subareas at different levels of priority.

Broadly the aspects ending up near the top are ones having to do with understanding SDI BM/C3 and preparing to preceed.

The second level has the try, learn, and improve thrust; C2 interfaces since these are nearer to current systems than the battle management ones that are at the top level; unique applications because of the small chance others will solve their problems; and the key technical and credibility problem of software reliability.

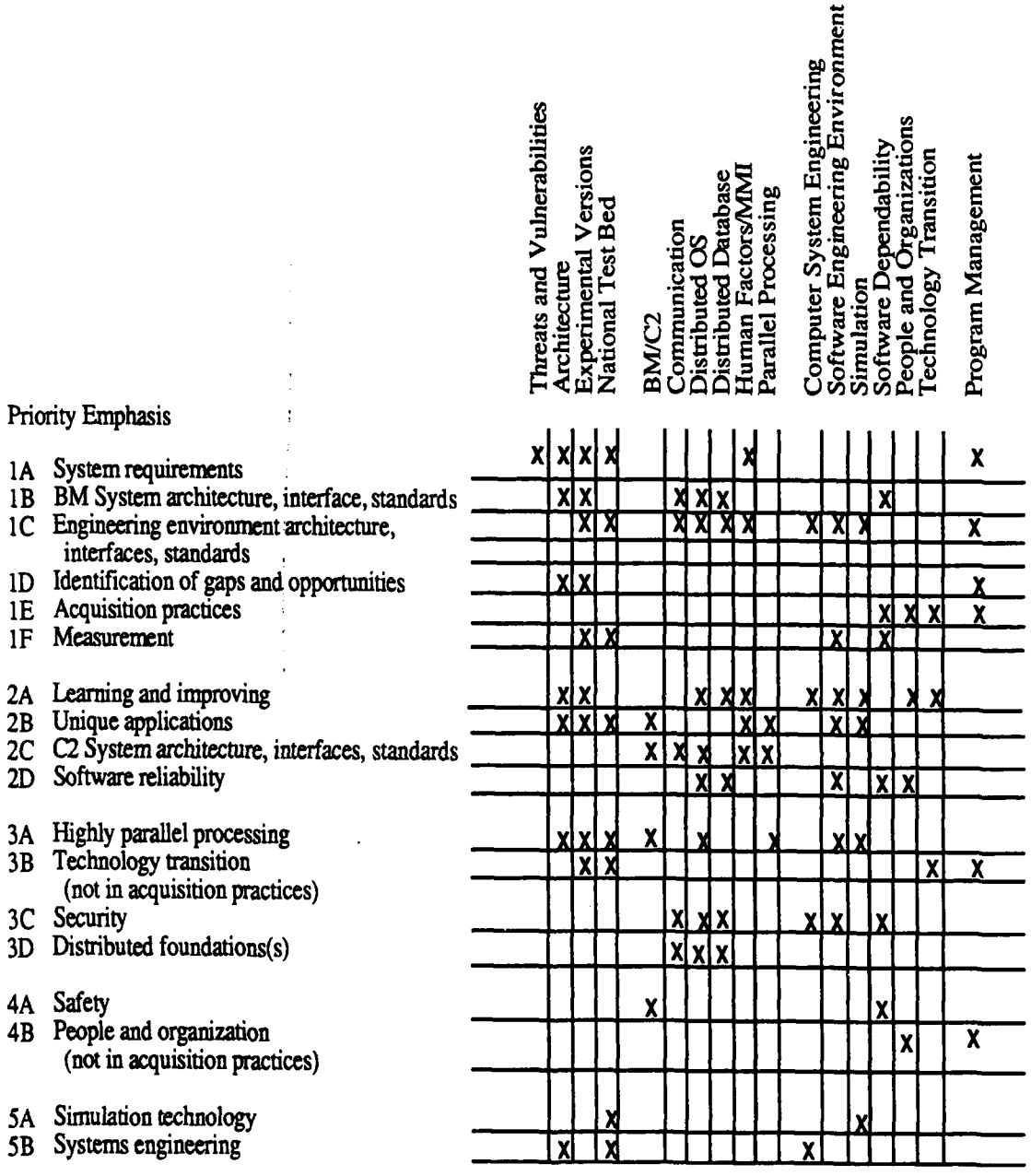


Figure 2. Priorities/Emphases Across Task Areas

The third level has three more important aspects: parallel processing, technology transition (outside the key contractors), security and distributed systems technology. The fourth level drops down substantially from the third and includes safety, and people and organization aspects outside the main contractors. The bottom level contains aspects that should be supported mainly outside the software technology sphere, systems engineering and simulation technology.

R&D Management Plan

The software technology R&D management plan, in conjunction with the prototyping/experimentation approach, pulls the technology areas together into a coherent program. It describes the general approach, roles of the SDIO, use of other organizations/programs, planning and control, quality assurance, and budget. It also recommends an annual cycle and management and reporting systems.

Three appendices conclude the report. Appendix A is a listing of projects the SDIO is currently funding categorized by task area. Appendix B constitutes the bulk of the report and contains the full software technology task area assessments, including SDI system requirements, current status of R&D, and recommendations. Appendix C contains the plan charts for each task area.

References

[Eastport 85] Cohen, D., et al., *A Report to the Director Strategic Defense Initiative Organization*, Eastport Study Group, Summer Study 1985.

[Fletcher 84] McMillan, Brockway, et al., *Report of the Study on Eliminating the Threat Posed By Nuclear Ballistic Missiles*, James C. Fletcher, Study Chairman, Volume V, Battle Management, Communications, and Data Processing, Brockway McMillan, Panel Chairman, February 1984.

1.0 INTRODUCTION AND BACKGROUND

This paper was prepared in response to a request from the Battle Management and C3 Office within the Strategic Defense Initiative Organization (SDIO) of the Department of Defense (DoD) for a software technology program plan to define software research & development (R&D) efforts required by the SDI and to provide the basis for integrating the SDIO software technology program with ongoing non-SDIO software programs. This is part of SDIO tasking to the Institute for Defense Analyses and emphasizes reviewing the ongoing software programs and plans within the Services, DoD, other federal government agencies, industry, and academia; identifying critical software technology areas required to meet the unique SDI requirements; and indicating priorities among the software technologies to meet attainability, productivity, and reliability goals as well as functional performance objectives.

1.1 Purpose and Scope of Document

The purpose of this document is to organize the SDI program's BM/C3 (Battle Management, and Command, Control and Communications) software R&D effort by presenting R&D recommendations and an R&D Management Plan. The document recommends research areas, emphases, R&D tasks, and partially identifies prime candidates for acquisition organizations (including U.S. government and foreign entities) for both the short and longer terms. It also recommends when to start longer-term R&D tasks. In the absence of an overall statement of SDI software requirements, this document postulates the SDI requirements for each technology area and sets R&D priorities based on these and programmatic objectives. Findings and recommendations made in this version of the plan may be revised as work progresses from year to year.

The Software R&D Technology Management Plan section prescribes a general approach for software technology R&D management, delineates the roles of the SDIO in software R&D management, and advocates using other organizations and programs to accomplish R&D. It describes planning and control, and quality assurance for software R&D management, and includes an overall budget.

1.2 Where BM/C3 and Software Fit into the SDI

Software is projected to play an important part in the SDI system. The Fletcher and Eastport study groups, among others, anticipate that the volume of this software will be greater than any to date, and its scope will be broad, spanning the spectrum of software technologies [McMillan 84; Eastport 85]. For these reasons, the SDI program needs a systematic software technology R&D plan. For economic reasons, this plan must be integrated with other existing and planned software technology efforts within the U.S. government. Sources that describe in detail the background of the SDI system include the Office of Technology Assessment reports and an article in IEEE Spectrum [OTA 85; OTA 85a; Adam 85].

A separate panel of the Defensive Technologies Study Team (DTST) [Fletcher 84] on Battle Management, Command, Control, and Communication, and Data Processing, headed by Brockway McMillan, studied the specific problem of Battle Management, Communications, and Data Processing. The report issued [McMillan 84], presents the panel's judgment of the critical problems and needs and outlines the program the panel recommended to address these needs.

The panel's report described the battle management system in terms of resources managed and functions performed, discussed issues related to the overall engineering and design of a battle management system, examined the rates of data flow and computation that might be encountered in a large BMD (Ballistic Missile Defense) system, addressed issues of software design and development, and discussed reliability and communication requirements. A final section outlined the panel's recommended technical program. The panel foresaw software design, development, and testing as one of the major engineering problems in the creation of an SDI system.

The SDI system is conceived as a multi-layered defense that destroys attackers in their boost, mid-course, and terminal phases. Three battle management functions are common to each layer of the defense. The first function is detection, acquisition, and tracking, which includes acquiring data on objects from multiple sensors, telling where objects not filtered out as uninteresting will be located at a given time, and keeping this information on objects. The second function, classification, determines what each object is. This requires discriminating reentry vehicles from balloons, decoys, and junk and adding this information to the track file. Another component of classification is kill or damage assessment. The third function, resource allocation, is a dynamic process of assigning sensors and weapons to targets. Part of this function is to produce a prioritized list of targets for each defending weapon platform [McMillan 84].

On a more global level, functions occur in all phases of battle or span multiple phases of battle management. The first of these functions, surveillance, takes output from a detection system and provides verification that an attack has been initiated and assesses the nature of the attack. The engagement function contains the doctrine for conducting the battle and selects appropriate responses to perceived threats. The delegation function passes control from operation to operation and coordinates various layers of defense. The mutual defense function coordinates and manages resources in the defensive system for survival of the system. Finally, the situation assessment function provides a current and ongoing assessment of the state of hostility and the status of the various defensive resources. It has three subfunctions: status of attacking forces, status of own forces, and presentation (assisting in presentation of an overload of information to command authority) [McMillan 84].

Software is integral to the performance of nearly all of these functions. Software is embedded in weapons and sensors and is responsible for boost-phase, mid-course phase, terminal phase, and overall battle management. In addition, many peacetime C3 and intelligence functions involving software must be performed to prepare and plan for battle, and to manage and maintain readiness.

1.3 The Problem

Many sources [Adam 85; Eastport 85; Elmer-DeWitt 85; Fletcher 84; McMillan 84; Marbach et al 85; Rensberger 85] acknowledge the importance and criticality of software to the SDI system. Its requirements, in terms of size, reliability, testability, security, and difficulty all push beyond the state of practice in software technology.

Figure 1 illustrates the problem. Meeting future SDI system requirements implies a required future software state of practice within the SDI effort. A potential gap between this future state of practice and future SDI requirements exists and must be closed to build the SDI with progress along the way supporting a full scale engineering development decision. Additionally, after initially building the SDI system, increasing requirements over time will necessitate further improvements to the state of the practice.

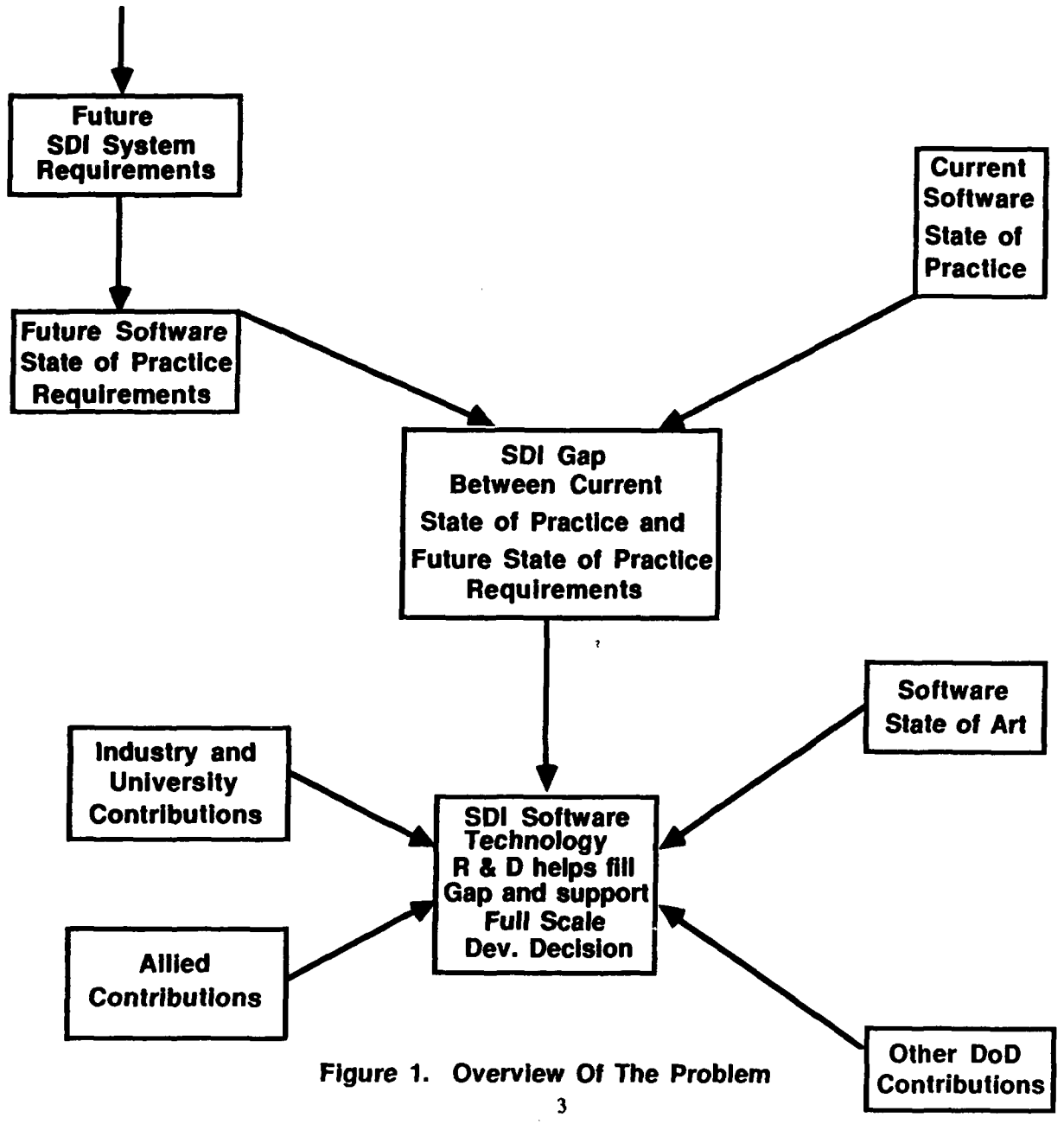


Figure 1. Overview Of The Problem

Boehm has made some rough, preliminary, and typical case estimates of effort, schedule, and person loading for SDI software development using the COCOMO (COnstructive COst MOdel) [Boehm 86, pp. 296-298]. For his estimated 20 million lines of operational code, Boehm estimates an effort of 900,000 man months, a development schedule of 17 years, and peak person loading of 7500 full-time equivalents to develop the SDI software. Thus, even ignoring the estimated 40 million lines of support code, the task could not be successfully performed unless steps are taken in preparation. He contends that certain actions can reduce these numbers, including using the results of software technology R&D, improving personnel, and adopting a spiral software process model with increasingly powerful prototypes/systems.

Finally, this preparation of technology and capability must be performed on a constrained budget. This necessitates careful attention to priorities despite the need for advances across a broad range of areas.

1.4 Organization of the Document

Section 2.0 of this document addresses objectives for SDI software technology R&D. These objectives are to gain information to make a decision whether or not to build the SDI as well as to achieve the capability to build an SDI system. Section 3.0 describes the strategy for R&D to develop the software and to develop the technology to design and implement the software. Section 4.0 summarizes the various technology R&D task areas examined in Appendix B. Information provided includes projected results of R&D in each area, what can be done with those results and how, why they are important, how the R&D supports the objectives outlined in Section 2.0, prime candidates to undertake the R&D, and issues of concern. Section 5.0 is an R&D management plan for SDI BM/C3 software technology.

Three appendices conclude the report. Appendix A shows what the SDI program is currently funding categorized by task area. Appendix B contains the full task area assessments, including SDI system requirements, current status of R&D, and recommendations. Appendix C contains plan charts for each task area.

2.0 OBJECTIVES

The primary objectives of a SDI software technology R&D program are to provide input to the decision of whether to build the SDI system and to develop the capability to provide the software to meet SDI requirements. Improvements in application-specific software technology, generic foundation software technology, and in the software engineering process are necessary to meet both of these objectives. Other considerations include improving understanding of requirements, how best to build an SDI system, and the value of the technology to other future DoD systems even if the SDI system is never built.

2.1 Full-Scale Engineering Development Decision

A full scale engineering development (FSED) decision involves both the technical feasibility of the BM/C3 software and the cost/benefit of the most attractive strategic defense system. Clearly, if the BM/C3 software is not technically feasible, then the decision should be negative (quit or delay). If the benefit of the most attractive feasible alternative does not outweigh its cost, then a negative outcome should also occur.

The decision to enter full scale engineering development involves several complex considerations. Figure 2 illustrates the time-phase relationships of the decision paths. Three primary considerations feed the full-scale decision: technical feasibility,

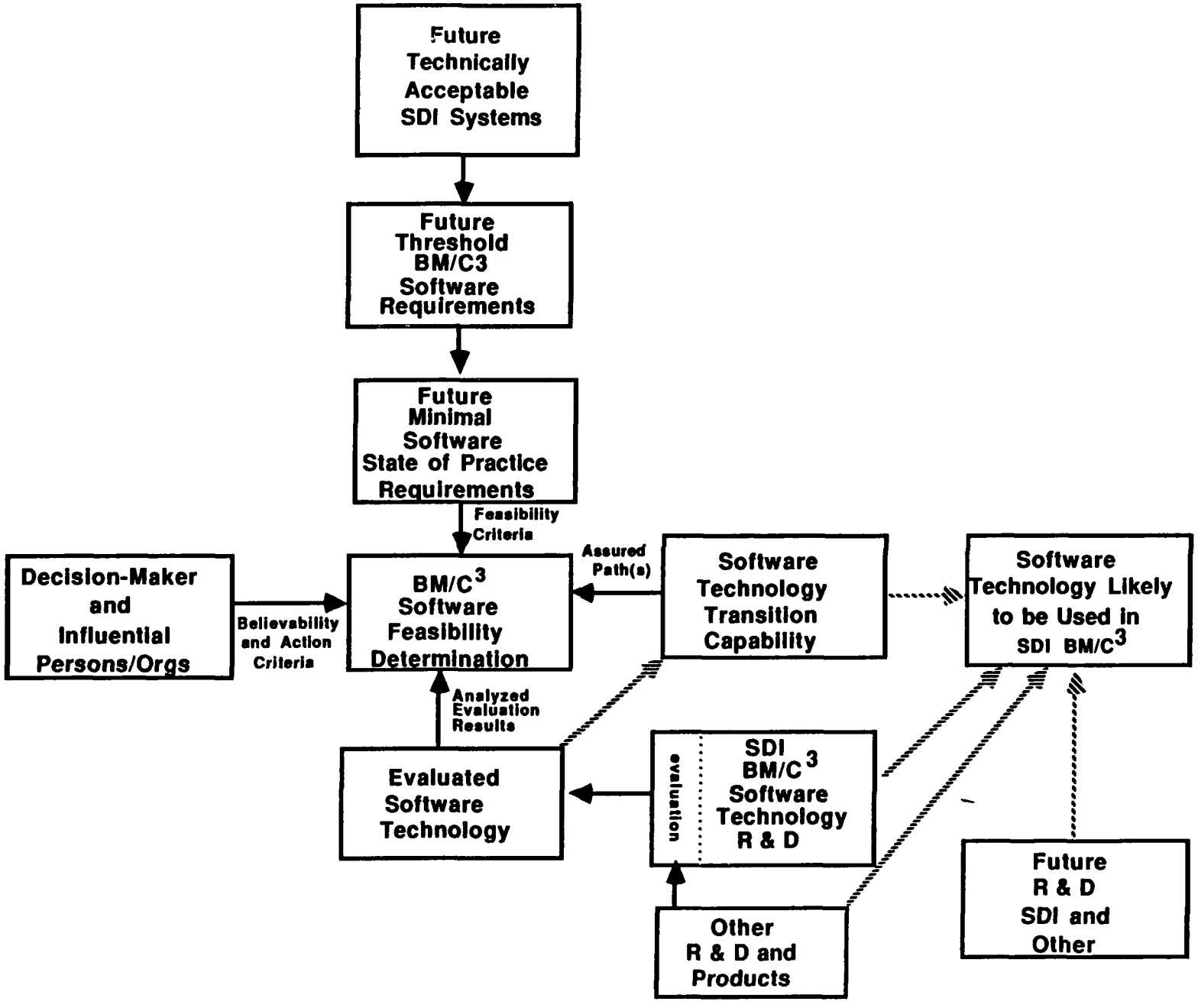


Figure 2. BM/C3 Software Technology Picture At Time Of Full-scale Development Decision

technological capabilities (existing and under development), and politics (cost/benefit, U.S. policy, U.S. decision making process, Soviet posture). A determination of the feasibility of building the BM/C3 software depends on a threshold definition for the SDI system (top of Figure 2). This threshold could range from minimal to sophisticated and will determine BM/C3 software requirements. This, in turn, will establish the necessary technologies which can then be evaluated based upon a set of feasibility criteria.

The necessary software technology must be mature enough to provide convincing evidence of its merit and workability for the SDI system (bottom left of Figure 2). Prototype software should have proven itself in large-scale tests and simulations that are as close to real conditions as treaties allow. Believability and decision criteria must be established for the decision makers and influential individuals and organizations that will be involved in the early 1990's decision. Unacceptable uncertainty may result in their making a negative determination. This exact threshold and evidence required must become clearer and more precise as SDI planning and R&D progress. Nevertheless, these considerations already provide some guidance to this plan.

A clear and certain technology transition path must exist for bringing each technology to production quality, integrating it into the SDI effort, and insuring its successful use by the time required. Although widespread use of the technologies is not required, the technologies will have to be integrated into multiple organizations within the SDI effort. The right side of Figure 2 and the gray arrows indicate that the state of technology used to demonstrate feasibility may not be the same as that used to build the system at a later date. Note that the software technology R&D effort should aim not at only meeting the minimal threshold, but at providing the basis for building as effective an SDI system as possible. This is a secondary aim when considering feasibility but an important one when considering total system attractiveness.

2.1.2 Criteria for BM/C3 Software Technology Feasibility Determination

From early reviews of requirements and the current states of art and practice in the relevant technologies, a preliminary list (Figure 3) has been composed of the types of criteria that appear most likely to cause a decision not to build the SDI system.

The top of Figure 3 lists key parts of the operational software that appear likely to be the most difficult. Next on the list are a number of key qualities or characteristics that could be showstoppers, for example, reliability, testability, and the sheer total size of the software. Most of the remainder of the list are managerial concerns such as cost, personnel, and schedule. Listed last are external concerns such as the credibility of the simulations and the requirements plus the need for all the individual tools, techniques, and people to work satisfactorily together.

These and other requirements are discussed further---along with possible solution technologies---in the remainder of this plan and the appendices.

Technology (software expressing "algorithms" plus computing resources) must satisfactorily solve the tough SDI BM/C3 application problems

a. Multisensor Correlation

b. Resource Allocation
   (1) Sensors
   (2) Weapons to Targets

Technology must provide foundation software functionality (on computing hardware resources).

a. Operating Systems

b. Data Management

c. Networking and Communications

These and other software must work together satisfactorily.

A number of technical characteristics/properties must be acceptable.

Reliability
Survivability
Security
Difficulties caused by large size of software
Testability/Verifiability
Maintainability
Evolvability

A number of managerial characteristics must be acceptable

Acquirability
Manageability
Affordability
Personnel Resources (availability and quality)
Schedule
Predictability/Risk

Several external concerns must be satisfied

Foreign Technology Transfer
Credibility of Simulations, Demonstrations, and Evaluations
Credibility of Requirements Specification

Tools, technologies, people and organizations used to meet all these criteria must work satisfactorily together

Figure 3. Tentative List of Types of Potential "Showstopper" Feasibility Criteria

2.1.3 Test and Demonstration of BM/C3 Software Technology

The three primary considerations - technical feasibility, technological integration, and politics - that feed the full-scale engineering decision -- and the criteria for determining the technical feasibility of BM/C3 software were discussed in the two previous subsections. The question still remains - how to evaluate such criteria to best support the full-scale engineering development decision?

The functional and performance requirements of the BM/C3 subsystem must be tested and demonstrated to the satisfaction of scientists and engineers as well as official decision makers and influential individuals and organizations. This must be accomplished with a strategic defense system that is not as yet deployed. Defense assets will be in various stages of development -- conceptual, design, simulated, emulated, prototype, and actual -- due to constraints of schedule, cost, and national policy. The offensive threat will be simulated and possibly scaled down. Reality and the physics of the engagements will also be simulated. In fact, parts of the BM/C3 subsystem itself will be as yet undeveloped and simulated.

The technical feasibillity of the BM/C3 subsystem needs to be determined with high confidence in such an environment. To support this determination, several projects are being developed through SDIO sponsorship, the Armed Services, and government contractors. The National Test Bed (NTB) will provide a "comprehensive capability to demonstrate and independently evaluate alternative Strategic Defense Initiative System and Battle Management/Command, Control, Communications (BM/C3) architectures and key defensive technologies" [NTB 86]. The Experimental Version (EV) program will develop a series of prototype BM/C3 subsystems to perform analysis of its tactical configurations and to demonstrate achievement of required technical performance. In addition, simulators have undergone intensive development for the past couple of years which model a wide variety of SDI components and typically generate graphic renditions of the engagement.

All of these simulations, prototypes, and other items must, through a series of experiments, tests, and demonstrations provide the evidence needed to make the full scale engineering development and other decisions.

2.2 Developing Capabilities to Build the SDI System

The second objective of an SDI software technology R&D program is to develop the capability to produce and support the software necessary to meet SDI system and software requirements. The functional capabilities that software must provide in the SDI BM/C3 subsystem are briefly mentioned in Section 1.0 and are discussed in some detail in Appendix B. But beyond these individual pieces of functionality, certain key overall requirements and properties drive the concerns for developing SDI BM/C3 software.

2.2.1 Capability to Change Over Time

A requirement of the SDI system is that it will change over time as requirements and technology change. Requirements will evolve as the result of (1) increased understanding of and desire for system functionality, (2) changes in the threat and countermeasures, and (3) political/policy changes. Technology will evolve as a result of R&D and technology transition activities that move state-of-the-art research into the state of practice.

The prototyping method of software development recommended by the Eastport Study Group accommodates evolutionary changes in both requirements and technology. A

prototype is an instance of a software version that does not exhibit all the properties of the final system. It is usually lacking in terms of functional or performance attributes [McDonald 86].

SDI system software development is expected to follow an evolutionary acquisition approach consistent with DOD-STD-2167, Defense System Software Development standard, making extensive use of prototyping and consisting of many overlapping life cycles. Software technologies that respond to this need must therefore be fostered. DOD-STD-2167 states that

> "During software development, more than one iteration of the software development cycle may be in progress at the same time. Each iteration represents a differnt version of the software. This process may be described as an 'evolutionary acquisition' or 'incremental build' approach. Within each iteration, the software development phases also typically overlap, rather than form a discrete termination-initiation sequence" [DOD-STD-2167, p. 11].

Both the Eastport Group and the DTST anticipated that technologies key to the SDI system will be demonstrated by the early 1990's in order to allow a decision on whether to develop and deploy an SDI system. The BM/C3 schedule, the gradual understanding of requirements, and the many and significant advances that must be made all point to the need for evolutionary software development.

2.2.2 Evolutionary Development by Parts Requires Open Architecture

The Eastport Group has further advocated an open architecture where parts may be changed, added or deleted independently of one another over time. Software not only provides functionality to the system components, it also provides the interfaces between the components.

The proper software architecture and interface standards are important for achieving an open systems capability. The SDIO will need to identify and capitalize on existing standards and, where necessary, set standards. Interface standards facilitate bulk data exchange, transfer of results, software transportability, functional scope, and transparency to users on one hand and security or access control, damage confinement, fault tolerance, and information hiding on the other. Here the SDI effort need not start from scratch. A number of information interface related standards exist or are under development [Nash 85]. The DoD Software Technology for Adaptable Reliable Systems (STARS) Program has been concerned with developing standard interfaces to assure technical compatibility of software engineering environments. The SDIO's interest in standard interfaces will be even broader.

The establishment of this open architecture with its expanding sets of upwards compatible interfaces over time is, therefore, a key element in the approach planned.

2.2.3 Assumptions

A number of assumptions about SDI system requirements can be found in prior studies/statements about the SDI. Some relate to the technology and some to management.

One assumption is that the architecture will ease software requirements. A strong recommendation of the Eastport Group was that SDI program should not depend on any

software breakthroughs, but instead rely on software technology that is available and emerging today for proving feasibility.

Another assumption is that the system will be coded in Ada and Common LISP. Ada is the DoD's standard HOL (High Order Language) for all mission critical applications. DARPA has recently standardized on Common LISP, and the American National Standards Institute (ANSI) and International Standards Organization (ISO) are expected to follow their lead.

Another assumption is that there will be multiple (as-yet-unknown) hardware architectures. Algorithms first implemented in software may later be implemented in hardware.

Other assumptions relate more to management/policy issues. One is that the SDI program will be able to invest substantial sums to supply computing and educational support to workers on the SDI software. Another is that the budget of the SDI program and the budgets and plans of other Government software technology programs will remain essentially unchanged from what is planned today.

2.2.4 Critical Properties of the SDI System from a Software Standpoint

A number of SDI system properties are critical from a software standpoint and will therefore affect software development including:

a. The real-time nature of the system

b. Its required robustness (in software as well as hardware)

c. The need for security

A fourth property that has been mentioned before, but is also closely linked to the software is the changing nature of the system.

Among the BM/C3 properties identified by the ESD (U.S. Air Force Electronic Systems Division) in their BM/C3 Architecture Definition Program are that the system be effective, reliable, survivable, secure/fail safe, flexible, reconstitutable, reconfigurable, and augmentable in the far term without radical design impacts [Ford 85]. These concerns plus concerns for performance, maintainability, adaptability, testability/verifiability, interoperability, and affordability imply certain software characteristics. Related concerns are the potential for losing control of the system and credible security.

Reliability and survivability are especially important to any battle management defense system. Both the response times and survivability required of the SDI system imply that it be a modular distributed system, some parts of which are highly autonomous. Reliability and survivability concerns demand a system that can provide for continuity of operations under and recovery from abnormal conditions, including physical attacks and countermeasures when one or more processors, storage units, or communication links is upset or fails.

The SDI communications network must cope with changing network topology and attrition of system resources [Offutt 85a]. One software-related concern is routing. Other networking concerns include the large numbers of platforms and processors involved and the changing relationships of satellites.

2.3 SDI Software Technologies

This document divides the software technologies pertinent to the SDI into three areas: application, foundation, and software engineering processes. The first two refer to parts of the system whereas software engineering processes refer to the development and support of the system.

2.3.1 Application vs. Foundation Technologies

Application software as defined by the *IEEE Standard Glossary of Software Engineering Terminology* [ANSI/IEEE 83] is software specifically produced for the functional use of a computer system. For example, software for navigation, gun fire control, payroll, and general ledger. This can be contrasted with the more generic foundation technologies concerning software to facilitate operation and maintenance of the computer system and associated programs (e.g., databases, operating systems, or communications). This document addresses software for both the application-specific and foundation areas of BM/C3.

In the area of foundations, the document discusses communications, distributed operating systems, data management systems, man-machine interfaces, and parallel processing. These foundation areas map to subject area specialities in computer science.

2.3.2 Software Engineering Processes

The software engineering activity comprises management and technical aspects. Environments in which software is developed and is evolved reflect tradeoffs between these management and technical aspects and provides the tools and methods for all the activities that constitute a software system's life cycle, including definition, design, construction, testing, installation, operation, and maintenance [Druffel 83].

The software life cycle is supported by a software engineering process typically formalized as a well-defined set of principles, practices, and procedures, which is in turn supported by general procedures for management and system acquisition. An automated software engineering environment covers the entire life cycle, providing generic automated tools and automated tools oriented toward specific management practices, methodologies, or applications [Druffel 83].

Software engineering process topics that this report addresses include automated hardware/software tradeoffs, software engineering environments, simulation, software dependability, people and organizations, and technology transition.

3.0 STRATEGY

This section describes a general strategy for the SDI software technology R&D effort to improve the DoD's ability to develop software to a level sufficient for the SDI system. The overall approach is one of try, evaluate, and improve. This fits with the prototyping approach planned for building the SDI large-scale demonstrations and SDI system. This approach to technology R&D encompasses foundation, application-specific, and software engineering process technologies plus R&D management. The strategy will exploit current technology, build on existing activities, and coordinate the collected expertise of people in many organizations.

3.1 Overall Try, Evaluate, Improve Approach

A major reason for adopting a try, evaluate, and improve strategy for SDI software technology R&D and software development is evolution. Not only will requirements evolve (requiring the software to evolve as well), but the software engineering process will also evolve as new techniques are developed and enter the state of practice. This is one reason for adopting an open system architecture approach that allows incremental improvements without changing the entire system.

Figure 4 illustrates the Technology Verification Strategy (TVS) that the SDIO has defined to permit the examination, evaluation, and validation of various defensive technologies in the context of candidate defensive system architectures. Using an Architecture Simulation/Engineering Facility (acquired separately from the NTB), the System Architect will simulate numerous configurations of weapons, sensors, and BM/C3 systems against many threat scenarios. These simulations will use the expected performance of the system elements being explored in the Technology Research Program. In this manner the System Architect will define a number of the most promising defense system architecture alternatives [NTB 86, p.2].

SDI's approach to software development will rely heavily on prototyping, simulation, and evaluation. Some R&D may be required to develop these capabilities. In addition, technologies selected to develop SDI software must be amenable to this approach.

Prototyping has some clear advantages over the lockstep waterfall development process (requirements definition - design - code - test). Because prototypes are on a smaller scale they can give results faster and cheaper than a full scale project, and it is often more cost-effective to try something than to analyze everything. Also, prototypes can incorporate changes more easily and therefore evolve with requirements and the understanding of the problem. Perhaps less obvious is the ability of people to think more creatively, and at the same time concretely, with a prototype in hand [Peters 82, p. 139-140]. As people experiment with a prototype, new ideas are spawned. At higher levels of abstract analysis, this is not always the case.

As Figure 4 indicates, prototyping is also an important concept for researching and developing the technologies necessary to develop the SDI software. As an R&D strategy, prototyping will be used to identify problems, conduct tradeoff analyses, and accelerate development of critical technologies with minimal expenditures of time and money. Technology prototypes may flow into system prototypes.

Although it will be possible for SDI to test weapons and sensors individually, the possibilities for testing a complete strategic defense system are limited. Simulation, therefore, will play a central role in the SDI development as a partial substitute for traditional testing and as a means of evaluation. The National Test Bed is being established for this purpose, but it will not be the only facility engaged in simulation. Simulation may also be used for training and as a technique for early evaluation of state-of-the-art software technologies for investment decision purposes. Simulation lends an evaluation capability to the R&D strategy and to the overall software engineering process.

Evaluation of both software technology R&D and the SDI software will be necessary for quality control and to ensure that evolving requirements are met. Evaluation measures progress toward meeting objectives. Effective evaluation is continuous, resulting in incremental improvements to the process or product being evaluated.

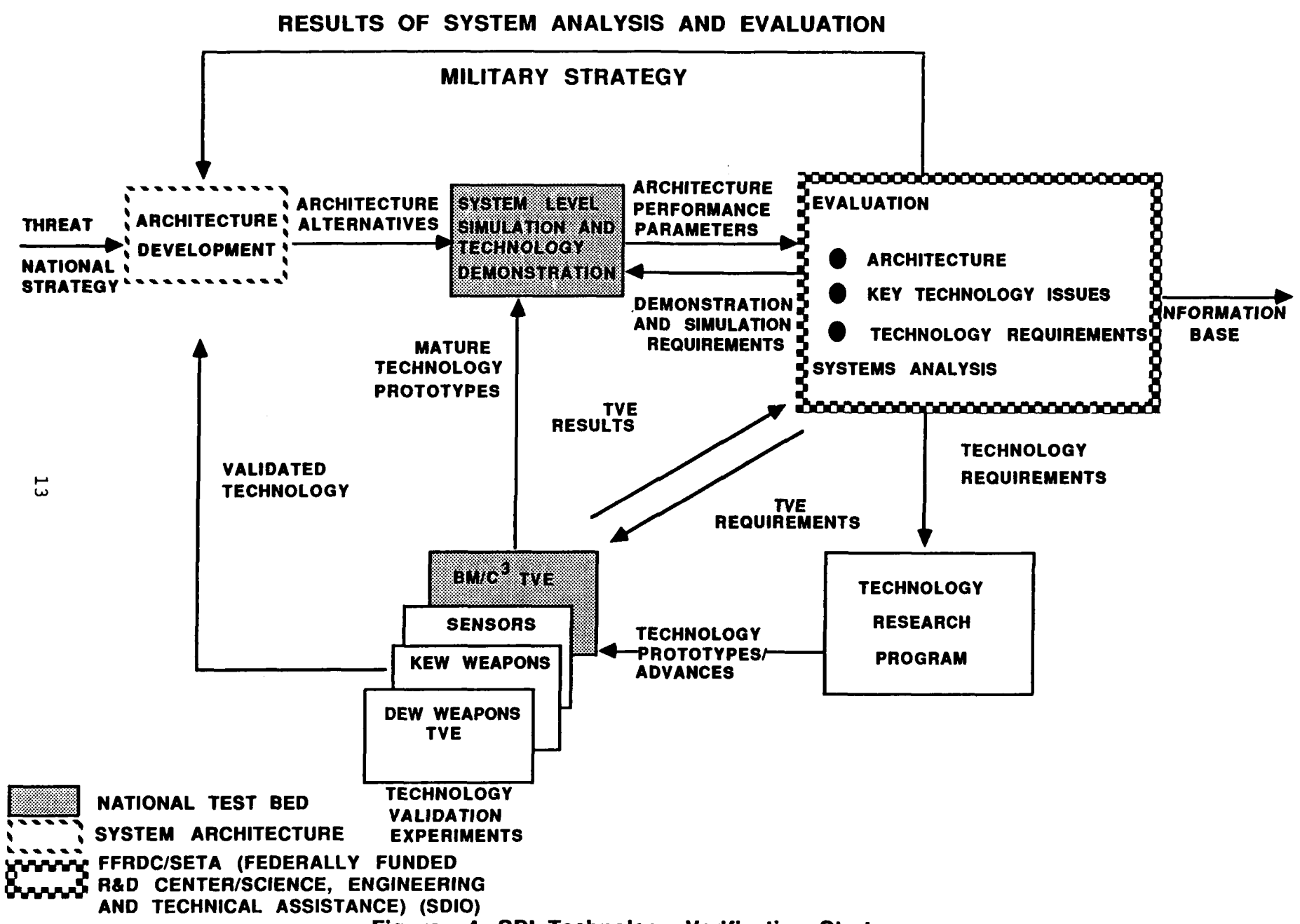


Figure 4. SDI Technology Verification Strategy

13

3.2 Three Part Approach Plus R&D Management

This plan includes a three part approach for identifying and satisfying SDI software technology needs (see Figure 5) plus a R&D Management Plan.

The small size of the SDIO staff requires a decentralized software technology R&D effort that relies on U.S. government agencies, the Services, joint programs, and foreign governments for implementation. Such decentralization underscores the importance of planning and control of the SDI software technology R&D. Methods that help assure quality R&D include evaluating prototypes, instituting an independent review process, and selecting competent people to do the work.

3.3 Principles

Some additional principles will guide the R&D strategy. These include building on existing efforts, coordinating currently planned efforts, mixing evolutionary/revolutionary technologies, building generations, using DoD organizations/programs to execute designated tasks, using the best technical talent available, and planning in more detail for near-term years.

3.3.1 Build on and Coordinate Existing and Planned Efforts

SDI should maintain interfaces with other DoD software programs and with the defense software community to leverage SDI funds with other funds and to ensure that the results of SDI investments are cumulative. Wherever possible, capabilities developed should be technically compatible and not restricted to the developer. Unwarranted duplication of effort among programs should be avoided. Interaction among software technology programs can also help ensure that requirements are met and speed the transition of technology into use. Figure 10 in Section 5.0 lists some existing software technology programs that the SDI software technology R&D program can build on.

Coordination of these efforts is an important dimension of the SDI software R&D strategy. To this end, an Interagency Group on Software Technology has been formed at the instigation of the SDIO to increase communication and coordination. Group members include representatives from the STARS Program, Ada Program, SEI, NASA, WIS, National Computer Security Center, DARPA, National Science Foundation (NSF), and SDIO. This group will work to identify and solve common problems, eliminate unwarranted duplication of effort and capitalize on one another's efforts and areas of expertise. A more coordinated approach to mission critical software technology in the federal government will benefit all members by increasing credibility, reducing overall costs, and enabling efforts to build on one another.

The SDI effort should also build on others' experiences by learning from large successful projects. One form would be to find agreement on large and successful DoD software projects and study the management and technologies used for their acquisition and development. Another would be to look at large companies like Bell and IBM, or even industry consortia like the Microelectronics and Computer Consortium (MCC) and the Software Productivity Consortium (SPC). Finally, the SDI should learn from large contemporary projects like the WIS, NASA Space Station, and FAA software development efforts.

TASK AREAS

- **Application -Specific**
  - **Battle Management/C3**
- **Foundations**
  - **Communications**
  - **Distributed Operating System**
  - **Data Management Systems**
  - **Human Factors/Man-Machine Interface**
  - **Parallel Processing**
- **Software Engineering Process**
  - **Hardware/Software Tradeoffs**
  - **Software Engineering Environment**
  - **Simulation and Evaluation**
  - **Reliability and Survivability**
  - **People and Organizations**
  - **Technology Transition**

Figure 5. Software Technology Areas

15

3.3.2 Mixture of Evolutionary/Revolutionary Technology

Another guiding principle of the R&D strategy for software technology is the notion that developments will be initially evolutionary, followed by more revolutionary developments in the future.

Software technology has already demonstrated that it is capable of effecting revolutionary change in both the military and commercial spheres. With the proliferation of software technology future revolutionary changes are likely. Projects like the DARPA Strategic Computing Initiative, the Japanese Fifth Generation, the MCC, and the European Economic Community's ESPRIT (European Strategic Research Program in Information Technology) are working toward such breakthroughs. Likewise, SDI-funded software technology efforts should aspire to revolutionary developments, without depending critically on their achievement.

Software sufficient to meet the requirements of the threshold technically acceptable SDI systems mentioned in Section 2.0 should not depend on revolutionary breakthroughs. With one or more revolutionary breakthroughs, however, a more sophisticated system with increased functionality, greater reliability, or lower cost might become possible.

3.3.3 Experimental Versions and System Generations

Beginning in 1988 a series of experimental versions of the SDI BM/C3 subsystem are planned. These prototypes will provide a means for practicing, experimenting, integrating, evaluating, learning and improving the technological, organizational, and other aspects related to BM/C3 software development.

This series of learning experience and improvements is essential to the chance for achieving success. In addition to preporatory education and technology R&D, the people and technology involved will also be expanded, upgraded, matured, and proven through attempting this series of increasingly capable versions. The result must be not only adequate software technology but the capability to use it successfully to develop the SDI system. This calls for the involvement and improvement of all elements from the contractor workforce with their automated aids and practices to the Government program office with their assisting organizations, and management and acquisition practices.

The SDI system is expected to evolve through several generations. Each generation's requirements will evolve and the R&D strategy should anticipate these changes to have the necessary technologies ready when they are needed. The R&D strategy must thus be flexible enough to accommodate change as well as future-oriented to satisfy the needs of the next generations of the system.

3.3.4 DoD Organizations/Programs Will Execute Designated Tasks

Recognizing that the task of software technology R&D is beyond the capabilities of any one organization, including the SDIO, another guiding principle of the R&D strategy is that DoD organizations/programs will execute designated tasks. This mechanism has the added advantage of enabling the SDIO to capitalize on the expertise resident in existing specialized programs and organizations.

The DoD Science and Technology Program has proved effective across a broad spectrum of technology development. The service and agency research and development community has produced technology ripe for exploitation and a distributed body of expertise. In the case of computer technology, particularly software, this technology is generally sharable,

offering enormous leverage to DoD. Assuming that other DoD (as well as industry and academic) research activities will continue as planned, the SDIO can complement these existing activities and provide funds to selected DoD organizations to execute and manage contracts and to accelerate or supplement existing activities in designated areas.

4.0 SOFTWARE TECHNOLOGY AREA ASSESSMENTS

This section summarizes the software technology R&D task area findings from Appendix B. The technologies are categorized by application-specific, foundation, and software engineering process technologies. Figure 5 lists the technologies covered in Appendix B.

The expected results of R&D for each software technology are discussed in this section including what can be done with the results and why they are important to the SDI effort. This section also presents an overview of how the R&D should be conducted, including when tasks should be undertaken. In the development of a computer system as large and as complex as that envisioned for the SDI system, standards play a crucial role. They facilitate the software development effort by enhancing product and programmer portability. Therefore, where appropriate, standardization issues are addressed. Prime candidates for performing organization/program are usually identified as well as issues of concern. Finally, the section states how the proposed R&D for each software technology supports SDI objectives.

Following the technology discussions is a summary of the recommendations of prior SDI studies showing that they tend to reinforce the recommendations given here. A discussion of interrelationships among task areas concludes the section. Appendix A relates the SDIO projects currently funded to the recommendation in the software technology task area assessment of Appendix B. Appendix B contains the full text of the technology investigations and recommendations. Appendix C contains plan charts in the form of time lines for the R&D tasks recommended in Appendix B and this section.

The task area assessments were first each done over roughly two months during the period of February to May 1986 and revised during the fall of 1986. They draw on the SDI contractor intermediate results, the technical literature in the areas, information from a number of government funding agencies, and extensive personal contacts with and reviews by individuals working in the areas. Section 4.0 ends with a discussion of the relationships among the task areas and their relative priorities.

4.1 Application-Specific

4.1.1 Battle Management/Command and Control (BM/C2)

It is essential that the SDI designers recognize that the engine driving software complexity is the SDI architecture. By giving more automony to system components many software problems could be eliminated. The BM/C3 software requirements for SDI will stress many existing areas of technology. For example, the shear volume of sensor data that will need to be processed during battle will force the use of parallel processing, necessitating the development of algorithms that exploit parallelism. The SDIO must become active not only in pushing the development of software technology, but in directing that development in many areas essential to BM/C3 applications.

More specifically the SDIO should:

a. Investigate the proposed architectures and identify where each architecture stresses existing software technology,

b. Provide a glossary of terminology for each organization participating in the Phase III BM/C3 architecture and system study,

c. Establish a Parallel Algorithm Test Center for research and evaluation,

d. Research parallel algorithms for multi-sensor data fusion,

e. Study and prototype the role that humans will play during the battle,

f. Investigate the possible applications of artificial intelligence for SDI, and

g. Identify the types of support systems required for SDI and the National Test Bed.

By pursuing these goals the SDIO will be in a position to meet the software requirements for BM/C3.

4.2 Foundations

The foundation software requirements of the SDI system are centered around three different environments: (1) the platform environment, (2) the platform management (or C2I) environment, and (3) the software engineering environment. Within each of these environments, requirements such as performance, security, integrity, distribution, etc., will be found. However, each environment will place varying amounts of emphasis on these requirements.

4.2.1 Network Communications

The network communications system of the SDI will operate in a constantly changing environment. These changes are due to (1) relative positional changes of platforms, (2) "normal" failures of processors, and (3) failures due to a hostile, changing environment, (explosions, jamming, etc.). Fault-tolerant techniques, including redundancy and excess capacity, must be used in the design of the communications system to provide a reliable and survivable system. In addition the deployed system will have to meet extensive access control and real-time performance requirements.

A commitment must be made to a protocol standard to achieve the desired interoperability among distributed computer communication networks. This standard must provide an open and extensible framework with provisions for the special requirements of the SDI specified above.

A variety of techniques currently exists for performing the required basic functions of a network communications system. The ability of these techniques to meet the dynamic reconfigurability, reliability, security, and real-time performance requirements of the SDI system must be determined. Algorithms for naming, routing, topology updating, and flow control are key to the success of the SDI system. Current algorithms have been developed under the assumption that (1) the network will be topologically fixed or require minor changes due to component failures or repairs and (2) communication demands will fluctuate slowly. New algorithms to perform these functions must be developed to meet the reliability, security, reconfigurability, and real-time performance requirements of the SDI.

The security requirements of the deployed SDI system are severe and unique. Existing systems do not address the needs of the SDI. Appropriate models for security of computer communication networks must be developed. These models must be available early in the design of the SDI system as they will be a basis for many design decisions.

The automatic generation of protocols will aid in the development of a secure and reliable communications system and assist in the evolutionary process of the SDI system. Research in this area should be continued.

The theoretical ability to solve the individual functional needs of the SDI system does not necessarily result in the ability to build a computer communications network that will perform appropriately. The integration of solutions to individual problems must be addressed. Prototype computer communications networks must be built. These systems should be integrated with the prototype distributed operating systems being developed for the SDI (see Section 4.2.2).

4.2.2 Distributed Operating Systems

The distributed operating system(s) required to support the development, maintenance, and operation of the SDI's BM/C3 is more extensive than any system built to date. Although distributed operating systems are well within the state-of-practice, systems that display the characteristics required by the SDI system are only now being considered.

The operating system(s) for the SDI must be distributed and fault-tolerant, while providing real-time performance. In addition, the system(s) must be rated secure to the A1 level as defined by the DoD's Trusted Computer System Evaluation Criteria.

To demonstrate feasibility and to obtain the technology and experience to build such a large and robust system, the SDIO should fund the development of up to four prototype secure, real-time, fault-tolerant, and reliable operating systems. The number of researchers with experience in developing distributed systems is quite limited. Multiple prototypes will increase the knowledge that is gained and enlarge the base of experience that will be needed for full-scale development.

Additional research projects in (1) modeling and analysis of distributed systems, (2) secure kernels, (3) rapid recovery and approximate recovery, and (4) real-time and dynamic scheduling/resource allocation must be tightly integrated with the development of these prototypes. This will tend to direct the studies in ways that are both more realistic and more applicable to the short-term needs of the SDI. Proposals should indicate the means for cooperating among the researchers.

While the prototype developments will yield short-term results and aid in the full-deployment decision, intermediate to long-term results require widespread and convenient means of experimentation. The SDIO should, therefore, fund the development of testbeds for distributed systems and widely distribute state-of-the-art software for distributed systems.

4.2.3 Data Management Systems

The data management requirements vary across three separate SDI "environments". The data management functional requirements of the space-based platform environment will primarily be implemented as high performance algorithms operating in high speed memory. Although the system will be distributed in nature, it should be as autonomous as possible. Much research needs to be undertaken in order to identify and fully understand the roles these algorithms will play. Prototypes of these algorithms should be constructed as soon as possible in order to determine critical areas where necessary software and hardware technology is lacking so that these can be addressed before the feasibility demonstrations of the early 1990's.

The platform management environment will provide the interface between the SDI system and human decision makers, as well as support for day-to-day managerial/logistical activities. This environment will emphasize requirements such as distribution and integrity.

Much of the current database research activity will be applicable within this environment. However, prototype distributed database management systems need to be developed early to identify areas where technology is lacking (e.g., data models, error recovery, etc.). Research in these areas will provide a better understanding of the level of distribution needed, as well as providing insight into the level of distribution necessary.

Within the Software Engineering Environment (SEE), much research is needed in developing a fully integrated object management system which will provide the capability for creating and storing strongly typed objects (such as test data, source code, output results, etc.). An SEE database will also be necessary for many functions within the environment. Research is needed to define the role this database will play, as well as the definition of interfaces, amount of data to store, the role of active components, the level of distribution, what types of security are necessary, etc. Many of these issues must be prototyped within an object management system. Prototypes of the database should be developed concurrently with prototypes of the SEEs.

It appears that several government agencies (WIS, STARS, NASA Space Station, etc.) may be able to provide leverage for several areas critical to the SDI. In addition, they may provide good opportunities for performing some of the research and development that will be needed in developing the various prototypes outlined above.

4.2.4 Man-Machine Interface

The man-machine interface (MMI) encompasses all inputs from the user to a computer, all outputs from the computer to a user, and the sequencing of such inputs and outputs [Foley 85]. The software technology area of MMI may be partitioned into six subareas:

a. Human factors

b. Interaction techniques

c. Workstation managers

d. Graphics packages

e. User interface management systems

f. Decision aids

The quality of the MMI has a profound effect on the performance of the user and hence on the overall utility of the computer system. In fact, the potential gains from an improved MMI may surpass those from advancements in other technological areas. These rewards more than outweigh the additional cost for the analysis, design, and implementation of an MMI with good human factors.

The acquisition of software foundation technology may facilitate the attainment of these goals. Certain actions are recommended for the SDIO to acquire relevant foundations for the SDI man-machine interface. The following recommendations are intended to assist the SDIO in making decisions about future directions for technical efforts.

An important human factors issue in the SDI environment is decision-making under stress. The SDI command will obviously operate in a highly pressured atmosphere. Research has indicated that humans are more susceptible to errors when stressed. Since errors committed while under attack could be catastrophic, it is essential that the potential for error

be reduced to a minimum. To fill the gaps in the current understanding of human performance under stress and to examine such performance in unique SDI war-time scenarios, SDIO should fund further research.

The SDI Command and Control Human Interface is being investigated at the Naval Research Laboratories. NRL is attempting the determine appropriate roles for human SDI commanders under the severe time constraints of an SDI scenario. In addition to establishing the feasibility of man-in-the-loop tasking, NRL is investigating alternative MMI mechanizations. The program should emphasize (1) monitoring human factors research, (2) identifying human performance requirements, (3) initiating a human factors TVE, (4) studying decision-making abilities under stress, (5) developing human factors standards for SDI, and (6) formulating an MMI evaluation methodology.

Given the enormous investment in the development of commercial workstations, SDIO should monitor the state of practice in interaction techniques. Then when required, commercial computer hardware and software that integrate the most sophisticated and useful interaction techniques may be acquired. Because of the potential utility of a user interface that employs artificial intelligence techniques, SDIO should monitor its state of the art and, in particular, the work sponsored by DARPA.

The U. S. Army Strategic Defense Command is developing a Command and Control Decision Aids Test Environment. The program has three main objectives [SDC 86]:

a. Determine where decision aids are feasible in an SDI BM/C3 subsystem.

b. Apply decision aids in a test bed environment.

c. Provide for technology transfer to BM/C3 TVEs.

SDIO should continue to fund the U. S. Army Strategic Defense Command's program for a Command and Control Decision Aids Test Environment. Their program is on track for the timely integration of decision aids into SDI foundation technology.

4.2.5 Parallel Processing

The parallel processing needs of the SDI effort are potentially extensive and require the further development of this technology. The use of parallel processing to increase system performance will grow as the demand for processing power increases. Many different architectures for parallel processing engines have been developed, although some exist only as paper machines, and the assets and liabilities of these architectures have yet to be determined. What is clear is that no single architecture is best for all applications and that each has a theater of applications in which it delivers its best performance.

The future of this technology will require further research and development of parallel processing engines, programming languages, compilers, operating systems, and algorithms. More specifically this should include:

a. Research of techniques for expressing parallelism, both explicitly and implicitly.

b. Development of Ada compilers targeted to a representative class of parallel architectures.

c. Development of operating systems to support the development of code for parallel machines.

d. Development of facilities for testing and evaluating algorithmic behavior on different parallel engines.

With this level of testing and evaluation it will be possible to discern those designs with potential and classify machine architectures by areas of application.

4.3 Software Engineering Process

4.3.1 Computer Systems Engineering Technology

For many years, the approach to computer systems design has been *ad hoc*. Components were designed in isolation and later integrated to form a complete system. Software, in particular, has almost always been designed on top of the hardware and not as an integral part of the entire system. With the increasing size and complexity of computer systems in general, and the SDI system in particular, this approach is neither desirable nor feasible. A method in which all system components are considered and designed together is necessary.

In an integrated system design paradigm, system design proceeds in two stages. Initially, the system to be designed is decomposed into subsystems or modules. At some point during the decomposition, decisions are made about how to implement the resulting modules. A hierarchy of implementation technology options exists. Deferring implementation decisions until late in the decomposition stage or even until after deployment gives the system increased flexibility.

The system requirements of the SDI indicate that an integrated approach to design is needed. This will require the development of design support environments and integrated toolsets. Input to the design system should be a system-level, implementation-independent specification generated from and traceable to a set of system requirements. Therefore it is recommended that the SDIO support:

a. Research to develop a method of requirements capture.

b. Research leading to a system specification language that is neutral with respect to eventual design implementations.

c. Research aimed at developing automatic verification (design meets requirements and specification) capabilities.

d. Research aimed at developing a design environment that provides support for maintaining multiple design alternatives, multiple functions to implementation assignments, multiple levels of design representation, and propagation of design changes across representations.

e. Research to develop advanced design aids such as automated system decomposition tools, tools to automate the assignment of functions to levels within the implementation technology hierarchy, simulators to assist in tradeoff analysis, and tools to permit transformation of representations such as microcode compilers and gate array compilers.

The degree to which requirement and design specification languages and engineering environments are standardized will impact the designs, development and maintenance of

computer systems. Typically, standardization facilitates the engineering process by, among other things, increasing communication capabilities, increasing the transfer and sharing of information and contributing to a larger and better pool of engineering support tools. Therefore, it is recommended that the SDIO determine:

a. An appropriate level of requirements specification language standardization.

b. An appropriate level of design specification language standardization.

c. An appropriate level of engineering environment standardization.

4.3.2 Software Engineering Environments

A software engineering environment (SEE) is composed of an integrated set of functional elements which supports a wide variety of users. In many of the areas pertaining to software engineering environments that are of interest to the SDI the state of practice is not adequate for the SDI's requirements. As such, the SDI should consider augmenting funds for research and development in those areas deemed critical due to lagging software technology or unique needs. This research will enable these areas to be addressed before the feasibility demonstrations expected in the early 1990's.

There are several government SEE efforts underway (STARS, WIS, NASA space station, etc.) which have requirements somewhat similar in nature to the SDI. It should be possible to coordinate activities with these organizations, thus leveraging much of the effort necessary in fielding a SEE.

The SDIO approach to SEE(s) must recognize that several generations of SEE technology will occur during its lifetime and that development SEE(s) can be less standardized than SEE(s) for test, evaluation, or maintenance. Early emphasis on standardization of software work product deliverable interfaces rather than numerous internal SEE interfaces fits these nicely. In addition, the programming languages Ada and Common LISP should be standardized upon as required implementation languages.

The SDI effort should position itself to benefit from several existing SEE efforts and maintain a posture that will be consistent with SEI and DARPA efforts in the longer term. The SDI effort should mainly be as evaluator, accelerator, addresser of special SDI needs, interface standardizer, and integrator. Indeed, the issue of SEE integration and extensibility should possibly be SDI's highest SEE research priority. In any case, an integration agent (contractor) should be specified early, and the development of prototype SEE's should begin as soon as possible.

At a lower level, technologies such as software methodologies, requirements development, and formal verification are not at an acceptable state-of-practice level for the SDI system. Studies of the various software methodologies must be undertaken to determine which are good candidates for the SDI system. The selection of this methodology should be made as early as possible. In the area of requirements development and formal verification, means must be developed for validating software requirements, both formally and informally. This involves developing efficient specification languages, simulators, animation techniques, etc. Formal tools which help automate the requirements and design aspects of software development should be developed in the long term.

Techniques in areas such as rapid prototyping and artificial intelligence must be fully explored. Methodologies and tools that support the rapid prototyping of software should be developed in the short term. In the long term, executable requirements and design

languages should be researched and developed. Studies should be performed to determine where AI technology may be applied within the SDI SEE. Although the short term may see few AI contributions, the long term may provide high payoff potential.

Strategies in areas such as testing and metrics will need to be developed. Basic research and development in testing will be needed as this branch of computer science is promising but relatively immature. Overall testing strategies and techniques must be developed in the short term for insuring high quality software. As testing will play a critical role in the SDI software system, it should be given a high priority. In the area of metrics, SDIO needs to institute a consistent measurement program across its efforts.

As can be seen, there are many areas of particular importance to the SDI relating to software engineering environments. These areas will require cooperation with other organizations and funding in the form of studies, research, and development. This must occur soon, however, in order to exploit the possibility of leveraging the work of other organizations and to field a prototype SEE that is tightly integrated, efficient, and provides the functionality needed for a system such as the SDI.

4.3.3 Simulation

Simulation has been defined as "the process of designing a computerized model of a system (or process) and conducting experiments with this model for the purpose either of understanding the behavior of the system or of evaluating various strategies for the operation of the system" [Shannon 75]. Simulation in the SDI environment will serve many purposes:

a. Supplementing the testing and debugging of battle management software.

b. Comparing the performance of alternative component designs.

c. Evaluating the performance of SDI components and subsystems.

d. Evaluating and comparing various designs for BM/C3 architectures.

e. Evaluating SDI system improvements and updates.

f. Exercising the SDI system in simulated attack scenarios to supplement traditional testing techniques.

g. Training users of the SDI system to ensure efficient operation during a real engagement.

h. Evaluating battle management strategies and tactics.

i. Analyzing the offensive, defense-suppression, and responsive Soviet threats.

j. Evaluating algorithms for tracking, targeting, kill assessment, etc.

k. Evaluating sensitivities of SDS performance with regard to underlying assumptions of threat, vulnerability, lethality, etc.

l. Evaluating computer resource and performance requirements.

Simulation modeling offers five advantages [Adkins 77]:

a. Controlled experimentation

b. Sensitivity analysis

c. Does not disturb the real system

d. Effective training tool

e. Time compression

The simulation approach, on the other hand, has four disadvantages [Adkins 77]:

a. Large costs may be incurred in manpower and computer time.

b. Development time may be extensive.

c. Model may diverge from reality.

d. Model parameters may be difficult to initialize.

These drawbacks are expected to impact the SDI project considering its magnitude. The first two, however, may be minimized through the use of software development tools and sound engineering practices. The latter two drawbacks may be reduced through the application of model validation techniques and through careful design of simulation experiments.

Virtually all of the functionality in SDI simulators will be embodied in computer software. To achieve strategic defense objectives for simulation, the software must be developed and experiments designed, performed, and analyzed with modern and capable tools, techniques, and concepts. Application of such software technology is essential for the coherent, efficient, and timely development of simulation systems, and thereby of the SDI system as a whole [Turner 85].

Valuable experience has been gained from the initial development of SDI simulators. However, the efforts were conducted by the Armed Services and government contractors without a coordinated plan. As a result, the simulators suffer from redundancy, increased development costs, inflexibility, and lack of interoperability and portability. Furthermore, several large simulation facilites have been proposed for SDI without a coordinated plan. The National Test Bed, the Architecture Simulation Center, and the BM/C3 Experimental Version have some overlap in functionality and similar requirements for hardware and software technology.

SDIO should examine the operational roles of the NTB, ASC, and EV programs for possible redirection to plug gaps and eliminate redundancy. The development and insertion of hardware and software technology into the NTB, ASC, and EV programs should be coordinated to optimize resource use.

Another important issue for the SDI software engineering process is a programming language for SDI simulations. Two important benefits would be reaped from committing to a simulation language standard for all SDI simulation applications:

(1) Increase program and programmer portability

(2) Enhance interoperability and interchangeability of simulators.

The first one reduces software development costs and enhances the robustness of the overall system. The second benefit ensures that different simulators can talk to one another and modules modeling a given component at varying levels of granuality can be interchanged.

The committment to Ada as the standard simulation would have additional advantages. An Ada simulation support environment supplements the Ada language kernel with high-level simulation functions and analytical facilities. Ada then becomes as robust and powerful as any special-purpose simulation language. Ada compilers will also be more widely available than those of special purpose simulation languages. Availability, which implies program portability, is an important consideration for a program as large and diverse as SDI. Finally, since Ada was designed to interface with other programming languages, existing simulations can be easily integrated into an Ada simulation portfolio.

4.3.4 Software Dependability

Dependability is a general term encompassing the many qualities of a system that relate to the ability to justifiably rely on its services. These qualities can relate to failure -- e.g., availability, maintainability, reliability, safety, and survivability. Or they can relate to the need to change the definition for a system's failure -- specification faultiness and system evolvability.

Software dependability requirements derive from system dependability requirements, but techniques for achieving software dependability are more immature than those for hardware. The critical, high dependability requirements for the SDI system combined with this immature status of software technology leads to the special attention provided by this section.

The mission of the SDI system demands ultra-high reliability and the ability to survive both natural and enemy caused upsets and failures. The software technology R&D involved in developing the capability to meet these demands covers failure definition (requirements), fault prevention (both avoiding putting faults in the software, and locating and removing them before deploying the software), measurement and modeling, software management and acquisition, and fault tolerance.

It is recommended that techniques and principles traditionally used in only one aspect of software develoment or test be applied. This comprehensive approach provides an integrating framework for all the individual technologies recommended and should be used starting with the prototyping efforts in a maximal approach combining the technologies.

In addition to the maximal, comprehensive approach recommendations cover failure definition (software requirements specification), fault prevention (software development and support), and fault tolerance (software execution). The systems requirements and engineering activities should include a strong awareness of software concerns and result in formal, machine analyzible software requirements specifications. SDIO should invest in and try several potentially suitable specification notations plus communication methods (e.g., animation, prototyping) for validation purposes. The specifications should address the range of possible implementations and evolutions, and operationally define the dependability qualities required including prerequisite knowledge such as usage distributions.

Software development and support related recommendations cover the means used and error detection. The concern for means has three facets: (1) quality technology and people must be selected and prepared, (2) technology and people components must be organized, specified, and certified, and (3) acquisition practices and contents must result in the desired contractor behavior. The error creation patterns of components and methods used in development and support should be characterized. Similarily, the error detection power of verification techniques should be characterized. These characterizations will allow fault minimizing combinations of components, methods, and techniques to be identified.

Investments should be made in technologies for software testing, simulating faults and recording error propagation, and formal verification using Ada. Automatic programming research should be monitored for potential future investment and use.

In addition, SDIO should invest in software dependability quality measurement and modeling research, use measurement and modeling in all its software efforts, and learn and improve over time.

Finally, SDIO should support R&D programs that cover fault tolerance at both the software and systems levels. In addition to the central analytical and empirical concerns with fault tolerance the issues of autonomy, decentralization, Ada definitional shortcomings in failure semantics, application-specific fail-forward and adaptive control mechanisms, and software safety should be addressed. Among the results should be level(s) of stable abstract machine(s) and a fault-tolerance harness that application software can use in a straight forward manner.

This area relates to all the other task areas since dependability attributes relate to all types of software technology. Many efforts in other areas will also contribute to dependability, ranging over the full spectrum of technologies from networking approaches to the selection of personnel capable of committing few errors. Conversely, however, care must be taken not to try to combine too many new or complex technologies causing reliability and survivability to inadvertently suffer.

The tasks in this area directly address one of the key concerns of the SDI Program--- one that is likely to be an important part the decision whether or not to build the SDI system.

4.3.5 People and Organizations

The projected size, difficulty, and criticality of the SDI effort will require large numbers of people and organizations performing at high levels. R&D will be required to identify and select the most effective people and organizations to build the SDI system, to improve individual, organizational, and interorganizational performance, to encourage innovation, and to improve the DoD business environment.

Because people and organizations are areas intimately involved in the contractor(s) internal processes, R&D in this area should be conducted through the contractor(s) to the extent possible.

The SDI software effort will require large numbers of expert software personnel with state-of-the-art knowledge. The SDI software R&D effort should fund research that identifies characteristics of excellent programmers and other software-related personnel and measures their relative effects. Such research suggests directions for selection and education of programmers. A related area that the SDI software R&D effort should monitor and learn from is superdesigner research going on at the MCC and elsewhere.

Education and training can effect some of the improvements in the workforce required by the SDI effort. SDI planners should monitor Ada, SEI, and other educational development activities, but will probably not need to initiate any new educational developments unless SDI system requirements impose education needs that vary widely from typical MCCR needs. For example, there may be a requirement to fund courseware development to communicate applications concepts to hardware and software personnel. However, given its needs for large numbers of workers with state-of-the-art knowledge, the SDIO must be prepared to fund educational upgrading of SDI software personnel.

The SDIO will also be concerned with evaluating SDI personnel. Based on the STARS Program and others work on measurement, it should find ways to measure people effectively, collect this measurement data, and learn from it. The SDIO should fund empirical studies of certification programs such as those of the ICCP to find out how effective such certification is in predicting job success.

In addition to excellent individuals, the SDI effort will need excellent organizations. To date, not much research has focussed on what makes organizations excellent, although the popular literature is replete with anecdotal evidence. The SDIO, therefore, should sponsor research in this area.

The relationship of project management to productivity and quality is of interest throughout the software industry. The SDI effort should monitor the research and data collection of software cost/size modellers and other researchers and incorporate the results in its organizations.

Improvements in the contracting and acquisition process are necessary to lower cost, improve quality, and introduce innovative technology in a timely fashion. The SDIO should monitor and capitalize on the efforts of the SEI in this area. Specifically, the SDIO needs to study what it can put in contracts to assure that the best people and organizations are used to build the SDI system.

Contractors should not only be responsible for people-and-organization-related R&D but also for achievement in this area. This necessitates the SDIO develop new acquisition practices and content that will ensure this achievement and encourage quality performance. This approach should include people and organization issues in the RFP, organization qualification review, proposal contents, and award evaluation criteria. It should also establish minimal initial requirements, intermediate human resource plans and improvement activities; thus 3-4 year-out advanced achievement levels appears feasible.

To increase awareness within the SDI software development community of the results of research in the area of People and Organizations and to coordinate the research recommended above, SDIO should fund an organization (possibly the development contractor) to fund, monitor, evaluate, and report on significant developments in this area. This could be done through the preparation of guidebooks or newsletters and by sponsoring conferences to address these topics.

4.3.6 Technology Transition

The technology transition task area will provide the methods and means to speed the transition of software technology from research to successful use. Since fundamental breakthroughs cannot be relied upon to occur and usually take decades to become usable in a DoD system, the advanced technology available for use in the demonstrations of feasibility and in the system's initial operational capability most likely will come from transitioning already existing research. Thus, the faster and more effectively this transition

can be done the more advanced will be the technology used. In addition as mentioned in Section 2.1, having the capability to do the final steps in the transition process may prove an important factor in the full scale engineering development decision.

Immediately, SDIO should emphasize establishing relationships with sources of technology, begin to augment the efforts of the DoD Software Engineering Institute (the FFRDC designated with the role of software technology transition promotion) to improve its coverage of SDI concerns, and establish the requirements for being ready for the early 1990's full scale engineering development decision. To address this last point SDI should establish preliminary versions of (1) requirements for software technology feasibility criteria, (2) the believability/decision criteria of the types of decision makers and influential persons and organizations likely to be involved in the decision, (3) the nature of the demonstration(s) needed to convince them.

In the near term, SDI should also address its technical strategies for transitioning technology particularly standards and compatibility, the provisions to place in contracts to encourage and ensure transition, the communication of its requirements to researchers, the technology evaluation capabilities for the different stages in the transition process, its SEE area strategy, and plans for transitioning the different technologies. Special attention should be given to establishing formal and informal links among the organizations through which the technology must flow.

The acquisition practices and content must encourage and to the extent possible ensure technology insertion by the software contractors. Selected contractors should have a history, culture, organizational structure and practices, and staff with demonstrated willingness and strong education foundation to adopt/adapt rigorous new software technology and use it successfully. (The government will probably have to help even the best qualified contractors to improve in this area.) The statement of work can call for specific tasks, suborganizations, and deliverables that improve technology insertion. These include requirements for technology insertion planning, technology insertion suborganizations, involvement in linking mechanisms, technical compatibility, and reports on progress.

The contractors should be required to cover in their proposals how they will introduce the required amounts and types of new software technology and this should be an important factor in the evaluation criteria. Adequate funding for technology insertion efforts by the contractors is essential, and incentive payments for outstanding performance in using new technology would be helpful.

In both the near and longer terms SDI should pursue actions, and research and experiments aimed at improving the transition of technology, observe their effectiveness and proceed accordingly. Persistent emphasis and measurement will result in continuing cumulative improvement.

In summary, SDIO needs to seriously plan and manage the technology transition process explicitly involving all the relevant organizations including the SEI and the contractors who will use the technology while paying particular attention to technical compatibility, evaluation, the software engineering environment, and the implications of the early 1990's full scale engineering development decision. Doing this should greatly improve the chances that the necessary BM/C3 software technology will be available and used.

4.4 Recommendations Support Selection of Tasks

A number of previous studies concerning the SDI have made recommendations that relate to SDI BM/C3 software technology. Figure 6 shows these recommendation arranged by task areas plus the areas of R&D management and general systems architecture. The prior recommendations shown in the columns come from the fifth volume of the Fletcher report [McMillan et al 84], the Eastport Study Group report [Eastport 85], a study performed for the Army Strategic Defense Command [Huntsville 1986], a proposal made to the National Research Council [NRC 86], and a JASON summer study [JASON 85].

While not always in total agreement, these recommendations from diverse groups provide a surprising level of support for the recommendations in this plan. This is true not only for the task areas but also the recommendations provided in Section 5.0 on R&D management.

4.5 Interrelationships Among Task Areas

The task areas are interrelated in two ways. The first is that they aim toward providing results for the series of prototypes leading up to the early 1990's demonstration of feasibility and to any eventual SDI system, and the second is that they meet or overlap technically either in subject matter or in the SDI operational system and its support system. The series of major experiments or demonstrations that will lead up to the full scale development decision are not yet defined. However, the near term efforts are aimed at providing results that can attempt to show feasibility in the early 1990's. Indeed, in this preliminary plan few of the tasks timelines shown in Appendix C have any specificity beyond 1995.

The SDI operational software can be thought of as application software sitting on top of a foundation distributed system including man-machine interface, data management, operating system and network communications (plus possibly special fault-tolerance) services, and operating partially on general purpose computers and partially on parallel processors (Figure 7). Thus, an important operational interface (possibly different) will exist in the platform and platform management environments between the application software and the foundation distributed system.

The foundation distributed system services will have a more intimate relationship with each other, but clean interfaces among their parts should nevertheless exist.

The software engineering environment will have interfaces to the computer system engineering tools; to simulation tools, systems, and testbeds; and to the operational system. In addition, it will have interfaces to its users in multiple organizations and to the research community (e.g., SDInet, Arpanet).

	DTST Vol. V	Eastport Study	BMD Study	NRC Proposal	Jason Summer Study
APPLICATION BM/C 3	• Specifying, generating, testing, and maintaining the software for a battle management system will be a task that far exceeds in complexity and difficulty any that has yet been accomplished in the production of civil or military software systems. • The battle management system and its software must be designed as an integral part of the BMD system as a whole, not as an applique. • Specific work is needed on algorithms related to critical battle management functions. • The speed and capacity needed for signal and data processors in a large BMD system exceed those available today. The basic technology is evolving rapidly and is likely to be available when needed. • The battle management system must provide a high degree of automation to support the accomplishment of the weapons release function.	• Develop the battle management system as an open system that allows rapid insertion of new assets and modification of existing pieces. The SDIO should sponsor research into the design of open systems. • SDIO should support a few prototyping projects for the development of battle management systems, with emphasis on the methods used, such that these efforts could be developed into full scale deployable systems. These prototypes must exhibit the properties important for the strategic defense system, such as responsiveness, robustness, network based, testability, and design for diversity and for evolution. • SDIO should manage the above battle management prototyping and the architecture projects such that they can influence each other.	Control theory a) Scientific Distributed/ Decentralized Control Theory b) *Distributed Dynamic Theory* Target Allocation (including Planning Algorithm) Estimation/Decision Theory Multisensor Correlation - Distributed - Attribute As Well As Position		
FOUNDATIONS COMMUNICATIONS		• Initiate a study of communications architectures beyond those implied by Volume V of the Fletcher Report. • Develop simulation of the spaceborne communication network to study system and deployment options. • Evaluate the dedicated communication network concept. If the evaluation shows promise, a space-borne prototype network should be designed, implemented, and deployed for final evaluation. The emphasis should be placed on low cost via mass production and deployment technique options, deployment strategies, and potential for an additional payload such as computing and sensors. • Pursue research and development of the algorithms for operating the recommended store-and-forward communications net work. • Address the development of protocols as early as possible. • Pursue research and development for advancing the communication technology toward the special requirements of the envisioned strategic defense System. • Pay early attention to development of new security systems because this is expected to be a very long process. • Establish a computer communication network (SDInet) to facilitate cooperation and information exchange among SDIO contractors.	Networks Network Management Algorithms and Protocols *Communications* a) Communication Coding and Waveform Development b) Security	Real-time operation Considering the system size, complexity, global development, and required *endurability("toughness")*, the demands for rapid communications and intra-systems response will be very high. This will include the processing and displays that support military commanders in deciding whether and how to commit system weapons.	Develop software standards for SDI (SDISS) - Communications Protocols - Network Protocols

Figure 6. Prior Recommendations

32

	DTST VOL V.	EASTPORT STUDY	BMD STUDY	NRC PROPOSAL	JASON SUMMER STUDY
RELIABILITY AND SURVIVABILITY	• The responsibility of currently available space systems and platforms, in a benign environment, approaches but does not meet the needs of a future large BMD system. The DoD must expand its effort on the development of electronic components hardened to a radiation environment. A new program is needed to develop hardened and fault-tolerant computers that have the speed and capacity needed for a BMD system. • There is no technical way to design absolute safety, security, or survivability into the functions of weapons release and ordnance safety. Standards of adequacy must, in the end, be established by fiat, based upon an informed consensus and judgment of risks.	• Develop technology for fast recovery from major upsets and loss of components. • Causes and types of failures in SDI/space environment must be integrated in designs from beginning. • Requirements necessitate extensive use of fault tolerant technology • Give adequate consideration to the programming aspects of fault-tolerant architectures. • Research software testing, sun time scheduling, verification and other mathematical proof techniques, system specifications and software for reliable systems.	Fault Tolerance a) Coverage, Connectivity, and Control b) Software Fault Tolerance c) Damage Tolerance d) Evaluation and Verification	Fault Tolerance The most careful design will not preclude occasional system inconsistencies, errors, and failures. Systems architecture and design, particularly for software, must ensure that these have small impact on overall systems effectiveness.	
SOFTWARE ENGINEERING ENVIRONMENTS	• Expanded efforts to generate software development tools are needed.	• Study should support research in software technology in the directions that are expected to influence and improve the development of the software for the strategic defense system. Section V.B.2 of this report has a list of such topics including •• Massive computing power for software development •• Software testing •• Software for reliable systems •• Runtime checking •• Probabilistic techniques •• Verification and other mathematical proof techniques •• System specifications •• Organizational issues •• Exploratory programming •• Parallel, concurrent, and distributed computing •• Software maintenance and reuse SDIO should manage these projects to assure their influence on the software development efforts • SDIO should not select a particular programming method (or style) as an exclusive method for strategic defense software development. • Experiment and measure development environments as part of prototyping		• Unique ways to "capture" and coordinate system specifications; • Interface specification tools and controls; and • Software production tools that will sharply reduce the cost of applications programming.	• Develop a Computer-Aided Specification System (CASS) containing an expert system core that can be adapted to various applications. Multiple proposals should be sought and evaluated. • Develop a Very High Level Language (BOOLE) similar to the way ADA was developed. Multiple proposals should be sought and evaluated. • Develop an Automated Programming (APS) in cooperation with the DARPA program. Multiple proposals should be sought and evaluated. • Develop a SDI library (SDIL) of reuseable subprograms, both independently and in cooperation with STARS. • Develop software standards for SDI (SDISS) for Ada.
PEOPLE AND ORGANIZATIONS		• Perform research on organizational issues in software development • Establish SDI net for sharing among contractors			• Establish Conferences, Summer Studies, "Schools", etc.; to train more professionals in advanced programming techniques and especially in the techniques of building automated specification and programming systems.
TECHNOLOGY TRANSITION		• SDIO should manage software research projects to assure their influence on software development efforts. • Create infrustruction for rapid technology insertion.			• Track and cooperate with the Space Station software work.

Figure 6 (Continued)

33

	DTST VOL V	EASTPORT STUDY	BMD STUDY	NRC PROPOSAL	JASON SUMMER STUDY
DISTRIBUTED OPERATING SYSTEMS					Develop Software Standards for SDI (SDISS) - Systems Calls
DISTRIBUTED DATABASES			Data Management Data Consistency/Performance Tradeoff		Develop Software Standards for: SDI (SDISS) - Database Formats and Organization
HUMAN FACTORS-MMI					Develop Software Standards for SDI (SDISS) - Graphics Display
ARTIFICIAL INTELLIGENCE			*Artificial Intelligence* a) Reasoning Over Time with Uncertainty b) Situation Assessment c) Planning Over Time with Uncertainty d) Strategy/Tactics Planning e) Diagnosis and Repair		
SOFTWARE FOR SUPERCOMPUTING					
SOFTWARE ENGINEERING PROCESS: HARDWARE/ SOFTWARE SYNERGY		• Establish centers for studying algorithmic, compilation, and *operating systems* issues for concurrent computer systems. • Massive computer power for software *development*			
SIMULATION AND EVALUATION	• Further emphasis is needed on simulation as a means to assist the design of battle management systems and software. • The problem of realistically testing an entire system, end-to-end, has no complete technical solution. The credibility of a deployed system must be established by credible testing of subsystems and *partial functions and by continuous monitoring* of its operations and health during peacetime.	• Construct diverse but coordinated simulators at several levels for testing and debugging the strategic defense system and evaluating BM/C3 strategies. • *Make the simulators available* to their users over computer communications networks. • Verify and validate simulation systems, including the attack scenarios and battle management architectures. • Develop weapon and sensor simulations through joint efforts of simulation specialists and contractors developing the weapons and sensors. • Implement immediately high-level simulators that permit the evaluation of battle management architectures and strategies under a set of attack scenarios.		• Powerful testing techniques that simulate credible situations that will only be encountered in wartime, while maintaining effectiveness in the face of continuing system change;	

Figure 6 (continued)

	DTST VOL V	EASTPORT STUDY	BMD STUDY	NRC PROPOSAL	JASON SUMMER STUDY
R & D MANAGEMENT		• Develop strength in computer science *within its own staff.* • Employ an independent, ongoing advisory panel. • Develop a communication network to encourage the development of a technical infrastructure among SDI contractors. • Establish contractor cross-checks. • Establish an independent technical support organization. • Sponsor research in program management and contracting. • An attempt has been made to discourage duplication of efforts under SDI in the areas were served by other initiatives. SDI participants however, must be encouraged to take the initiative in using the results of those other efforts. The proposed infrastructure will help the participants confidently reuse the results of current *research.*		• In addition, the acquisition strategy for the SDI will need to meet exceptionally stringent demands. The SDI system will consist of numerous sensing, weapon, communication, and management elements which will phase in and out over decades, will be continually modernized, and will need to interoperate effectively. • Review the SDI information systems situation and provide advice, guidance, and recommendations that will hel research and development activities achieve information processing systems that meet the SDI system requirements.	
GENERAL ARCHITECTURE ISSUES		• Find architectures that are admissible but rely on greatly reduced amounts of software • Reject any architecture that is not admissible, feasible, and testable. • Work toward developing a decentralized strategic defense architecture. • Sponsor additional research investigating possible defense system architectures. In particular, this research should examine the consequences of reducing the assumed degree of coordination and interdependence among system parts.	System Theory a) Modeling - Complex, Multi-hybrid System b) Hierarchical MOE/MOP Computer Systems Spaceborne Computer Architecture - Parallel/Distributed	• Architectures that support the size, complexity, unpredictability, endurability, fault-tolerance, and real-time operation *characteristics and* requirements while allowing systems changes that incorporate new requirements, subsystems, and technologies;	

Figure 6 (Conclusion)

35

4.6 Priorities and Emphases Across Task Areas

The budget for BM/C3 software technology is currently insufficient to support all the efforts that could be beneficially undertaken and this is expected to continue to be the case for at least the next several years. This and other considerations imply the need to consider the comparitive merit of efforts across task areas in addition to the consideration already given within each area.

Figure 8 lists the major considerations for setting priorities across task areas. As can be seen a large number of considerations exist, and they are not all precisely knowable or easily reconcilled with each other. This makes it inadvisable to approach the across task area comparison through (spuriously accurate) quantitative analysis. Rather, certain qualitative conclusions and strategic and tactical programmatic and technical approaches can be suggested.

A number of qualitative conclusions can be readily drawn

- Unclear or unknown technology requirements can make it difficult to make technology R&D decisions, but these tend to be only regarding major aspects of the system such as degree of centralization and approaches to key systems difficulties such as security, midcourse discrimination, or weapons assignment. Within task areas even today's vague understanding was not particularly hampering except for causing a few problems to be addressed that ultimately may not need as much attention and causing subareas to be addressed on a more abstract basis than might have been the case.

- Uncertainties should be explicitly stated and factored into decisions.

- Both the parts of the SDI system and pieces of the technology will be developed in a highly parallel fashion and it is, therfore, important that the need and capability to integrate parts of the system and technologies and evolve them over time be addressed and solved early. This means an early concern for interface standards and open architectures for both the system and the (software) engineering environment.

- Emphasis should be given to items that are essential to the system and that are important for SDI Program credibility (Figure 3's list of showstoppers is relevant as is Figure 2 on the FSED decsion).

- If the development of the SDI system decreases in probability (either for technical, program credibility and advocacy, or international considerations), then added concern should be given to usefulness of the results in programs other than SDI.

- Problems unique to SDI have less likelihood of being solved by others.

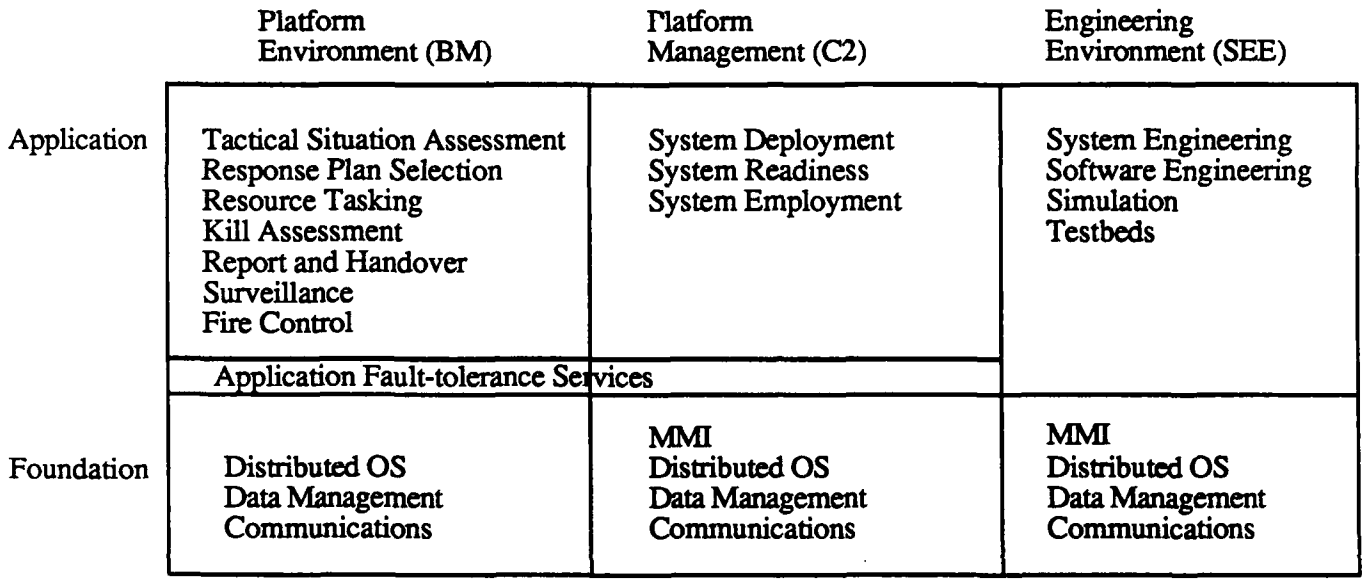


Figure 7. Functional Structure of SDI System

- Definition of Technology Requirements
  - System/Software Requirements Over Time
  - Architectures and Models
  - Evolutionary Strategy
  - Technological Gap Identification
  - Technological Opportunity Identification
- Relationships Among Parts of System
  - Among Parts
    - Uses (invokes)
    - Uses Output
    - Integrating Capability
  - Among Technology
    - Capability
    - Prerequisite
    - Integrating Capability
- Benefits/Costs
  - Gap or Opportunity Size
  - Essentiality
  - Impact on Understanding of Requirments
  - Impact on Architecture
  - Impact on Evolutionary Strategy
  - Impact on System Performance/Cost
  - Impact on Development sand Support Capabilities
  - Impact on Creditability
  - Likelihood Others will Solve
  - Cost of Technology R&D
  - Timeline of Technology R&D
  - Predictability of Technology R&D
  - Quality of Technology R&D Performers
  - Likelihood and Costs of Use
  - Enhancement to Technology Transition Capabilities
- Type of Use
  - Arms Control Negotiation
  - Budget/Program Advocacy
  - Policy, Standards, and Acquisition Practices
  - Prototypes
  - Full Scale Engineering Development Decision
  - Development and Support of SDI System
    - Response to Early Breakout
    - Nominal
  - Systems other than SDI
- Time of (First) Use
  - 1988 - 1990 (Early Prototypes)
  - 1980 - 1993 (Mid-term Prototypes and FSED Decision)
  - 1993 - 2000
  - 2000 - 2010
  - 2010 +

Figure 8. Considerations for Setting Priorities

- Obviously preferred are low cost, early payoff, less risky, high impact, non-duplicative, high quality efforts with easy to transition, low operating cost results having potential across all architectures and types of uses and that are either essential or fit well with the open architectures, and evolutionary strategies and greatly increase credibility.

- Building on or accelerating existing or planned work by others will often yield a more cost effective approach than going it alone, and will allow addressing subareas that otherwise might not be addressed adequately.

- Much of the policy, standards, and particularly acquisition practices are easier to implement efficiently and effectively at the beginning of acquisitions rather than later.

- Acquisition, people and organizations, and technology transition aspects can cause failure even if adequate technology exists.

- Measurement, comparison, evaluation, learning, and improvement are central to software technology strategy.

- Given the difficulty of SDI's problem only the highest quality efforts can be expected to meet SDI's needs, and even they may need several iterations and special educational, equipment, and other improvement assistance.

So what can be said from all of this about the relative priorities and emphasis across task areas? Figure 9 attempts to combine all the considerations and list aspects in order of priority or emphasis down the lefthand side and the task areas involved across the columns.

They are divided into five major levels (numbered 1-5) and in order within each level (as indicated by letter). The differences within levels are smaller than between levels. A single task area may have aspects or subareas at different levels of priority.

Broadly the aspects ending up near the top are ones having to do with understanding SDI BM/C3 and preparing to preceed.

The second level has the try, learn, and improve thrust; C2 interfaces since these are nearer to current systems than the battle management ones that are at the top level; unique applications because of the small chance others will solve their problems; and the key technical and credibility problem of software reliability.

The third level has three more important aspects parallel processing, technology transition (outside the key contractors), security and distributed systems technology. The fourth level drops down substantially from the third and includes safety, and people and organization aspects outside the main contractors. The bottom level contains aspects that should be supported mainly outside the software technology sphere, systems engineering and simulation technology.

This overall plan is still preliminary and will itself be improved in several ways (as well as there being the creation of more detailed yearly planning documents). One of the important ways it will be improved is by a more detailed description of the interworking among the tasks. Section 5 outlines the schedule and nature of planning intended to be done over the coming period.

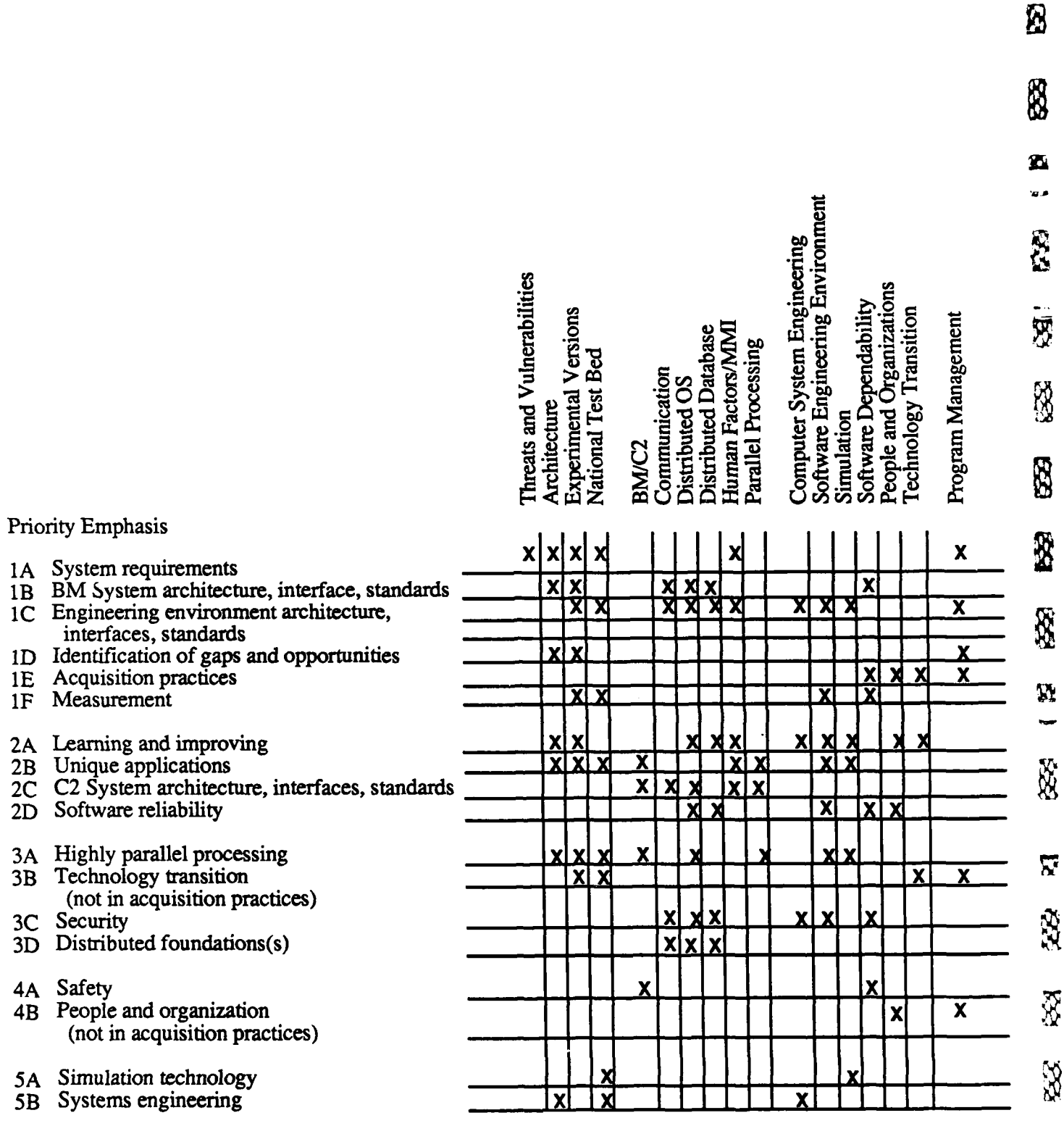


Figure 9. Priorities/Emphases Across Task Areas

5.0 R&D MANAGEMENT PLAN

5.1 Introduction

This section outlines a software technology R&D management plan for the SDIO. It describes the general approach, roles of the SDIO, use of other organizations/programs, planning and control, quality assurance, and budget. In addition, it recommends an annual cycle and management and reporting systems.

5.1.1 Purposes of R&D Management for SDI Software Technology

The ultimate purposes of software technology R&D management are to acquire the capability to develop the software for the SDI system and enough information to make a full scale development decision in an efficient and timely fashion. R&D management implements the strategy described in Section 3.0 and provides mechanisms to ensure that the strategy works.

The functions of R&D management are to plan, organize, direct, track, assess, take corrective action when necessary, exploit unexpected opportunities, and assure that results of R&D are transitioned into practice. R&D plans, funds transfers, tasking orders, and progress reports are among the tools it uses to perform these functions.

5.1.2 Context

The BM/C3 software technology R&D management plan should be viewed within the larger contexts of the overall SDI technology R&D management effort, other software technology R&D efforts, and the SDI, DoD, and Congressional decision-making process. Software is also a concern of other elements of SDIO for weapons, sensors, and innovative science and technology.

The SDI effort seeks to demonstrate and independently evaluate alternative SDI system and BM/C3 architectures and key technologies, in accordance with the SDI Technology Verification Strategy (TVS), to support an informed full scale development decision. Section 3.0 describes the TVS in detail; Figure 4 illustrates the major components of the TVS. Coordination among SDI technology R&D management efforts is critical since the technologies covered by one area are related to technologies covered by other areas. For example, software is related to computer and communications hardware.

The larger context of other software technology R&D efforts includes DoD programs, other Federal Government programs, private industry including consortiums, and international efforts all intended to improve the state of the art and the state of practice in software technology. SDI software technology R&D management should coordinate with these programs and obtain results from them to exploit these results and to avoid unwarranted duplication of efforts. Figure 10 lists some of these programs.

The coordination effort should be systematic and deliberate, striving for current awareness of technological developments through such actions as translating foreign technical information, requesting to be put on key organizations' mailing lists and on automatic distribution for their technical publications, and initiating personal contacts with representatives of other programs for formal and informal technical interchanges. Whenever possible, it should establish arrangements with private consortia, such as the MCC or the Software Productivity Consortium (SPC) to learn of research results early. It

is in U.S. industry's interest to keep the SDI Program, as a potential buyer of innovative technology, aware of technological developments.

Finally, the context of SDI, DoD, and Congressional decision-making should not be overlooked. BM/C3 SDI software technology R&D management must provide timely input to these processes as well as react swiftly to their decisions.

5.2 General Approach

The general approach to software technology R&D management assumes that the SDIO will provide oversight and guidance at a high level of abstraction with support from FFRDCs and contractors. Project management and acquisitions will be tasked to other organizations and programs with the appropriate expertise and mechanisms in place. The approach also calls for independent R&D reviews.

5.3 Roles of the SDIO

The SDIO will play external relations, management systems, and decision making roles. The SDIO's small size necessitates a limited role supported heavily by other sources including other programs/organizations, consultants, FFRDCs, and Advisory Committees.

One of the most important roles of the SDIO is its external relations role. This includes proposing, informing, and advocating the SDI program upward in the DoD and Congress, and outward to the technical community, and the general public. A related role is that of coordinating and negotiating with other DoD, foreign and national implementing organizations to perform and manage various research projects or areas (See Figure 10).

SDIO's role as a decision maker is supported by its management system. The SDIO will be the ultimate decision authority on tasking and funding as well as the approver of the R&D management plan. It will also take responsibility for selecting and establishing organizational arrangements.

The SDIO must have appropriate and adequate information on a timely basis to make these decisions, to meet requirements levied by its management or Congress, and to monitor activities under its control. The SDIO has responsibility for setting up a management system, ensuring that it works well, and for taking corrective action as needed.

This system should require the preparation of an R&D management plan. Once the plan is approved and arrangements are made with outside entities to carry out various areas of research, the system should be set up to move funds and receive progress reports expeditiously. The management system should also establish standards, guidelines, and directives to promote uniformity and good practices across projects. It will set and adjust limits within which the projects operate by delegating appropriate levels of detailed planning, control, and corrective decisions to the implementing programs and organizations.

DoD

- Ada Program
- STARS Program
- Software Engineering Institute
- VHSIC Program
- ASCENT Program
- WIS Program
- Army Computing Technology R&D
- Navy Computing Technology R&D
- Air Force Computing Technology R&D
- Software Test and Evaluation Project

U.S. Government

- National Computer Security Center
- NASA
- National Bureau of Standards
- National Science Foundation

U.S. Industry

- MCC
- Software Productivity Consortium
- Other

Foreign

- Alvey (U.K.)
- ESPRIT (Europe)
- ICOT
- Sigma (Japan)
- Software Plant Project (Brazil)
- Joint Software Engineering Project (Singapore)
- Italian program with five projects

Figure 10. SDI Software Technology Related Programs

Several advisory or liaison groups exist. A BM/C3 Advisory Panel and the Tri-Service Working Group advise on the entire BM/C3 effort, not just software. An Inter-Agency group is concerned with coordinating U.S. Government software technology R&D. The contractual effort of the Eastport Group has considered the BM/C3 software effort as a whole. Several tri-service working groups concerned with particular issues or task areas have been established including ones for standards, MMI/graphics, and distributed systems. These contrasts with the BM/C3 Software Technology Task Area review groups discussed later that each are planned to be composed of outside experts.

5.4 Planning and Control

Planning and control for the SDI program includes an annually updated overall plan supplemented by annual plans. The SDIO, its advisory groups, and implementing organizations will all play roles in this annual planning process. An important aspect of the annual planning and budgeting cycle is its need to be integrated into the budget cycles of programs and organizations the SDI is funding, as well as those of the SDIO, DoD, President, and Congress.

Control components of the annual cycle include reports, briefings, meetings, tasking statements, and funding mechanisms. The principal planning and control resource that the SDIO will employ, however, will be the managerial and technical expertise of the organizations acquiring and doing the R&D.

5.4.1 Start-Up

In fiscal year 1986 and to some extent in fiscal years 1987 and 1988, installation of full planning mechanisms will have been delayed; planning for those years will be at a disadvantage. As Figure 11 shows, in addition to the fourth and fifth years of the five year budgeting process, the full, more detailed DoD planning cycle takes almost three years. Therefore, planning for FY 89 begins in 1986. SDIO has been remarkably effective in rapidly beginning a large R&D effort; now the need is for more complete and systematic planning and management. Figure 12 lists some of the types of actions that are called for in the overall and annual plans.

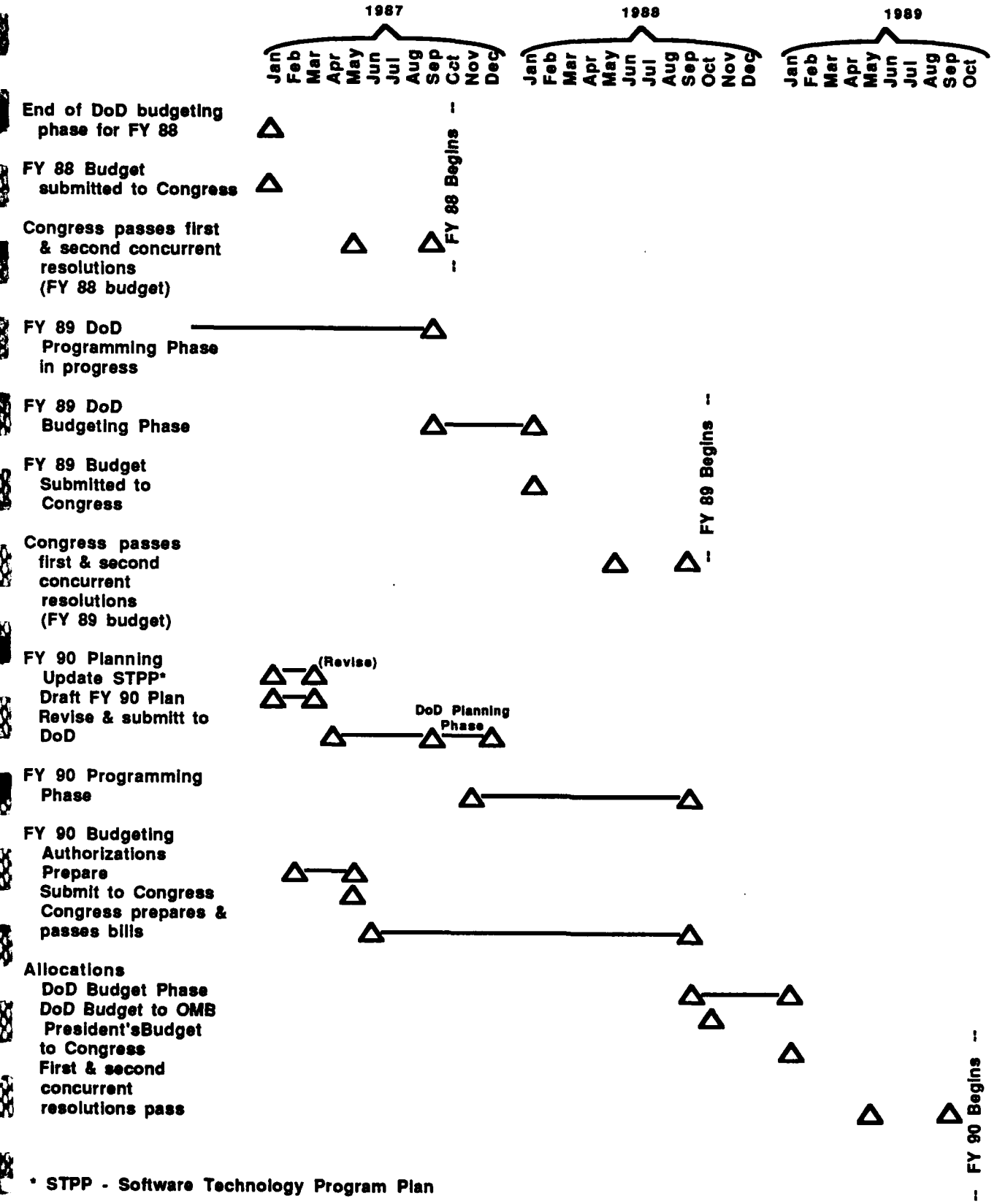


Figure 11. Overlap of Annual Cycles FY 88-90

- Study
  - Current State of the art or practice
  - Current SDI funded work
  - Particular other work in an area (e.g., NASA, STARS, VHSIC, WIS, Services, Industry)
  - Utility of SDI undertaking parallel efforts with other work or internal
  - Potential impact of efforts on SDI Program
  - Plans of others and how match SDI schedule and requirements
- Monitor
- Establish Standards
  - Review and adopt/adapt current standards
  - Develop standards: new or accelerate efforts already underway
- Research
  - Theoretic
  - Empirical
- Build Prototypes
- Evaluate
- Develop
- Integrate
- Productize
- Include in SDI system prototypes or generations

Figure 12. Some Types of Actions SDI Software Technology Plans May Call For

The first version of a SDI software technology plan was distributed in July 1986. Also in July, SDIO sponsored a week long session of proposal presentations bringing all of the old and new performers together. During the period from July through August, the SDIO reviewed the proposals and annual plan. This process continued through September.

FY87 work should have commenced in October and funds were transferred as soon after as possible.

5.4.2 Annual Cycle

The annual cycle comprises planning and control components. It will consist of quarterly reports from all of the research projects as well as semi-annual briefings, one of which will take place at an annual SDI BM/C3 Software Technology Week. The purpose of this week is to increase communication among all of the players for better utilization of results as well as planning and control including presentations on any changes in requirements. A by-product will be the relationships and understandings that result from interpersonal contacts. Figure 13 illustrates the annual cycle, the beginning of which coincides with the beginning of the fiscal year (October 1).

5.4.2.1 WPD Reports

The programs will produce their monthly Work Program Description reports. These reports will compare results to the operations plan outlining milestones, staffing, and budgets and any anticipated problems. If the work continues from the prior year, an October report will be the final progress report for that year. Subsequent reports will include work performed during the quarter, significant results, problems, and plans for the next period.

5.4.2.2 Semi-Annual Briefings

Projects will give semi-annual briefings to the SDIO on project plans, progress, and problems. The first of these will take place in January and will cover research results to date. The second briefing will take place in July at the annual BM/C3 Software Technology Week and will focus on research results. Projects not continuing on to the next year will submit final reports and briefings in November following the end of the fiscal year.

5.4.2.3 Planning

All plans will be based on the long-term Software Technology Program Plan annual updates of which will be available each February. One key input will be changes in requirements. Concrete planning for the following year and less detailed planning for future years will begin then. Revisions of Work Program Descriptions (WPD) at program, task, and certainly subtask levels may occur [WPD 86]. These plans will reflect changes in SDI requirements and the results of research as reported in briefings and reports.

The SDI's annual planning cycle must fit in with the DoD PPBS (Planning, Programming, and Budgeting System), the Federal Budget Process, and the planning cycles of the organizations with whom the SDIO will be coordinating and working. The Office of Management and Budget (OMB) begins the Federal Budget Process in May, 17 months before the beginning of the fiscal year. The Departments (including the DoD) prepare their budget requests for OMB review in October. For example, decision making on the fiscal

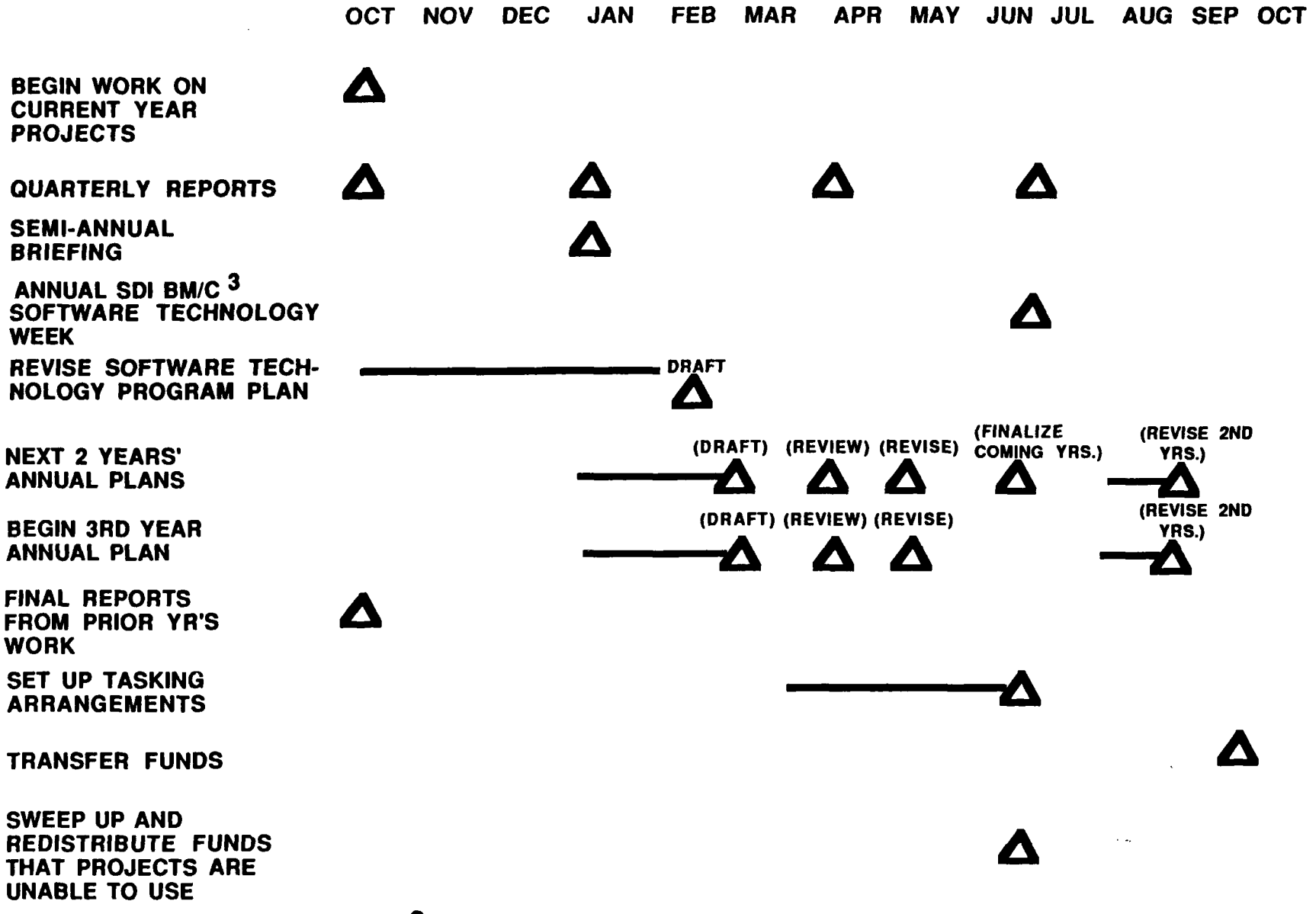


Figure 13. SDI BM/C^3 Software Technology R&D Annual Cycle

year 1986 budget was essentially completed in January 1985 when the budget was submitted to Congress.

In order to feed into to the DoD PPBS, abstract planning must begin two years prior to the beginning of the fiscal year. This means that in any given year, the SDIO will need to be working on three annual plans in addition to the Software Technology Program Plan. This longer-term plan should cover at least 7 years and provide a basis for the annual plans.

Once the Software Technology Program Plan is updated (February), the SDIO will distribute it to potential performers who will follow its guidance to draft proposed task descriptions for work to be performed over the next three years. These proposals will be submitted to the SDIO in March 1987. SDIO will negotiate with the performers and assemble the proposals into annual plans for the next three years, updating the draft annual plans for the next two years and beginning a plan for the third year. The results of these drafts will be used to update the Software Technology Program Plan.

The three year annual plans will be reviewed by the independent reviewers (described later in 5.5). Their suggestions will be incorporated into revised versions in May. In June 1987, in time for the annual BM/C3 Software Technology Week, the SDIO will tentatively approve the coming year's plan. However, some fine tuning of the plan may occur as the result of the briefings and discussions during the week. In July, the SDIO will sweep up funds that projects will have been unable to use and redistribute them to exploit unexpected opportunities. By September, the SDIO management system will have revised the second and third years' plans to reflect the finalized plan for the coming year.

Work on these programs and transfer of funds will begin in October. However, since Congressional appropriations acts are frequently not completed before 1 October, prior year funding will normally be provided through at least November.

The SDIO annual cycle takes into account all these planning cycles plus the planning cycles of the organizations with whom it will be coordinating and working.

5.4.3 Quick Response

An important aspect of planning and control for software technology R&D management is that it yield quick responses to evaluations, problems, and unexpected successes and opportunities and changes in SDI requirements. In addition to the formal reporting mechanisms described above, it is expected that managers of projects will bring unusual events on their projects (both good and bad) to the attention of the appropriate people. Outside review mechanisms such as FFRDCs and independent review groups may bring problems and opportunities to light in the course of their evaluations. In addition, the SDIO's policy should give the SDIO and its agents access to project performers at all times in order to observe progress on projects. A moderated ability to sweep up unused funds for careful redistribution as necessary is another aspect to quick response.

5.4.4 Mechanisms

In addition to the planning and control mechanisms mentioned so far, the principal planning and control mechanism will be the managerial and technical expertise of the organizations doing the research, including Government programs and contractors. Planning and control systems work best when people communicate regularly and informally as well as formally, either by phone or electronic mail. The SDI BM/C3 Software Technology Week should facilitate the process.

5.5 Quality Assurance

The regular planning and control system described above will bring some quality problems to attention, but when issues are highly technical, a more in depth technical quality assurance function must support the planning and control system.

This technical quality assurance function should involve reviewers that are independent from the performers of the work being reviewed and directly responsible to the SDIO. Other organizations that sponsor research, such as the National Science Foundation, also use independent reviewers.

In addition, the quality assurance function should set objectives and use simulation/test/demonstration results, measurement and evaluation to assure that both quality resources and processes are used. Finally, the quality assurance function should interact with the planning and control system.

A three-fold strategy for quality assurance combines acquisition practices and expert independent review with test and evaluation. Independent review will be the main mechanism for assuring quality. Figure 14 lists the review groups planned and their tentative size. They will aid in setting technical objectives and review progress toward accomplishment of these objectives. SDI software technology objectives generally derive from the SDI system and software requirements. At one level, they address the technology necessary to achieve a minimal SDI system. On a more ambitious level, technical objectives are to more predictably build a more advanced, higher quality SDI system quicker and for less cost.

Using the NTB and other capabilities, the SDI software and software technology will be tested, measured, and evaluated. Time, size, and performance objectives are relatively easy to measure compared with quality objectives such as reliability or maintainability. Measurements and evaluations are central to the overall strategy of developing the SDI system and software technology because they provide evidence that the technology being tried is adequate or inadequate to do the job. The SDIO can take steps to ensure quality work include the following.

First, efforts should consider alternatives. Taking action on the first alternative proposed is a narrow approach unlikely to result in success. Second, alternatives should be done in sufficient depth to evaluate them. The "horse race" contracting strategy, while it produces alternatives, risks shallow treatment of these alternatives if each "racer" is not given sufficient funds and time to consider each alternative in depth. Third, the SDIO should direct programs to use appropriate evaluation criteria in awarding contracts to assure excellent people. The limited number of "excellent people" and the stiff competition among government and private software efforts for them demands that the SDIO coordinate with its competition to find ways to share scarce resources effectively. Finally, independent Validation and Verification (V&V) contractors can be used to evaluate the requirements, designs, products, and the processes used to create the technology that is embodied in software products.

Because quality assurance and planning and control have implications for each other, they should interact regularly. For example, an implication that quality assurance has for planning and control is that contracts be awarded to competent people. Accordingly, the planning and control function should produce reports dealing with this issue. Similarly, quality review reports by independent reviewers should flow into the planning and control system like other program reports.

Task Area	Estimated Size
BM/C3	6
Communications Distributed Operating Systems Distributed Databases	9
Man/Machine Interface	3
Artificial Intelligence	3
Software for Supercomputing	3
Hardware/Software Synergy	3
Software Engineering Environment	6
Simulation and Evaluation	4
Reliabiltiy and Survivability	4
People and Organizations Technology Transition	4
Total	45

Figure 14. Independent Review Groups

Quality assurance is as important to the SDI software technology R&D effort as planning and control. Acquisition practices and expert independent review coupled with testing and evaluation will assure that the right technologies are being researched by the right people and provide evidence that these technologies meet SDI system requirements. Quality assurance includes setting objectives, using quality people, organizations, and tools to achieve these objectives, and conducting reviews, tests and evaluations of the technologies being developed as well as the methods being used to develop them. The results of quality assurance should feed back into the planning and control loop to influence future actions.

5.6 Budget

Figure 15 shows rough tentative estimates (in $millions) covering FY87-FY91 for the recommended SDI BM/C3 software technology R&D program. It was constructed in a bottom up fashion from the plan charts in Appendix C and takes into consideration the priorities shown in Figure 9. Portion falling into the "tech" or technology budget line and the "EV" or experiment budget line are indicated.

As limited time has been spent on the estimate and the tasks are not fully defined, clearly they should be treated with caution at this time. The estimates are based on the approach of building on the work of other government programs including using a basic automated software engineering environment developed elsewhere---for example, the software support environment to be developed by NASA for the Space Station.

		FY88	FY89	FY90	FY91	FY92	5-YEAR TOTAL
BM/C2	Tech	16.0	16.0	16.0	12.0	12.0	72.0
	EV	8.0	12.5	12.5	16.5	16.5	66.0
Network Communications	Tech	4.0	5.0	5.0	4.0	3.0	21.0
	EV	6.5	6.5	6.5	6.5	6.5	32.5
Distributed Operating Systems	Tech	12.0	12.0	5.0	5.0	5.0	39.0
	EV	6.5	6.5	6.5	6.5	6.5	32.5
Data Management Systems	Tech	3.0	4.0	4.0	4.0	4.0	19.0
	EV	6.5	6.5	6.5	6.5	6.5	32.5
Man-Machine Interface	Tech	2.5	2.5	2.0	2.0	2.0	11.0
	EV	.5	4.0	4.0	4.0	4.0	16.5
Parallel Processing	Tech	8.0	8.0	8.0	8.0	8.0	40.0
	EV	4.5	1.5	1.5	4.0	4.0	15.5
Computer System Engineering	Tech	6.5	10.0	10.0	10.0	8.0	44.5
	EV					2.0	2.0
Software Engineering Environments	Tech	20.5	20.5	20.5	20.5	15.0	97.0
	EV	5.0	9.5	11.0	12.5	13.5	51.5
Simulation	Tech	6.0	6.0	6.0	6.0	6.0	30.0
	EV/NTB	6.5	9.0	9.0	9.0	6.0	39.5
Software Dependability	Tech	12.0	12.0	12.0	12.0	10.0	58.0
	EV	8.0	8.0	8.0	8.0	10.0	42.0
People and Organizations	Tech	5.5	5.5	3.0	3.0	1.0	18.0
	EV	5.0	12.0	12.0	12.0	15.0	56.0
Technology Transition	Tech	5.0	5.0	5.0	5.0	5.0	25.0
	EV	9.0	14.0	16.5	16.5	16.5	70.0
R&D Program Management	Tech	6.0	6.0	6.0	6.0	6.0	30.0
Total	Tech	107.0	112.50	102.50	97.50	85.0	504.50
	EV/Other	66.0	90.0	91.50	102.0	107.0	456.50

Figure 15. SDI BM/C3 Software Technology Budget Estimates ($Millions)

6.0 REFERENCES

[Adam 85] Adam, John A. and Fischetti, Mark A., "SDI: The Grand Experiment," *IEEE Spectrum* (September 1985), pp. 34-64.

[Adkins 77] Adkins, Gerald and Pooch, Udo W., "Computer Simulation: A Tutorial," *IEEE Computer* 10, 4 (April 1977), pp. 12-17.

[ANSI/IEEE 83] IEEE Computer Society, IEEE Standard Glossary of Software Engineering Terminology, ANSI/IEEE STD 729, 1983.

[Boehm 86] Boehm, Barry, "SDI Software Development," SDI Architecture Study Briefing, 16-17, 1986.

[DOD-STD-2167] U.S. Department of Defense, *Defense System Software Development, Military Standard, DOD-STD-2167* (June 4, 1985).

[Druffel 83] Druffel, Larry E., et al., "The STARS Program: Overview and Rationale," *IEEE Computer* (November 1983), p. 21.

[Eastport 85] Cohen, D., et al., *A Report to the Director Strategic Defense Initiative Organization*, Eastport Study Group, Summer Study 1985.

[Elmer-DeWitt 85] Elmer-DeWitt, Phillip, "Star Wars and Software," *Time* (July 22, 1985), p. 73.

[Foley 85] Association for Computing Machinery, *Twelfth Annual Conference and Exhibition on Computer Graphics and Interaction Techniques*, San Francisco, July 22-26, 1985, Course notes on "How to Design Computer-User Interfaces" and Advanced Topics - Human Factors in Computer Graphics Systems."

[Ford 85] Ford Aerospace and Communications Corporation, ESD BM/C3 Architecture Definition Program Status Briefing (U) (December 13, 1985), (Secret classification)

[Fletcher 84] Fletcher, James C. et al., *Report of the Study on Eliminating the Threat Posed By Nuclear Ballistic Missiles*, (February 1984).

[Huntsville 86] U.S. Army, *SDI Large-Scale System Technology Study*, Final Report, Army SDC Huntsville, AL (March 28, 1986).

[JASON 85] The MITRE Corporation, *JASON, Report on the Workshop for Automated Software Programming*, JSR-85-927, McLean, VA (December 2, 1985).

[McDonald 86] McDonald, Cathy, and Redwine, Samuel T. Jr., *STARS Glossary: A Supplement to the IEEE Standard Glossary of Software Engineering Terminology, Version 3.0*, Institute

for Defense Analyses, Paper P-1846 (January 1986) (draft), p. 49.

[McMillan 84] McMillan, Brockway, et al., *Report of the Study on Eliminating the Threat Posed By Nuclear Ballistic Missiles*, James C. Fletcher, Study Chairman, Volume V, *Battle Management, Communications, and Data Processing*, Brock way McMillan, Panel Chairman, (February 1984).

[Marbach 85] Marbach, William D. et al, "The STAR Warriors," Newsweek (June 17, 1985), pp. 34-45.

[Nash 85] Nash, Sarah H. and Redwine, Samuel T., Jr., *Information Interface Related Standards, Guidelines, and Recommended Practices, SEE-INFO-004*, Institute for Defense Analyses, Paper P-1842 (July 1985).

[NRC 86] National Academy of Sciences, National Research Council, Commission on Engineering and Technical Systems, Board on Telecommunications and Computer Applications, *Proposal No. 86-146 for the Establishment of a Committee on SDI Information Systems Technology* (March 1986).

[NTB 86] *Statement of Work Strategic Defense Initiative (SDI) National Testbed Concept Definition and Preliminary Design*, RFP SDIO84-86-0001, Attachment 1 (January 10, 1986).

[Offutt 85a] Offutt, Commander James and Danese, Dr. J. Bryan, "Communications for SDI," *Signal* (July 1985), pp. 27-34.

[OTA 85] U.S. Congress, Office of Technology Assessment, *Ballistic Missile Defense Technologies*, Government Printing Office, (September 1985).

[OTA 85a] U.S. Congress, Office of Technology Assessment, *Anti-Satellite Weapons, Countermeasures, and Arms Control*, Government Printing Office, September 1985.

[Peters 82] Peters, Thomas J. and Waterman, Robert H., Jr., *In Search of Excellence: Lessons Learned from America's Best-Run Companies*, Harper and Row, 1982.

[Rensberger 85] Rensberger, Boyce, "Computer Bugs Seen as Fatal Flaw in 'Star Wars'," *Washington Post* (October 30, 1985), p. A1

[SDC 86] U.S. Army Strategic Defense Command, Contract #DASG 60-86-C-0054, July 1986.

[Shannon 75] Shannon, Robert E., "Simulation: A Survey with Research Suggestions," *AIIE Transactions* 7, 3 (September 1975), pp. 289-301.

[Turner 85] Turner, Robert D., *Assessment of C3 Capabilities for Strategic Defense Initiative*, Institute for Defense Analyses, Alexandria, VA (July 1985).

[WPD 86] SDIO, *WPD User Documentation, Version 1* (February 1986).

Appendix A

What SDI is Currently Funding (Categorized by Recommendations)

Introduction

In this appendix, each of the projects funded by the BM/C3 element of SDIO in FY87 is listed, ordered by Work Package Directive (WPD) number. The objective of each project is given, along with the budget allocations and the recommendations from this plan that are related to the ongoing work. Each recommendation is identified by the Task area in which the recommendation was made and the number of that individual recommendation.

Further information on these recommendations, including the text of the recommendation, the motivation for the recommendation, and further details can be found in Appendix B.

FY87 FUNDED EFFORTS

BY PRIMARY TASK AND WORK PACKAGE DIRECTIVE

TITLE OF EFFORT	DESCRIPTION OF EFFORT	APP FY87$	AGENCY	RELATED RECOMMENDATIONS
** PRIMARY TASK: ALG				
* WORK PACKAGE DIRECTIVE: B412				
TARGET CLASSIFICATION	IDENTIFY TECHNIQUES AND ALGORITHMS FOR IMPROVED TARGET CLASSIFICATION; DEFINE REQUIREMENTS FOR BM/C3 SYSTEM TO INITIATE PASSIVE OR ACTIVE RESPONSE	300	SDC	BM/C3 4
MULTIPLE INFORMATION SET TRACKING & CORRELATION (MISTC)	DEVELOP ADVANCED APPROACHES FOR MULTIPLE SENSOR FUNCTIONAL CORRELATION AND TRACKING	2000	SDC	BM/C3 4
TARGET ASSIGNMENT ALGORITHM	DEVELOP CANDIDATE SET OF ALGORITHMS FOR WEAPON-TARGET PAIRINGS LATE MIDCOURSE AND TERMINAL PHASES; IMPLEMENT AND EVALUATE SELECTED ALGORITHMS IN REALISTIC ENVIRONMENT	1000	SDC	BM/C3 4 PARALLEL PROCESSING 5

TITLE OF EFFORT	DESCRIPTION OF EFFORT	APP FY87$	AGENCY	RELATED RECOMMENDATIONS
*** WORK PACKAGE DIRECTIVE: B413**				
BM BENCHMARK ALGORITHMS	COMPARE DATA PROCESSING ARCHITECTURE IMPLEMENTATIONS	100	RADC	PARALLEL PROCESSING 5
ALGORITHM DESIGN DEVELOPMENT EVALUATION & REFINEMENT ALGORITHM AND PROCESSOR UPGRADE PROGRAM		300	RADC	BM/C3 3
BM MANAGEMENT STRUCTURE		1200	RADC	
*** WORK PACKAGE DIRECTIVE: B414**				
TRACKER/CORRELATOR	DEVELOP STATE OF THE ART HYBRID TRACKER/CORRELATOR SYSTEM THAT COMBINES OPTIMUM STATISTICAL ESTIMATION TECHNIQUES WITH HEURISTIC REASONING AND KNOWLEDGE-BASED SYSTEM TECHNIQUES; ESTABLISH FEASIBILITY OF SDI APPLICATION BY EXPERIMENTALLY DETEMINING FUNCTIONAL PERFORMANCE, SENSITIVITY, AND COMPUTATIONAL RESOURCE REQUIREMENTS	1400	NRL	SOFTWARE DEPENDABILITY BM/C3 4, 6

TITLE OF EFFORT	DESCRIPTION OF EFFORT	APP FY87$	AGENCY	RELATED RECOMMENDATIONS
SITUATION ASSESSMENT AND STRATEGIC PLANNING MANAGEMENT	DEVELOP STATE OF THE ART SYSTEM FOR SDI SITUATION ASSESSMENT AND STRATEGIC PLANNING; ESTABLISH FEASIBILITY FOR SDI APPLICATION BY EXPERIMENTAL DETERMINATION OF FUNCTIONAL PERFORMANCE, SENSITIVITY AND COMPUTATIONAL RESOURCE REQUIREMENTS	1200	NRL	BM/C3 3
WEAPON ALLOCATION AND RF SPECTRUM MANAGEMENT	DEVELOP SET OF ALGORITHMS FOR RESOURCE ALLOCATION IN SDI BATTLE MANAGEMENT APPLICATION; ESTABLISH FEASIBILITY FOR SDI APPLICATION BY EXPERIMENTAL DETERMINATION OF FUNCTIONAL PERFORMANCE, SENSITIVITY, AND COMPUTER RESOURCE REQUIREMENTS	1200	NRL	BM/C3 3
TECHNOLOGY TESTBED	PROVIDE SIMULATION FRAMEWORK IN WHICH BATTLE MANAGEMENT ALGORITHMS CAN INTERACT IN CONTROLLED MANNER TO PERMIT EXPERIMENTAL DETERMINATION OF SYSTEM LEVEL FUNCTIONAL PERFORMANCE, SENSITIVITY AND RESOURCE REQUIREMENTS	2200	NRL	BM/C3 3
HYBRID COMPUTING	DEVELOP TECHNIQUES FOR COMBINING NUMERICAL AND SYMBOLIC COMPUTING ON PARALLEL ARCHITECTURE MACHINES	1700	NRL	PARALLEL PROCESSING 5
COMMAND AND CONTROL HUMAN INTERFACE	DETERMINE APPROPRIATE ROLES FOR HUMAN SDI COMMANDER UNDER SEVERE TIME CONSTRAINTS OF SDIO SCENARIO AND INVESTIGATE ALTERNATIVE MECHANIZATIONS FOR HUMAN/MACHINE INTERFACE; ESTABLISH FEASIBILITY OF	1300	NRL	MAN-MACHINE INTERFACE 1 SOFTWARE DEPENDABILITY BM/C3 5

TITLE OF EFFORT	DESCRIPTION OF EFFORT	APP FY87$	AGENCY	RELATED RECOMMENDATIONS
	MAN-IN-THE-LOOP TASKING AND MECHANIZATIONS			
SYSTEM ENGINEERING AND INTEGRATION	PERFORM SYSTEM INTEGRATION, ENGINEERING AND EXPERIMENT DESIGN EFFORTS REQUIRED TO ESTABLISH FEASIBILITY OF AUTONOMOUS BATTLE MANAGEMENT SYSTEM	800	NRL	COMPUTER SYSTEMS ENGINEERING TECHNOLOGY 4 SOFTWARE DEPENDABILITY
*** WORK PACKAGE DIRECTIVE: B418**				
ARTIFICIAL INTELLIGENCE		1700	DARPA	MAN-MACHINE INTERFACE 2
TRANSFER				TECHNOLOGY TRANSITION

TITLE OF EFFORT	DESCRIPTION OF EFFORT	APP FY87$	AGENCY	RELATED RECOMMENDATIONS
**** PRIMARY TASK: NET**				
* WORK PACKAGE DIRECTIVE: B412				
DISTRIBUTED DATA MANAGEMENT SYSTEM	DEVELOP CONCEPTS FOR DISTRIBUTED INFORMATION MANAGEMENT SYSTEM TO ENSURE NECESSARY ACCESS TO RAPIDLY VARYING DATA	1300	SDC	DATA MANAGEMENT 2
MANAGEMENT OF DISTRIBUTED SYSTEMS	DEVELOP AND EVALUATE ADVANCED RESOURCE CONTROL ALGORITHMS FOR CANDIDATE NODE ARCHITECTURES	1200	SDC	OPERATING SYSTEMS 4
* WORK PACKAGE DIRECTIVE: B413				
COMMUNICATIONS NETWORK	DEVELOP COMPUTER COMMUNICATIONS TECHNOLOGY FOR DISTRIBUTED PROCESSING BACKBONE FOR SDI BM/C3	700	RADC	NETWORK COMMUNICATIONS 2,5
DISTRIBUTED INFORMATION PROCESSING	DEVELOP DISTRIBUTED SYSTEM RESOURCE MANAGEMENT MECHANISMS TO CONTROL LOCATION, EXECUTION, MOVEMENT, AND ACCESS TO SDI PROCESSING FUNCTIONS	3500	RADC	OPERATING SYSTEMS 4
DISTRIBUTED DATABASES	DEVELOP DATABASE TECHNOLOGY TO SUPPORT REAL TIME DATA HANDLING IN HIERARCHICALLY CLUSTERED SYSTEM WHICH INCORPORATES MULTIPLE REPLICATED AND/OR PARTITIONED DATABASES	400	RADC	DATA MANAGEMENT 1

TITLE OF EFFORT	DESCRIPTION OF EFFORT	APP FY87$	AGENCY	RELATED RECOMMENDATIONS
DISTRIBUTED SYSTEM SECURITY	DEVELOP SYSTEM SECURITY MECHANISMS TO PROVIDE MULTIPLE LEVELS OF SECURITY COMMENSURATE WITH SINGLE HOST SYSTEM	3400	RADC	OPERATING SYSTEMS 4
*** WORK PACKAGE DIRECTIVE: B414**				
TRACK FILE DATA BASE MANAGEMENT SYSTEM	DEVELOP MECHANISMS FOR ENSURING SYNCHRONIZATION OF PHYSICALLY DISTRIBUTED DATABASES; DEVELOP PARALLEL TRACK FILE DBMS SUITABLE FOR SDI APPLICATION; EXPERIMENTALLY ESTABLISH FEASIBILITY	1200	NRL	DATA MANAGEMENT 1
ADAPTIVE COMMUNICATION NETWORKING	DEVELOP STATE-OF-THE-ART SDI NETWORK CONTROL ALGORITHMS THAT ARE ADAPTIVE, AUTONOMOUS, AND WHICH IMPLEMENT PHYSICALLY DISTRIBUTED NET CONTROL STRUCTURE; ESTABLISH FEASIBILITY OF APPLICATION BY EXPERIMENTALLY DETERMINING FUNCTIONAL PERFORMANCE, SENSITIVITY, AND COMPUTATIONAL RESOURCE REQUIREMENTS	1400	NRL	NETWORK COMMUNICATIONS 2, 5
COMMUNICATIONS/ GRAPHICS	DEVELOP COMMUNICATIONS CONCEPTS (AT INFORMATION LEVEL) REQUIRED FOR SDI SYSTEMS (PROTOCOLS, TRADEOFFS,...), DEVELOP GRAPHICS TECHNIQUES TO AID IN SITUATION COMPREHENSION AND EVALUATION OF STATUS AND CAPABILITIES	1900	ONR	MAN-MACHINE INTERFACE 6

TITLE OF EFFORT	DESCRIPTION OF EFFORT	APP FY87$	AGENCY	RELATED RECOMMENDATIONS
	OF SDI ABM SYSTEM OVER TIME			
*** WORK PACKAGE DIRECTIVE: B417**				
SYSTEM ENGINEERING AND TECHNICAL ASSISTANCE		400	NSA	
SECURITY ARCHITECTURE STUDIES	ADDRESS SECURITY ISSUES IN COHESIVE MANNER AND PROVIDE FOCUS FOR TECHNOLOGY PROGRAM	2000	NSA	OPERATING SYSTEM 4 SOFTWARE DEPENDABILITY
*** WORK PACKAGE DIRECTIVE: B418**				
TRUSTED REAL-TIME OPERATING SYSTEM	DELIVER TRUSTED REAL-TIME OPERATING SYSTEM IN ADA TO ARC	2200	DARPA	OPERATING SYSTEM 4, 6

TITLE OF EFFORT	DESCRIPTION OF EFFORT	APP FY87$	AGENCY	RELATED RECOMMENDATIONS
****				
**** PRIMARY TASK: PRO				
* WORK PACKAGE DIRECTIVE: B413				
PROCESSING	FSED SPECIFICATION OF HP, FT SURVIVABLE COMPUTER ARCHITECTURE CAPABLE OF SUPPORTING AUTONOMOUS, TRUSTED BM PROCESSING IN SPACE	2000	RADC	COMPUTER SYSTEMS ENGINEERING TECHNOLOGY 2
FORMAL VERIFICATION TECHNOLOGY	DESIGN, DEVELOP AND DEMONSTRATE TOOLS, TECHNIQUES AND METHODOLOGIES FOR FORMAL VERIFICATION SYSTEMS	3100	RADC	SIMULATION 4 COMPUTER SYSTEMS ENGINEERING TECHNOLOGY 3, 4 SOFTWARE ENGINEERING ENVIRONMENTS 5
TRUSTED SYSTEM DESIGN	EXPLOIT SOA TECHNOLOGIES IN DEVELOPMENT OF TOOLS AND TECHNIQUES TO BE USED IN DESIGN OF SECURE SYSTEMS	2400	RADC	COMPUTER SYSTEMS ENGINEERING TECHNOLOGY 4
* WORK PACKAGE DIRECTIVE: B414				
PARALLEL COMPUTING		2000	NRL	PARALLEL PROCESSING 5
CENTER FOR EXPERIMENTAL RESEARCH IN PARALLEL ALGORITHMS SOFTWARE AND SYSTEMS		5000	ONR	BM/C3 3 PARALLEL PROCESSING 5 OPERATING SYSTEMS 5

TITLE OF EFFORT	DESCRIPTION OF EFFORT	APP	AGENCY	RECOMMENDATIONS
UNIVERSAL SYSTOLIC PROCESSOR		400	ONR	
MASSIVE MEMORY MACHINE		400	ONR	
*** WORK PACKAGE DIRECTIVE: B415**				
VHSIC MULTIPROCESSOR TECHNOLOGY	DEMONSTRATE OPERATING SYSTEM METHODOLOGY FOR COUPLING SELF-TESTABILITY WITH RANGE OF USER SELECTABLE FAULT TOLERANCE IN MULTIPROCESSOR APPLICATIONS	500	NASA	SOFTWARE DEPENDABILITY
*** WORK PACKAGE DIRECTIVE: B418**				
FAULT-TOLERANT FOR SCALABLE ARCHITECTURES		3400	DARPA	SOFTWARE DEPENDABILITY
FAULT-TOLERANT EXTENSION TO MIPS		2000	DARPA	

TITLE OF EFFORT	DESCRIPTION OF EFFORT	APP FY87$	AGENCY	RELATED RECOMMENDATIONS
RP III		2500	DARPA	

** PRIMARY TASK: COM

* WORK PACKAGE DIRECTIVE: B413

TRANSMISSION & RECEPTION	PROVIDE COMMUNICATIONS LINKS (RF & LASER) BETWEEN ELEMENTS OF SDI, BOTH SPACE-SPACE & SPACE-EARTH	2000	RADC	
SYSTEM MANAGEMENT	PROVIDE OVERALL CONTROL OF COMMUNICATIONS SYSTEMS	800	RADC	
LASERCOM POINTING, ACQUISITION & TRACKING		150	RADC	
EARTH-SPACE LINKS		100	RADC	
BM COMMUNICATIONS DEVELOPMENT		500	RADC	
AGILE BEAM LASERCOM EXPERIMENT		250	RADC	

TITLE OF EFFORT	DESCRIPTION OF EFFORT	APP FY87$	AGENCY	RELATED RECOMMENDATIONS
LASERCOM RISK REDUCTION		1000	RADC	
60 GHZ PHASED ARRAY		700	RADC	

*** WORK PACKAGE DIRECTIVE: B415**

SEMICONDUCTOR LASER TECHNOLOGY FOR OPTICAL COMMUNICATIONS	DEVELOP AND DEMONSTRATE HIGH POWER, HIGH DATA RATE, AND SINGLE LOBED DIFFRACTION LIMITED BEAM PHASED ARRAY SEMICONDUCTOR PHASED ARRAY LASER FOR OPTICAL COMMUNICATIONS	500	NASA	

**** PRIMARY TASK: SWE**

*** WORK PACKAGE DIRECTIVE: B412**

DISTRIBUTED COMPUTING DESIGN SYSTEM (DCDS)	COMPLETE METHODOLOGY WITH INTEGRAL SOFTWARE DEVELOPMENT ENVIRONMENT THAT IS BEYOND CURRENT STATE-OF-THE-ART	1600	SDC	COMPUTER SYSTEMS ENGINEERING TECHNOLOGY 4 SOFTWARE ENGINEERING ENVIRONMENTS 5

TITLE OF EFFORT	DESCRIPTION OF EFFORT	APP FY87$	AGENCY	RELATED RECOMMENDATIONS
DCDS ACCELERATOR		500	SDC	SOFTWARE ENGINEERING ENVIRONMENTS 5
SOFTWARE QUALITY, RELIABILITY TOOLS		200	SDC	SOFTWARE ENGINEERING ENVIRONMENTS 5

*** WORK PACKAGE DIRECTIVE: B418**

ADA ENGINEERING ENVIRONMENT	DELIVER ADA ENGINEERING ENVIRONMENT BASED ON USING OBJECT MANAGEMENT SYSTEM RUNNING ON MACH	3400	DARPA	COMPUTER SYSTEMS *ENGINEERING TECHNOLOGY* 4 SOFTWARE ENGINEERING ENVIRONMENTS 4

**** PRIMARY TASK: EXP**

*** WORK PACKAGE DIRECTIVE: B532**

TITLE OF EFFORT	DESCRIPTION OF EFFORT	APP FY87$	AGENCY	RELATED RECOMMENDATIONS
BM/C3 CONCEPT DEFINITION	DEVELOP BM/C3 ARCHITECTURES FOR TERRESTRIALLY BASED SYSTEMS FOR CONUS & ALLIED DEFENSE; DEFINE CONSENSUS BM/C3 ARCHITECTURE; DEFINE EXPERIMENTAL BM/C3 ARCHITECTURE REQUIREMENTS	4500	SDC	
BM/C3 COMPOSITE ARCHITECTURE SELECTION TEAM (CAST)	DERIVE TERRESTRIALLY BASED BM/C3 COMPOSITE ARCHITECTURES FOR CONUS AND ALLIED BMD FROM CONCEPT DEFINITION CONTRACTS AND SYSTEM REQUIREMENTS STUDIES; DEFINE BM/C3 SYSTEM EXPERIMENTS	500	SDC	
NEAR TERM BM/C3 ARCHITECTURAL EXPERIMENTS	DEFINE AND EXECUTE NEAR TERM (FY88) DEMONSTRATION AND VALIDATION OF BM/C3 CAPABILITY IN REALISTIC, GEOGRAPHICALLY DISTRIBUTED MANNER; PROVIDE VISIBLE AND CONVINCING PROGRESS OF BM/C3 CAPABILITY	25000	SDC	SIMULATION 1 MAN-MACHINE INTERFACE 6
*** WORK PACKAGE DIRECTIVE: B543**				
BM/C3 ARCHITECTURE STUDY (PHASE II)		4000	ESD	
EXPERIMENTAL VERSION - LEVEL I	*BUILD AND DEMONSTRATE BM/C3* PROTOTYPE FOR PROOF-OF FEASIBLITY; PROVIDE VEHICLE FOR BM/C3 EXPERIMENTS IN SYSTEM CONTEXT	*10000*	*ESD*	*SIMULATION 1*

TITLE OF EFFORT	DESCRIPTION OF EFFORT	APP FY87$	AGENCY	RELATED RECOMMENDATIONS
*** WORK PACKAGE DIRECTIVE: B561**				
BM/C3 ARCHITECTURE AND REQUIREMENTS SPECIFICATION		10000	SDIO	
SETA SUPPORT - UK		2000	SDIO	
SETA SUPPORT - OTHER		1800	SDIO	
*** WORK PACKAGE DIRECTIVE: B568**				
MULTIPLE SATELLITE SYSTEM (MSS)		10000	DARPA	

TITLE OF EFFORT	DESCRIPTION OF EFFORT	APP FY87$	AGENCY	RELATED RECOMMENDATIONS
**** PRIMARY TASK: MGT				
* WORK PACKAGE DIRECTIVE: B411				
SETA SUPPORT		5000	SDIO	
BM/C3 INFRASTRUCTURE DEVELOPMENT		4700	SDIO	
* WORK PACKAGE DIRECTIVE: B413				
MANAGEMENT SUPPORT		3600	RADC	
* WORK PACKAGE DIRECTIVE: B543				
SUPPORT - MITRE		3000	ESD	SOFTWARE DEPENDABILITY
SUPPORT - SETA		1000	ESD	

TITLE OF EFFORT	DESCRIPTION OF EFFORT	APP FY87$	AGENCY	RELATED RECOMMENDATIONS

**** PRIMARY TASK: ALY**

* **WORK PACKAGE DIRECTIVE: B562**

THEATER BM/C3 ARCHITECTURE	DETERMINE FEASIBILITY OF EUROPEAN BMD BY EARLY ADDRESSAL OF ISSUES, TECHNOLOGIES, SYSTEMS AND CONCEPTS APPLICABLE TO EUROPEAN BM/C3 ARCHITECTURE	200	SDC	

Appendix B

Technology Task Area Assessments

B.1.1 Introduction

This appendix contains the twelve software technology areas assessments. Each area is reviewed to determine the state of the art and the state of practice. In addition, each area is analysed to determine how the requirements of the SDI system might not be supported by available technology. Recommendations are made for research in each area to address the needs identified.

The first section of this appendix is a consolidation of all the recommendations made in the task area assessment of Sections B.2 - B.13. In addition, each recommendation is followed by a list of projects currently funded by the SDIO BM/C3 office that are related to the recommendation.

While it is hoped that this organization will help in identifying technology issues that are not being addressed, a few caveats are in order. A recommendation may have several projects related to it and yet not be adequately addressed. For example, the related projects might address only a portion of the area identified for research. Similarly, a recommendation may have no funded projects related to it and yet be adequately addressed through other mechanisms and not as the programmatic approach.

All projects funded by the BM/C3 element of SDIO and their FY87 funding levels are listed in Appendix A.

B1.2 Battle Management/C3

Recommendation 1: Each of the architectures which have been investigated to date should identify which of the BM/C3 software modules are considered to be stressing technology. Alternative configurations of sensors,weapons, etc. should be identified which significantly reduce or eliminate the BM/C3 application software difficulties.

Recommendation 2: A glossary of terminology should be provided by the Government to each organization participating in the Phase III BM/C3 architecture and system study.

Recommendation 3: An Parallel Algorithm Test Center for research and evaluation should be established, possibly as part of the National Test Bed.

Related SDIO Funded Projects

Name: Situation Assessment and Strategic Planning Management
Agency: NRL
Objective: Develop state of the art system for SDI situation assessment and strategic planning; establish feasibility for SDI application by experimental determination of functional performance, sensitivity and computational resource requirements.

Name: Weapon Allocation and RF Spectrum Management
Agency: NRL
Objective: Develop set of algorithms for resource allocation in SDI battle management application; establish feasibility for SDI application by experimental determination of functional performance, sensitivity, and computer resource requirements.

Name: Algorithm Design Development Evaluation and Refinement Algorithm and Processor Upgrade Program
Agency: RADC

Name: Technology Testbed
Agency: NRL
Objective: Provide simulation framework in which battle management algorithms can interact in controlled manner to permit experimental determination of system level functional performance, sensitivity, and resource requirements

Name: Center for Experimental Research in Parallel Algorithms Software and Systems
Agency: ONR

Recommendation 4: Research algorithms for multi-sensor data fusion.

Related SDIO Funded Projects

Name: Target Classification
Agency: SDC
Objective: Identify techniques and algorithms for improved target classification; define requirements for BM/C3 system to initiate passive or active response.

Name: Multiple Information Set Tracking & Correlation (MISTC)
Agency: SDC
Objective: Develop advanced approaches for multiple sensor functional correlation and tracking.

Name: Target Assignment Algorithm
Agency: SDC
Objective: Develop candidate set of algorithms for weapon-target pairings late midcourse and terminal phases; implement and evaluate selected algorithms in realistic environment.

Name: Tracker/Correlator
Agency: NRL
Objective: Develop state of the art hybrid tracker/correlator system that combines optimum statistical estimation techniques with heuristic reasoning and knowledge-based system techniques; establish feasibility of SDI application by experimentally determining functional performance, sensitivity, and computational resource requirements.

Recommendation 5: The amount of human intervention should be carefully thought out as early as possible and prototyped to determine the feasibility of developing the application software.

Related SDIO Funded Projects

Name: Command and Control Human Interface
Agency: NRL
Objective: Determine appropriate roles for human SDI commander under severe time constraints of SDIO scenario and investigate alternative mechanizations for human/machine interface; establish feasibility of man-in-the-loop tasking and mechanizations.

Recommendation 6: The SDIO should support some artificial intelligence research, most probably in the area of decision support systems.

Related SDIO Funded Projects

Name: Tracker/Correlator
Agency: NRL

Objective: Develop state of the art hybrid tracker/correlator system that combines optimum statistical estimation techniques with heuristic reasoning and knowledge-based system techniques; establish feasibility of SDI application by experimentally determining functional performance, sensitivity, and computational resource requirements.

Recommendation 7: The types of decision support systems required for the SDIO system and National Test Bed should be identified. To eliminate, or reduce to a minimum, changes to the DSS, as architectures mature, the DSS design should be based upon open architecture principles.

B1.3 Network Communications

Recommendation 1: An open and extensible protocol standard that meets the reliability, security, reconfigurability, and real-time performance requirements of the SDI must be developed early.

Recommendation 2: New algorithms for naming, routing, topology updating, flow control, and clock synchronization must be developed to meet the reliability, security, reconfigurability and real-time performance requirements of the SDI.

Related SDIO Funded Projects

Name: Communications Network
Agency: RADC
Objective: Develop computer communications technology for distributed processing backbone for SDI BM/C3.

Name: Adaptive Communication Networking
Agency: NRL
Objective: Develop state-of-the-art SDI network control algorithms that are adaptive, autonomous, and which implement physically distributed net control structure; establish feasibility of application by experimentally determining functional performance, sensitivity, and computational resource requirements.

Recommendation 3: Appropriate models for security of computer communication networks must be developed. These models must be available early in the design of the SDI system as they will be a basis for many design decisions.

Recommendation 4: The SDIO should fund research in the area of automatic generation of reliable, secure, high-performance protocol software.

Recommendation 5: Prototype computer communications networks must be built. These systems should be integrated with the prototype distributed operating systems being developed for the SDI.

Related SDIO Funded Projects

Name: Communications Network
Agency: RADC
Objective: Develop computer communications technology for distributed processing backbone for SDI BM/C3.

Name: Adaptive Communication Networking
Agency: NRL
Objective: Develop state-of-the-art SDI network control algorithms that are adaptive, autonomous, and which implement physically distributed net control structure; establish feasibility of application by experimentally determining functional performance, sensitivity, and computational resource requirements.

B1.4 Distributed Operating Systems

Recommendation 1: The area of distributed modeling and analysis should be monitored to ensure that funding at the current level continues in spite of any budget cuts.

Recommendation 2: Assist hardware (in the form of surplus computing power and resources) should be used to reduce software complexity. A simplier and significantly more reliable system will be obtained if 70-80% hardware efficiency is required rather than 100%.

Recommendation 3: Dynamic assignment and scheduling research should be monitored to ensure that current work continues.

Recommendation 4: The development of four prototype fault-tolerant, real-time, secure distributed operating systems should be funded. At least one should be based on MACH. Past (successful) experience in the development of distributed systems should be a primary consideration in the awarding of these contracts. Proposals should indicate how the development of the prototypes will be integrated with projects on real-time scheduling/allocating, dynamic scheduling/allocating, access control, remote updating, rapid, approximate recovery, and modeling.

Related SDIO Funded Projects

Name: Management of Distributed Systems
Agency: SDC
Objective: Develop and evaluate advanced resource control algorithms for candidate node architectures.

Name: Distributed Information Processing
Agency: RADC
Objective: Develop distributed system resource management mechanisms to control location, execution, movement, and access to SDI processing functions.

Name: Distributed System Security
Agency: RADC
Objective: Develop system security mechanisms to provide multiple levels of security commensurate with single host system.

Name: Security Architecture Studies
Agency: NSA
Objective: Address security issues in cohesive manner and provide focus for technology program.

Name: Trusted Real-Time Operating System
Agency: DARPA
Objective: Deliver trusted real-time operating system in Ada to ARC.

Recommendation 5: Multicomputer systems with state-of-the-art operating systems and programming language environments for straightforward parallel programming should be placed in at a large number of research sites. Research sites that already have multicomputer systems should be provided access to the same state-of-the-art operating systems and programming language environments. This should be coordinated with the development of additional parallel processing testbeds recommended in Appendix B, Section 8.

Related SDIO Funded Projects

Name: Center for Experimental Research in Parallel Algorithms Software and Systems
Agency: ONR

Recommendation 6: Software developed by the prototype operating systems projects should be widely distributed.

Related SDIO Funded Projects

Name: Trusted Real-Time Operating System
Agency: DARPA
Objective: Deliver trusted real-time operating system in Ada to ARC.

B1.5 Data Management Systems

Recommendation 1: SDI should prototype platform data management architecture models. This will aid in determining database structure, data elements that will need to be distributed (and the level of distribution), and strategies for reconstitution.

Related SDIO Funded Projects

Name: Distributed Databases
Agency: RADC
Objective: Develop database technology to support real time data handling in hierarchically clustered system which incorporates multiple replicated and/or partitioned databases.

Name: Track File Data Base Management System
Agency: NRL
Objective: Develop mechanisms for ensuring synchronization of physically distributed databases; develop parallel track file DBMS suitable for SDI application; experimentally establish feasibility

Recommendation 2: SDI should fund research into areas such as data model selection, homogeneity, and error recovery for distributed systems. The SDIO should prototype distributed DBMSs that experiment with different aspects of these issues. In addition, the modular architecture for DBMSs discussed in Section 5.2.4.2 is recommended for implementation. Research and development prototypes of systems developed around this concept should be funded.

Related SDIO Funded Projects

Name: Distributed Data Management System
Agency: SDC
Objective: Develop concepts for distributed information management system to ensure necessary access to rapidly varying data.

Recommendation 3: Issues, such as the definition of various interfaces, concurrency control and deadlock resolution algorithm selection, should be prototyped to determine the merits of different algorithms.

Recommendation 4: It is highly recommended that programs such as WIS, NASA space station, and STARS continue to be analyzed to determine the potential synergy between other government agencies involved with data management systems and the needs of the SDIO

Recommendation 5: With regards to software engineering environment data management requirements, SDIO should standardize on interfaces between DBMS's and users, tools and data. Other issues need to be resolved (as discussed in Section 5.3.3) within the short term, such as the role of active components, choice of data model, amount of project data to store, etc. These issues should be resolved early to that their results may be applied to other projects.

Recommendation 6: It is recommended that prototype object management systems be developed, in Ada, similar to the ODIN [Clemm 84] and KEYSTONE [Osterweil 84] systems.

Recommendation 7: SDIO should fund studies to determine whether or not multi-level secure DBMS technology is required in the program. In any case, a production quality multi-level secure DBMS is probably several years ahead in the future.

B1.6 Man-Machine Interface

Recommendation 1: SDIO should continue its support of the Command and Control Human Interface Program at the Naval Research Laboratories.

Related SDIO Funded Projects

Name: Command and Control Human Interface
Agency: NRL
Objective: Determine appropriate roles for human SDI commander under severe time constraints of SDIO scenario and investigate alternative mechanizations for human/machine interface; establish feasibility of man-in-the-loop tasking and mechanizations.

Recommendation 2: Because of the potential utility of a user interface that employs artificial intelligence techniques, SDIO should monitor its state of the art and, in particular, the work sponsored by DARPA.

Related SDIO Funded Projects

Name: Artificial Intelligence
Agency: DARPA

Recommendation 3: SDIO should monitor government development efforts for workstation managers, window managers, graphics packages, and user interface management systems.

Recommendation 4: SDIO should monitor current standardization efforts for graphics packages.

Recommendation 5: SDIO should survey commercially developed and research-oriented UIMSs for adaptation and integration into the strategic defense system.

Recommendation 6: SDIO should continue to fund the U. S. Army Strategic Defense Command's program for a Command and Control Decision Aids Test Environment.

Related SDIO Funded Projects

Name: Communications/Graphics
Agency: ONR
Objective: Develop communications concepts (at information level) required for SDI systems (protocols, tradeoffs, etc.); develop graphics techniques to aid in situation comprehension and evaluation of status and capabilities.

Name: Near Term BM/C3 Architectural Experiments
Agency: SDC
Objective: Define and execute near term (FY88) demonstration and validation of BM/C3 capability in realistic, geographically

distributed manner; provide visible and convincing progress of BM/C3 capability.

B1.7 Parallel Processing

Recommendation 1: In the short term, Ada should be the principal language for the development of software for parallel architectures. In the long term, basic research on languages better suited to the expression of parallelism, both explicitly and implicitly, should be pursued.

Recommendation 2: With the current emphasis on the development of Ada compilers, SDI should piggy-back on such efforts and develop Ada compilers targetted to parallel processing engines.

Recommendation 3: SDI should pursue the development of retargetable parallel code generators, with a particular emphasis on the development of architectural description languages for parallel architectures.

Recommendation 4: In the short term, SDI should pursue the development of operating systems to support the development of code for parallel processing engines.

Recommendation 5: SDI should support the development of facilities to allow investigation into parallel algorithms. These facilities should be coordinated with those of the National Test Bed and those of the NSF and DARPA's Super Computing Centers, as well as the Algorithm Test Center recommended in Section 2, Appendix B.

Related SDIO Funded Projects

Name: Target Assignment Algorithm
Agency: SDC
Objective: Develop candidate set of algorithms for weapon-target pairings late midcourse and terminal phases; implement and evaluate selected algorithms in realistic environment.

Name: BM Benchmark Algorithms
Agency: RADC
Objective: Compare data processing architecture implementations.

Name: Hybrid Computing
Agency: NRL
Objective: Develop techniques for combining numerical and symbolic computing on parallel architecture machines.

Name: Parallel Computing
Agency: NRL

Name: Center for Experimental Research in Parallel Algorithms Software and Systems

Agency: ONR

B1.8 Computer Systems Engineering Technology

Recommendation 1: In the near-term, the SDIO should examine existing and ongoing work in system-level requirements capture. Methods specifically for either software requirements or hardware requirements capture should be examined to determine whether this work may be extended to accommodate the other aspect of the system. Based upon the quality and applicability of existing or ongoing work to SDIO needs, the SDIO should either provide continued funding for these projects, or initiate a project to construct prototypes for system-level requirements capture.

Recommendation 2: In the near-term, the SDIO should examine existing and ongoing work in system-level specification languages/notations. Methods specifically for either software specification or hardware specification languages/notations should be examined to determine whether this work may be extended to accommodate the other aspect of the system. Based upon the quality and applicability of existing or ongoing work to SDIO needs, the SDIO should either provide continued funding for these projects, or initiate a project to construct prototypes for system-level specification languages/notations.

Related SDIO Funded Projects

Name: Processing
Agency: RADC
Objective: FSED specification of HP, FT survivable computer architecture capable of supporting autonomous, trusted BM processing in space.

Recommendation 3: In the far term, following the development of a reasonable set of requirements capture and specification language prototypes, the SDIO should support research aimed at beginning to develop automatic verification (design meets requirements and specification) capabilities.

Related SDIO Funded Projects

Name: Formal Verification Technology
Agency: RADC
Objective: Design, develop and demonstrate tools, techniques and methodologies for formal verification systems.

Recommendation 4: In conjunction with the work outlined in Recommendations (1) and (2), the SDIO should examine the tools and engineering environments provided for using the requirements capture and design specification languages/notations. Based upon the quality and

applicability of these tools and environments the SDIO needs, the SDIO should either provide funding for continued work or support the construction of engineering environment and tool prototypes.

Related SDIO Funded Projects

Name: System Engineering and Integration
Agency: NRL
Objective: Perform system integration, engineering and experiment design efforts required to establish feasibility of autonomous battle management system.

Name: Formal Verification Technology
Agency: RADC
Objective: Design, develop and demonstrate tools, techniques and methodologies for formal verification systems.

Name: Trusted System Design
Agency: RADC
Objective: Exploit SOA technologies in development of tools and techniques to be used in design of secure systems.

Name: Distributed Computing Design System (DCDS)
Agency: SDC
Objective: Complete methodology with integral software development environment that is beyond current state-of-the-art.

Name: Ada Engineering Environment
Agency: DARPA
Objective: Deliver Ada engineering environment based on using object management system running on MACH.

Recommendation 5: Based on the above recommendations, the SDIO should make policy decisions on the level of requirements specification language standardization, design specification language standardization, and engineering environment standardization.

B1.9 Software Engineering Environments

Recommendation 1: Place early emphasis on standardizing software work product interfaces for interchange among SEEs. Use Ada and Common LISP (for AI) as the standard programming languages.

Recommendation 2: Apply only mild constraints to SEE(s) needed immediately (e.g., initial NTF) that must be composed from what is currently available.

Recommendation 3: Position the SDI SEE effort to benefit over the next few years from several SEE efforts. These include the NASA Space Station, WIS, SEI, and DARPA efforts.

Recommendation 4: SDI should continue to fund research in developing effective, efficient, and extensible methods of integrating software tools into an environment. Work such as the Arcadia environment provides a good framework to begin developing integrated environments. This may be the most important issue confronting the SDI SEE(s), and should be given a high priority.

Related SDIO Funded Projects

Name: Ada Engineering Environment
Agency: DARPA
Objective: Deliver Ada engineering environment based on using object management system running on MACH.

Recommendation 5: SDI should fund tools to incorporate into the standard environment framework of recommendation 4. These include tools for methodology, requirements development, rapid prototyping, support, visual programming formal verification, code generation/compiling, testing, metrics and data base support. The state of practice in most of these areas is not suitable for an SDI SEE.

Related SDIO Funded Projects

Name: Formal Verification Technology
Agency: RADC
Objective: Design, develop and demonstrate tools, techniques and methodologies for formal verification systems.

Name: Distributed Computing Design System (DCDS)
Agency: SDC
Objective: Complete methodology with integral software development environment that is beyond current state-of-the-art.

Name: DCDS Accelerator
Agency: SDC

Name: Software Quality, Reliability Tools
Agency: SDC

B1.10 Simulation

Recommendation 1: SDIO should examine the operational roles of the NTB, ASC, and EV programs for possible redirection to plug gaps and eliminate redundancy.

Related SDIO Funded Projects

Name: Near Term BM/C3 Architectural Experiments
Agency: SDC
Objective: Define and execute near term (FY88) demonstration and validation of BM/C3 capability in realistic, geographically

distributed manner; provide visible and convincing progress of BM/C3 capability.

Name: Experimental Version-Level I
Agency: ESD
Objective: Build and demonstrate BM/C3 prototype for proof-of-feasibility; provide vehicle for BM/C3 experiments in system context.

Recommendation 2: SDIO should choose Ada as the standard simulation programming language. All new simulation software should be coded in Ada; exceptions require government approval.

Recommendation 3: SDIO should standardize an Ada Simulation Support Environment for SDI simulation and evaluation.

Recommendation 4: SDIO should fund basic research into techniques for validating models of large-scale systems.

Related SDIO Funded Projects

Name: Formal Verification Technology
Agency: RADC
Objective: Design, develop and demonstrate tools, techniques and methodologies for formal verification systems.

Recommendation 5: SDIO should standardize a testing methodology for SDI software and hardware components.

B1.11 Software Dependability

Recommendation 1: Adopt a maximal, multi-tiered, fault-tolerant approach covering requirements, development and support (including the means used), and run-time to defend against software failures for the experimental prototypes including diversity and systematic measurement and improvement.

Recommendation 2: Systems requirements and engineering activities should include a strong awareness of software concerns and result in -- initially multiple -- software requirements specifications that are formal, comparable, machine analyzible and manipulatable, explicitly address the range of possible systems implementations and evolutions, operationally define software-failure-related quality requirements and include prerequisite knowledge such as usage distributions.

Recommendation 3: Invest in multiple efforts toward the development or improvement of specification technologies potentially well suited for use in accurately representing and communicating SDI BM/C3 system/software requirements and try them in the experimental version efforts.

Recommendation 4: Ensure the proper organization of the processes of technology development and use, and personnel selection

and development including characterization of their error patterns and their certification for use.

Recommendation 5: Identify the error detection power of verification techniques; support the enhancement for use on SDI BM/C3 of technology for software testing, simulating faults and recording error propagation, and formal verification with Ada; and monitor automatic programming research.

Recommendation 6: Invest in quality measurement/modeling research. Use measurement and modeling on all BM/C3 software efforts, and learn and change.

Recommendation 7: Support R&D programs that cover fault tolerance at both the software and systems levels; address Ada definitional shortcomings in failure semantics, autonomy, decentralization, atomic synchronization of distributed processes, application-specific fail-forward and adaptive control mechanisms, and software safety; and result in stable level(s) of machine abstraction and a fault-tolerance harness that applications software can use straightforwardly.

Related SDIO Funded Projects

Name: Tracker/Correlator
Agency: NRL
Objective: Develop state of the art hybrid tracker/correlator system that combines optimum statistical estimation techniques with heuristic reasoning and knowledge-based system techniques; establish feasibility of SDI application by experimentally determining functional performance, sensitivity, and computational resource requirements.

Name: Command and Control Human Interface
Agency: NRL
Objective: Determine appropriate roles for human SDI commander under severe time constraints of SDIO scenario and investigate alternative mechanizations for human/machine interface; establish feasibility of man-in-the-loop tasking and mechanizations.

Name: System Engineering and Integration
Agency: NRL
Objective: Perform system integration, engineering and experiment design efforts required to establish feasibility of autonomous battle management system.

Name: Security Architecture Studies
Agency: NSA
Objective: Address security issues in cohesive manner and provide focus for technology program.

Name: VHSIC Multiprocessor Technology
Agency: NASA

Objective: Demonstrate operating system methodology for coupling self-testability with range of user selectable fault tolerance in multiprocessor applications.

Name: Fault-Tolerant for Scalable Architectures
Agency: DARPA

Name: Support - Mitre
Agency: ESD

B1.12 People and Organizations

Recommendation 1: To the maximum extent appropriate People and Organization R&D should be conducted through the contractor(s) performing the BM/C3 software effort.

Recommendation 2: The SDI BM/C3 contractor(s) should be responsible for research that identifies characteristics of excellent (and super) programmers and other software-related personnel, and development that results in validated means of selection including empirical studies of certification programs such as the ICCP's to find out how effective such certification is in predicting job success.

Recommendation 3: The SDIO should fund educational upgrading of SDI software personnel as well as help universities involved. It should take measures to assure that contractors educate and upgrade SDI workers.

Recommendation 4: The SDI BM/C3 contractor(s) should be responsible for R&D that supports organizational design decisions, organizational and team learning, and computer mediated work and MMI for software workers as well as for monitoring software, software cost/size/quality data collection and models, and other research to improve software productivity and quality.

Recommendation 5: The SDIO should fund studies of the acquisition process to find mechanisms for assuring that organizations working for it meet the needed people and organization-related requirements.

Recommendation 6: To increase awareness within the SDI software development community of the results of research in the People and Organizations area, SDIO should fund an organization to monitor, evaluate, and report on significant developments in this area.

Recommendation 7: SDIO and other government personnel involved with SDI software as well as assisting personnel in FFRDC's and SETAs should be of outstanding quality or be upgraded or replaced. Continual upgrading is required for just staying current and should be funded by SDIO.

B1.13 Technology Transition

Recommendation 1: Formulate technical strategies and plans for transitioning technology for application technologies, foundation technologies, and software engineering technologies. Cover compatibility, integratibility, standards and conventions, ease of transfer, quality assurance, and interoperability. Plans should indicate paths for transitions each technology within each task areas and fit closely with contractor technology insertion plan.

Recommendation 2: Establish relationships with all the relevant organizations; encourage (and in some cases require) speeding up of internal activities, establishing linking mechanisms, and accelerating the transfer of technology toward use in the SDI effort.

Recommendation 3: Include provisions in the qualifications, statement of work, requirements for proposal content, evaluation criteria, funding and other provisions of acquisitions to encourage and assure the needed technology insertion and successful use. The SDIO should establish what such provisions should be to prepare for future contracting.

Recommendation 4: Scan for promising software technology and products. The results of this scanning plus information on SDI supported efforts should be put in suitable form and communicated throughout the SDI community.

Recommendation 5: In the near term, ensure that the SDI can piggyback on the NASA Space Station Software Support Environment (SSE) effort and any major DoD SEE efforts. In the longer term, position SDI to benefit from multiple sources of software tools.

Recommendation 6: Ensure that all the parties involved in the BM/C3 software R&D are familiar with the SDIO software needs and with the properties of a software technology that make it more likely to be used.

Recommendation 7: Ensure that the technology evaluation processes used at the different stages, while knowledgeable and appropriately independent, properly reflect the requirements and needs of the full scale engineering development decision, the SDI system, and the users of the technology.

Related SDIO Funded Projects

Name: Transfer

SECTION B2

Battle Management/C³

Prepared by Michael I. Bloom and John Chludzinski

Topics covered in Section B2:

2.1 Introduction
2.1.1 Purpose
2.1.2 Scope
2.1.3 Background Considerations
2.1.3.1 Effects of System Architecuture on BM/C3 Software Requirements
2.1.3.2 System Decomposition

2.2 BM/C3 Functional Requirements
2.2.1 Sensor Data Processing
2.2.1.1 Sensor Network Interface to BM/C3
2.2.1.2 Sensor Network Design Model
2.2.1.3 Multi-Sensor Fusion Techniques
2.2.1.3.1 Partitioning of Fusion Functions
2.2.1.3.1.1 Association and Correlation
2.2.1.3.1.2 Classification
2.2.1.3.2 Fusion Approaches
2.2.1.3.2.1 Centralized
2.2.1.3.2.2 Autonomous
2.2.1.3.2.3 Hybrid
2.2.1.4 Tracking
2.2.1.4.1 Track Initiation
2.2.1.4.2 Track Maintenance
2.2.1.4.3 Track Termination
2.2.1.4.4 Track File Design and Storage
2.2.2 Weapon System Data Processing
2.2.2.1 Threat Evaluation
2.2.2.2 Weapon Assignment
2.2.2.2.1 Determine Defensive Strategy
2.2.2.2.2 Apply Release Criteria
2.2.2.2.3 Prioritize Threats
2.2.2.2.4 Maintain Weapons
2.2.2.2.4.1 Weapon Consumables Inventory
2.2.2.2.4.2 Status Monitoring
2.2.2.2.4.3 Backup
2.2.2.2.5 Allocate the Weapons
2.2.2.2.6 Terminal Guidance and Illuminator Assignment
2.2.2.3 Kill Assessment and Revised Damage Estimate
2.2.2.4 Defensive Phase Weapon Handover
2.2.3 Self-Defense
2.2.4 System Performance Monitoring, Reporting

2.2.5 Platform Control
2.2.6 Decision Support Systems (DSS)
2.2.6.1 Taxonomies of Decision Problems
2.2.6.2 Decision Support System Information Criterion
2.2.6.3 Types of Decision Support Systems
2.2.6.4 Lanugage Interface
2.2.6.5 Interactive Graphics
2.2.7 Operational Coordination
2.2.7.1 Defensive and Offensive Coordination

2.3 Current Status
2.3.1 Situation Assessment
2.3.2 Algorithms for Multi-Sensor Data Fusion
2.3.3 Weapon Assignment
2.3.3.1 Determine Defensive Strategy
2.3.3.2 Allocate the Weapons
2.3.4 A Framework for Developing Decision Support Systems
2.3.4.1 Role of the National Test Bed
2.3.4.2 Process-Independent Approach Toward Decision Support
2.3.4.3 Database Management System
2.3.4.4 Model Base Management System
2.3.4.5 Dialog Generation and Management System

2.4 Recommendations

2.5 References

2.6 Bibliography

List of Figures

Figure	Title	Page
1	Hybrid Fusion.....	29

List of Tables

Table	Title	Page
I	Comparison of Representative Association and Correlation Methods	27
II	Assessment of Application Software Requirements vs. Current Software Technology	41-42

2.0 BATTLE MANAGEMENT/C^3

2.1 Introduction

This portion of Appendix B gives a task area assessment of Battle Management/Command, Control and Communications (BM/C3) application-oriented software technology.

2.1.1 Purpose

This study addresses the question: Which BM/C3 application software functions could be:

a. Implemented within the scope of present-day software engineering and algorithm design techniques,

b. Implemented with difficulty because they stress the present state-of-practice in software engineering and algorithm design, or

c. Not implemented because they require software engineering and algorithm design techniques that push the state-of-the-art?

For those application-oriented software functions where software engineering and/or algorithm design techniques push the state of the art, what will help develop needed technologies and/or provide the information needed to inform decisions about full scale development?

2.1.2 Scope

This portion of the study is limited to those software issues that are germane solely to BM/C3. It will not repeat conclusions voiced elsewhere in this report. Thus, this portion of the study will not address general foundation topics such as distributed operating systems, communications, and so on, and neither will it address issues shared with other SDI software components, such as productivity, project management, and verification. It will not address non-software issues, such as hardware survivability, communications security, and so on, unless they have a direct bearing on software.

For the purposes of this study, BM/C3 consists of those functions that maintain awareness of the status of the system, perceive what threat the system has to counter, select a plan to counter the perceived threat, inform the system which plan to execute, monitor the progress of the plan through the actions of sensors and weapons, and select and notify appropriate portions of the system of modifications to the plan. The conclusions and recommendations in this task area assessment are preliminary.

2.1.3 Background Considerations

2.1.3.1 Effects of System Architecture on BM/C3 Software Requirements

The determination of which BM/C3 application software is within the state-of-art is dependent on system architecture. For example, if an architecture has data from many sensors going to relatively few nodes for fusion to form a master track file, and the number of object tracks being fused at a node is large, i.e., several tens of thousands to hundreds of thousands, studies at the SDIO and the other services suggest that traditional algorithms,

run on sequential processing computers, will be unable to perform within the stringent time constraints and accuracy requirements.

By comparison, architectures which present a constrained quantity of sensor fusion data to a node reduce or eliminate this particular stress in the BM/C3 application software. The reason for the reduced stress lies in restricting the span of BM/C3 responsibility for a given node.

The example given is not intended to suggest which architectures are superior to others. It is intended to suggest that the risks inherent in BM/C3 application software development can be reduced if sufficient systems analysis has been performed to isolate those factors which make the software difficult, and if system architectures are chosen that will ease or eliminate problems. The Eastport Group and others have stated that the BM/C3 software would be the most difficult part of a space defense system. Experience in the DoD has shown that one of the largest costs over the life time of major weapons systems has been the maintenance of the software. These observations provide motivation for developing architectures that are sensitive to difficulties in developing BM/C3 software.

2.1.3.2 System Decomposition

Section 2.2 of this report uses a decomposition of sensor and weapon functions to organize the report and highlight the BM/C3 functions. This decomposition is not exhaustive. It carries the decomposition of a function to only one or two levels. In an exhaustive decomposition of the sensor and weapon components, the number of levels reaches six or more.

The decomposition is not unique. Other schemes may have different names for the same function, functions may be placed at different levels, and functions decomposed to different levels of granularity. Recommendation Two (Section 4.2) suggests the need for a standardized decomposition.

2.2 BM/C3 Functional Requirements

For purposes of analysis, BM/C3 functions have been segmented into pertinent subtopics or subfunctions. The following subtopics or subfunctions are considered in the following subsections.

a. Sensor Data Processing (Section 2.2.1)

b. Weapon System Data Processing (Section 2.2.2)

c. Self-Defense (Section 2.2.3)

d. System Performance Monitoring, Reporting (2.2.4)

e. Platform Control (2.2.5)

f. Decision Support Systems (Section 2.2.6)

g. Operational Coordination (Section 2.2.7)

2.2.1 Sensor Data Processing

The BM/C3 function requires application software to perform or provide the following functions.

a. Sensor System Status Verification: Reports are accepted from the sensor system and verified, i.e., incoming data from the sensors is of the correct type, accuracy, quantity, etc. Included in the verification are checks to ensure that illegitimate data streams, i.e., Electronic Countermeasures (ECM), are detected and disposed of in a manner which does not interfere with required operations. When sensor(s) are not producing proper data streams, they will be engaged in necessary dialogue to ascertain the nature of the problem and advise a designated controller (human or machine) of the problem. The software challenge will be in performing the necessary functions with negligible time delay. This may be an application area where parallel processing with forward feeding of corrective action is necessary. This function should require some advance in the current state-of-practice but is well within the state of the art.

b. Object Discrimination: Determine which of the object tracks represent re-entry vehicles with warheads, and which tracks represent other non-lethal objects. In terms of the applications software the degree of stress is highly architecture dependent. If the BM/C3 application processing is concentrated at one or a few nodes, the throughput requirement could be stressing. If an architecture can be used where the interface is distributed then the software requirement appears to be within the current state-of-art. However, the physical mechanisms used to discriminate objects are beyond the state-of-art for reasons not related to software or BM/C3.

c. Coverage Control: Construct the pointing, staring, scanning, and tracking instructions. These instructions must account for the orbital and evasive movement of the space based sensor platforms and their targets. The challenge to the BM/C3 software technology will include the need for high speed logic processing to develop the necessary sequence of commands. There will also be the need for high speed determination that commands are not given which request exceeding physical limitations such as gimbal slew rates, and traversal limits. Development of software to perform this function is within the state-of-practice.

2.2.1.1 Sensor Network Interface to BM/C3

The distribution of functions between the sensor constellation(s) and BM/C3 is a major decision effecting the selection of a deployable system. The distribution affects the risk inherent in developing BM/C3 application software to meet reliability and performance requirements.

Specific factors that are effected by the distribution are:

a. The number of variables passed to BM/C3 for processing,

b. The types of variables(binary or logical),

c. The complexity of data transformations necessary,

d. The required accuracy of the transformations,

e. The speed with which the transformations have to be made, and

f. The size of database for search, sort, insertions, and required concurrancy across distributed databases.

Further, the distribution will have a direct influence upon the choice or necessity of parallel versus sequential processing.

2.2.1.2 Sensor Network Design Model

The Eastport Group suggested that "the most plausible organizations for a strategic defense battle management system are hierarchic" (meaning the structure can be portrayed graphically in tree diagrams, such as organizational charts). This tree structure of a battle management system is rooted in the command authority and branches to the subsystems.

Another candidate for the sensor systems is the network design. With the network design, a message can be routed through many different paths, thereby assuring that the destruction of a single node does not isolate entire segments of the system. However, it lacks the authority of chain of command.

A third candidate system for the sensor network is a hybrid structure. With the hybrid design, the leaves of a hierarchical (tree) design will be networked together. If a link is destroyed, alternative paths can be used to reroute messages.

2.2.1.3 Multi-Sensor Fusion Techniques

Software for multi-sensor object data fusion performs correlation, association and object track identification with data from homogeneous or heterogeneous classes of sensors. The software must also manage the data flows into the process, ascertain if the process is approaching overload, and manage the flow of track files for weapons allocation. The difficulties inherent in producing this software are highly architecturally dependent and cannot be determined at this time.

2.2.1.3.1 Partitioning of Fusion Functions

In examining the multi-sensor fusion problem, especially in a dense target environment, the computational problems that frequently lead to ambiguous results are association, correlation, and classification.

2.2.1.3.1.1 Association and Correlation

One important problem facing a multiple sensor tracking system in a multiple target environment is the unique identification of the same target as observed by multiple sensors. There are two approaches to this problem. The first approach attempts to correlate the existing track files (state estimate of position and velocity based on past measurements) with current measurements. The second approach is to directly correlate the set of measurements from the i-th sensor with that of the j-th sensor.

The fusion process serves as a mechanism for forming and updating composite (fused) target states with the respective filtered or unfiltered target states received from each sensor. If we have N objects in the fusion track file and M input reports, this process involves MxN comparisons. The resultant pairs of potential input report/track combinations serve as inputs to the correlation process. The correlation process decides which pairs represent a valid track update.

Decision logic has been based predominantly on a minimum distance criterion or hypothesis testing methods such as:

a. Euclidean distance,

b. Statistical distance,

c. Maximum likelihood hypothesis test, and

d. Probabilistic data association filter (PDAF).

Table I summarizes a comparison of a few representative methods showing the software advantages and disadvantages of each technique.

None of these techniques is clearly superior or applicable to all target environment conditions.

2.2.1.3.1.2 Classification

The classification decision executed by the multi-sensor fusion process allows for two options. The first option fuses and upgrades the declarations generated by those sensors that are capable of classification and generates a composite decision that may be the same as those of individual sensors but with a higher confidence factor. If declarations are not unique, the fusion process combines them into a new resultant decision.

The second option allows for either continuous or selective retrieval of observed classification parameters in those cases where a single sensor's parameters are not adequate to yield a reliable declaration. Thus, this option is identical to a conventional single sensor classification procedure using static or dynamic parameters.

The techniques used for fusion of declarations from various sensors generally fall into four categories:

a. Polling,

b. Maximum likelihood,

c. Bayesian, and

d. Dempster-Shafer.

The major differences in these methods is the amount of *a priori* information they use.

The polling method is the simplest. It relies on majority voting using three or more sensor declarations. It is the least accurate since it does not take into account quantitatively the sensor recognition conditional probabilities or target distribution statistics.

The maximum likelihood method looks for the maximum response. It takes into account conditional sensor recognition capabilities and the discriminant value of features.

Table I. Comparison of Representative Association and Correlation Methods

KEY CANDIDATE TECHNIQUES	ADVANTAGES	DISADVANTAGES
Template Matching	Simple comparison	Large number Necessary, Ad-hoc
Nearest Neighbor Euclidean Distance	Simplest computation; memory does not grow	Does not account for measurement, system errors.
Nearest Neighbor	Decision rule takes into account updated system errors; memory does not grow.	More computation than Euclidean. Must keep track of system noise covariance.
Track Split (Maximum Likelihood)	Decision made after several trial tracks formed.	Growing memory and throughput.
PDAF (all neighbor) No scan Back	Bayesian statistics. Memory size,computations slightly exceed Kalman filter.	Suboptimal, no history; needs verification.
PDAF (all neighbor) with history	Optimal Bayesian method, substantially better then Nearest Neighbor.	Substantially greater memory and throughput.

The Bayesian procedure goes further, since it also includes the expected probabilities of target types in any given theater of operations. It evaluates the probability of declaring target M given a sensor response R. The Bayesian method should be more accurate, given that all the a priori data is accurately known.

The Bayesian method distributes a unit of "personal judgement" across a set of mutually exclusive and exhaustive propositions. The Dempster-Shafer approach does not have such strong assumptions. Instead, a belief is represented by an "evidential interval". The lower bound represents the degree to which the evidence supports the proposition; the upper bound represents the degree to which the evidence fails to refute the proposition. It has the ability to express the level of ignorance.

2.2.1.3.2 Fusion Approaches

The three principle approaches to data fusion are:

a. Centralized, or global,

b. Autonomous, or local, and

c. Hybrid.

2.2.1.3.2.1 Centralized

The centralized method uses the data directly from widely distributed sensors before the state estimation, i.e., generation of position and velocity vectors. It attempts to fuse all multi-sensor data to generate composite traces. This method is the most complex in terms of the fusion function.

2.2.1.3.2.2 Autonomous

The autonomous method fuses all sensor data on board a single platform. It relies on independent detection and state estimation performed within each sensor's signal processor and tracker. The resultant tracks and track declarations from each sensor are then fused as before to form composite tracks. In both the centralized and autonomous methods, the fusion results can be fed back to the sensor subsystems to provide queuing, threshold control, etc., for the purpose of optimizing the overall response.

2.2.1.3.2.3 Hybrid

The hybrid method, shown in Figure 1, combines the benefits of both the centralized and autonomous methods. It allows data reduction prior to fusion, as in the autonomous system, but it also permits extraction of raw observations to allow for selective enhancement of target declarations and resolution of ambiguities.

2.2.1.4 Tracking

This problem is sometimes referred to as scan-to-scan correlation or track data association. This problem can be divided into three phases. The first phase is track initiation, the second is track maintenance, and the third is track termination.

2.2.1.4.1 Track Initiation

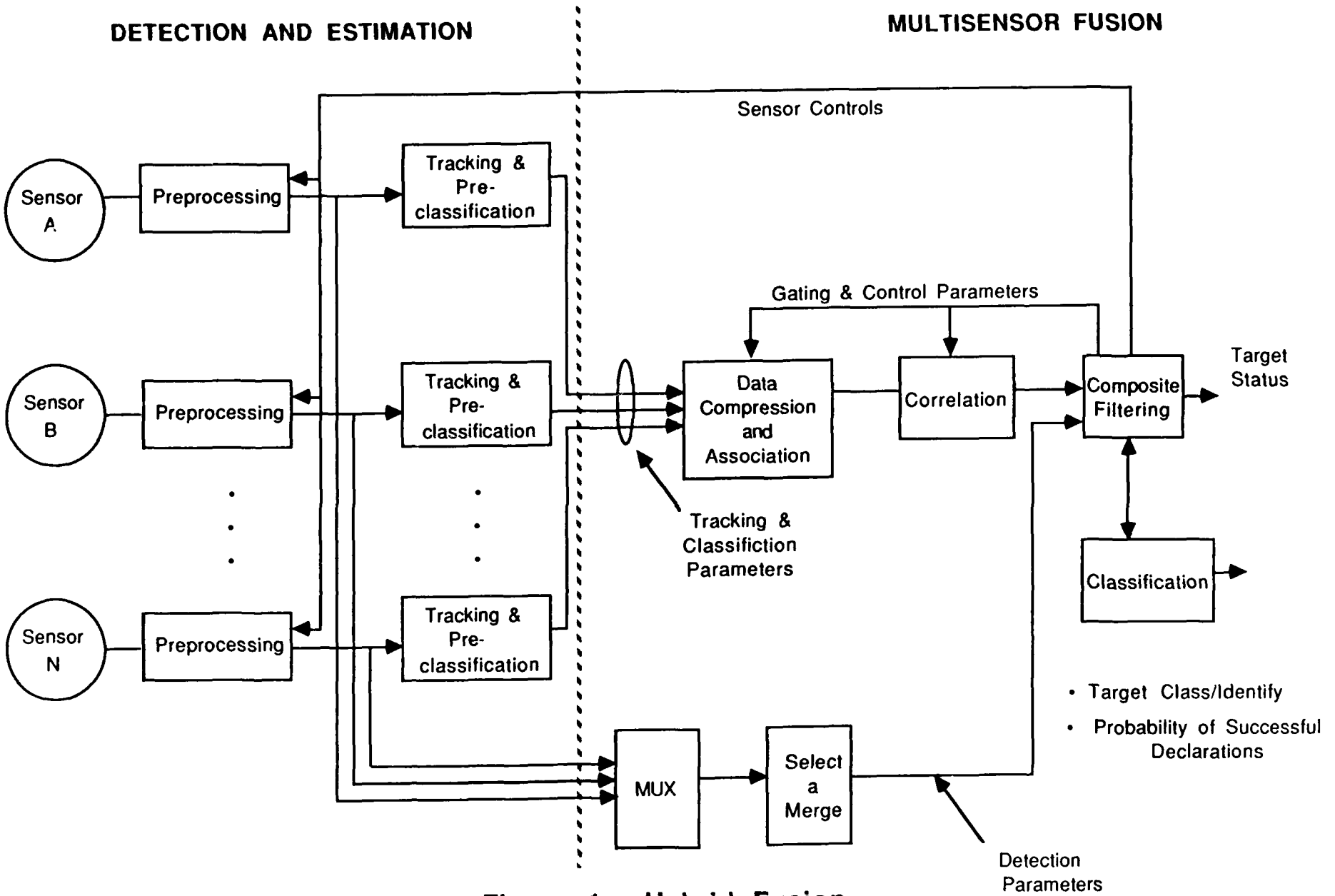


Figure 1. Hybrid Fusion

Consider the case of a scanning sensor. The first and second scan produce N1 and N2 detections, respectively. The problem is to associate the two sets of detections to form min($N1, N2$) number of track files. The computational requirement of the problem is nontrivial. It is usually done by the N-dimensional assignment scheme or the maximum likelihood method. The software for this function must be fast enough to associate the scan to scan detections and to accept the detections from the next scan(s) with the buffering capacity allocated for the process. The efficiency requirements for this software are dependent on the system architecture selected. For those architectures that present this process with min($N1, N2$) on the order of a few tens of thousands, advances in the state-of-practice will be necessary. Where this parameter is on the order of hundreds of thousands, advances in the state-of-art will be necessary.

2.2.1.4.2 Track Maintenance

Once the track files have been established, the computational requirement is greatly reduced. The reason is that each track file only requires searching the "admissible" regions determined by the covariance of the residual filter. The state-of-practice of software, used for track maintenance in phased array radars for multi-object tracking, should be sufficient for this process.

2.2.1.4.3 Track Termination

The result of kill assessment is associated with the track file. An indicator of the "kill" eliminates the track file as the candidate in the next scan processing. The challenge for this processing will be performing the necessary routing of the kill assessment to all databases that contain the track to be eliminated. Some form of directory system will be necessary for routing the elimination notice, or for the appropriate databases to recognize a track needs to be eliminated, if broadcasting messages is used. The software for this processing is within the state-of-practice.

2.2.1.4.4 Track File Design and Storage

Track file storage and organization is dependent upon the time interval between updates of the tracker, and the dimensionality of the multi-sensor measurements. Multi-sensor track files must be organized for dual access by the correlator/classifier functions and must permit concurrent updating by the asynchronous activities of each. Track storage capability is normally determined by the target density and search volume. For multi-sensor track data storage, the storage per correlated target must accommodate the multiple measurements and feedback identification decisions made along the track. Although the computer memory storage for tracks may be large, the sizing of the requirement is within the state-of-practice and the software for control of dual access is within the state-of-art.

2.2.2 Weapon System Data Processing

The BM/C3 requires application software to perform the following functions:

a. Threat evaluation (Section 2.2.2.1),

b. Weapon assignment (Section 2.2.2.2),

c. Kill assessment (Section 2.2.2.3), and

d. Defensive phase weapon handover (Section 2.2.2.4).

2.2.2.1 Threat Evaluation

Threat evaluation is the act of determining the nature of the attack. Although some functional decompositions may place the anti-satellite attack in this area, it will be treated as a separate function due to its potential for requiring advances in the state-of-practice and state-of-art. In this decomposition, threat evaluation is subdivided into the following functions.

a. Launch Categorization: Threatening launches are distinguished from non-threatening ones. For the space defense system, this function will have to be performed in real time to provide the battle manager with a full range of options. Provisions for human review and confirmation/override of the order to attack will probably exist. Given that a sensor network is in place, which provides the necessary coverage in the spatial, temporal, spectral space, the challenge to software technology is to process the data in the required short period of time and present information for human review. Both of these software functions are within the state-of-art, with advances in the state-of-practice necessary.

b. Threat Categorization, Notification, Verification: Given parameters such as the location of the launch(s), track(s), and current intelligence estimates, this function estimates who is being attacked and which offensive strategy an adversary is implementing. For example:

   (1) Is the adversary using a strategy where the majority of missiles are salvoed in the shortest time?

   (2) Is the strategy to fire the missiles uniformly over a time period?

   (3) Are the missiles being fired in two major salvos with the first being used to draw down the defensive system weapon inventory? or

   (4) Is the strategy to fire a concentrated anti-satellite attack and assess the result before selecting the timing for firing warhead carrying ballistic missiles?

   These examples are not meant to be an exhaustive description of the possible strategies. Other strategies are likely to be known or developed.

   In general, the software requirements may be different depending upon the architecture. For architectures where this function is performed at one or several nodes, the databases should be identical to avoid confusion and ambiguity in the results.

   If the architecture is based upon several autonomous groupings of sensors, weapons, and BM/C3 capabilities, and each group's responsibility is limited by geographic or other means, then their databases need not be identical. In this case, there will be a requirement for the databases to change as the group(s) ground track moves (assuming that no group is in geostationary orbit).

c. Track Propagation: Given the track history of an object, orbital mechanics can be used, after boost phase, to extrapolate the site and time to impact, plus the related confidence limits. This function is within the state-of-practice.

d. Target Damage Estimation: The estimated impact points from different tracks are correlated to determine the total threat to a single site. This can be used to

estimate the expected damage at a given site. This function can also be used to evaluate the probable effects of defensive strategies. Given the expected effectiveness of each defensive asset, this function determines the most likely number of offensive weapons to survive. From this determination, the damage estimation function can derive an indication of the expected damage from a attack-defense cycle. This information is also useful if a defensive triage strategy is necessary or to implement a preferential defense. Algorithms to perform this function are within the state-of-practice.

2.2.2.2 Weapon Assignment

Once the opponent's offensive strategy is known, the SDI system should begin the task of deploying defensive assets to counter the threat. This task can be broken down into the steps.

a. Step 1: Determine defensive strategy.

b. Step 2: Apply release criteria.

c. Step 3: Prioritize threats.

d. Step 4: Maintain weapons.

e. Step 5: Allocate the weapons.

f. Step 6: Perform terminal guidance and illuminator assignment.

2.2.2.2.1 Determine Defensive Strategy

Given the volume and apparent objectives of the attack and the time to impact, defensive strategies can be selected. In a tiered system the defensive strategies may be subdivided into several tactical implementations (in each tier), each designed to support the overall system strategy. A major function of the BM/C3 will be to ensure that each tier has received and understands which of the selected strategies is to be employed.

The indications are unlikely to be absolute and unambiguous. Hence, the challenge of developing BM/C3 application software will be determining which elements of information to use and their variances. Given the challenge, software techniques which produce iterative answers may be applicable. A time budget for this function, expressed as a function of the number of objects, would assist in judging the effectiveness of candidate software approaches. This function probably requires advances to the software state-of-art.

2.2.2.2.2 Apply Release Criteria

Given the nature of the attack and the defensive conditions in effect at the time of the attack, the release criteria are applied, and authorization is sought. If the indications were definite and precise, then release criteria could be algorithmic because they would have been analyzed in advance. The pressures of making such a critical decision are such that the decision maker must have a set of pre-planned options. Since the situation is likely to contain ambiguities and uncertainties, decision support techniques should be used to present optional explanations of the situation, explain them, correlate these to pre-analyzed scenarios, and indicate alternatives to help the decisionmaker. The software state of practice will require some advances, especially for the decision support.

2.2.2.2.3 Prioritize Threats

The priority of each threat can be identified by parameters such as launch points, targets, projected burn time, time-of-impacts, yields and accuracy, flight phase (boost, mid-course, terminal) and the overall offensive strategy. Maintaining priorities will probably be difficult because they may change quickly with new information. For the software this function will be complex in terms of the extent to which changes have to be propagated to defensive system elements, which is dependent upon the architecture. Some advances to the state of practice will be necessary.

2.2.2.2.4 Maintain Weapons

The defensive assets require maintenance in the following areas:

a. Weapon consumables inventory,

b. Status monitoring, and

c. Backup.

2.2.2.2.4.1 Weapon Consumables Inventory

The weapon inventory at each platform is limited both by the non-renewable resources consumed by each shot and also by the rate by which renewable resources are regenerated. These elements represent absolute limits constraining permissible firing lists. These may be maintained by algorithms derived from the physical design of the weapon. A database of the amount of consumables present at each platform must be maintained dynamically throughout deployment . Levels of these consumables will change when they wear out, when they are expended against a threat, and when they are destroyed. This table will be critical to the construction of valid firing lists. Consumables to be tracked include ammunition, retargeting propellant, weapons delivery power, and so on. The software to calculate the inventories is within the state-of-practice.

2.2.2.2.4.2 Status Monitoring

The defensive assets under SDI control should routinely go through an operational test cycle. Each asset may be one of the following:

a. Ready,

b. Busy, i.e. either deployed or recovering/recharging from deployment, or

c. Dead.

A database will have to be maintained to reflect the status of each asset. Generation and maintenance of a database is within the software state-of-practice. Other factors, such as communications of the required data, especially while the system is in the midst of a battle, will complicate this function.

2.2.2.2.4.3 Backup

The system is expected to be able to reconfigure itself whenever a link in the network fails. Because of the complexity of the system, reconfigurations from scratch may be too time-

consuming. Backup tables can give a set of pre-calculated routines to be followed in the event of certain classes of failure. The routines may not be optimal, but they can provide a known baseline configuration, and they would be a fast approach to reconfiguration. Software for the creation, maintenance and implementation of backup tables is within the state-of-practice.

2.2.2.2.5 Allocate the Weapons

Given an understanding of the nature of the attack, the trajectory of each threat, the priorities of the targets, and the locations and statuses of each defensive weapon, firing lists have to be generated. The firing lists will be sets of orders to each defensive platform giving the time and target for each weapon to be used in the defense. The firing list will be extremely sensitive to the phase of the attack. Boost phase is the most desirable time in which to defend, because the plume is extremely easy to track, the re-entry vehicles (RV's) have not yet deployed. Consequently, destroying a missile during boost phase will prevent the release of RV's, decoys, and chaff.

This function's impact on the state-of-practice and state-of-art is also architecture dependent. For centralized architectures (which allocate weapons across several tiers) the process of generating firing lists may require iteration. If the firing list fails to cover all the threats, then a slightly different strategy may be used to generate another firing list. If all strategies fail to cover all threats, then the notion of triage may be used.

Given all the firings lists under consideration, the system will require metrics upon which to select the actual solution. As new threats appear and as kill assessments indicate misses, further firing lists must be computed. This process will continue throughout the battle. This approach will require advances in the state of art. Decentralized architectures will not require advances in the state-of-art, but some extension of the state-of-practice would likely be required.

2.2.2.2.6 Terminal Guidance and Illuminator Assignment

As with several other functions, the necessity for state-of-practice or state-of-art advances is architecture dependent. Some architectures may require a sizable computation load (large data volume, limited time for algorithmic solution, or look-up table search) to manage the terminal guidance of kinetic energy weapons. This load is likely to be heavier if enemy targets take evasive actions. The assignment of an illuminator to a threat may involve many of the same considerations named in the allocation of defensive weapons to threat, specifically coverage, energy, and inventory limits.

2.2.2.3 Kill Assessment and Revised Damage Estimate

Kill assessment is the act of evaluating the destructive impact of a defensive weapon against a threat. Even the best defense will not be 100% effective, and so this function is required. This function should continuously evaluate the expected damage based on the current threat. When an individual threat is killed, the expected damage is revised downwards. When an individual threat survives a defensive action, this function activates further defenses. Software for this function should not require advances to the state-of-art or state-of-practice.

2.2.2.4 Defensive Phase Weapon Handover

Each of the defensive phases (boost, post-boost, mid-course, and re-entry) is so different in terms of applicable sensors and defensive assets that each phase calls for its own

strategy. The defensive phase weapon handover involves the coordination of strategy and the sharing of data consistent with the goal of reducing leakage across all phases. The complications of this problem can be illustrated by the problems involved in changing from the boost to the post-boost phase. First, different sensors are involved in the two phases, one phase requiring sensors to track bright objects in the atmosphere, the other requiring sensors to track dark objects against the background of space. Second, different tracking centroids are involved because the plume is tens of feet from the bus in post-boost while it is closer to the booster body in boost phase. Finally, booster release gives a push to the bus, introducing a small change in velocity readings returned by the boost sensors. These factors suggest that a common coordinate system be used so that time is not required in software to perform coordinate transformations. Software for this function will probably require some advances to the software state-of-practice.

2.2.3 Self-Defense

The SDI system must distinguish between launches of missiles carrying RVs and launches of anti-satellite attack vehicles. Once ASATs are identified, prediction of the threatened space and time window(s) will have to be made for purposes of selecting a self-defense strategy, and commanding its execution throughout the appropriate portion of the SDI system. The battle managers' response options to a many-on-many ASAT attack will probably take into account a projection of the residual capabilities of the system after the self-defense battle and how it effects the battle management in all tiers of the system. Some of the common software requirements in this process are calculations of ASAT trajectories after boosted flight, orbital parameters after boosted flight, lethal volume and its intersection with SDI space platforms.

The software applications design of the system must permit an evolutionary response to new types of attacks.

Once it has been determined that a threat is directed against the system platforms, the attacked platform and/or some other cooperating platform may either:

a. Make evasive changes in their positions,

b. Take an offensive response, or

c. Deploy a counter measure, i.e., ECM, decoy, etc.

The selection of an appropriate response, monitoring of its effect and follow on actions will require decision support software in the earlier phases of the engagements, and more automated function support as time constraints and situation complexity limit the degree of human intervention. Advances in the decision support software state of art will be necessary for this function, as well as advances in state-of-practice at lower levels of software execution.

2.2.4 System Performance Monitoring, Reporting

In the midst of an attack, the system must be capable of monitoring its performance on a continuous and reliable basis. This self-consciousness is intended not only to detect failures but also to detect bottlenecks and slow-downs. In some cases, the system must be reconfigured to improve performance. Software to monitor and report, in contrast to being fault tolerant, should be within the state-of-art with improvements to the state-of-practice being necessary.

2.2.5 Platform Control

Platform control refers to maintaining the routine operation of the platform. These include housekeeping functions, e.g., thermal control, telemetry control, etc.; stationkeeping functions, e.g., ephemeris maintenance, navigation, and guidance; and autonomy functions aimed at keeping the platform alive in the event it loses contact with the ground stations.

Years of experience with space platforms have resulted in considerable knowledge about platform control under normal conditions. The state-of-art should be extended to cover platform control in hostile environments. Special attention must be paid to autonomy considerations in the event of damage to ground-based support facilities.

2.2.6 Decision Support Systems (DSS)

Identification of the functions where human intervention will exist effect the BM/C3 application software. Recommendation five (Section 2.4) suggests that this be addressed at the earliest possible time in the Battle Management architectures for the SDI system. Equally important is the identification of the types of decisions and the characteristics of the information that the decision maker requires. An adaptive decision support system concept will be necessary for the range of decision types and change in the experience level of the decision makers.

2.2.6.1 Taxonomies of Decision Problems

Numerous disciplinary areas have contributed to the development of decision support systems. These include computer science and engineering, which provide the hardware and software tools necessary to implement decision support system design constructs. The fields of management science and operations research have provided the theoretical framework in modeling, optimization, and decision analysis that is necessary to design useful approaches to choicemaking. The area of management information systems has provided the database design tools that are needed. The areas of organizational behavior and cognitive science have provided a source of information concerning how humans and organizations process information and make judgments in a descriptive fashion.

Thus, the design of a decision support system must be considered as a systems engineering effort. There have been many attempts to classify different types of decisions. Among the classifications of particular interest is the decision taxonomy of Anthony. He describes four types of decisions [Anthony 65]:

a. Strategic Planning Decisions, which are decisions related to making the highest level policies and objectives, and associated resource allocations,

b. Management Control Decisions, which are decisions made for the purpose of assuring effectiveness in the acquisition and use of resources,

c. Operational Control Decisions, which are the decisions made for the purpose of assuring effectiveness in the performance of operations, and

d. Operational Performance Decisions, which are the day-to-day decisions made while performing operations.

Simon has described decisions as structured or unstructured depending upon whether or not the decision making process can be explicitly described prior to the time when it is necessary to make the decision [Simon 80]. This taxonomy lends itself to the formalization

of expert skills and rules used for judgements. Generally, operational performance decisions are more likely to be pre-structured than strategic planning decisions. Thus, expert systems can usually be expected to be more appropriate for operational performance and operational control decisions than for strategic planning and management planning decisions. In a similar way, decision support systems will often be more appropriate for strategic planning and management control than for operational control and performance.

It is important to note that expertise is a relative term which depends upon familiarity with the task and the operational environment into which it is embedded. Since decision environments change and novices eventually become experts, it is clear that there should be many areas in which the proper knowledge base support is a hybrid between an "expert system" and a "decision support system". This suggests that there will be a variety of decision making processes in practice and that an effective support system should support multiple decision processes. In a similar way, the information requirements for decision making can be expected to be highly varied and an effective support system should provide for a variety of database management needs. An approach is to conceptualize the decision support components needed for evaluation of alternative courses of action. This is believed to be appropriate in that most of the major decisions made in the SDI system are unstructured or semi-structured. This suggests an eventual augmentation of the system with some form of expert system capability enhancing the efficiency of structured operational decisions and providing a better language interface to the system.

2.2.6.2 Decision Support System Information Criteria

A decision support system should provide the abilities to:

a. Formulate or frame the decision situation in the sense of recognizing needs,

b. Identify appropriate objectives with which to measure successful resolution of an issue, and

c. Generate alternative courses of action that will resolve the needs and satisfy objectives.

It should also provide support to enhance the abilities of the decision maker in obtaining the possible impacts on needs of the alternative courses of action. This analysis capability must provide the decision maker with an interpretation of the impacts on the objectives. The ability to interpret these impacts will lead to an evaluation of the alternatives and a selection of a preferred option. Providing this capability within the SDI system is unquestionably desirable. It is essential that the BM/C3 software and hardware be able to implement the chosen alternative course of action.

There are many variables that will affect the information that is obtained from any given decision situation. These variables are very clearly task dependent. Keen and Morton identify eight variables [Keen 78].

a. Inherent accuracy of available information - Operational control situations will often deal with information that is relatively certain and precise. The information in strategic and tactical planning situations is often uncertain, imprecise and incomplete.

b. Needed level of detail - Often very detailed information is needed for operational type decisions. Highly aggregated information is often desired for strategic

decisions. There are many difficulties associated with information summarization that need attention.

c. Time horizon for information needed - Operational decisions are typically based on information over a short time horizon, and the nature of the control may be changed very frequently. Strategic decisions are based on information and predictions based on a long time horizon.

d. Frequency of use - Strategic decisions are made infrequently, although they are perhaps refined fairly often. Operational decisions are made quite frequently, and are relatively easily changed.

e. Internal or external information source - Operational decisions are often based upon information that is available to the internal organization, whereas strategic decisions are much more likely to be dependent upon information content that can only be obtained external to the organization.

f. Information scope - Generally operational decisions are made on the basis of narrowly scoped information, relating well-defined events internal to the organization. Strategic decisions are based upon broadly scoped information and a wide range of factors that often cannot be fully anticipated prior to the need for the decision.

g. Information quantifiability - In strategic planning, information is very likely to be highly qualitative, at least initially. For operational decisions, the available information is often highly quantified.

h. Information currency - In strategic planning, information is often rather old and it is often difficult to obtain current information. For operational control decisions, very current information is needed.

2.2.6.3 Types of Decision Support Systems

Whether a system should be called a management information system (MIS), a predictive management system (PMIS), or a decision support system (DSS) will depend on the extent to which it possesses the capability to formulate, analyze, and interpret issues. In a classical management information system, the user inputs a request for a report concerning some question and the MIS supplies that report.

When the user is able to pose a "what if *X* " question and the system is able to respond with an "if *X* then *Y*" response, then we have a predictive management information system. In each case, there is some sort of formulation of issues and this is accompanied by some analysis capability.

The classical MIS need only be able to respond to queries with reports. Search of an electronic file cabinet or a relational database, would provide information with which a report generator could construct the desired report. The MIS would include the capabilities to:

a. Focus on data processing and structured data flows at an operational level, and

b. Produce summary reports for the user.

The predictive management information system would include an additional amount of analysis capability. This might require an intelligent database query system or perhaps just the simple use of some sort of spreadsheet model.

To obtain a decision support system, we would need to add the capability of model-based management to a MIS. But much more would be needed than just the simple addition of a set of decision trees and procedures to elicit decision maker utilities, as might be believed from examination of the paradigms of decision analysis. We would also need a system that is flexible and adaptable to changing user requirements. An example of this is the capability of providing support for different decision styles as they change with task, environment, and experience. Most decision situations are fragmented, in that there are multiple decision makers rather than just a single decision maker. The decision processes may also be seperated by time and/or space. Further, users of the SDI system will typically have many more activities than decision making to occupy themselves with, and it is necessary for appropriate DSSs to support many of these other information related functions as well.

It should be noted that the form of the decision maker is not explicitly identified. For the space defense system, exclusive of National Defense Policies dictating where a human must exercise positive control, i.e., dealing with the release of nuclear weapons, the selection of the proper form of decision maker should be an evolutionary process.

It is likely that the SDI system will require all types of decision support systems. For those functions where an MIS is necessary, the state-of-art will not require advancement; however, the state-of-practice may need some refinement. The state-of-art in both PMIS's and DSS's will most likely require advancement.

2.2.6.4 Language Interface

A language interface stands between the controller and the system. As mentioned above, the controller cannot be expected to learn programming, so the system must be capable of interacting with him in a language which has a minimum of ambiguity, especially during execution of the battle.

The use of "natural" language presents problems for the space defense system. Natural language are ambiguous, both intentionally and inadvertently. It does serve human purposes because syntactic and semantic redundancy and feedback can be used to disambiguate a message. However, in an operational environment the price to be paid for building in these features effects efficiency and cost. The potential size of the necessary knowledge base, along with its maintenance and search costs, will be factors which will adversely effect the efficiency and cost of the system.

In current applications it has been found sufficient to limit the language interface to a lexicon which is meaningful to the human user and tightly defined to the computer. Application of this technique to the space defense system should be the goal. The SDI should monitor developments in this technology area.

If the language interface is based upon a limited lexicon and a restricted semantics, then the state-of-practice and state-of-art will not require any advances. However, if a "natural" language interface is required the software state-of-practice and state-of-art will require advances.

2.2.6.5 Interactive Graphics

The use of interactive graphics will complement a limited lexicon for language interface. Three-dimensional graphics (holography) displaying objects as they move in time would aid the human cognitive process. A capability for the user to use rubber banding, lassoing, zoom, and pan with three-dimensional graphics would lead to a capability of the user issuing instructions to the system via the graphics. In two-dimensional graphics there are three prevailing technologies: vector graphics, raster graphics, and bit-mapped graphics. Bit-mapped graphics overcome both the flicker problems associated with vector graphics as well as the resolution problems associated with raster graphics.

For three-dimensional graphics, beyond the hardware advances necessary, the software state-of-art will also require advances to process the greatly expanded bandwidth of information in a timely fashion. For two-dimensional graphics, the software state-of-practice will likely require some improvement, exclusive of hardware improvements.

2.2.7 Operational Coordination

Operational coordination is required to integrate the SDI system into the overall U.S. and allied offensive and defensive forces posture.

2.2.7.1 Defensive and Offensive Coordination

The SDI system is only one part of our defensive system, and is itself a layered system. Notifications of defensive actions taken by other defenses must be relayed to the SDI system to prevent it from interfering. Coordination between layers will be imperative. The primary requirement of offensive coordination is that the SDI system not shoot down U.S. or allied missiles.

System integration requirements ensure that both of these functions are properly performed. A software issue for this integration is the trade-off between communications efficiency and security of those communications. In general the software state-of-art will probably not require advances to support the operational coordination functions. However, the software state-of-practice will probably require numerous upgrades to satisfy the exchange of information.

2.3 Current Status

Each of the requirements identified in Section 2.2 is summarized in Table II. Due to the introductory or tutorial nature of some paragraphs, they are not shown in the table. Only those paragraphs which explicitly treat the requirements for software state-of-practice or state-of-art are shown. This table shows whether the requirements:

a. Can be met with state-of-practice of software technology, or

b. Can be met with some advances in the state-of-practice but are within the state-of-art, or

c. Requires advances in the state-of-art, or

d. Are highly architecture dependent and cannot be judged at this time.

At best these conclusions are tentative since the complexity of the necessary BM/C3 applications software is dependent upon the architecture selected. Those requirements that will require advances in state-of-art will be discussed in the remainder of Section 2.3. Only

Table II. Assessment of Application Software Requirements vs. Current Software Technology

	SOP	SOP+	SOA+	?
2.2.1 Sensor Data Processing				
2.2.1.a Verify Sensor System Status		X		
2.2.1.b Object Discrimination				X
2.2.1.c Coverage Control	X			
2.2.1.3 Multi-Sensor Fusion				
2.2.1.3.2.1 Association and Correlation				X
2.2.1.3.2.2 Classification				X
2.2.1.4 Tracking				
2.2.1.4.1 Track Initiation				X
2.2.1.4.2 Track Maintenance	X			
2.2.1.4.3 Track Termination	X			
2.2.1.4.4 Track File Design and Storage		X		
2.2.2.1 Threat Evaluation				
2.2.2.1.1 Launch Categorization		X		
2.2.2.1.2 Threat Categorization, Notification,Verification	X			
2.2.2.1.3 Track Propagation	X			
2.2.2.1.4 Target Damage Estimation	X			
2.2.2.2 Weapon Assignment				
2.2.2.2.1 Determine Defensive Strategy			X	
2.2.2.2.2 Apply Release Criteria		X		
2.2.2.2.3 Prioritize Threats		X		
2.2.2.2.4.1 Weapon Consumables Inventory	X			
2.2.2.2.4.2 Status Monitoring	X			
2.2.2.2.4.3 Backup	X			
2.2.2.2.5 Allocate the Weapons			X	
2.2.2.2.7 Terminal Guidance and Illuminator Assignment		X		
2.2.2.3 Kill Assessment and Revised Damage Estimate	X			
2.2.2.4 Defensive Phase Weapon Handover		X		
2.2.3 Self-Defense			X	
2.2.4 System Performance Monitoring, Reporting		X		

Table II. (Continued)

	SOP	SOP+	SOA+	?
2.2.5 Platform Control			X	
2.2.6 Decision Support Systems				
2.2.6.3 Types of Decision Support Systems				
MIS	X			
PMIS/DSS			X	
2.2.6.4 Language Interface				
Limited Lexicon	X			
Natural Language			X	
2.2.6.5 Interactive Graphics				
Three Dimensional			X	
Two Dimensional	X			
2.2.7.1 Defensive and Offensive Coordination		X		

selected items from those requirements that are highly dependent upon architecture will be discussed.

2.3.1 Situation Assessment

For the SDI system to coordinate its action based on the current status of the battle, the situation (battle) must be constantly monitored. Coordination between layers (boost, mid-course, and terminal layers) will be imperative. The Naval Research Laboratory is currently funding work to develop state-of-art software for situation assessment and strategic planning.

2.3.2 Algorithms for Multi-Sensor Data Fusion

The software challenges involved in multi-sensor data fusion will involve finding computationally efficient techniques to perform the association, correlation, and classification functions. Algorithmic effeciency will be particularly significant in making decisions concerning which approach(es) lead to satisfactory processing of the anticipated data volume. The Strategic Defense Command is funding work to develop advanced approaches for multiple sensor functional correlation and tracking.

Unfortunately, target distribution statistics are rarely known with precision. In this case, for classification, association and correlation, heuristic inputs could provide the missing intelligence, allowing not only for the choice of the preferred algorithm, but also for the choice of estimated parameters or even alternative rules.

Each of the armed services has ongoing programs for the development of algorithms. Test beds for algorithms are a common feature of these programs. However, in several of the approaches it appears that the test beds for measuring the algorithms do not take sufficient cognizance of the interplay between the computer architecture, i.e., parallel processing versus single thread, and the efficiency of the algorithms. The Strategic Defense Command is funding an effort to identify techniques and algorithms for improved target classification and define requirements for the BM/C3 system to initiate passive or active responses.

The Naval Research Laboratory has a program in progress investigating the use of parallelized track files. They are using a monotonic logical grid to investigate the efficency of the nearest neighbor method, where these nearest neighbors are organized in physical memory for parallel processing. Analyses to date have shown the advantages of this method over more traditional serial processing methods used for track correlation.

2.3.3 Weapon Assignment

Most of the subfunctions in weapon assignment can probably be accomplished within the state-of-practice, or with some advances in software technology. Two subfunctions, the determination of a defensive strategy and the allocation of weapons, may require new software techniques for solution if hierarchical architectures are used. The Strategic Defense Command is currently funding work to develop a candidate set of algorithms for weapon-to-target pairing in late midcourse and terminal phases, and evaluate selected algorithms in realistic environments.

2.3.3.1 Determine Defensive Strategy

The determination of specific defensive strategies in response to various offensive strategies will evolve over time as the realism of simulations improves and the National Test Bed facilities are exercised. These results can be used to make an *N*-ary decision

trees, e.g., complex look up table, for each unique defensive strategy. Experience has shown that the greatest value derived from the cycle of simulation and decision tree generation at the weapons level is its assistance in evolving basic weapon system strategy and later tactics.

As systems analyses and simulations progress to define the offensive and/or defensive strategies, one capability required in the battle management software will be to determine which of the strategies is being pursued. Allowing that there can be a large ensemble of sensor data representations (allowing for incomplete data due to loss of assets or contradictory data due to countermeasures) for any offensive strategy and that the time to algorithmically sort these will be constrained, the battle management software has to have the parallel capability of classifying offensive strategies at an abstraction level above sensor data and which is oriented toward human comprehension. In essence, the higher level of abstraction will help the battle manager to choose strategies early enough to allocate resources (weapons, processing, communications), while the sensor are algorithmically processing the track files for weapon pointing and tracking functions.

There will be a need to have battle management advisories for those instances where the counter-strategy has been worked out in advance. For each of these a sequence of decisions can be constructed, including what information the decision maker should seek, its range of expected values, what variables the decision maker will influence, and the range of values (not necessarily numeric) that should be expected as a result of a decision. Beyond this capability will be the need to interpolate and extrapolate advisories. This will require that the system possess meta-level logic which would be used to generate object-level rules that would be used to form the new advisories. As the simulation and Test Bed facilities are exercised, the software will have the opportunity to learn and formulate new meta-level knowledge and refine it.

The value of meta-level knowledge for guiding the invocation, construction, and explanation of object-level rules in an expert system has been demonstrated [Davis 76]. More recent work [Fu 84] has generated meta-rules automatically and were used to improve the efficiency of a diagnosis system by selecting a set of the meta-rules.

Finally, it will be necessary to determine what part a human can and should play during the actual battle; this should become part of the defense strategy. The Naval Research Laboratory is currently funding on effort to determine the appropriate role for a human SDI commander under the severe time constraints of the SDI scenerio and investigate alternative mechanisms for human-machine interface. This effort will also include work to establish the feasiblity of the role NRL defines for the human SDI commander.

2.3.3.2 Allocate the Weapons

This function will be beyond the state-of-art in software technology if an architecture is used which depends upon a single BM/C3 node to generate firing lists that are optimal. To reduce the quantity of weapons that must be orbited, the obvious strategy would be to optimize the firing list according to threat priority, weapon coverage, etc.; this solution is NP-complete. Two tractable alternatives are:

a. Algorithms which approximate optimal solutions, and

b. Delegatation of autonomy to low levels.

The first option, using approximation algorithms, accommodates the time tradeoffs. Optimal algorithms are impractically slow, while approximation algorithms may have

acceptable time complexity for the accuracy required. The accuracy required of the approximation algorithm must be balanced against the computational resources required to deliver this accuracy. The Naval Research Laboratory is currently funding the development of and experimentation with weapon allocation algorithms for SDI use.

A special case of approximation algorithms is an expert system. Once the National Test Bed is established, it will be able to generate offensive scenarios. These scenarios can be subjected to analyses to generate rules for the allocation of weapons.

The techniques of knowledge acquisition, knowledge representation, and resolution could be used to mimic the human reasoning processes on a computer. The Naval Research Laboratory is funding work to develop a state-of-art hybrid tracker/correletor system that combines optimum statistical estimation techniques with heuristic reasoning and knowlegde base system techniques. Additionally, this work will establish feasibility of SDI applications by experimentally determining functional performance, sensitivity, and computational resourse requirements.

Finally, it may be more tractable to use an architecture whereby weapon allocation is delegated to the lowest level consistent with a low-leak defense. Using this approach the BM/C3 interface is distributed throughout the system. At the distributed interface the BM/C3 application software will be presented with a reduced number of variables. In this case optimization algorithms may become tractable, or the situation could reduce to the use of look-up tables. The effect of the BM/C3 application software problems should be analyzed as to their effect upon the whole system.

2.3.4 A Framework for Developing Decision Support Systems

There are three principal components of a Decision Support System (DSS):

a. Database Management System (DBMS)

b. Model Base Management System (MBMS)

c. Dialogue Generation and Management System (DGMS)

Also, there are three technology levels at which a DSS may be considered. The first of these is the level of DSS tools themselves. This level will contain the hardware and software elements and those system science and operations research methods that will be needed to design a specific decision support system. The purpose of these DSS tools is to design a specific DSS that will be responsive to a particular task or issue.

Often, the best designers of a decision support system are not the specialists familiar with the DSS tools. The principal reason for this is that it is difficult for one person to be very familiar with a great variety of tools and also the requirements needed for a specific DSS. This suggests a second, intermediate level of technology: the decision support generator. The DSS generator is a set of software tools for constructing a DSS. A DSS generator would contain an integrated set of tools to support inquiries, modeling languages, optimization and statistical (and other) analyses, graphic displays, and report preparations. Some work has been done to determine the characteristics of successful decision support systems, but these experiments do not reproduce the stress likely to be encountered by humans during an attack.

2.3.4.1 Role of the National Test Bed

There are many people that can become involved in the design and use of a decision support system. At a minimum, these include the DSS users and their staffs, the DSS designer, the technical support people, and the specialists in computer and system science. The advantage of the DSS generator is it enables the DSS designer to interact directly with the user group. This eliminates, or at least minimizes, the need for DSS user to interact with the specialists in computer and system science. Remember that the user will seldom be initially able to specify the requirements for a DSS, consequently the advantage of having a DSS generator for use by the DSS designer in interacting with the DSS user becomes apparent.

In the SDI system, the DSS generator concept can be envisioned to be implemented through the role of an analyst team at the National Test Bed. The analyst team acts as a go-between for decision maker(s) and DSS tools, to enable selection of the most appropriate set with which to resolve a specific problem. The three primary ingredients determining the software support that must be supplied to the analyst team to make this process effective and efficient are the time requirements, system complexity, and the professional skill of the analysts.

2.3.4.2 Process-Independent Approach Toward Decision Support

The design and development of a DSS can be patterned after the stages of the design process. These stages include preliminary conceptual design, and evaluation and testing of the decision support system. A decision support system is intended to be used by decision makers with varying experience with a particular task. For this reason, it is important that a DSS designer consider the variety of issue representations, the operations that may be performed on these representations, the automated memory aids that support retention of the various results, and the control mechanisms that assist decision making.

A useful control mechanism results in the construction of heuristic procedures to enable development of efficient and effective standard operating policies. Other control mechanisms are intended to help the decision maker use the DSS and acquire additional skills. This process independent approach toward the development DSS's is due to Sprague and Carlson and is known as the ROMC approach. It also serves to specify the essential features a DSS generator must have to be potentially capable of building an effective decision support system.

2.3.4.3 Database Management System

A database management system is one of the three fundamental components of a decision support system. An appropriate database management system must be able to work with data that is both internal and external to the organization. Whenever there are multiple decision makers, there will be a need for personal databases, local databases, and system wide databases. Some of the desirable characteristics of a DBMS include the ability to cope with a variety of data structures that allow for probabilistic, incomplete, and imprecise data, and data that is unofficial, and personal. The DBMS should also be able to inform the user of the types of data that are permissible and how to access them.

To construct a database, we must first identify a data model. A data model is a collection of data structures and integrity rules which are used to constrain permissible data values. There are at least five models that may be used to represent data. The most elementary of these is the individual record model. The relational model is a powerful generalization of the record model. A relation is a fundamental data structure in the relational model, and there may be a number of fields in any given relation.

The hierarchical data model is a relatively efficient representation of data. In a hierarchical model, the structure of information is additional information, which in the relational model is contained in the fields. Because of the structure of hierarchical the model, it is necessary to repeat some of the data that need be stored only once in a relational model. The network model is a generalization of the hierarchical model in that there are links between records which enables a given record to participate in several relationships.

Due to the potential need to accommodate expert system capabilities in the decision support system, it is desirable to consider a production rule model as a fifth data model. This will enable inferences to be made. A more extensive treatment of DBMSs and their applicability to SDI is presented in the Section B5, Distributed Data Management Systems.

2.3.4.4 Model Base Management System

The desire to provide recommendations in a decision support system leads to consideration of model base management systems. It is through the use of MBMS that one is able to provide for sophisticated analyses and interpretations in a decision support system. The single most important characteristic of a MBMS is its ability to allow the decision maker to explore the decision situation through use of a model base system. This can occur through the use of modeling statements and data abstraction models. This latter approach is close to the expert system approach in that there exist element, equation, and solution procedures that will together constitute an inference engine.

It will be desirable to use multiple models to accommodate the desire of the decision maker for flexibility. To provide flexibility, the MBMS should maintain various prewritten models found useful in the past. It should be possible to perform sensitivity tests of model outputs, and to run models with a range of data to obtain the response to a variety of "what if" questions.

The purpose of the MBMS component of a DSS is to provide effective, efficient, explicable, and flexible access to mathematical optimization and modeling tools. This feature greatly distinguishes a DSS from the much more traditional MIS in that the MBMS augments the data retrieval features of a MIS with powerful systems science and operations research tools for information analysis and interpretation.

Models have traditionally been represented most often as subroutines, and more recently as statements in a very high level command language which invoke the appropriate subroutine. Software packages for statistical analysis, such as SPSS or SAS, are basically collections of subroutines which are used to represent models. IFPS is a prototypical example of models represented as statements in modeling languages.

Models can also be represented as three different types of data: elements, equations, and solution procedures. The distinction between modeling languages and data retrieval now in large part disappears. It then becomes possible to conceive of a natural language or dialog that describes needed data. Alternatively, we can think in terms of data retrieval languages that are procedural and relational, and which allow manipulation of data through the use of models. A prototype of this sort of effort is the PLATOFORM system developed by Exxon to manage mathematical programming models through shared databases, modular construction of models, storage and retrieval of analysis results, case study presentations, and other capabilities.

Structural management is a major concern in an effort of this sort. It is needed to enable representation of a large number of models within the confines of a single database. Some modeling constructs are based on quite different logics; for example, a linear programming

model and an econometric model are based on different logics. Yet, each of these would need to have access to at least some of the same data. Alternatively, we could have two models that are based on the same theoretical constructs but which use data at different hierarchical levels; for example, two versions of the same linear programming or economic model may be constructed at different levels of aggregation.

The typical operations performed on databases will be needed in a MBMS as well as restructuring operations to enable the integration of two or more models. There will also be a need to "exercise" the model, something that does not appear to have any counterpart in a standard DBMS. There are numerous related questions of initiating, controlling, and describing the model solution process and results; there are similar concerns for the storage and recall of the process and results.

Solving a model will require linking the DBMS to the modeling process. A significant part of the effort will consist of determining design principles. Afterwards, it will possible to construct a workable MBMS that is linked to the DBMS, allowing the determination of new types of data or functions, and new data manipulations or model solutions.

2.3.4.5 Dialog Generation and Management System

The dialog generation and management system (DGMS) portion of a DSS is designed to satisfy knowledge representation and to control and communicate requirements of the DSS. The DGMS is responsible for presentation of the outputs of the DSS to the decision makers and for acquiring and transmitting their inputs to the DBMS and the MBMS. Thus, the DGMS is responsible for producing DSS output representation, for obtaining the decision maker inputs that result in the operations on the representations, for interfacing to the *memory* aids, and for explicit provision of the control mechanisms that enable the dialog between user input and output and the DBMS and MBMS. Thus the is *strongly involved* in each of the ROMC approaches to the analytic design of decision support systems.

There are a number of possible dialogs. These are inherently linked to the representational forms that are used for the DBMS and MBMS. Menus, spreadsheets, trade off graphs, and production rules are some of the formats that may be used as a basis for dialog system design. Generally, several of these should be used as the support system. A user may wish to shift among these formats as the nature of issues and experience with them changes. The DGMS should be sufficiently flexible to allow review and sensitivity analysis of past judgments, and to be able to provide partial judgments based upon incomplete information. The DGMS should be "user friendly" by providing various HELP facilities prompting the decision maker.

2.4 Recommendations

Recommendations 1 and 2 involve standardization of terminology to assist the SDIO in the Phase III evaluation. Recommendation 3 involves the establishment of an Parallel Algorithm Test Center to test the performance of parallel algorithms as a function of the architecture on which they are run. The evaluations from the Parallel Algorithm Test Center would help in deciding which parallel architectures best meet the communication requirements of a particular parallel algorithm, contributing to the full scale development decision. Recommendation 4 recommends support to software research for parallel algorithms performing sensor data processing functions assessed to stress software state of art for some architectures. Recommendation 5 is to identify criteria for selection of BM/C3 functions to be performed by humans. Recommendation 6 is to have SDIO support research into a limited class of applications of artificial intelligence. Recommendation 7 is

to develop a Decision Support System based on an open architecture and the use of flexible tools.

Recommendation 1: Each of the architectures which have been investigated to date should identify which of the BM/C3 software modules are considered to be stressing technology. For each of these, alternative configurations of sensors,weapons, etc., should be identified which significantly reduce or eliminate the BM/C3 application software difficulties.

In parallel, the modifications to the sensor, weapon, etc. component should be analyzed to determine if the modification can be subsumed in the preliminary design.

Recommendation 2: A glossary of terminology should be provided by the Government to each organization participating in the Phase III BM/C3 architecture and system study.

Recommendation 3: An Parallel Algorithm Test Center for research and evaluation should be established, possibly as part of the National Test Bed.

The Parallel Algorithm Test Center should also serve as a library for reusable BM/C3 application software modules. One alternative to achieve this goal would be tasking the system and BM/C3 architecture contractors with recommending specific algorithm implementations for those processes that have been identified within the alternatives. Algorithms in the library would be tested/certified and have, as a minimum, descriptions of:

a. Accuracy,

b. Reliability (particularly for AI applications), and

c. Performance, e.g., memory requirements, computational time requirements -- sublinear, linear, polynomial, or NP-complete, etc.

The library should have a common algorithm specification language. Proof of concept prototypes could be written in any computer language; however, by the time an algorithm enters the library it should be implemented in a common language, Ada being the most likely candidate to conform to DoD directives. This constraint will allow for performance comparisons and facilitate reusability.

Artificial intelligence algorithms should be included in the Algorithm Test Center. This approach would ameliorate the need for a separate AI Test Center and allow an integrated assessment of approaches (heuristic and deterministic).

Recommendation 4: Research algorithms for multi-sensor data fusion.

The multi sensor fusion function can be made more tractable by selecting architectures tending to have smaller autonomous groupings of platforms (space or ground based). However, analyses to date do not suggest that these architectures will have acceptable overall system capabilities (boost through terminal defense). Recognizing that the association, correlation and classification functions will stress existing software technology, the SDIO should support algorithm research directed toward improving these functions.

The extent of human intervention in the BM/C3 process is an issue which will effect the specification of application software.

Recommendation 5: The amount of human intervention should be carefully thought out as early as possible and prototyped to determine the feasibility of developing the application software.

Criteria and methods to allocate the discretionary decision processes between human and machine components should be a primary concern of near-term BM/C3 research.

The National Test Bed would be a good source for obtaining information to make the allocations. A software strategy for decision support should focus early on developing the tools that will be required in the National Test Bed to address this issue.

Recommendation 6: The SDIO should support some artificial intelligence research, most probably in the area of decision support systems.

The SDIO should support investigating the use of artificial intelligence techniques for projects where there exists a base of human expertise. Additionally, the SDIO should monitor progress in applied AI research.

Recommendation 7: The types of decision support systems required for the SDIO system and National Test Bed should be identified. To eliminate, or reduce to a minimum, changes to the DSS, as architectures mature, the DSS design should be based upon open architecture principles.

As described in Section 2.3.3.2, and elaborated in the description of the conceptual Model Based Management System (Section 2.3.3.4.1), a process independent approach should be taken to the DSS system. The approach should focus on the design of an integrated set of software tools that can be flexibly tailored to provide decision support through the use of models in conjunction with DBMS and DGMS.

The development of the Model Based Management System structure and software should be in parallel, and closely coordinated with the Parallel Algorithm Test Center activities (see Section 2.4.3). Through such coordination a prototype MBMS, supporting decision support systems, could be undertaken in the near-term.

2.5 References

[Anthony 65] Anthony, R.N., *Planning and Control Systems: A Framework for Anlysis*, Harvard University Graduate School of Business Administration, Boston, 1965.

[Davis 76] Davis, R., *Applications of Meta-Level Knowledge to the Construction, Maintenance and Use of Large Knowledge Bases*, PhD dissertation, Computer Science Department, Stanford University.

[Fu 84] Fu, Li-Min, and Buchanan, Bruce G., "Enhancing Performance of Expert Systems by Automated Discovery of Meta-Rules," *First Conference on Artificial Intelligence Applications*, IEEE, 1984.

[Keen 78] Keen, P.G.W. and Morton, M.S. Scott, *Decision Support Systems: An Organizational Perspective,* (1978).

[Simon 80] Simon H., "Cognitive Science: The Newest Science of the Artificial," *Cognitive Science* 4 (1980).

2.6 Bibliography

Alspach, Daniel, "An Approach to the Multi-Sensor Integration Problem," *EASCON 16th Annual IEEE Electronics and Aerospace Systems Conference and Exposition Proceedings,* IEEE, 1983.

Blackman, Samuel S., *Multiple Target Tracking with Radar Applications,* Artech House, 1986.

Bowman, Christopher, "Maximum Likelihood Track Correlation for Multisensor Integration," *Proceedings of IEEE Conference Decision Control,* IEEE, 1979.

Chang, Chaw-bing, "Application of State Estimation to Target Tracking," *IEEE Transactions on Automatic Control* (February 1984).

Institute for Defense Analyses, *Software Requirements for the Computational and Analytic Segment (CAS) of the WIS Smart Advisor for Planning and Execution Decisions (WISSAPED)* (Revised), IDA Paper P-1893, Volume VII, (December 1986).

Rome Air Development Center, *Signal Processing: Air Force C3 Requirements, State of the Art Assessment, and Investment Strategy,* RADC Report #RADC-TR-84-31.

Reiner, Julius, *Application of Expert Systems to Sensor Fusion,* 1985.

Strattan, Robert, "Target Identification from Radar Signatures," *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing,* IEEE, 1978.

Waltz, Edward L., "Computational Considerations for Fusion in Target Identification Systems," *Proceedings of the IEEE 1981 National Aerospace and Electronics Conference* , IEEE, 1981.

Wertz, James R. (ed.) *Spacecraft Attitude Determination and Control,* D. Reidel, Boston, 1985.

Wolf, Paul, "Elements of Photogrammetry," SDIO BM/C3 Workshops, 9 (16-17 April, 29 April-1 May 1986).

SECTION B3

Network Communications

Prepared by Cathy Jo Linn

Topics covered in Section B3:

3.0 Network Communications

3.1 Introduction
3.1.1 Purpose and Scope
3.1.2 Background
3.1.3 Relationship to Other Foundation Areas
3.1.4 Organization of Section B.3

3.2 SDI Requirements
3.2.1 Platforms and Platform Management Systems
3.2.2 Development and Maintenance System
3.2.3 Summary of the SDI's Network Needs

3.3 Current Status
3.3.1 Introduction
3.3.2 Satellite Communications Systems
3.3.3 Computer Networks
3.3.4 Protocols
3.3.5 Routing
3.3.6 Naming
3.3.7 Topology Updating
3.3.8 Flow Control
3.3.9 Deadlock and Livelock
3.3.10 Clock Synchronization
3.3.11 Distributed Consensus
3.3.12 Encoding

3.4 Recommendations
3.4.1 Introduction
3.4.2 Funding Recommendations

3.5 References

3.0 NETWORK COMMUNICATIONS

3.1 Introduction

3.1.1 Purpose and Scope

The purpose of this section is twofold. First, a comparison of the current state-of-the-art and state-of-practice in network communications to the needs of the SDI is made. Second, research and development projects in network communications are recommended. These recommendations are aimed at aiding in (1) making the full scale development decision of the early 1990s, and (2) obtaining the technology needed to develop the SDI system.

3.1.2 Background

In this section the basic switching techniques and protocols of communication systems are reviewed. These concepts are used in identifying the network communication needs of the SDI.

The switching techniques used in wide-area communications networks are (1) circuit switching, (2) message switching, or (3) packet switching. In a circuit switching network, a route through the network is physically set up before any data is sent and is held until the message is completed. This technique is used in the telephone system. If the communication tends to be short bursts of data, as in computer networks, the overhead of allocating a circuit and the remaining time that is not used to send data can result in poor utilization of the network.

In both message-switched and packet-switched networks, the data is individually routed from each node in the network to the next. More efficient utilization of the circuits can be made in this way. In message switching, complete messages are sent from node to node. However, the variable size of messages makes the allocation of buffers somewhat difficult. In packet switching, the messages are divided into equal size packets, and the individual fixed size packets are sent from node to node. Packets of a single message may take different routes to the final destination and may arrive in any order. Therefore, the message must be reconstructed after all packets have arrived.

A packet broadcast system such as a packet radio network (PRN) extends the concept of packet switching to multiple-access channels. A message is broken up into packets, as in packet switching and then broadcast to all nodes listening on the communications channel. Access to the channel must be managed to minimize overlapping transmission. In some PRNs, nodes are mobile and, therefore, the topology of the network is constantly changing.

A "protocol" is a set of conventions or rules that govern the operation of functional units to achieve communication, [IEEE 83]. The International Standards Organization (ISO) Reference Model for Open Systems Interconnection (OSI) defines a model structure for communications protocols in seven layers [Tanenbaum 81]:

a. The physical layer provides low-level communication of bits.

b. The data link layer provides framing and checksumming.

c. The network layer provides internetwork addressing and routing of packets.

d. The transport layer provides reliable message communication and host-to-host addressing. This layer shields the upper layers (5-7) from the details of the carrier layers (1-3).

e. The session layer provide connection management, synchronization and process-to-process addressing.

f. The presentation layer provides data formatting, encryption, and compression.

g. The application layer is application dependent.

The layered, modular structure of the OSI model has often been used in the design of protocols. Some protocols exist, however, that do not easily map onto an individual layer of this model. They provide functions from several of these layers. An example of such a protocol is the Transmission Control Protocol/Internet Protocol (TCP/IP) discussed in Section 3.2.3. A good survey of communications protocols is given in [Tanenbaum 81].

3.1.3 Relationship to Other Foundation Areas

The network communications area is most closely related to the foundation area of operating systems. A primary function of the operating system(s) for the SDI is to provide support for network communication. This support must provide a foundation for the real-time and fault-tolerant performance needed by the communications protocols. The issues of the SDI operating system(s) are discussed in Appendix B, Section 4, Distributed Operating Systems.

As the platforms of the SDI system circle the earth and as processors fail, the configuration of the SDI network constantly changes. Unpredicted and undesired patterns may develop. The user interface of the SDI must provide for "the visualization of the network's dynamic status and [support] manual corrective actions" [Eastport 85].

The SDI system will operate in a hostile environment. The overall system must survive and perform reliably even if individual links or entire clusters of the network are lost. The general techniques for designing reliable fault-tolerant software are discussed in Appendix B, Section 11, Reliability and Survivability. Examples of research in fault-tolerant distributed operating systems are reviewed in Appendix B, Section 4.

3.1.4 Organization of Section B.3

Section 3.2 discusses the software network communication needs of the SDI. Included are the general architectural characteristics of the SDI communication network and the resulting performance characteristics. Section 3.3 reviews work in network communications of interest to the SDI to determine the state of the art and the state of practice. In Section 3.4, funding recommendations are made, calling for support of work in fields critical to the SDI's success. Long-term and short-term objectives of the SDIO are considered.

3.2 SDI Requirements

The network for the SDI must meet the needs of the three different environments of the SDI. These environments are: (1) the platforms, (2) the platform management system, and (3) the development and maintenance system.

3.2.1 Platforms and Platform Management Systems

The platforms and platform management system require "secure, survivable, high-performance communications among all the spaceborne and ground-based assets: sensors, weapons, and battle management computation" [Eastport 85]. Such a system must include support for the following battle-management functions [Turner 86]:

a. Relay of automated network control data, i.e., sensor coverage and reporting assignments, communications routes and address structures associated with the battle-management process, and computer tasking.

b. Transmittal of sensor-derived event reports for engagement assessment, and sensor-derived tracking data for fusion and initiation and maintenance of multi-sensor kinematic track files.

c. Relay of kinematic target track data, engagement prioritization assessments and decisions, weapon designation and initial aiming data, and database update information.

d. Transmittal and acknowledgment of directive messages which govern and may override automated phases of the battle-management process.

A hierarchical network has been suggested for the platforms and platform management system to perform these functions. A fast local communications network is needed between the processors of a single platform. The platforms are grouped with neighboring platforms into clusters that cooperate and communicate frequently. These clusters may overlap and, therefore, an individual platform may be a member of more than one cluster. The formation of the clusters may be static, or they may be formed and reformed dynamically. All clusters, and therefore the platforms, are connected via the system network to communicate with the platform management system. This layered network, or network of networks, is called an internet.

It has not yet been determined whether the architecture of the SDI will provide a dedicated communication backbone or whether the entire communications network will be "piggybacked" on the platforms for sensors and weapons. A distinct advantage of a dedicated communications backbone is the ability to develop, deploy, and thoroughly test the communications system before deploying any other part of the SDI.

The topology of the platform and platform management system internet is constantly changing. These changes are due to (1) relative positional changes of platforms, (2) "normal" failures of processors, and (3) failures due to a hostile, changing environment, (jamming, explosions, etc.). Communication links may be broken unexpectedly for any length of time. Fault-tolerant techniques, including redundancy and excess capacity, must be used to provide a reliable and survivable communications network. Routing tables and communication protocols must quickly adapt to this changing environment and must provide a quick way for links to be reestablished as soon as possible.

In such a dynamic environment, the internet protocol must provide reliable end-to-end error control, assuring either the accurate delivery of messages or a time-out mechanism to prevent deadlock. The internet must also provide a means of flow control to prevent overloading of the communications network and the resulting poor performance.

Several other DoD communication networks, such as WWMCCS (the World Wide Military Command and Control System), already exist and the SDI communication internet needs to cooperate with these systems to provide the maximum functionality. The protocol must also be extensible so that the internet can evolve with the SDI. The operating systems of

the different parts of the SDI may have different Interprocess Communication (IPC) mechanisms. The protocol must be compatible with all of these IPCs. A modular and hierarchical design (a network of networks) is needed to provide this type of extensibility and interoperability.

The internet for the platforms and platform management system must also be secure. The concept of a secure network is relatively new and DoD is currently developing a set of evaluation criteria similar to those used in the evaluation of operating system and a draft is currently available, [TNEC 85]. These evaluation criteria divide computer networks into four divisions, closely paralleling the divisions used in the Trusted Computer System Evaluation Criteria, [TCSEC 83]

Division ND - Minimal protection: Evaluated networks that do not meet the requirements for any higher level are placed in this division.

Division NC - Controlled access protection: These systems provide a single security level, access to be permitted at the discretion of the owner, and auditing facilities to monitor user access to information.

Division NB - Mandatory protection: These systems provide the policies of Division NC, provide mandatory separation of security levels, and document a security model on which the implementation is based.

Division NA - Verified design: These systems provide the policies of Division NB and verification of the design and the code.

The SDI internet requires security at the NA level, but possibly some portion without multi-level functionality.

The internet for the platforms and platform management system requires very extensive performance characteristics. The Eastport group estimated that the required communication rate is "1-10Mbit/sec with less than a few tens of milliseconds delay among closely coupled (neighboring) assets and less than a second or two delay between any remote parties" [Eastport 85]. An analysis of the traffic capacity requirements for the SDI architecture proposed by IDA resulted in estimates that "appear to be well within the current state of the art for digital communication technology" [Turner 86]. More recent architectural studies have estimated that the required communication rate will reach 80Mbit/sec. While this data rate may be within the state-of-the-art, the reliable, secure software protocols to handle this amount of data do not currently exist. Further, it is unknown whether the application of state-of-the-art protocol techniques will result in the real-time performance that the data rate requires.

3.2.2 Development and Maintenance System

The network of the development and maintenance system is also an internet. The workstations of an individual contractor are linked together via a local area network, while many contractors are linked via a wide area net. The linking of the contractors facilitates communication during the development of related software to ensure compatible interfaces. The topology of the development system is relatively static, changing only as often as new contractors are added to the system.

As in the other SDI environments, the development and maintenance environment requires compatibility with the existing networks, in this case the commercial networks of the contractors and/or ARPANET/MILNET. The fault-tolerence, reliability, and performance

requirements of the development and maintenance network are not as extensive as those of the platforms and platform management system. The security requirements however, are just as extensive, requiring multi-level and compartmentalized functionality at the NA level.

3.2.3 Summary of the SDI's Network Needs

The platform and platform management systems require a secure, fault-tolerant, and survivable communications network. This network will constantly change topology as nodes fail and satellites circle the earth. Under battle conditions, when the system is under a heavy communications load, these reconfiguration requirements will increase dramatically, as large areas of the communications network are lost, temporarily and permanently. While the fault-tolerant, survivable, and reconfigurable aspects of the development and maintenance system are not as extensive as those of the platform and platform managment systems, all will require interoperability with existing networks.

3.3 Current Status

3.3.1 Introduction

In this section the current status of networking research is reviewed. The state-of-practice in communications networking is represented by a discussion of current satellite communication systems and computer networking projects.

State-of-the-art research and development project in subareas of research of interest to the SDI are reviewed. These subareas include protocols, routing, naming, topology updating, flow control, deadlock and livelock detection, clock synchronization, distributed consensus, and security. Time constraints have prevented a thorough review of all work in these areas. Where reports of ongoing research projects were not yet available, researchers have simply been identified.

3.3.2 Satellite Communications Systems

The commercial and military satellite communications systems of today use very limited software techniques. Satellite systems are only currently being expanded from simple links to and from ground stations to autonomous space communication stations with dynamic satellite-to-satellite links. Routing techniques used in computer networks are just now being applied to communication systems under design, (MILSTAR, and NASA's Advanced Communications Technology Satellite).

The MILSTAR system, currently being designed by the DoD, "will stretch the state-of-the-art in communications satellites in several key areas," [Ricci 86]. This system will include four satellite in geosynchronous orbit over the equator as well as three satellites in elliptical polar orbits. The MILSTAR system will have the ability to dynamically create crosslinks between the satellites and dynamically route messages. These satellites will be the first able to function autonomously, maintaining their positions for months, in case the control centers on earth are destroyed. The dynamic routing, error-correction encoding, encryption, and frequency hopping used by the MILSTAR system will provide security, reliability, jamming resistance, and survivability. At the same time, however, it will reduce the data transmission rate to 1Mbps.

3.3.3 Computer Networks

The development of the Internet system was sponsored by DARPA as a research tool to interconnect a set of diverse networks sponsored by DARPA, including the well known

ARPANET. It was designed to meet the needs of the defense community and the protocols that define its architecture have been adopted as DoD standards, (the Transmission Control Protocol (TCP), and the Internet Protocol (IP)). The IP level provides the ability to send datagrams, a packet of data unrelated to any other packet. The arrival of these datagrams is not guaranteed. The TCP level uses acknowledgment, retransmission, and other techniques to provide reliable, ordered arrival of related packets of data. Placing the reliability facility at a relatively high level of the protocol allows the DARPA Internet to connect with networks, such as Packet Radio networks, that provide only the datagram level. The Internet has been in use for over five years and currently connects hundreds of networks worldwide.

WIS, funded by the Joint Project Management Office, is revising WWMCCS. The protocol needs of this system are very similar to those of the SDI system. The network is composed of a moderate number (less than 50) local area networks located worldwide but grouped at a number of sites, The Defense Data Network (DDN) is used as the long haul backbone. Interoperability is required with other networks, domestic and foreign, commercial and military. WIS requires multi-level security, priority, reliability, reconfigurability, and distributed databases.

WIS has implemented an Ada version of the DoD's TCP/IP protocol as well as the ISO transport layer protocol with the connectionless internet protocol. (This internet protocol is currently being defined as a subprotocol of the network layer, [WIS 86].) WIS has also proposed research directed at the automatic generation of Ada protocol software. The short-term goal of the proposed work is to gain experience in developing protocols in Ada and to establish a technical baseline for proceeding later to a prototype system [WIS 86].

RADC is initiating efforts to develop communication capabilities for BM/C3 application. This work will focus on system integrity, survivability, and fault tolerance in a dynamic real-time environment.

The SDIO has announced plans for an SDINet. This network will first serve the contractors during the development of the SDI software. It will later be used to provide a testbed for the protocols of the SDI communications network. The development of specifications for the SDINet are currently being planned [Cohen 86].

3.3.4 Protocols

Research in network protocols is primarily taking place in universities. C. V. Ramamoorthy at Berkeley [Ramamoorthy 85] is studying protocol analysis and synthesis. Analysis tools are used to verify properties of protocols. Protocol synthesis refers to the automatic generation of protocols. Ramamoorthy is focusing on the generation of reliable and error-recoverable protocols and bases his work on Petri net models.

Raymond Miller at Georgia Tech [GaTech 85] is also developing formal techniques for protocol construction. He is using the general structure of protocols to simplify the automatic analysis and design of new protocols.

Mike Liu at Ohio State University is developing multi-destination and internet protocols for efficient and reliable communication systems. He is also developing automatic or semi-automatic tools for verifying and implementing protocols [Liu 84].

Jie-Yong Juang at Northwestern University [NW 85] is developing protocols for real-time processing on local area networks. Emphasis is being placed on fault-tolerant protocols that maximize throughput while meeting real-time deadlines.

Protocols for real-time processing are also being studied by Mario Gerla, [UCLA 85]. The emphasis here is on the merging of conventional traffic and real-time traffic. Analytical and simulation models for performance evaluation are also being developed.

The analysis of traffic on a local-area network [MIT 83] has determined that the distribution of packet interarrival times is "decidedly non-Poisson". Raj Jain is exploring a more realistic network packet arrival model called "packet trains" in which the interval between related packets has a different distribution from inter-train intervals. Reservation sharing protocols designed to exploit this new model result in more efficient resource utilization.

Additional protocol researchers include:

a. S. Lam, Don Good and Bob Moore at the University of Texas, Bochmann at Universite de Montreal, Shankar at the University of Maryland, and Carl Sunshine at the University of Southern California (automatic generation and verification of protocols)

b. Towsley at University of Massachusetts and Jaffe at IBM T.J. Watson Research Center (multi-destination protocols)

c. Lechowsky at CMU, LeLann at INRIA, and Schwartz, et al at Columbia (real-time protocols).

d. Pursley and Hajek at Illinois, Geraniotis at Maryland, Towsley and Kurose at Massachusetts, and Silvester and Li at University of Southern California, Boorstyn, et al. at New York Polytechnic Institute, (protocols for packet radio networks).

3.3.5 Routing

Routing is the task of finding a (best) path for transmission of data from source to destination in a communications network. In most systems today, tables are maintained indicating one or more routes to travel from point A to point B. These tables may be relatively static, centrally updated, updated by the continuous broadcasting of state information, or updated by information of the status of neighbors only. The ARPANET, for example, regularly floods or broadcasts information on topological changes throughout the network. It is impossible, in a dynamic environment for every node to know the correct network topology and, therefore, for the routing tables to be completely accurate at all times. Any routing algorithm used by the SDI system must deal with these transient inconsistencies as well as the sudden and dramatic changes in the topology expected under battle conditions.

Other techniques not based on system state also exist. One example is the "hot potato" algorithm that sends out any message on the link with the shortest queue, regardless of where it is going. The performance of these routing techniques in a hostile environment has not been determined.

The routing studies at Harris Corp., funded by NRL, focus on survivability. Multiple copies of each message are sent over at least three node-disjoint paths [Althouse 86].

The WIS program developed specifications for the development of routing algorithms for systems having multiple objectives. Real-time constraints, minimum bandwidth, high reliability, and different priorities are addressed. The algorithm will adapt to changes in the

traffic load and the topology of the system. Multiple destination routing as well as the standard single destination routing would be provided.

The Exterior Gateway Protocol (EGP) was developed for routing information between gateways of the DARPA Internet [MIT 83] and is expected to become a DARPA standard. It provides for a restricted form of dispersing updates to routing tables.

The Berkeley Routing Simulator [Ramamoorthy 85] is a flexible tool for studying the performance of routing algorithms. It is highly parameterized and can simulate a wide variety of hierarchical and non-hierarchical algorithms on arbitrary topologies. Currently four routing algorithms have been implemented and studied on this system. Additional studies are planned.

Additional research in routing algorithms includes:

a. The development of routing techniques for large packet radio networks and public data networks, by Leonard Kleinrock [UCLA 85].

b. Design and performance evaluation of routing and flow control procedures by Mario Gerla [UCLA 85].

c. The development of optimal hierarchical routing algorithms by Oliver Ibe, [GA Tech 85].

d. The design of hierarchic routing for large networks, and routing in interconnected networks by Wescott at BBN.

e. The decentralized computation of optimal routing protocols by Gallager and Bertsekas at MIT.

f. The development of optimal and adaptive routing and flow control techniques by Srikanta Kumar [NW 85].

3.3.6 Naming

Naming algorithms are those used to map names of objects to locations or addresses. When objects in a distributed system move from one site to another, the mapping function must be updated appropriately. Current techniques provide this function for relatively stable distributed environments. However, in the SDI system, as clusters of the network are destroyed or temporarily disconnected and reconnected, large-scale relocation of processes and resources has to take place. Current techniques must be analyzed to determine whether or not existing naming algorithms will apply in this dynamic environment.

3.3.7 Topology Updating

The basic structure of the communications network of the SDI is a hierarchical one. The grouping of nodes of a network into clusters is a common technique. However, the dynamic requirements of the SDI system require the ability to dynamically recluster. C. V. Ramamoorthy [Ramamoorthy 86] has addressed this need in his research and has proposed a clustering algorithm based on a modified form of B-Trees. This technique results in balanced clusters that can be dynamically rebalanced and support efficient routing and naming algorithms.

Ballistic Missile Defense at Huntsville is also initiating research into algorithms for dynamic reconfiguration of a network. The ability to provide flow control and routing are key issues being addressed [BMDSOW 86].

Past work on packet radio networks has focused on the problem of reconfigurable networks. While this work has not necessarily addressed hierarichal networks, it is a source of experience that should be used by the SDI developers.

3.3.8 Flow Control

A flow control algorithm monitors traffic on a network and forces reduction of traffic when necessary to prevent congestion. Most systems currently use "windows", a restriction on the number of unacknowledged messages or packets that may exist between two nodes on a network. However, in satellite systems, with potentially large propagation delays, large windows are needed to prevent this flow control method from degrading performance under heavy traffic. The routing studies at Harris address flow control by decreasing the number of copies of each message that are sent [Althouse 86].

A current area of research is the development of decentralized flow control mechanisms. Work is being conducted at MIT by Gallager and at the IBM T.J. Watson Research Center by Jaffe.

3.3.9 Deadlock and Livelock

If a process is blocked, waiting for an event that will never occur, it is said to be deadlocked. If a process is blocked, waiting for an event that may not occur for an arbitrarily long amount of time it is said to be livelocked. Both of these conditions can easily develop in a dynamic network, as processes can be waiting on messages from other processes that have been destroyed or temporarily disconnected. Techniques for the prevention and detection of deadlock currently exist and must be included in the design of the SDI communications network. Scheduling techniques to prevent livelock exist and must also be included in the SDI's design.

3.3.10 Clock Synchronization

The synchronization of clocks in a geographically distributed network is required to enable the distributed system to arrive at a consistent view. An example of this requirement is provided in [Ramamoorthy 85]. If a missile is detected at time 12:45 by detector 2 and at time 12:42 by detector 1, it would be assumed that the missile was heading in the direction from detector 2 to 1. However, if the clock of detector 1 is 0:06 behind the clock of detector 2, the missile is actually heading in the opposite direction.

Clock synchronization is also required for the communication security technique known as "frequency hopping". In this technique, signals transmitted at a given frequency are rapidly shifted to another frequency. The MILSTAR system determines the frequency-hopping pattern from a code at the start of each message. For the receiver to be listening to the correct frequency at the correct time, the clocks of the sender and receiver must be synchronized.

C. V. Ramamoorthy, [Ramamoorthy 85], has developed two groups of clock synchronization algorithms. The first group is to be used in environments without malicious faults while the second is for environment with malicious faults. Both groups assume that clock drifts are within known bound. Performance analysis of these algorithms is continuing and the algorithms are also being modified to apply to hierarchical networks.

Jennifer Lundelius and Nancy Lynch [MIT 83] are also studying the clock synchronization problem. The have obtained a lower bound on "the closeness with which clocks can be synchronized in the presence of uncertainty of message delay."

3.3.11 Distributed Consensus

Distributed Consensus is the problem of reaching agreement among all nodes in a distributed system in which different processor have different clocks and information, possibly conflicting, about the state of the system. Lynch, Dwork, and Stockmeyer have recently developed an algorithm to reach consensus in a partially synchronous environment, [MIT 83].

T.V. Lakshman and Ashok Agrawala have presented an algorithm to achieve a consensus in in two rounds of O(n*sqrt(n)) messages. Previous techniques have required one round of O(n**2) messages [Lakshman 86].

3.3.12 Encoding

A variety of data encoding techniques exist that adequately address the needs of the SDI. The distribution of keys in a dynamically reconfigurable distributed network for the decoding of messages is currently a research issue.

Blacker, supported by the National Security Agency is a multi-level secure packet switching system. This system uses an end-to-end, real-time encryption technique and may be applicable to the SDI system.

3.4 Recommendations

3.4.1 Introduction

A variety of techniques currently exist for performing the required functions of a network communications system, (e.g., protocol support, routing, naming, flow control, synchronization, etc.) The ability of these techniques to meet the reliability, security, reconfigurability, and real-time performance requirements of the SDI system must be determined. New techniques or modifications to existing techniques must be developed where current algorithms are found not to provide these characteristics.

3.4.2 Funding Recommendations

"The importance of a unified set of consistent, compatible protocols can not be overstated," [Eastport 85]. A commitment must be made to a protocol standard to achieve the desired interoperability among distributed computer communication networks. The security techniques to be used by the SDI are likely to have an effect on the design of the low level protocols and therefore must be taken into consideration.

> Recommendation 1: An open and extensible protocol standard that meets the reliability, security, reconfigurability, and real-time performance requirements of the SDI must be developed early.

Algorithms for naming, routing, topology updating, flow control, and clock synchronization are key to the success of the SDI system. Current algorithms have been developed under the assumptions that (1) the network will be topologically fixed or require minor changes due to component failures or repairs and (2) communication demands will

fluctuate slowly. The SDI environment does not fit these assumptions. Major emphasis must be placed on the real-time constraints of the SDI network. Active, (not passive), fault-tolerance and fault-detection techniques must be used throughout the design.

Recommendation 2: New algorithms for naming, routing, topology updating, flow control, and clock synchronization must be developed to meet the reliability, security, reconfigurability and real-time performance requirements of the SDI.

The security requirements of the deployed SDI system are severe and unique. Existing systems do not address the needs of the SDI.

Recommendation 3: Appropriate models for security of computer communication networks must be developed. These models must be available early in the design of the SDI system as they will be a basis for many design decisions.

The automatic generation techniques will provide a means of quickly developing verifiable, reliable, secure, high-performance protocols. The strict requirements of the SDI system indicate that these techniques should be used.

Recommendation 4: The SDIO should fund research in the area of automatic generation of reliable, secure, high-performance protocol software.

The theoretical ability to solve the individual functional needs of the SDI system do not necessarily result in the ability to build a computer communications network that will perform appropriately. The integration of solutions to individual problems must be addressed.

Recommendation 5: Prototype computer communications networks must be built. These systems should be integrated with the prototype distributed operating systems being developed for the SDI.

3.5 References

[Althouse 86] Althouse, Ed., Personal Correspondence (August 1986).

[BMDSOW 86] "Network Reconfiguration Algorithms," *Scope of Work, U.S. Army Strategic Defense Command,* BMD, Huntsville, Alabama, (February 10, 1986).

[Cohen 86] Cohen, Danny, Personal Conversation at IDA, (April 21, 1986).

[Eastport 85] Eastport Study Group, *A Report to the Director, Strategic Defense Initiative Organization*, Summer Study, (1985).

[GaTech 85] Georgia Institue of Technology, *Information and Computer Science*, (1985).

[IEEE 83] IEEE Computer Society, *IEEE Standard Glossary of Software Engineering Terminology*, ANSI/IEEE Standard 729-1983, (1983).

[Lakshman 86] Lakshman, T. V. and Agrawala, A. K. "Efficient Decentralized Consensus Protocols," *IEEE Transactions on Software Engineering* (May 1986), pp. 600-607.

[Liu 84] Liu, Ming T., "Computer Communication Protocols for Advanced Communication Systems," *Second Semi-Annual Report, Research and Development Technical Report CECOM-83-K542-2*, (1984).

[MIT 83] Massachusetts Institute of Technology, *Laboratory for Computer Science Progress Report*, (July 1983-June 1984).

[NW 85] Northwestern University, *Research Activities*, Department of Electrical Engineering and Computer Science, (December 1985).

[Ramamoorthy 85] Ramamoorthy, C. V., "Requirements for a Distributed Operating System," Slides from a presentation (1985).

[Ricci 86] Ricci, Fred J., and Schutzer, Daniel, *U.S. Military Communications: A C3I Force Multiplier*, Computer Science Press, (1986).

[Tanenbaum 81] Tanenbaum, Andrew S., "Network Protocols," *ACM Computing Surveys* (December 1981), pp. 453-489.

[TCSEC 83] U.S. Department of Defense - Computer Security Center, *Trusted Computer Systems Evaluation Criteria*, (1983).

[TNEC 85] U.S. Department of Defense - Computer Security Center, *Trusted Network Evaluation Criteria*, Draft, (1985).

[Turner 86] Turner, Robert D., Yanni, David, and Freitag, Harlow, *Functional Analysis of SDI Battle Management Processes*, IDA, (1986).

[UCLA 85] University of California, Los Angeles, *The UCLA Computer Science Department Quarterly*, (1985).

[WIS 86] Institute for Defense Analyses, *Software Requirements for WIS Network Protocol Prototypes*, IDA, P-1893, Volume IX, Alexandria, VA, 1986.

SECTION B4

Distributed Operating Systems

Prepared by Cathy Jo Linn

Topics covered in Section B4:

4.0 Distributed Operating Systems

4.1 Introduction
4.1.1 Purpose and Scope
4.1.2 Background
4.1.3 Relationship to Other Foundation Areas
4.1.4 Organization of Section

4.2 SDI Requirements
4.2.1 Introduction
4.2.2 Platform System Requirements
4.2.3 Platform Management Requirements
4.2.4 Development and Maintenance Requirements
4.2.5 Two Approaches to Meeting SDI Operating System Needs
4.2.6 Other Required Characteristics
4.2.7 Functional Requirements
4.2.8 Summary of SDI Requirements

4.3 Current Status
4.3.1 Introduction
4.3.2 Modeling and Analysis
4.3.3 Scheduling and Resource Allocation
4.3.4 Access Control and Auditing
4.3.5 Distributed Operating Systems Projects
4.3.6 Distributed Testbeds
4.3.7 Summary of Current Status

4.4 Recommendations
4.4.1 Introduction
4.4.2 Modeling and Analysis
4.4.3 Scheduling and Resource Allocation
4.4.4 Access Control and Auditing
4.4.5 Distributed Operating Systems Development
4.4.6 Distributed Testbeds
4.4.7 Summary of Recommendations

4.5 References

4.0 DISTRIBUTED OPERATING SYSTEMS

4.1 Introduction

4.1.1 Purpose and Scope

The purpose of this section is twofold. First, a comparison of the current state-of-the-art and state-of-practice in distributed operating systems to the needs of the SDI is made. Second, research and development projects in distributed operating systems are recommended. These recommendations are aimed at aiding in (1) making the full scale engineering development decision of the early 1990s, and (2) obtaining the technology needed to develop the SDI system.

4.1.2 Background

In this section basic operating systems concepts and terminology are reviewed. These concepts are then used in later sections in identifying the distributed operating systems needs of the SDI.

The operating system of a computer supports the applications programmer by supplying basic functions needed in all programs. It provides easy access to the I/O devices of the system and supports the long-term storage of data by providing a filing system.

A multi-programmed system is one that provides the interleaved execution of two or more programs on a single processor. In this case, the operating system must coordinate the use of the hardware resouces among the running user programs. The coordination of the use of processor time is called scheduling. The coordination of all other resources, (memory, file storage, I/O devices, etc.), is called resource allocation. A multi-programmed system must protect each user from other users that are executing during the same time period. In addition, many multi-programmed systems provide support for interprocess communication (IPC), and sharing of data between cooperating processes.

In computer systems consisting of more than one processor and more than one memory unit, the problem of coordinating the use of these hardware resouces is further complicated. This resource allocation/scheduling function can be performed in one process of the system (centralized), or by a set of processes running on a set of processors in the system (distributed). Centralized techniques are simpler and similiar to techniques used in single processor systems. For this reason they are often used in tightly-coupled, multi-processor systems. However, the resource allocator/scheduler process itself is a system resource. The single instance of this resource often creates a bottleneck in the system and results in poor system reliability. In addition, this technique requires total system state information to reside in one location. This is often very difficult in loosely-coupled, wide-area networks of processors.

Distributed techniques overcome these problems. The task of resource allocation/scheduling is shared among many processes. This alleviates the bottleneck, provides better system reliability (if one resource allocator/scheduler has failed, the others can continue to function), and does not require the state information on the entire system to reside in one location. However, the coordination that is required between this cooperating set of resource allocators/schedulers adds a level of complexity to the system.

A complete introduction to operating systems is given in [Peterson 83]. Additional terms will be defined in this section as they are used.

4.1.3 Relationship to Other Foundation Areas

The operating system is most closely related to the foundation areas of (1) database management, (2) communication, and (3) fault tolerance and reliability. These relationships are briefly discussed below.

There is evidence suggesting that implementing database management systems (DBMSs) on top of existing operating systems can lead to significant performance problems. Some researchers [Stonebraker 81] advocate that the operating system be supported by the DBMS rather than the reverse, as is customary today. DBMS requirements are discussed in Appendix B, Section 5, Distributed Data Management Systems.

A primary function of the operating system(s) for the SDI is to provide support for network communication. The software requirements for network communication, i.e., protocols, object naming, message routing, and clock synchronization, are discussed in Appendix B, Section 3, Network Communications.

The SDI system will operate in a hostile environment. The overall system must survive and perform reliably even if individual systems or entire clusters of the network are lost, temporarily or permanently The recovery required in the SDI system, however, can be approximate in the sense that not all state information will be recovered. In fact, perfect recovery is probably not feasible and the right degree of approximate recovery needs to be determined. The system must be designed to operate in the context of errors and inaccurate information.

The general techniques for designing reliable fault-tolerant software are discussed in Section 11, Appendix B. Examples of fault-tolerant distributed operating systems are reviewed here, in Section 4.3.5.

4.1.4 Organization of Section 4.0

Section 4.2 discusses the operating system needs of the SDI including the general structure and functions of the operating system(s). Section 4.3 reviews the current status of important areas in operating systems research and development. Section 4.4 makes funding recommendations for further work in operating system design.

4.2 SDI Requirements

4.2.1 Introduction

The SDI operating system(s) must support three different environments. These environments are (1) the platform systems, (2) the platform management system, and (3) the development and maintenance system. While the design of the software needed for each of these environments can not be done in isolation, a separate analysis of each along with an analysis of their interactions will aid in determining all the issues that need to be addressed. A brief description of these three environments and the resulting requirements are given in Sections 4.2.2 through 4.2.4.

Section 4.2.5 discusses alternative approaches to designing a set or family of operating systems meeting the needs of the SDI. General characteristics of the operating system(s) that apply across all of the environments are discussed in Section 4.2.6. Section 4.2.7

specifies functions that are generic across all operating system(s) (even non-SDI systems), emphasizing the impact of the SDI environments on these functions.

4.2.2 Platform System Requirements

The platform systems will be geographically distributed, potentially through space as well as on earth. The individual platforms will be linked by an inter-platform communications network. The operating system must support communication both among platforms and between the platform management system and the platforms. Each individual platform functions autonomously in many ways. It determines both its own scheduling and its local resource allocation. Global system commands, however, such as "fire weapons" or "reprogram", are initiated from the platform management system via the network. In addition, the network is required to transfer data and possibly processes from one platform to another. The software requirements of network communications are discussed in Appendix B, Section 3.

Many requirements are posed by the platform system. It must support real-time processing with guaranteed deadlines. The system must be fault-tolerant, reliable and robust. Recovery techniques must allow rapid system restarting without high overhead and without completely accurate information. The system will be continually upgraded. Thus, the system must support reprogramming the deployed processors remotely while attempting to minimize system unavailability. The platform systems communicate both with the platform management system and with the development and maintenance system; but they need not interface directly with people.

The platform system itself requires a local network to connect a variety of processors and multi-processors. Each of these individual systems contains specialized control programs that are sometimes referred to as operating systems. For the purpose of this paper, however, the platform operating system is defined as the set of all software controlling the processors and other computing/communication resources of the platform.

Centralized control with replication among the processors on a single platform for fault tolerance provides an simple and reliable system in this environment. Static scheduling and/or fully dedicated processors can be employed to reduce the complexity of this system. In this way, unwanted contention for processing and memory may be eliminated and error-causing side effects may be minimized.

4.2.3 Platform Management Requirements

The platform management system is centralized around a small number of control sites. Appropriate personnel are allowed to monitor the current global state of the system and to authorize platforms to perform appropriate actions. In spite of the centralized nature of this function, fault tolerance must be ensured. Real-time access to large geographically distributed and replicated databases must be provided, and real-time deadlines must be guaranteed for a variety of processing tasks. The user interface must be designed so that large amounts of data are presented appropriately for varying stress levels; additionally, security must be maintained although multiple levels of security may not be needed.

4.2.4 Development and Maintenance Requirements

The software engineering environment for the development and maintenance of the SDI will be continually evolving. Standards for interfacing to these changing environments (such as the Common APSE Interface Set (CAIS) to support Ada environments) are

currently being developed. These, in turn, must be supported by the operating system for the development and maintenance environment.

The security system in the development and maintenance environment must support both multi-level and compartmented ("need to know") bases for restricting access. Auditing facilities must also be provided. The scheduling of jobs on the development system might be straightforward (e.g., every programmer has a dedicated system), or a sophisticated dynamic distributed scheduler might be desired.

4.2.5 Two Approaches to Meeting SDI Operating System Needs

The three environments of the SDI present a variety of needs for operating systems support; sometimes, these needs are conflicting. There are two basic approaches to dealing with these conflicts. One approach is first to develop a standard set of interfaces between the three environments and then to develop a set of three specialized operating systems. In this way real-time performance requirements of the platforms and platform management systems can greatly influence the overall design of their operating systems. At the same time the need for a state-of-the-art software engineering environment can be one of the driving forces behind the design of the development and maintenance operating system.

An alternative approach that is intuitively more attractive is to develop a modular operating system that can be customized to meet the needs of the different environments. A multi-level security module and a load balancing scheduler can be included for the development and maintenance system while these same modules can be left out or replaced with simpler more streamlined modules in the other two environments. It is not clear, however, that a design can be developed in which functions such as security can be completely captured in a module. Also, it is not known if an appropriately general and modular design can be developed to satisfy the performance requirements of the SDI. Thus, it is more likely that the variety of needs can be more easily met in the near term by employing the first approach. Further analysis is needed to make the final design decision.

4.2.6 Other Required Characteristics

Several characteristics are required for all parts of the SDI operating system.

a. The operating system must be very modular. The evolution and maintenance of the SDI system require that upgrades and replacements of parts of the system be handled easily. Therefore, this modularity must relate not only to the logical functions of the operating system, but also to the physical modularity of the SDI system architecture. In this way, physical partitions of the system can be temporarily deactivated while new software is installed. In addition, a modular design also lends itself to hardware assist. Individual functions, such as scheduling, can be off-loaded to a co-processor, if they are written in a modular fashion.

b. The operating system must be portable so that as the hardware is changed, as firmware is modified or as new platforms are added, the operating system can be more easily ported.

c. Interfaces between different modules of the operating system and between the operating system and application programs must be designed so as to minimize potential programming errors and to prevent propagation of errors from one module to another.

d. It is not likely that the entire operating system will be formally verifiable. It is desirable, however, that critical parts of the system undergo formal verification.

4.2.7 Functional Requirements

The functions required by any distributed operating system include:

a. Scheduling and resource allocation

b. Access control and auditing

c. Process management

d. Transaction management

e. Interprocess communication

f. I/O drivers

g. Support for installation of upgrades

h. Database support

i. Network communication support

These functions and the specific needs of the SDI are discussed below.

a. Scheduling and Resource Allocation

The scheduling and resource allocation needs of the SDI operating system(s) may potentially require both a static scheduler/allocator and a dynamic scheduler/allocator. In a static scheduler/allocator, the assignments of all tasks to processors and time slots and resources to tasks are fixed prior to the initiation of any task. Therefore, real-time deadlines such as those of the platform systems can be guaranteed if maximum computation loads are known. A dynamic scheduler/allocator uses techniques such as load balancing to determine these assignments based on the current state of the distributed system. These techniques are very useful when the mix of tasks on the system is constantly changing and when obtaining full use of the computing resources is more important than meeting any particular deadline.

In addition, the scheduler/allocators, whether static or dynamic, must take into account the constantly changing network architecture. A scheduler for a platform system must utilize fault-tolerance techniques to allow a task to be reassigned if a processor fails. Tasks must be moved from one platform to another in case an entire platform is lost. Resources must be reclaimed from tasks that no longer exist. In addition, the system must be able to handle temporary loss of platforms due to radiation.

Deadlock and livelock avoidance techniques must be used in the design of the overall system. If a task is blocked, waiting for an event that will never occur or a resource that will never be available, it is said to be deadlocked. If a process is blocked waiting for an event that may not occur or a resource that may not be available for an arbitrarily long amount of time, it is said to be

livelocked. Static scheduling techniques can greatly reduce the deadlock problem for the allocation of processors, but the allocation of resources, such as communication buffers, is an area where deadlock must still be addressed. Schedulers/allocators must ensure that no tasks are delayed due to deadlock or livelock.

b. Access Control and Auditing

The operating system of the SDI must provide a secure base on which to build a secure system. The DoD requires that information be labeled with (1) a sensitivity level (Confidential, Secret, Top Secret) and (2) a set of compartments used to control access on a "need to know" basis (NATO, NASA, or SDI, for example). Clearance for the specified level and authorization for the specified compartments is required to access any information.

The DoD developed criteria to be used to evaluate the degree to which computer systems support both the labeling of information and the auditing of access to information [TCSEC 83]. These evaluation criteria divide computer systems into four divisions. Divisions A, B, and C are further divided into subdivisions, representing smaller increments in the protection facilities provided.

Division D - Minimal protection: Evaluated systems that do not meet the requirements for any higher level are placed in this division.

Division C - Discretionary protection: These systems provide a single security level, access to be permitted at the discretion of the owner, and auditing facilities to monitor user access to information.

Division B- Mandatory protection: These systems provide the policies of Division C, provide mandatory separation of security levels, and document a security model on which the implementation is based. B3 systems demonstrate that a top level design implements this security model.

Division A - Verified protection: These systems provide the policies of Division B and formal verification of the design (A1) or the code (A2).

The current criteria were designed to evaluate centralized computing facilities. The SDI system must have similar criteria for evaluating distributed operating systems. The security needs of the SDI platform systems are limited because there is little direct user access. The remote updating of software/firmware and verification of incoming signals, however, are two unique requirements of these systems.

The SDI platform management system might only need a single level but compartmented system, supporting the current national policy for command decisions. For example, the agreement of several key personnel may be required to initiate a defense action. This system, however, should be verified.

The maintenance and development system must provide a variety of people with access to the software. Therefore, a B3 or A1 system will be required.

Additional security measures must be provided by the network and the DBMS. The DoD is currently developing evaluation criteria for networks [TNEC 85]. A similar effort is being initiated for DBMSs. These issues are discussed in Sections 3 and 5, Appendix B.

c. Process Management

The process management provided by the SDI operating system must provide operations to create, to control, and to destroy processes. In addition, it must isolate non-communicating processes from one another to prevent the propagation of errors.

d. Transaction Management

The transaction manager must provide techniques for coding atomic units of execution. Such a facility may be used for checkpointing and recovery.

e. Interprocess Communication

The SDI operating system(s) should provide a flexible interprocess communication (IPC) technique with relatively low overhead such as pipes. More advanced techniques may be layered on top of this basic system if they are needed.

f. I/O Drivers

I/O drivers are required for a wide variety of peripheral devices envisioned in the SDI environment, possibly including weapons and sensor systems. A modular design is required so that new drivers can easily be installed as the hardware is upgraded.

g. Support for installation of upgrades

The operating systems must provide a general facility for the installation of upgrades to both the application programs of the platform systems and the different parts of the operating systems themselves. The entire system certainly cannot be brought to a halt while upgrades are installed, and the unavailability of any individual portion of a system must not seriously impact SDI's ability to carry out its mission.

h. Database Support

The SDI operating system must be designed with the DBMSs to assure adequate performance for the SDI. The database requirements are discussed in Appendix B, Section 5.

i. Network Communication Support

Network communication must be supported by the operating system. These requirements are discussed in Appendix B, Section 3.

4.2.8 Summary of SDI Requirements

The operating system of the SDI must be fault tolerant, reliable, survivable, correct, and secure, and it must support real-time programming and real-time access to large databases. The three different environments of the SDI emphasize these characteristics differently. It is not known if a single operating system can be designed to meet all these needs, and, in the short term, it is expected that a set of operating systems with standard interfaces will be employed.

4.3 Current Status

4.3.1 Introduction

In this section the subareas of operating systems research of interest to the SDI are reviewed. State-of-the-art research and development projects in each subarea are identified. These subareas include modeling and analysis of distributed systems, distributed scheduling and resource allocation, and distributed access control and auditing.

Next, the integration of solutions to individual problems into total distributed systems is addressed. State-of-practice distributed systems are discussed. State-of-the-art projects in the design and development of distributed operating systems focusing on fault tolerance, real-time programming, or security are reviewed.

Lastly, existing testbeds for distributed systems are surveyed. These provide a means for comparing and analysing techniques to be used in distributed operating systems.

4.3.2 Modeling and Analysis

Adequate formal models of distributed computing are still in the research phase. Such models must address specifications, theoretical limitations, reliability and, most importantly, must accurately predict performance. Several efforts are currently under way to develop models that capture one or more of these issues.

Dr. Wesley Chu at UCLA has developed a task response time model for distributed systems that considers precedence relationships, interprocessor communication, interconnection network delays, module scheduling policy, and assignment of modules to computers, [UCLA 85]. The model has been validated via simulation; this work is currently funded by TRW and the State of California.

Movaghar uses Petri nets for the performance analysis of distributed real-time systems, [Movaghar 84]. This model addresses concurrency, fault tolerance, and degradable performance.

SDC is also funding work in modeling. Their goal is to develop continuous models of distributed systems.

Several academic research groups are attempting to develop useful models of distributed systems. The Georgia Institute of Technology Research Program in Fully Distributed Processing Systems is working in the formal modeling of extremely loosely coupled systems with a high degree of control autonomy [GaTech 85]. The Theory of Distributed Systems Group at MIT is working on the formalization of reliability and time in distributed systems [MIT 83]. A recent dissertation from this group [Stark 85] presented a mathematical model of distributed systems to be used for specification and verification. Two recursive structuring principles have been developed at Newcastle-upon-Tyne University that allow the incorporation of fault tolerance in design to be greatly simplified, [Randell 82].

4.3.3 Scheduling and Resource Allocation

The techniques developed in this area have primarily been labeled as scheduling algorithms. Most of the results, however, are easily mapped from the scheduling of tasks on processors to the scheduling, (allocating), of resources to tasks.

A variety of static assignment and scheduling algorithms for distributed systems have been proposed. These algorithms determine the assignment and scheduling of tasks on processors of a distributed system prior to execution. This assignment is based on known execution time and communication characteristics. Such assumptions are often realistic for dedicated processors such as those in platform systems. A class of algorithms typified by [Stone 77] and [Bokhari 81] does not consider parallelism but minimizes the communication time between processes. Another class of algorithms, [Chow 82, Lint 79, Efe 82], considers both parallel execution and communication time in attempting to provide a schedule to optimize performance.

Dynamic assignment and scheduling techniques that base decisions on the current state of the system generally are not designed to meet real-time needs. Ramamritham and Stankovic have addressed real-time constraints but cannot guarantee deadlines [Ramamritham 84].

SDC is sponsoring a project entitled "Real-Time Control for SDI BM/C3" [Johnson 85]. This system supports dynamic reconfiguration and load balancing while attempting to minimize the overhead of scheduling. They are studying the application of list scheduling techniques to a real-time system but do not include hard deadlines.

RADC is funding an effort to develop resource management techniques for controlling location, execution, movement, and access to SDI processing functions.

NASA is supporting research [Liestman 80] that addresses hard deadlines in periodic real-time systems. The proposed deadline mechanism trades off accuracy for timing precision by supplying two algorithms for each service. The primary (accurate) algorithm is used if it can be completed by the deadline; otherwise the secondary (adequate and fast) algorithm is used. In this way, the scheduling algorithm guarantees scheduling deadlines.

4.3.4 Access Control and Auditing

The state-of-practice in access control and auditing techniques is represented by several systems currently used by the government that have been rated at the A1 level. The state-of-the-art is harder to characterize. Neither adequate evaluation criteria for distributed systems nor prototype secure systems for real-time processing in distributed environments have been develped to date. Accordingly, this section focuses first on existing state-of-practice secure systems and then on research projects that address state-of-the-art approaches to particular security issues.

The design of the Kernelized Secure Operating System (KSOS) was developed by the Ford Aerospace and Communications Corporation (FACC) and Logicon and funded by NSA, DARPA, and the Navy [Landwehr 83]. This system was verified and rated at the A1 level, but performed poorly.

The Honeywell Secure Communication Processor (SCOMP) also received an A1 rating. This system used a hardware box called the SPM, security protection module, to monitor transfers on the bus without CPU interference resulting in faster performance than KSOS

achieves. This work was funded by Honeywell, NSA, DARPA, Defense Communications Agency, and the Navy. The system is currently used by several government agencies [Fraim 83].

The Message Flow Modulator (MFM) system is based on a model of message filtering. It was developed at the University of Texas using Gypsy and the Gypsy tools. Both the design and the code have been verified. In spite of this state-of-the-art verification, the MFM system is only rated at the C2 level. This is because the message filtering model does not include the multi-level and compartmented labeling facilities required for the B level. This work was funded and is currently being used by the Navy [Landwehr 83].

The Probably Secure Operating System (PSOS) is a capability-based system. The design was formally specified and verified but an implementation was not initiated. The design was developed by FACC and Honeywell and was funded by NSA [Landwehr 83].

RADC is currently funding a variety of projects that address the specific security needs of SDI [RADC 86]. These include (1) security of remote software updates (Rockwell), (2) guarded computing or the use of a dedicated computer to verify the behavior of a second computer (Intelligent Software), (3) the use of AI techniques to detect penetration attempts (TRW), and (4) the use of formal software specification and verification to develop a trusted and secure system for the SDI (RCA).

4.3.5 Distributed Operating Systems Projects

The state-of-practice in distributed operating systems is represented by a variety of systems that were developed in the 1980's. A complete list and categorization of these systems is given by Stankovic [Stankovic 85]. Many of these systems have been in use for more than three years and are very stable. They are, however, general purpose systems; and, as such, they do not address the specific issues of fault tolerance, reliability, reconfigurability, and real-time programming needed for the SDI.

The state-of-the-art in distributed systems is represented by a variety of ongoing projects. The status of these projects ranges from paper design to currently running. In this section, those projects that consider one or more of the specific needs of SDI are reviewed.

The Eden Project [Almes 85] is attempting to design, build, and test a distributed system that supports both personal computing and a high degree of integration. In Eden distributed applications are built as collections of objects accessed by capabilities. It is expected that the modularity afforded by the object based style of programming will provide an environment appropriate for the construction of distributed applications. Eden supports limited fault tolerence through checkpointing of objects. The Eden system is under development at the University of Washington in Seattle and is supported in part by the National Science Foundation.

The V Kernel [Cheriton 84] has been running at Stanford University since 1982. Its development was motivated by the wide availabilty of powerful personal workstations. The V Kernel was designed as a "sofware chassis" into which services could be "plugged". These services include facilities often embedded in other operating systems, such as a file server or resource management system, in addition to the more traditional application level services. This modular approach is a key advantage of the V Kernel.

The Archons project at Carnegie-Mellon University (CMU) is funded by several DoD agencies and industrial concerns. This system differs from most other distributed systems in that it attempts to provide a system-wide, logically centralized operating system in a

distributed environment. Probabilistic models of the whole system state are maintained at each node. This information is used to help meet the demands for resources at critical times. In this way, the system is better able to meet real-time needs at exactly the times most real-time systems fail---that is, under heavy load. Pieces of the kernel are scheduled to be functional in 1986 and the entire prototype ArchOS kernel is scheduled to be functional by Fall, 1988 [CMU 85].

The MACH is an operating system currently running at CMU [Baron 85]. It is a UNIX-like multi-processing system for highly parallel computing to meet the needs of real-time processing. Due to previous work with Accent, a communication-oriented operating system also developed at CMU, the MACH system was designed for network extensibility. The combination of highly parallel computing and a distributed environment directly addresses the needs of the SDI.

DARPA supported the original design and development of MACH and is continuing to support the current testing of the distributed system. Other research programs being supported by DARPA are being redirected to result in software that will run on the MACH system. DARPA expects the MACH system to become a standard for distributed operating systems [Squires 86].

Additional work on the MACH is addressing the DoD security requirements for operating systems [TCSEC 83]. It is expected that the MACH system will be secure to the B3 level.

The Advanced Information Processing System (AIPS), currently under development at Draper Labs, comprised fault and damage-tolerant processors, a fault and damage-tolerant network, and a fault-tolerant power distribution system. The system is managed hierarchically, with a System Manager allocating functions to the individual processors and the individual processors handling local functions independently. The system allows functions to migrate from one processor to another in case of a reconfiguration due to a processor loss, or a major change in the system load [Brock 86].

The PHOENIX Project, at the University of Virginia, is researching the problems associated with (1) developing a high-performance UNIX for embedded applications with real-time response requirements, (2) modifying an operating system and applications remotely without halting the computer, and (3) recovering after power failure or crash. Currently a high-performance version of UNIX, written in Modula, is being used to evaluate techniques that address these problems. This project is supported by the Army Research Office [Cook 85a].

NASA is supporting the Embedded Operating System Project at the University of Illinois [Campbell 85]. This work is directed at the development of real-time embedded operating systems for aerospace applications. Real-time scheduling and fault tolerance in distributed systems are major focuses of this project. Tools are currently being developed to aid in the construction of a family of real-time systems.

The WIS Operating System, funded by the WIS Joint Project Management Office, is designed to aid in the modernization of the WWMCCS. As such, it is designed for an environment very similar to the SDI BM/C3I [IDA 86]. It provides a minimal kernel to optimize local area network IPC and provide fast context switches. It provides clearly delineated address spaces, basic mandatory access control, and communication control to help support security in the kernel. Outside the kernel, security is supported by "alias" processes and an authentication agent. Its multi-level, modular design is suited for evolutionary change. A prototype, written in Ada, is planned but not yet funded.

The Swift Operating System Project at MIT was motivated by major performance problems with protocol software, [MIT 83]. It attempts to provide the appropriate support for highly interactive parallel software through the use of a new programming style allowing a more flexible flow of control. In addition, a real-time scheduler is included to meet the needs of network protocols and other applications requiring a large number of small computations. The Swift Operating System has been ported from a VAX to a 68000 processor and modifications to improve performance are currently being explored.

The Cronus general purpose distributed operating system is being extended in projects funded by RADC [RADC 86]. Cronus is an object-oriented system designed to support distributed application development [Gurwitz 86]. Improvements to the monitoring and control components, interprocess communication function, resource management function and network interface modules are being made by Bolt Beranek and Newman. A seperate project will provide extensions to support a multicluster configuration, specialized machine architectures, database applications, multi-host transaction management, global resource management and distributed system software development tools.

The ISIS system is an extension of a conventional distributed operating system to support fault tolerance. It insulates the application programmers from the details of fault tolerant programming. This system is currently running at Cornell and current work is aimed at integrating the ISIS more fully into the operating system. The project is supported by DARPA and NSF [Birman 85].

4.3.6 Distributed Testbeds

A distributed systems testbed provides a means of testing and comparing various techniques of scheduling, resource allocation, IPC, etc. While the individual distributed operating systems projects provide some facility for testing, a more general and parameterizable testbed is required to appropriately analyze and evaluate operating systems techniques.

The CRYSTAL Project at the University of Wisconsin provides a software partitionable multi-computer for use as a research tool. The software kernel allows the hardware to be logically partitioned for independent, simultaneous use by different researchers. This project is funded by NSF and is currently being used to study a variety of distributed algorithms [Wis 85].

The Distributed Ada Run-Time Package (DARP) is a reconfigurable, distributed system. It provides researchers with the ability to (1) assess the dynamic state of the system, (2) specify events to be recorded, (3) control the allocation of tasks and data to processors, and (4) dynamically reconfigure the system. The project is funded by DARPA and is part of a larger project to produce an Ada language system for a distributed network of computers [Fisher 86].

The StarLite software prototyping environment is currently being used at the University of Virginia. This environment suports the creation of concurrent and distributed applications and has been used to develop systems software "ranging from disk scheduling and operating systems construction to distributed file servers and network routing" [Cook 85b]. This work is currently being funded by the Office of Naval Research, Burroughs Corporation, Modula Corporation, and the Virginia Center for Innovative Technology.

4.3.7 Summary of Current Status

The research areas of (1) modeling and analysis of distributed systems, (2) scheduling/resource allocation, and (3) access control and auditing do not yet address the specific needs of the SDI. Testbeds are not yet widely available and have held back research in these areas.

Several of the development projects currently under way address some of the concerns of the SDI; but no existing project considers the total problem.

4.4 Recommendations

4.4.1 Introduction

This section makes recommendations for the funding of research and development projects related to the operating system needs of the SDI. The analysis of current research projects given in this section are the result of a preliminary study. Some of the recommendations are general and do not identify any particular candidates for funding. In the cases where candidates are identified, further analysis of their current projects, including trips to interact with the researchers, must be undertaken to assure that the opinions expressed in this document are accurate.

4.4.2 Modeling and Analysis

A general theory of distributed systems that can be used in modeling and predicting performance in fault tolerant, real-time distributed systems will probably not be developed in the short term. Funding directed to this area is more likely to result in meeting long term needs. Theoretical projects inherently require less funding than development projects. NSF has traditionally been able to support work of this nature.

> Recommendation 1: The area of distributed modeling and analysis should be monitored to ensure that funding at the current level continues in spite of any budget cuts.

In the short term, additional funding should be directed at theoretical studies that are closely linked to the prototype distributed systems. In this way the theoreticians will learn more about real systems and what models are useful, while lending their insights to the design of the prototypes. (See Recommendation 4)

4.4.3 Scheduling and Resource Allocation

The combination of assist hardware and simple real-time schedulers will address the short-term needs of the SDI.

> Recommendation 2: Assist hardware (in the form of surplus computing power and resources) should be used to reduce software complexity. A simpler and significantly more reliable system will be obtained if 70-80% hardware efficiency is required rather than 100%.

The development of simple real-time schedulers/allocators (based on current projects or new projects) should be funded. These projects, however, should be tightly coupled with the development of the prototype distributed operating systems and should (1) assume massive computing resources under normal circumstances, and (2) address the issue of prioritizing of deadlines for situations that arise when many platforms or processors have been disabled. (See Recommendation 4)

Dynamic assignment and scheduling research is progressing adequately without large government grants. However, most of the current work consist of theoretical or simulation studies. Implementations on real systems are rare.

Recommendation 3: Dynamic assignment and scheduling research should be monitored to ensure that current work continues.

The implementation and analysis of dynamic techniques for scheduling and resource allocation should be funded. These projects, however, should be tightly coupled with the development of the prototype distributed operating systems and analysed under realistic workloads. (See Recommendation 4)

4.4.4 Access Control and Auditing

There is currently a great deal of research in the area of access control and auditing. This work provides a foundation for solving the specific problems of the SDI. Projects for the development of distributed real-time secure kernels should be initiated. These projects should be integrated with the development of the prototype distributed operating systems and should address the problem of secure remote updates. Performance requirements for these systems should be obtained from the architectural studies. (See Recommendation 4)

4.4.5 Distributed Operating Systems Development

The majority of short-term funding for distributed operating systems should be used to augment and accelerate current projects for the development of fault-tolerant, real-time, distributed system prototypes. In addition to the specific research projects recommended above, the research into reliablity and rapid and approximate recovery techniques discussed in Section 11 of this appendix must be integrated with these prototype projects.

Projects directed specifically at the environment of BM/C3 and utilizing the current work as a bases should yield workable prototypes in the short term. These projects should be undertaken by researchers with substantial experience in developing distributed systems. While the number of such researchers is quite limited, multiple prototypes will increase the knowledge that is gained and enlarge the base of experience that will be needed for full-scale development. Research projects on "pieces" of an operating system (schedulers, remote updating, performance prediction, etc.) should be closely linked with these prototype development projects. DARPA's support of a UNIX-like distributed operating system standard, along with early reports of commercial UNIX-like distributed operating systems under development, indicate that MACH is one potential starting point for a prototype SDI operating system. The WIS operating systems design should also be carefully considered. At least three of these prototypes should be written in Ada and conform to either a CAIS-like or UNIX-like standard.

Recommendation 4: The development of four prototype fault-tolerant, real-time, secure distributed operating systems should be funded. At least one should be based on MACH. Past (successful) experience in the development of distributed systems should be a primary consideration in the awarding of these contracts. Proposals should indicate how the development of the prototypes will be integrated with projects on real-time scheduling/allocating, dynamic scheduling/allocating, access control, remote updating, rapid, approximate recovery, and modeling.

4.4.6 Distributed Testbeds

While the prototype developments will yield short-term results and aid in the full-deployment decision, intermediate to long-term research results require widespread and convenient means of experimentation.

Recommendation 5: Multicomputer systems with state-of-the-art operating systems and programming language environments for straightforward parallel programming should be placed in at a large number of research sites. Research sites that already have multicomputer systems should be provided access to the same state-of-the-art operating systems and programming language environments. This should be coordinated with the development of additional parallel processing testbeds recommended in Appendix B, Section 8.

Recommendation 6: Software developed by the prototype operating systems projects should be widely distributed.

4.4.7 Summary of Recommendations

The SDIO should:

a. Monitor the work in:

   (1) Modeling and analysis of distributed systems

   (2) Access control and auditing

   (3) Scheduling/resource allocation to ensure that current funding and progress in the fields continue at the current rate

b. Fund the development of four prototype secure, real-time, fault-tolerant, and reliable operating systems, leveraging off past developmental experience, and incorporating subprojects to research:

   (1) Modeling and analysis of each prototype

   (2) Secure kernals

   (3) Rapid recovery and approximate recovery, and real-time and dynamic scheduling/resource allocation

c. Fund the development of testbeds for distributed systems and widely distribute state-of-the-art software for distributed systems

4.5 References

[Almes 85] Almes, Guy T., Black, Andrew P., Lazowska, Edward D., and Noe, Jerre D., "The Eden System: A Technical Review," *IEEE Transaction on Software Engineering* SE-11, 1 (January 1985) pp. 43-59.

[Baron 85] Baron, Robert V., Rashid, Richard F., Siegel, Ellen H., Tevanian, Avadis Jr., and Young, Michael W., "*MACH: A Multi-Processor-Oriented Operating System and Environment*," Technical Report, Carnegie-Mellon University, Department of Computer Science, (1985).

[Birman 85] Birman, Kenneth P., "Replication and Fault-Tolerance in the ISIS System," *Proceedings of the Tenth ACM Symposium on Operating Systems Principles* (1985), pp. 79-86.

[Bokhari 81] Bokhari, Shahid H., "A Shortest Tree Algorithm for Optimal Assignments Across Space and Time in a Distributed Processor System," *IEEE Transactions on Software Engineering* SE-7, 6, (November 1981), pp. 583-589.

[Brock 86] Brock, Larry, D. and Lala, Jaynarayan, "Advanced Information Processing System: Status Report," *Proceedings of NAECON*, (May 1986).

[Campbell 85] Campbell, Roy H., "The Embedded Operating System Project," *1985 Mid-Year Report*, University of Illinois at Urbana-Champaign, (1985).

[Cheriton 84] Cheriton, David R., "The V Kernel: A Software Base for Distributed Systems," *IEEE Software* (April 1984), pp. 19-42.

[Chow 82] Chow, Timothy C. K. and Abraham, Jacob A. "Load Balancing in Distributed Systems," *IEEE Transactions on Software Engineering* SE-8, 4, (July 1982), pp. 401-412.

[CMU 85] "Faculty Research Guide 1985-1986," Carnegie-Mellon University, (1985).

[Cook 85a] Cook, Robert P. "PHOENIX, A High-Performance UNIX With an Emphasis on Dynamic Modification, Real-time Response and Survivability," submitted to Army Institute for Research in Management, Proposal No. CS-DOD/ARMY-3184-86, (July 1985).

[Cook 85b] Cook, Robert P. and Auletta, Richard J., StarLite, *A Visual Simulation Package for Software Prototyping*," Technical Report, University of Virginia, (1985).

[Efe 82] Efe, K., "Heuristic Models of Task Assignment Scheduling in Distributed Systems," *IEEE Computer* (June 1982), pp. 50-56.

[Fisher 86] Fisher, David A. and Weatherly, Richard M., "Issues in the Design of a Distributed Operating System for Ada," *IEEE Computer* (May 1986), pp.38-47.

[Fraim 83] Fraim, L. J., "SCOMP: A Solution to the MLS Problem," *IEEE Computer* (July 1983), pp. 26-34.

[GaTech 85] Georgia Institue of Technology, "Information and Computer Science," 1985.

[Gurwitz 86] Gurwitz, R., Dean, M., and Schantz R., "Programming Support in the Cronus Distributed Operating System," *Proceedings of the Sixth International Conference on Distributed Computer Systems* (May 1986) pp. 486-493.

[IDA 86] Institute for Defense Analyses, *Ada Foundation Technology, Vol. VI: Software Requirements for WIS Operating System Prototypes*, P-1893, Alexandria, VA, 1986.

[Johnson 85] Johnson, C. D., Briefing for BMD by SDC, (1985).

[Landwehr 83] Landwehr, Carl E., "The Best Available Technologies for Computer Security," *IEEE Computer* (July 1983), pp. 86-100.

[Liestman 80] Liestman, Arthur L., and Campbell, Roy H., *A Fault-Tolerant Scheduling Problem*, Technical Report UIUCDCS-R-80-1010, University of Illinois at Urbana-Champaign, Department of Computer Science, (1980).

[Lint 79] Lint, Bernard, "Communication Issues in Parallel Algorithms and Computers," Ph.D. Dissertation, Computer Science Department, University of Texas, (May 1979).

[MIT 83] Massachusetts Institute of Technology, *Laboratory for Computer Science Progress Report*, (July 1983-June 1984).

[Movaghar 84] Movaghar, A., and Meyer, J.F., "Performability Modeling with Stochastic Activity Networks," *Proceedings of the Real-Time Systems Symposium* (1984), pp. 215-224.

[Peterson 83] Peterson, James L., and Silberschatz, Abraham *Operating System Concepts*, Addison-Wesley Publishing Company, (1983).

[RADC 86] Contract Descriptions by RADC, (1986).

[Ramamritham 84] Ramamritham, K. and Stankovic, John A. "Dyamic Task Scheduling in Distributed Hard Real-Time Systems," *IEEE Software* (July 1984).

[Randell 82] Randell, B. *Structuring of Distributed Computing Systems*, Technical Report, Newcastle upon Tyne University, (1982).

[Rushby and Randell 83] Rushby, J. M., and Randell, B. "A Distributed Secure System," *IEEE Computer* (July 1983), pp. 55-67.

[Squires 86] Squires, Steven, Conversation at Interagency Meeting, (March 1986).

[Stankovick 86] Stankovick, Jack, Roby, Clyde, Cheriton, David, Liu, Mike, Smith, Alan, Salasin, John, Ada Foundation Technology. Vol. VI: Software Requirements for WIS Operating System Prototypes. P-183, 1986.

[Stonebraker 81] Stonebraker, Michael, "Operating System Support for Database Management," *Communications of the ACM* (July 1981), pp. 412-418.

[Stankovic 85] Stakovic, John A., Ramamritham, Krithivasan and Kohler, Walter H. "A Review of Current Research and Critical Issues in Distributed System Software," *Distributed Processing Technical Committe Newsletter* (March 1985), pp. 14-47.

[Stark 85] Stark, Gene, "Foundations of a Theory of Specification for Distributed Systems," Ph.D. Dissertation, Massachusetts Institute of Technnology, (1985).

[Stone 77] Stone, Harold, "Multi-processor Scheduling with the Aid of Network Flow Algorithms," *IEEE Transactions on Software Engineering* SE-3, (January 1977), pp. 85-93.

[TCSEC 83] U.S. Department of Defense, Computer Security Center, "Trusted Computer Systems Evaluation Criteria," (1983).

[TNEC 85] U.S Department of Defense, Computer Security Center, "Trusted Network Evaluation Criteria," Draft, (1985).

[UCLA 85] University of California, Los Angeles, *The UCLA Computer Science Department Quarterly*, (1985).

[Wis 85] University of Wisconsin, Madison, *Computer Sciences Department Research Review*, (September 1985).

SECTION B5

Distributed Data Management Systems

Prepared by Bill Brykczynski

Topics covered in Section B5:

5.0 Distributed Data Management Systems

5.1 Introduction
5.1.1 Purpose
5.1.2 Background
5.1.2.1 Overview of Database Management Systems
5.1.2.2 Database Management Systems Models
5.1.2.3 Data Management System Architecture
5.1.2.4 Distributed Database Management System
5.1.3 Relationship to Other Foundation Areas
5.1.4 Organization of Section 5.0

5.2 SDI Requirements
5.2.1 Platform System Requirements
5.2.2 Platform Management Requirements
5.2.3 Software Engineering Environment Requirements
5.2.4 Characteristics of Data Management Systems
5.2.4.1 Integrity
5.2.4.1.1 Semantic Integrity
5.2.4.1.2 Concurrency Control
5.2.4.1.3 Recovery
5.2.4.2 Modifiability/Modularity
5.2.4.3 Other Characteristics

5.3 Current Status
5.3.1 Platform Environment
5.3.1.1 Issues
5.3.2 Platform Management Environment
5.3.2.1 Issues
5.3.2.1.1 Distribution
5.3.2.1.2 Integrity
5.3.2.1.3 Security
5.3.2.1.4 Modular Design
5.3.2.1.5 Interfaces
5.3.2.1.6 Portability
5.3.3 Software Engineering Environment
5.3.3.1 Issues

5.3.4 Distributed Database Management Systems Projects
5.3.4.1 ENCOMPASS
5.3.4.2 SDD-1
5.3.4.3 R*
5.3.4.4 DDM
5.3.4.5 Distributed-INGRES
5.3.4.6 Multibase

5.4 Recommendations
5.4.1 Platform Environment
5.4.2 Platform Management Environment
5.4.3 Software Engineering Environment
5.4.4 Summary of Recommendations

5.5 References

5.0 DISTRIBUTED DATA MANAGEMENT SYSTEMS

5.1 Introduction

5.1.1 Purpose and Scope

This portion of the appendix examines the data management needs of the SDI. First, a brief introduction to database management systems (DBMSs) and distributed database management systems (DDBMSs) is given. Second, different types of data management systems needed by the SDI, along with their functions and characteristics, are identified. Next, research activities in data management systems of interest to the SDI are reviewed in order to determine the state of the art and state of practice. Finally, funding recommendations are made, with regards to areas that are critical to SDI's needs.

5.1.2 Background

This section of the report presents a brief overview of the purpose, functions, and features of both database management systems and distributed database management systems. Although none of the environments within the SDI will require all of the advances made in DBMS technology, many of them will be based on concepts elaborated in this section.

5.1.2.1 Overview of Database Management Systems

Before the advent of database technology, computer applications typically were not concerned with data redundancy, logical and physical data independence, or the sharing of data with other applications. Each application had its own private files, and was not concerned with other applications' data needs. Consequently, the data used for these applications was often costly (in terms of storage, retrieval, etc.), inconsistent, and difficult to modify. Beginning in the late 1960's, the concept of integrating data into a dedicated subsystem evolved into what is now known as database management.

A database has been defined as "a collection of interrelated data stored together without harmful or unnecessary redundancy to serve multiple applications; the data are stored so that they are independent of programs which use the data; a common and controlled approach is used in adding new data and in modifying and retrieving existing data within the database" [Martin 77]. Several key objectives spurred the development of database technology:

a. Reduction or elimination of harmful data redundancy. Redundancy increases cost, the number of update operations required, and greatly increases the potential for inconsistent information.

b. Logical and physical separation of the application and the data it processes. This separation allows applications to be written independently of how the data is physically stored (e.g., storage hardware and physical storage techniques can be changed without modifying existing applications) or logically stored (e.g., new data items may be stored, or the overall logical structure may be modified, without existing applications having to be modified).

c. Centralized control over data. With a centralized control over data, concepts such as protection, reconstructability, and auditability have become achievable. Today's DBMSs run on a wide variety of computers (e.g., large mainframes to personal computers). Database machines (special purpose systems providing

high performance) that achieve greatly improved performance for some critical tasks through dedicated hardware are also available. This improvement is gained through portions of the software within the database being placed into dedicated hardware and reducing the flow of data to the main processor. Significant performance gains can be achieved in this manner.

5.1.2.2 Database Management System Models

Databases are typically described in terms of which conceptual model is employed to represent how data items are related to each other. A conceptual model provides the user with an environment with which he may interact. There are three main types of conceptual models widely used today in DBMSs. They are the relational model, the hierarchical model, and the network model. Many extensions to these types, such as the entity relational model, the extended relational model, etc. have also been developed.

In the relational model, developed by E. F. Codd [Codd 70], the data are viewed as a collection of non-hierarchical, time-varying relations. Means are provided for describing data with its natural structure. In addition, high level data languages that provide independence between machine representation and data organization are supported. Late binding of applications requirements to the data structures provides this independence and flexibility. Some sophistication is required in optimizers for relational databases to overcome the cost associated with such flexibility. The relational data model has been almost the exclusive model of choice for recent experimental and commercial systems [Vick 84].

The hierarchical data model defines data as a collection of relations that are connected together by logical associations to form trees. These logical associations are directional associations, the segment pointed to by the logical association is referred to as the member segment while the other segment is referred to as the owner segment. The hierarchical model was typically used in the early experimental and commercial DBMSs, and is still being successfully used within many commercial systems. It is effective when one application or "view" of the database dominates.

The network model is a generalized form of the hierarchical model. In the network model, a segment may have more than one owner segment. In general, the segments are grouped as in the hierarchical model, but logical associations may exist between any segments. A network is effective when more than one application view is bound to the database.

5.1.2.3 Data Management System Architecture

In discussing data management system architectures, it is useful to divide the architecture into a sequence of functional layers. Each layer represents a distinct interface between the data management system and other entities (e.g., applications, users, the data management system itself, etc.). Friedman and colleagues have defined a reference model for Ada interfaces to database management systems [Friedman 86].

This reference model identifies eight layers of database interfaces. These eight layers are, from highest to lowest:

a. Heuristic: Access for applications using AI

b. Object: Developmental DBMSs for applications such as VLSI

c. Relation: Typical commercial, off-the-shelf DBMS

d. Tuple Algebra: Minimum layer for communication interface

e. File: File management system

f. Record: Defined Ada I/O

g. OS: Defined Ada I/O

h. Hardware: Bare Machine

The highest level, the heuristic level, incorporates AI technology in performing its processing. Approximations and probabilistic assertions are used within this level to deal with complex and uncertain information, as may be needed for strategic planning and decision-making under conditions of incomplete knowledge. At the lowest level, the operating system level, external storage is viewed as being partitioned into files. Data are accessed by their file addresses, and the operating system is not aware of data context, record structure, or of any interrelationship between files. As can be seen, there is a wide difference between the heuristic level and the operating system level.

Many, if not all, of these layers will be found within the SDI. Different environments will have different layer requirements. For this reason, data management systems must be implemented in a fashion such that the basic framework remains the same, while added functionality simply involves adding modules to the system. This concept of modularity will be discussed in Section 5.2.4.2.

5.1.2.4 Distributed Database Management Systems

A distributed database is a collection of data which are distributed over different computers of a computer network. Each site of the network usually has autonomous processing capability and can perform local applications. Each site also participates in the execution of at least one global application, which requires accessing data at several sites using a communication subsystem [Ceri 84].

There a several features of distributed DBMSs (DDBMSs) that differ significantly from, or are not contained in, centralized DBMSs. These features are:

a. Less emphasis on centralized control. Control over a centralized DBMS lies with the database administrator. In DDBMSs, however, it is rare for a global database administrator to have authority over all local DBMSs. Much control may be left to the individual sites [Wiederhold 84].

b. Distribution transparency. Applications should execute as if the DDBMS were actually a centralized DBMS. Possibly the only different characteristic observed between a DDBMS and a centralized DBMS is a slower execution speed.

c Data redundancy as a desirable feature. In centralized DBMSs, most data redundancy was considered harmful, and thus eliminated. In DDBMSs, however, redundancy potentially increases survivability and application locality [Apers 84].

d Recovery is much more difficult. If a transaction fails in a DDBMS, the steps required to rollback and recover are much more complicated than with centralized DBMSs.

There are several types of design for DDBMSs, with regard to the method of storing and updating data: fully replicated, partially replicated, primary copy updated, and all copies updated. A fully replicated DDBMS keeps copies of the values of all data items located at all nodes. This approach is in practice only used for some selected portions of a distributed database (e.g., for relational tables). Data that are updated regularly are replicated only at the nodes where they are frequently accessed or must be rapidly available. Such partially replicated DDBMSs contain nodes that are much more autonomous. The second alternative regards update. A primary copy mechanism [Minoura and Wiederhold 83] updates a primary copy, often on the local machine, and subsequently updates flow to nodes where the data are replicated. A higher degree of global consistency is obtained by updating all nodes before any further access to the affected data is permitted, although performance and availability under unreliable conditions are greatly reduced. Where nodes operate autonomously, updates occur as determined by local nodes, and any consistency benefit obtained will be minor.

5.1.3 Relationship to Other Foundation Areas

The data management systems for SDI will be closely tied to the foundation areas of operating systems and communications. The relationships among these three areas are discussed below. In addition, the platform system will be closely tied to the BM/C3 system architecture.

A typical database must interact with an operating system in many ways. Some researchers [Stonebraker 81] feel that many of the functions provided by an operating system (e.g., input/output) should be placed within the DBMS itself. Efficiency and portability are two given reasons for such a configuration. The motivation for this view is in part the inadequacy of operating systems optimized towards user and CPU time-sharing for data management. For data management, the critical resources are memory and storage access. These are best managed by a class of systems called Transaction Processing Systems, which schedule active transactions with a priority so that they will release their resources rapidly. Many of the SDI platform databases may incorporate operating system functions for reasons of efficiency. In addition, if a multi-level secure DBMS is required (this is discussed in Section 5.4.5.3), a secure operating system will probably be needed.

A primary focus of the SDI system data management needs will be centered around the operational system requirements (i.e., weapons, sensors, BM/C3, etc.). Given the time critical nature of SDI, the data management systems employed must be highly efficient. This calls for extremely high performance among the data management system and its associated algorithms, and the external communications network.

The software requirements for network communications is described in Section 3, Appendix B. In addition, the exact implementation of the platform system will be closely tied to the BM/C3 architecture that is selected. Many design decisions will be made after the basic architecture has been defined. This report will attempt to discuss the data management aspects of the platform system in the most architecture-generic means as possible.

5.1.4 Organization of Section 5.0

This section of the appendix discusses the data management needs of the SDI. Section 5.2 gives a brief overview of three different environments that will exist within the SDI. In addition, general characteristics of data management systems are presented. Section 5.3 discusses issues within the three environments presented in Section 5.2. References are

provided that survey relevant research in these areas. In addition, a survey of existing distributed DBMSs is provided. Section 5.4 lists funding recommendations relevant to each environment. Areas found to be lacking in data management systems,with respect to SDI requirements, are enumerated. Section 5.5 presents a list of references used in compiling this section of the appendix.

5.2 SDI Requirements

There are at least three different environments within the SDI where database technology will be applied. These are (1) the platform environment, (2) the platform management environment, and (3) the software engineering environment. A description of these three environments and the resulting data management requirements is given in Section 5.2.1 through 5.2.3. General characteristics of typical data management systems are discussed in Section 5.2.4.

5.2.1 Platform System Requirements

Of the various environments where data management technology might be employed, the space-based platform environment will be the most critical. While it is intuitive that the platform databases must be distributed in nature, the level of distribution may not be very high. Although the Fletcher Report [Fletcher 84] advocated a very central architecture, it seems that few functions need to be maintained at the global system level (e.g. central command authority, global situation assessment). The Eastport Group [Cohen 85] determined that tight coordination of the systems seems to provide only slight improvement in system performance. In addition, tight coordination would be a potential source of many technical difficulties in performance, reliability, and testability. Thus, it seems that each battle group (a group of platforms) should be made as autonomous as possible.

The space-based platform data management systems should not resemble today's monolithic centralized DBMSs. Instead, the functional requirements of the system (e.g., track targets, assigning weapons, launch determination, etc.) should probably be implemented as high performance algorithms (e.g., tracking files, fire control, launch doctrine). The need for performance is evident, given that the engagement lasts for a relatively short time while computations are enormous. In this light, probably the only functional layer (as discussed in Section 5.1.2.3) supported within this environment will be the operating system layer. Other layers are either unnecessary or would result in performance losses.

While the space-based platform databases will be distributed in nature, they should be as autonomous as possible. Certain data (e.g., track file, kill assessment) must be distributed to other platforms and/or battle managers, while other data (e.g., target selection) may not need to be distributed. Issues such as distribution, integrity and data consistency will be more fully described in Section 5.3 of this report.

5.2.2 Platform Management Requirements

The platform management environment will consist primarily of typical C3 applications. A similar capability is found in the World Wide Military Command and Control System (WWMCCS) Information System (WIS). A network of platform status sensors, command center computer systems, and telecommunications will provide the military with assessment of the operational system.

The data management needs within this environment are extensive. Real-time support among the platforms must be provided during engagement, logistical support must be

provided as the system becomes operational, and managerial support must be provided at all times. Each of these areas has differing functional requirements (e.g., distribution, performance, fault-tolerance, etc.). All functional layers described in Section 5.1.2.3 will be implemented within various data management systems in this environment. It is within this environment where traditional database research will be most utilized.

5.2.3 Software Engineering Environment Requirements

In building an integrated software engineering environment (SEE), one of the most important decisions that must be considered will be the level of database support provided. Traditional environments have seemed to simply "add on" database capability as an afterthought. This resulted in poorly integrated systems. It will be necessary for the SDI to develop a fully integrated database as the core of the software engineering environment.

Providing database support within the SEE offers many advantages over previous environments where the database was not incorporated. A coherent and consistent organization for data structures and interrelationships is provided via data models [CCA 85]. This relieves many problems that dependence on conventional files creates. In addition, logical and physical independence between the tools comprising the SEE and the access methods used within the database provides for a more extensible environment.

Also, the Software Engineering Environment Database (SEED) must provide the capability for handling multiple types of information, such as text, code, test data, management data, graphics, attributed syntax trees, etc. These types of data should be typed objects within the SEED, and have certain operations that may be performed on them. Finally, the inclusion of database support within an environment allows a uniform method of accessing data. This leads to increased security, integrity and consistency.

In the past, DBMS technology research has received much attention. However, little of that attention has been focused on the role the database should play within the environment. Opportunities exist for all of the functional layers, as defined in Section 5.1.2.3, to be implemented within this environment. There are several key issues which need to be explored more fully to understand this role. Section 5.3.3 will discuss some of these issues.

5.2.4 Characteristics of Data Management Systems

There are a number of characteristics of database management systems that, to some degree, will be required in almost all environments under consideration. Several of these requirements are enumerated in this section of the report. Section 5.3 of this report discusses how each requirement affects the different environments.

5.2.4.1 Integrity

Within the field of database technology, the term "integrity" has several connotations. Accuracy, correctness, validity, consistency and security all have meanings which can be bundled together under the term integrity. The fundamental goal of integrity is guarding the database against invalid data. Invalid data may result from several different types of errors (hence the different denotations of integrity). Integrity may be compromised by incorrect data entry, by a system failure, by program error, by operator error, by timing errors, or by hostile alteration (although this is usually discussed as a security issue).

Integrity within database systems may take two forms, one dealing with the interaction of transactions and the other dealing with validating individual transactions. One aspect in

ensuring integrity deals with multi-user environments. Most systems developed today employ techniques that attempt to guarantee that incorrect results will not be produced from two (correct) concurrently executing transactions.

However, a more important aspect of integrity is the validation of an individual transaction. A transaction may fail for a variety of reasons. The user may terminate the transaction, in which case the state of the database must be brought back to just before the transaction's execution. The system may crash, in which case logging techniques may help restore the database. The application program executing the transaction may fail. The contents of the transaction may violate semantic integrity constraints placed upon the data.

Three main facets of integrity will be discussed in further detail: semantic integrity, concurrency control, and recovery.

5.2.4.1.1 Semantic Integrity

Semantic integrity involves checking that updates maintain the database in a reasonable state. Erroneous updates may result from faulty data, sensor readings, incorrect application programs, or user error. Semantic integrity checks can verify that an update is self-consistent and that it is consistent with the current state of the database. It is not possible to verify that the update is actually correct, as the database system has no independent information about the real world other than the data and knowledge in the database. However, when it is possible to circumscribe the reasonable states and transitions in a database, semantic integrity checks can help prevent storage of erroneous data. For example, position coordinates within a track file can be constructed such that only above ground positions would be valid. This would ensure against positions below the earth's surface being placed in the track file.

We distinguish between two categories of semantic integrity constraints, static and dynamic. Static constraints refer to each database state individually. Examples of such constraints are domain (e.g., age must be greater than 0), tuple (e.g., price times quantity equals total), relational (e.g., there can only be one platform numbered 56824), and inter-relational (e.g., a platform must be controlled by a specific battle manager). Dynamic constraints refer to database state transitions (e.g., no object may move faster than the speed of light) or to the history of database states (e.g., a tracked object may not unexpectedly change velocity by a large amount).

While constraints assist retrieval requests by increasing the confidence level in the data or by permitting more efficient retrieval algorithms to be used, they sometimes slow down updates because the constraints indicated may require additional work to check. Some constraints are well understood and easy to implement efficiently while others are less well understood or do not yet have efficient implementations.

5.2.4.1.2 Concurrency Control

Integrity may be lost within a database due to errors resulting from concurrent execution of transactions. There are three main types of problems encountered in this area: lost updates, "dirty" reads, and unrepeatable reads [Fernandez 81].

The lost update results from interleaving of transactions. Data may be read by two transactions, processed, then immediately written such that the first update is lost. A "dirty" read occurs when a transaction operates on data that has not yet been committed (committed refers to locking data needed by a transaction so it may not be accessed by another transaction). If the transaction holding the commit fails, the second transaction

may be operating on "bad", or "dirty", data before and after another transaction updates the same data. An unrepeatable read occurs when a transaction reads a data item, another transaction updates that data item, and then the first transaction reads that data item again. The first transaction will have read two different values for the same data item. All three of these types of problems present integrity problems within a distributed database.

5.2.4.1.3 Recovery

Recovery refers to the ability to restore the database to the state prior to a failure. There are several different forms of failure that may take place (e.g., transaction failure, system failure, or media failure). Recovery is concerned with two functions: (1) preparing for failure, and (2) restoring a database to its prior state after a failure.

A database may prepare for a failure in several ways. It may keep a log of actions performed. A save point may be placed within the transaction, saving the environment prior to the execution of the transaction. There are many other methods, but logging seems to be the most common.

The steps necessary to restore the database to its prior state after a failure involve the actual undoing of the action that caused the failure. Or, if the failure was a system failure, backup procedures or compensation techniques may be used.

5.2.4.2 Modifiability/Modularity

It is likely that the data management needs within the platform management and software engineering environments will evolve overtime. Requirements for the database systems will be created, modified, or removed. In addition, there will probably be many different instances of database systems within, at least, the platform management system. While the basic framework of the database systems will remain the same in these instances, particular implementation strategies or functional requirements may vary. For instance, if a database system is to operate in a secure system-high (system-high meaning all information processes are treated as classified material) environment, features of the system dealing with multi-level security for various classes of security might be extraneous. A multi-level security capability may be desirable. If the system were constructed in a modular fashion, this capability (and others) might be more easily modified. To this end, it seems that the SDI's database systems should be constructed in a modular fashion that enables functional and performance considerations to determine implementation strategies. This is not being done to a large extent in today's systems.

Wiederhold and colleagues have proposed a framework for the design of a database system that provides a collection of loosely coupled, highly cohesive modules as the basis from which the database system may be constructed [Wiederhold 86] . Given the wide range of database systems that the SDI may need, this type of architecture seems appropriate.

5.2.4.3 Other Characteristics

There are other characteristics of data management systems that will be involved. The two most critical to the platform system will be distribution and performance. These characteristics will be discussed within Section 5.3.1. Of special importance to the software engineering environment database will be issues of security and interfaces. These characteristics will be discussed within Section 5.3.3.

5.3 Current Status

In this section the different environments are more fully reviewed. In areas where relevant research is being performed, a reference citing the performer is listed. In addition, a survey of existing distributed DBMSs is provided.

5.3.1 Platform Environment

This section of the report discusses the current status of data management support technology that will be required within the platform environment. For reasons that will be enumerated below, the implementation of the data management functions will not resemble what is commonly referred to today as a DBMS. Instead, specialized, high-performance algorithms operating on directly addressable data will likely perform the activities associated with the management of data. This will closely tie hardware/software synergy (discussed in Section 8, Appendix B), as well as the operating system (discussed in Section 4, Appendix B), in with the implementation of data management functions.

As has been often said, the performance demands placed upon the space-based platforms will be extreme. There are several reasons for the severity of the performance demands: (1) short engagement duration (especially boost phase), (2) large number of computations necessary during engagement, and (3) a distributed system architecture. Although little can be done about the length of engagement, the system software architecture and the methods of computation are within our control.

Although the Fletcher report advocated a central architecture, current thought seems to support a distributed, decentralized system. Within this architecture, a battle manager is tasked with receiving data from sensors, and transmitting data to weapon platforms. There are currently many different ideas/scenarios being put forth, with many different implications for the implementation of the data management functions. Within this section of the report, only those assumptions of the architecture set forth above are presumed.

During engagement, the number of updates/accesses to the data management system will be enormous. From a software standpoint, there are two approaches to addressing this problem. First, highly optimized algorithms may be used to speed these processes. However, the Eastport Group [Cohen 85] warns that "the most error-prone part of software stems from trying to optimize its performance. " This optimization is usually performed at the machine-code level. Another method of handling large processing intensive requirements would be to apply "brute-force" techniques. Supposing computing cycles will be abundant, simple procedures may be used many times for complicated procedures. This method is advantageous in that the software is likely to be much more reliable.

There are other issues that must be addressed within the scope of the platform environment. These issues will be discussed in the following section.

5.3.1.1 Issues

Given the functional requirements and the performance demands placed upon the platform system, it appears that little of today's research in traditional distributed DBMS areas will be applicable to the SDI. Typical areas involved with traditional distributed DBMS research include concurrency control, deadlock resolution, recovery, and types of models supported. Procedures developed for solving problems associated with these areas include such things as two-phase commit locking, time-stamp ordering, wait/die protocols, modified relational models, etc. However, many of these procedures are not well suited for an environment where an operational system is engaged for only a few minutes in processing huge amounts of data. IBM's (IMS) fast path is a good example of a system

that has been optimized for huge volumes of short transactions. This is a commercial, hierarchical database system, and several of the techniques used may be applicable to the SDI.

The track file needed for supporting critical functions such as target acquisition and tracking, kill assessments, weapon assignments, etc., is a good example for illustrating the difficulties involved with handling distributed data. The Fletcher report estimated that the track file would need to be designed to handle 30,000 objects, each represented by an entry of 200 bits. This would mean a track file composed of roughly 6,000,000 bits. It would be impractical to use a two-phase commit strategy to ensure all updates to the track file were handled synchronously across the SDI platform system. In addition, techniques used to determine data consistency may defeat their original purpose. If, for instance, replicated data were found to be inconsistent, then the time needed to bring that data back to a consistent state would be much longer than the time to next update that data. In addition, techniques such as these would drastically lower the performance of the system. It is therefore necessary that the system not be centralized (as envisioned in the Fletcher Report), but decentralized, and that the nodes of the system be as autonomous as possible.

It is clear that a position between fully centralized and fully autonomous, distributed battle managers is necessary. As it will take several different sensor inputs to correlate key elements of the track file, it seems probable that a battle manager, perhaps within a group, will assimilate sensor data before distributing it to its assigned weapon platforms. During engagement, portions of the track file (or simply a target position) would be disseminated continually among weapons platforms (for targeting), and other platforms capable of assuming battle manager status (for survivability). There are few efforts underway exploring the effects of distribution in environments such as the SDI. The Naval Research Laboratory [Jajodia 86] is performing interesting research in the area of target correlation and tracking. They have developed data structures, called monotonic logical grids, that provide for the storing, calculating, and projecting of object tracks. NRL has combined parallel algorithms with these data structures and have achieved orders of magnitude performance gains over traditional methods of track correlation. In addition, they are investigating means of replicating data (specifically the track file) to achieve high degrees of survivability and availability. In addition, RADC is currently analysing the effects of distributed processing with regards to replicated and/or partitioned databases.

A traditional DBMS research area has been the area of data consistency. Data consistency is an aspect of integrity usually associated with the concurrent execution of transactions. It does not appear that the battle management system will need to keep data consistent among its weapon platforms or sensors. It does appear necessary, however, to keep the weapon platforms and sensors "up to date." If an attempt was made to keep data consistent among platforms, procedures such as two-phase commit protocols would be necessary in order to insure that all platforms contained the exact same data. This approach would not be able to meet any real-time constraints. Another approach would be to simply attempt to keep platforms, and possibly sensors, up to date with the most current applicable information. In this scenario, data would be constantly transmitted from the battle manager to the platforms/sensors without being concerned about whether one platform contains slightly old data (the data may be determined to be old due to clock time, input cycles, mission cycles, etc.). Care would be taken to ensure that if a platform stopped receiving data from a particular battle manager, that it would attempt to begin receiving from an alternate battle manager.

An event that the SDI data management system will have to be prepared for is reconstitution. During engagement, if a platform loses communication, the remaining platforms must acknowledge this loss, and take appropriate steps within the battle

management group. In addition, if the isolated platform is still able to function despite its loss of communication, it should still attempt to perform its assigned roles to the best of its abilities. In the event of regained communication, the isolated platform must be reconstituted. This will mean a complete transfer of relevent data to the formerly isolated platform from the other platforms in the group.

5.3.2 Platform Management Environment

This section of the report discusses the data management needs of the platform management environment and the current status of relevant technology. This environment, as opposed to the platform environment or the software engineering environment, will have a number of varied application areas. Real-time support for the command, control and communication of the whole SDI system must be provided, as well as managerial and logistical support for the day-to-day operations. These differing applications will possess correspondingly different data management functional requirements and implementations.

During engagement of the SDI system, for example, much of the data generated from the sensors, weapon platforms, battle managers, etc, must be assimilated and presented in a fashion readily understandable by human decision makers. This will involve real time responses almost as fast as those required within the platform environment.

For day to day operations, managerial and logistical support will be needed on a vast scale. The scope of this segment of the environment is similar to that of the WIS. Activities such as information collecting, processing, and display, as well as activities such as equipment accounting, maintenance schedules, and cost analyses will be required.

There are many issues that must be addressed within the scope of the platform management environment. These issues will be discussed in the following sections.

5.3.2.1 Issues

In surveying the data management requirements within the platform management environment, many of the traditional areas of research will be applicable to the SDI. This is due to the wide range of applications found within this environment. Requirements such as integrity, distribution, high performance, user friendliness, etc., are found within the platform management environment. In addition, more advanced data models, database machines, workstations, etc., will be required for effective operations of this portion of the SDI system. Although the environment will be primarily ground based, segments of it may be located elsewhere. Airborne, spaceborne, and oceanborne segments may compose part of this environment.

Many of the logistical/managerial functions required by the SDI are implementable within today's state of practice technologies. Current commercial off the shelf DBMSs would satisfy many of the requirements set forth. However, there are several areas that would render these DBMSs unacceptable. These areas, and others relating to the real-time segment of the platform management environment, will be individually discussed in the section following. The first area, distribution, deals with the effects of distribution on DBMSs. Subsequent areas will deal mainly with issues regarding DBMSs in general.

5.3.2.1.1 Distribution

As in the platform environment, the platform management environment has distributed requirements. These requirements are not absolute, however, as initial DBMS prototypes may be centralized for lack of acceptable distributed versions. The real-time segment of the

platform management environment will utilize techniques similar in nature to those employed on the sensors, weapons, and battle managers within the platform environment. As with those systems, current research in distributed systems may not be applicable since they focus on integrity/consistency, not performance. Techniques developed for the platform systems should be used within this environment. A discussion of distribution concerns for the platform environment is found in Section 5.3.1.1.

For the logistical/managerial data management needs of the platform management environment, distribution is a desirable objective. Initial prototypes may not be distributed, but later versions may be implemented in this fashion. There are several areas of research associated with the development of distributed DBMSs. Concurrency control, heterogeneity, recovery, and deadlock resolution are all issues that have not been completely resolved, and will need to be addressed early. It should be noted that many efforts are underway in these areas, and that they may be resolved independently before development of the initial SDI prototypes.

Concurrency control and deadlock resolution are two areas within distributed processing that are fairly well developed. Many algorithms and approaches exist for solving these problems and it is only a matter of choosing one over another. A more difficult problem is determining which implementation is best for the SDI systems.

Techniques for recovery after failure need to be improved for distributed DBMSs. Current techniques introduce unacceptable levels of overhead. Methods must be developed so that precautionary procedures do not greatly degrade normal processing. Current research indicates that decentralized control provides a means for failure recovery. This is a relatively new area of research, and much more investigation is necessary before its effectiveness is understood.

SDC is currently looking at several of the aforementioned issues, such as concurrency control mechanisms, and will be demonstrating the effectiveness of various concepts in realistic environments. In general, however, wide-spread experience in distributed DBMSs is lacking. Few systems today deal with real world applications, and none of these perform at acceptable levels.

5.3.2.1.2 Integrity

Integrity considerations dealing with the real time portion of the platform management environment will be similar to those found in the platform environment. Due to performance requirements, and also the type of interaction between the database and the objects which will supply data to it, classical methods of ensuring integrity are not possible (nor even necessary). Steps must be taken, however, to ensure the data received is actually that which was transmitted from the intended sensor, weapon, etc.

Within the logistical/managerial portion of the platform management environment, integrity concerns will be much more achievable. Due to a relaxing of performance requirements, emphasis can be placed on guarding the database from invalid data.

The various forms of integrity listed in Section 5.2.4.1 are traditional areas of database research. Many implementations of various strategies designed to ensure the different types of integrity have been incorporated within commercial systems. While these strategies have not been perfected, they are currently at an acceptable level for use by the SDI. Contributions from industry over the next few years may be expected, furthering techniques in this area.

5.3.2.1.3 Security

As discussed in Section 5.3.3.1, the issue of security within DBMSs has been a major concern in the last several years. However, it is not clear what the requirements for security will be or whether the benefits derived from a multi-level secure system outweigh the enormous expenditure that would be required in order to obtain one. It may be best to simply run in a system-high mode. In any case, a multi-level secure DBMS is many years away.

5.3.2.1.4 Modular Design

As enumerated in Section 5.2.4.2, the databases implemented in the platform management environment must be designed in a modular fashion. This will greatly facilitate the inevitable transition from prototype to prototype, and from different versions of operational systems. If different implementations of modules may be benchmarked against one another, prototype systems will be better suited for their applications. In addition, this will allow for the constructing of different classes DBMSs using the same basic framework. If the basic framework of the database is modular in nature, portions of it may be upgraded or implemented in different fashions without affecting other modules or the basic framework. For instance, different file access implementations may be used for different types of databases.

5.3.2.1.5 Interfaces

An important aspect of the platform management environment will be the interfaces selected between the system and it's users. Natural language interfaces for human users is desirable. For language interfaces, interesting work has been performed for the WIS JPMO [Friedman 86]. WIS has developed a compilable Ada-SQL interface that allows many of the benefits of Ada to be exploited within its database interface. This method seems to be promising.

5.3.2.1.6 Portability

The platform management system environments will be located on several different types of hardware. Emphasis must be placed on designing and implementing portable data management systems. This can be achieved through the incorporation of various standards. Programming language standardization must be achieved so that systems and applications are easily rehosted on various hardware configurations. Interface standardization is also critical [Morton 86]. Information generated from various applications must be available for many uses.

5.3.3 Software Engineering Environment

This section of the report discusses the database management support that will be required within the SEE. The database used within this environment will be called the Software Engineering Environment Database, or SEED as it has been referred to in the literature.

The SEED will include information about managerial, technical and clerical aspects, such as products and their versions, e.g., documents, programs; resources, e.g., dollars, personnel, equipment; management controls, e.g, milestones, schedules, responsibilities; configuration controls, etc. [Oberndorf 85].

Research into issues concerning database support for SEE's is a relatively new field. In 1980, the DoD formulated the Stoneman requirements [Stoneman 80] which provided a

general definition of software support environment requirements for Ada. TRW used the Stoneman requirements as a baseline in assessing their internal corporate objectives, which led to the design and development of the "Software Master Database" [Boehm 84]. A binary relational model has been examined for use in describing information concerning software projects [Meyer 85]. A software environment named ODIN has been developed which contains a system for managing typed objects [Clemm 84]. This object manager provides capabilities (such as the definition of object types and operations allowed on those types) similar to the needs of the SEED. Finally, CCA, under contract for STARS, has compiled a list of database management issues connected with STARS SEE requirements [CCA 85].

In the sections that follow, characteristics and requirements of the SEED will be elaborated, as well as where research is lacking in these areas. The information presented will be drawn heavily from [SEI 86], [Oberndorf 85], [CCA 85], as well as [Boehm 84], [Meyer 85], and others.

5.3.3.1 Issues

There are a number of issues that have not been resolved relative to characteristics and requirements that an environment database might exhibit. Other issues relate to what type of features and characteristics the DBMS must provide in order to perform the requirements demanded by the SEE.

A fundamental research issue that has not been resolved is the extent to which project information is stored and collected. It may not be possible for all information concerning a project to be placed within SEED control (especially projects such as those that might be within the SDI). Current technology (and predicted technology) in hardware and software may not be able to meet these needs. Therefore, research must be undertaken to understand the basic role of the SEED. The amount and type of historical information that will be collected and defined must be determined. Methods of documenting the historical information collected need to be improved, as do the actual method of collection (e.g., manual or automatic).

In addition, the role of active components must be defined. Active components are "mechanisms which monitor the data and take action when some constraint or rule is met" [Oberndorf 85]. There are many times when the modification of data (e.g., requirements or design specifications, baselines, etc.) results in the modification of subordinate data. For example, the modification of a requirements or design specification would mean the modification of the code and possibly documentation. An active component would alert the fact that the two may be out of synchronization. Active components may be implemented as part of the underlying DBMS, the SEED, or tools within the environment. It seems probable that the SEED would be the most likely candidate for performing this operation. However, it remains to be seen if this is true.

An important aspect of the SEED, as in the platform management environment database, will be its interfaces with other entities. For human users interacting with the SEED natural interfaces are desireable. Interfaces between tools need to be agreed upon. A major effort in this area is the CAIS standard [CAIS 85]. This standard provides specifications for tool interfaces in an Ada environment. This standard will make tools within Ada environments much more portable. Finally, interfaces need to be developed in order to access data needed by the project, yet not resident within the SEED. This data will most likely reside in heterogeneous databases (databases with different implementations of models, interfaces, data formats, etc.), or other such dissimilar formats. Methods must be developed which enable the efficient transformation from one type of format to another.

Morton and Redwine have defined several different categories of interfaces, in addition to providing examples of efforts to achieve them [Morton 86].

A major issue within database research, potentially relevant to the SEE, is that of security. There are many facets of security within DBMSs, including such categories as discretionary control, mandatory control, multi-level security, and inference protection. Up until a few years ago, little research was being performed in order to solve some of the more difficult security problems. For the SDI, it is not necessarily certain that a multi-level secure DBMS is necessary. In any case, it does not appear that a multi-level secure database will be developed within the next five years. One reason for this delay is that the criteria for a secure DBMS has not been established. Guidelines similar to the Computer Security Centers Trusted Computer System Evaluation Criteria [TCSEC 83] need to be established.

There is as yet no consensus as to the type of architecture the SEED will employ. While is is widely recognized that a distributed SEE (and therefore SEED) is desired, it is not known when this will be achievable. Current SEE's are typically centralized, supporting few projects, or are decentralized allowing for slight communication between sites. There are several issues relating to a distributed SEED that must be resolved.

As enumerated in [SEI 86], concurrency control and recovery will require significant amounts of new research. Locking schemes may need to be developed due to the amount of data that will be held for prolonged periods of time. In addition, recovery mechanisms may need to be modified in order to enable data to be recoverable over longer periods of time.

5.3.4 Distributed Database Management Systems Projects

In this section of the report, a number of existing distributed database management system efforts are reviewed. This section is not meant to be exhaustive, but attempts to survey some of the more interesting or successful endeavors. Although there is still some dispute over the characteristics of a DDBMS, the systems discussed meet most of the criteria associated with DDBMSs.

5.3.4.1 ENCOMPASS

ENCOMPASS is a commercial, homogeneous distributed database management system marketed by Tandem Corporation. The system is based upon the GUARDIAN operating system under Tandem's multi-computer Non-Stop computer architecture. The basic architecture of the GUARDIAN and the Non-Stop is distributed, thereby providing easy communication between processes from different CPUs [Borr 81].

5.3.4.2 SDD-1

SDD-1, developed by Computer Corporation of America, was the first prototype of a distributed database management system. SDD-1 is a relational database, and many different techniques with regards to DDBMS were first experimented on SDD-1. As such, system performance is poor, and several features of SDD-1 are almost outdated [Rothnie 77].

5.3.4.3 R*

Developed at the IBM San Jose Research Laboratory, R* is a distributed relational database composed of autonomous sites. These sites may or may not be geographically separated.

Fragmentation (the splitting of global relations among sites) and replication among sites is not supported, although location transparency is achieved from the users viewpoint. R* is a prototype system, and as such is being used for research in areas such as view protection, snapshots, fragmentation, and replication [Ceri 84].

5.3.4.4 DDM

The Distributed Database Manager, DDM, is being developed by the Computer Corporation of America. DDM supports horizontal fragmentation, as well as replication among its sites. A language named Adaplex is used for accessing and manipulating the database from within Ada programs through preprocessing. DDM is based on several architectural features found in SDD-1.

5.3.4.5 Distributed-INGRES

Developed by the University of Berkeley, Distributed INGRES is a DDBMS based on the relational DBMS INGRES. Fragmenting of relations is supported within Distributed INGRES, as is allocation transparency. Query processing is achieved through the non procedural Query Language (QUEL). Two-phase locking is utilized for concurrency control.

5.3.4.6 Multibase

Multibase, developed by the Computer Corporation of America, is a system for retrieving data from heterogeneous databases. Users access data using the language Daplex, and are provided with a uniform view of the database. Multibase provides support for heterogeneous read-only applications which do not affect data allocation. Update procedures are the responsibility of the local sites. Many advanced concepts are not implemented in Multibase, as the objective of the system is to interface with existing systems without modifying them.

5.4 Recommendations

This section of the report identifies critical areas in data management where the SDI should consider funding research and development prototypes. Some of the recommendations are general and do not identify any particular candidates for funding. In the cases where candidates are identified, however, further analysis of their current research projects should be undertaken to ascertain that the opinions expressed in this document are accurate.

5.4.1 Platform Environment

Of the three environments considered in this report, the data management functions in the platform environment are the most critical. Unfortunately, it is also the least researched and least understood. The work being performed by the Naval Research Laboratory [Jajodia 86] regarding replication of data and track file correlation is relevant and should continue to be funded.

In addition, it seems that one of the SDI phase II architecture candidates [Carlin 86] has drafted a "Strawman" document describing their interpretation of the data management functions and requirements for the SDI platform system. Although this document was not available for inclusion in this report, it is of some significance as it is the first unified set of functions/requirements describing a platform system. This document should be critiqued for its strengths and weaknesses.

Although there is much interest in research for distributed data management systems, little of it is concerned with issues relevant to the SDI platform system.

Recommendation 1: SDI should prototype platform data management architecture models. This will aid in determining database structure, data elements that will need to be distributed (and the level of distribution), and strategies for reconstitution.

5.4.2 Platform Management Environment

Within the field of data management, probably the most difficult research issue to the SDI will be that of implementing an efficient, reliable distributed DBMS. This technology will be required within the platform management environment, and few systems in existence today satisfy the perceived requirements in this area. Funding for research in this area should be a high priority item.

Recommendation 2: SDI should fund research into areas such as data model selection, homogeneity, and error recovery for distributed systems. The SDIO should prototype distributed DBMSs that experiment with different aspects of these issues. In addition, the modular architecture for DBMSs discussed in Section 5.2.4.2 is recommended for implementation. Research and development prototypes of systems developed around this concept should be funded.

Recommendation 3: Issues, such as the definition of various interfaces, concurrency control and deadlock resolution algorithm selection, should be prototyped to determine the merits of different algorithms.

Many issues relevant to the platform environment are also relevant to the platform management environment. For the most part, the recommendations contained in Section 5.4.1 are applicable for the real-time segments of this environment.

Finally, there are a number of existing DoD programs that contain several of the data management requirements similar to that of the SDI platform management environment.

Recommendation 4: It is highly recommended that programs such as WIS, NASA space station, and STARS continue to be analyzed to determine the potential synergy between other government agencies involved with data management systems and the needs of the SDIO.

5.4.3 Software Engineering Environment

Recommendation 5: With regards to software engineering environment data management requirements, SDIO should standardize on interfaces between DBMSs and users, tools and data. Other issues need to be resolved (as discussed in Section 5.3.3) within the short term, such as the role of active components, choice of data model, amount of

project data to store, etc. These issues should be resolved early to that their results may be applied to other projects.

SDI must have an object management system for strongly typing and controlling objects created within the environment or the user.

Recommendation 6: It is recommended that prototype object management systems be developed, in Ada, similar to the ODIN [Clemm 84] and KEYSTONE [Osterweil 84] systems.

Recommendation 7: SDIO should fund studies to determine whether or not multi-level secure DBMS technology is required in the program. In any case, a production quality multi-level secure DBMS is probably several years ahead in the future.

5.4.4 Summary of Recommendations

Within the platform environment, a description of the actual data management functions should be drafted. Continuous development of algorithms for data replication and track file correlation is recommended. Methods for increasing the efficiency of data management functions should be explored. The effects of parallel computer architectures on these data management functions should be explored.

A central concern of the platform management environment will be the development of an efficient, reliable distributed DBMS. Issues relating to DDBMSs such as data model selection, homogeneity, error recover, etc., need to be understood more fully. Algorithms for concurrency control and deadlock resolution, and various DBMS-related interfaces should be studied. Prototype DBMSs implemented using a modular architecture should be funded.

For the software engineering environment, many issues pertaining to the role of the SEED need to be explored. The issue of whether or not a multi-level secure DBMS is required needs to be resolved.

5.5 References

[Apers 84] Apers, P. and Wiederhold, G., "Transaction Classification to Survive a Network Partition," Submitted to *ACM Transactions on Database Systems (TODS)*, Stanford CS report, (September 1984).

[Boehm 84] Boehm, B., Penedo, M., Stuckle, E., Williams, R. and Pyster, A., "A Software Development Environment for Improving Productivity," *IEEE Computer* 17, 6 (June 1984), pp. 30-44, .

[Borr 81] Borr, A., "Transaction Monitoring in ENCOMPASS: Reliable Distributed Transaction Processing," *Proceedings of the Seventh Very Large Database Conference*, Cannes, (1981).

[CAIS 85] *Military Standard Common APSE Interface Set (CAIS)*, Department of Defense, Proposed MIL-STD-CAIS, (January 1985).

[Carlin 86] Carlin, R., "Battle Management C3 Information Flow," *SDI Update '86 Conference*, (March 1986).

[Ceri 84] S. Ceri and G. Pelagatti, *Distributed Databases, Principles and Systems*, McGraw-Hill, (1984).

[CCA 85] Computer Corporation of America, "STARS Software Engineering Environment (SEE) Database Management," Technical Report SEE-ARCHSTDY-011, STARS Joint Program Office, (June 21, 1985).

[Clemm 84] Clemm, G., "ODIN - An Extensible Software Environment," University of Colorado, Department of Computer Science Technical Report CU-CS-262-84, (1984).

[Codd 70] Codd, E.F., "A Relational Model of Data for Large Shared Data Banks," *Communications of the ACM* 6 (June 1970), pp. 377-387.

[Cohen 85] Cohen, D., "A Report to the Director Strategic Defense Initiative Organization," Eastport Study Group, (December, 1985).

[Fletcher 84] Fletcher, J., "Report of the Study on Eliminating the Threat Posed by Nuclear Ballistic Missiles," (February, 1984).

[Fernandez 81] Fernandez, E., Summers, R., Wood, C., "Database Security and Integrity," Addison Wesley, (1981).

[Friedman 86] Friedman, F., Keller, A. Salasin, J., Wiederhold, G., Berkowitz, M., Spooner, D., "Reference Model for Ada Interfaces to Database Management Systems," *Second IEEE Conference on Data Engineering*, Los Angeles CA, (February 1986), pp. 492-506.

[Friedman 86] Friedman, F. and Brykczynski, B., "Ada/SQL: A Standard, Portable Ada-DBMS Interface," *Second IEEE Conference on Data Engineering*, Los Angeles CA, (February 1986), pp. 515-522.

[Jajodia 86] Jajodia, S. and Meadows, C., "NRL Research in Database Management for the SDI Battle Management System," submitted for MILCOM 86 Conference.

[Martin 77] Martin, J., *Computerized Data-Base Organization*, Prentice-Hall, (1977).

[Meyer 85] Meyer, B., "The Software Knowledge Base," *The 8th International Conference on Software Engineering*, IEEE, (August 1985), pp 158-165.

[Minoura 82] Minours, T. and Wiederhold, G., "Resilient Extended True-Copy Token Scheme for a Distributed Database System," *IEEE Transactions on Software Engineering* SE-8, 3 (May 1982), pp. 173-188.

[Morton 86] Morton, R. and Redwine, S., "Information Interface Standards for Software Engineering Environments," *IEEE Computer Standards Conference*, San Francisco, (May 1986).

[Oberndorf 85] Oberndorf, P. and Penedo, M., "Future Ada Environment Workshop - Summary of Project Database Working Group Discussions," *ACM SIGSOFT Software Engineering Notes* 10, 2 (April 1985), pp. 92-98.

[Osterweil 84] Osterweil, L., Clemm, G., Helmbigner, D., Williams, L., "KEYSTONE: A Federated Software Environment," University of Colorado, Department of Computer Science Technical Report CU-CS-284-84, (1984).

[Rothnie 77] Rothnie, J. and Goodman, N., "An Overview of the Preliminary Design of SDD-1: A System for Distributed Databases," *Proceedings 2nd Berkely Workshop on Distributed Data Management and Computer Networks*, (1977).

[SEI 86] "Technology Identification and Assessment: A Survey of Software Development Environment Technology," Draft, Software Engineering Institute, (February 1986).

[Stonebraker 81] Stonebraker, M., "Operating System Support for Database Management," *Communications of the ACM* (July 1981).

[Stoneman 80] *DoD Requirements for Ada Programming Support Environments, STONEMAN*, Higher Order Language Working Group, (February 1980).

[TCSEC83] Department of Defense - Computer Security Center, "Trusted Computer System Evaluation Criteria," (1983).

[Vick 84] Vick, C., and Ramamoorthy, C., *Handbook of Software Engineering*, Van Nostrand Reinhold Publishers, (1984).

[Wiederhold 84] Wiederhold, G., "Databases," *IEEE Computer, Centennial Issue* 17, 10 (October 1984), pp. 211-223.

[Wiederhold 86] Wiederhold, G., Keller, A., Navathe, S., Spooner, D. Berkowitz, M., Brykczynski, B., Salasin, J., "Modularization of an Ada Database System," submitted for

Advanced Database Symposium, Tokyo, Japan, (August 29-30, 1986).

SECTION B6

Man-Machine Interface

Prepared by Michael R. Kappel

Topics covered in Section B6:

6.1 Introduction
6.1.1 Purpose
6.1.2 Background
6.1.3 Organization of Document

6.2 SDI Requirements

6.3 Current Status
6.3.1 Introduction
6.3.2 Human Factors
6.3.2.1 Computer-Human Interaction Models
6.3.2.2 MMI Evaluation
6.3.2.3 MMI Design
6.3.2.4 Future Prospects
6.3.2.5 Performers and Funders
6.3.3 Interaction Techniques
6.3.3.1 Issues
6.3.3.2 State of the Art
6.3.3.3 State of Practice
6.3.3.4 Future Prospects
6.3.3.5 Performers and Funders
6.3.4 Workstation Managers
6.3.4.1 Issues
6.3.4.2 State of the Art
6.3.4.3 State of Practice
6.3.4.4 Future Prospects
6.3.4.5 Performers and Funders
6.3.5 Graphics Packages
6.3.5.1 Issues
6.3.5.2 State of the Art
6.3.5.3 State of Practice
6.3.5.4 Future Prospects
6.3.5.5 Performers and Funders
6.3.6 User Interface Management Systems
6.3.6.1 Issues
6.3.6.2 State of the Art

6.3.6.3 State of Practice
6.3.6.4 Future Prospects
6.3.6.5 Performers and Funders
6.3.7 Decision Aids
6.3.7.1 Issues
6.3.7.2 Future Prospects
6.3.7.3 Performers and Funders

6.4 Recommendations
6.4.1 Human Factors
6.4.2 Interaction Techniques
6.4.3 Workstation Managers
6.4.4 Graphics Package
6.4.5 User Interface Management Systems
6.4.6 Decision Aids

6.5 References

6.0 MAN-MACHINE INTERFACE

6.1 Introduction

6.1.1 Purpose and Scope

This section assesses foundation technology for the man-machine interface (MMI) of the Strategic Defense Initiative (SDI). The results of a preliminary investigation into the current theory and practice in MMI are presented. Those technical aspects of MMI that pertain to unique SDI software requirements are identified and assessed. This section does not address specific applications areas; it considers basic MMI technology across all SDI applications. It provides broad guidelines and direction for research efforts into technical areas of the man-machine interface that are promising as foundations for SDI software technology.

6.1.2 Background

The man-machine interface comprises all interactions between human and computers and hence should not be considered in isolation. By definition, MMI influences all users of all computer systems within the SDI environment. A relationship exists between MMI and all other subareas of SDI foundation technology as well as all application-specific task areas.

In particular, MMI forms a close association with the foundation technology of artificial intelligence (AI). Future advances in AI that may be applied to MMI are natural language, speech understanding, stroked character recognition, and expert user interface management systems.

MMI is closely related to the software engineering environment (SEE) (see Appendix B, Section 9). Many issues addressed by SEE to enhance programmer productivity fall under the heading of MMI.

MMI also relates to the software engineering process task area of simulation (see Appendix B, Section 10). The simulation of real scenarios not only supports user training but provides feedback to assess the effectiveness of the man-machine interface. Experimental analyses of user-system interactions on a simulator will provide insight into the design and integration of tasks to optimize human performance.

Finally, an application-specific task area such as battle management/command, control, and communications (BM/C3) (See Appendix B, Section 2) inherently has an MMI. The design of an interface for effective interaction between user and computer of such applications is the crucial concern of MMI investigators.

6.1.3 Organization of Document

Section 6.2 proposes generic requirements for the man-machine interface of SDI computer systems. Section 6.3 details various technological subareas of MMI. For each subarea, issues relevant to SDI are delineated, the state of the art and the state of practice are documented, prospects for future developments are conjectured, and key participants in on-going research and development efforts are listed. Section 6.4 tenders recommendations for the best course for SDI development and acquisition of foundation technology for MMI software. Section 6.5 contains bibliographic references.

6.2 SDI Requirements

SDI requirements for the man-machine interface span the SDI application task areas. SDI users will fill numerous and diverse operational roles. There will be many users with varying skills located at remotely distributed sites. To complicate matters further, they will perform under stringent time and reliability requirements. Since this section addresses foundation technology, such requirements shall be specified in the most general terms. More specific requirements may be identified when the SDI application task areas are formally analyzed.

While much emphasis has and will be placed on hardware and software technology, humans must play a critical role in the strategic defense system. The list of functions to be performed by people includes [Carlin 86]:

a. Establishing or changing alert status

b. Enabling, terminating, or inhibiting engagement

c. Altering defense strategies and tactics

d. Monitoring battle and equipment status

e. Providing system support and maintenance

f. Managing the force

g. Controlling functions

h. Delegating functions

i. Setting constraints

j. Resolving ambiguities

These functions would be performed by a variety of command and support personnel located at:

a. National Command Authorities (NCA)

b. Joint Chiefs of Staff (JCS)

c. Commanders-in-Chief of the Strategic Air Command, Aerospace Defense Command, and of the European, Atlantic and Pacific Commands

d. SDI Command and Control Centers

The effective performance of these functions by human agents introduces certain requirements for the SDI man-machine interface. A key requirement is for striking an appropriate balance between automated components for timeliness and responsiveness and humans for positive control. This requirement not only determines the form and extent of the man-machine interface, but influences SDI architectural designs and strategies. The conceptual line separating man and machine in the SDI system should be determined in an evolutionary manner. This process entails prototyping, simulation, experimentation, and redesign in an interactive manner with early and continued involvement of the users [Bailey 85].

An important requirement for the SDI man-machine interface is the reduction of error susceptibility. The SDI command in a war-time scenario will obviously operate in a highly pressured atmosphere. Research has indicated that decision-making abilities tend to break down under stress. Since errors committed while under attack could be catastrophic, it is essential that the potential for error be reduced to a minimum. This requirement can be realized by designing a computer system with particular attention to the human element.

Given the size and complexity of the SDI system, the amount of output data will be voluminous. Data will be generated by sensors, received from communications channels, displayed by status monitors, etc., in a continuous and concurrent manner. This information may require human assimilation and interpretation for real-time decisions. Hence a requirement of the man-machine interface is to facilitate this process so that the information load does not become overwhelming and critical factors can be distinguished rapidly and accurately.

In addition to SDI requirements for the man-machine interface, requirements exist for the software used to implement that interface. Such requirements for SDI MMI software technology include:

a. Facilitate achievement of human factors objectives.

b. Reduce software development effort.

c. Provide powerful, high level tools.

d. Facilitate prototyping and experimentation.

e. Promote device independence.

f. Support program and programmer portability.

g. Support application's function/MMI separation.

h. Support evolution of the SDI MMI.

The satisfaction of these requirements may be facilitated through the use of software technology for the man-machine interface as detailed in the following section.

6.3 Current Status

6.3.1 Introduction

The construction of a man-machine interface that achieves human factors objectives may be facilitated through the application of software technology. The collection of software tools for the MMI comprises:

a. Interaction techniques

b. Workstation managers

c. Graphics packages

d. User interface management systems (UIMS)

e. Decision aids

The degree to which these tools facilitate the construction of a user interface that meets SDI requirements for human factors and software development is analyzed in the following subsections. The current state of the art and state of practice are documented and the prospects for their future development are conjectured. Finally, the key participants in on-going research and development efforts are listed.

6.3.2 Human Factors

Human factors concern how humans interact with equipment and encompass a broad spectrum of issues [Foley 85] including:

a. Hardware design and functionality

b. Software design and functionality

c. Experimental design

d. Ergonomics

e. Cognitive psychology

f. Computer science

g. Artificial intelligence

Human factors is not a software tool as are the other MMI subareas addressed in the following sections (6.3.3 - 6.3.7). On the other hand, human factors issues guide the system implementor in the effective use of such tools. Through experimental research and empirical evaluation, useful data is acquired to drive the construction of a user interface that achieves human factors objectives. The potential improvements in task performance more than outweigh the additional costs for the analysis, design, and implementation of such an interface.

6.3.2.1 Computer-Human Interaction Models

Card, Moran, and Newell studied the interactions between computers and humans by modeling a person as a processor of information [Card 83]. The components of the human information processing model include a perceptual system, a motor system, and a cognitive system. This model provides a common framework for the study of memory, problem solving, and perception.

A language model was developed by Foley and Van Dam [Foley 82]. The user-computer dialogue is divided into four layers:

a. Conceptual: key applications concepts which must be mastered by the user

b. Semantic: detailed functionality

c. Syntactic: sequence of inputs and outputs

d. Lexical: formation of input and output tokens from hardware primitives

The model proposed by Norman [Norman 84] takes a cognitive psychology view of the interaction process. Four stages are identified:

a. Intention: set of goals in the user's conceptual model

b. Selection: set of actions or methods chosen to achieve the goal

c. Execution: physical act of inputting selections

d. Evaluation: examination of feedback to determine further activity

6.3.2.2 MMI Evaluation

An understanding of what constitutes a man-machine interface with good human factors has been gained from the cumulative efforts of computer scientists, human factors scientists, psychologists, social scientists, systems designers, and end users. One set of metrics to evaluate the quality of a man-machine interface was proposed by Moran [Moran 81]:

a. Functionality of the user interface

b. Ease of learning

c. Time it takes to perform a task

d. Error sensitivity

e. Quality of the resulting output

Requirements for an MMI should be stated in quantitative terms using the above metrics instead of employing qualifying phrases as "friendly" or "effective". Then a given MMI may be evaluated empirically to determine if it satisfies the requirements.

6.3.2.3 MMI Design

A set of design principles for a man-machine interface has evolved through research and experience. The following set was compiled from a variety of sources [Cooper 86]:

a. Know the user. Adapt to the user's capabilities; adapt wordiness of both input and output to the user's needs; allow choice of entry patterns; provide shortcuts for knowledgeable user; provide guidance when desired; use the user's model for the problem domain.

b. Consider personal worth of user. Leave action initiation and control (e.g., abortion of command) with user; allow user to decide on dialogue technique and to tailor the dialogue pattern; give quick and meaningful feedback on state and progress of interaction and execution; do not require the user to supply information that is already available; maintain a well-balanced social element in the dialogue.

c. Reduce learning. Provide consistency and clear wording; reduce modality; maintain uniformity and homogeneity; avoid overloading of commands; provide self-explanatory commands and help when requested or needed.

d. Engineer for errors. Provide consistency and uniformity; handle all possible input combinations; provide good error messages; eliminate common errors; provide protection from costly errors; provide support for correction of errors; provide context-sensitive online assistance.

e. Know the application. Present a natural image of the application system; provide problem-oriented powerful commands; carry forward a representation of the user's knowledge base about the application's problem domain.

Sidney Smith of Mitre has generated a set of design guidelines for the user interface to computer-based information systems [Smith 83] under contract by the Electronic Systems Division of the U.S. Air Force. Their report revises and enlarges previous compilations of guidelines in six functional areas: data entry, data display, sequence control, user guidance, data transmission, and data protection.

A collection of general guidelines for designing a user interface should next be mapped to the requirements of a specific system environment. General guidelines are strained through a design filter which accounts for the specific characteristics of a system including users, hardware, and applications. This tailoring is documented in a user interface style guide. A style guide facilitates the design and implementation of a human-computer interface consistently styled across a range of applications [McCormick 84]. A style guide addresses global issues such as functional requirements and any known constraints of the system; interaction tasks and techniques; semantics, or the meanings of objects, actions, and attributes encompassed in a system; the input and output syntax; and user guidance, which would include the topics of help, error recovery, documentation, and embedded training [McCormick 84].

6.3.2.4 Future Prospects

The future prospects for human factors in computer systems were addressed at a workshop in Vail, Colorado in April 1983 sponsored by the Human Factors Society [Atwood 84]. The workshop participants established goals, metrics, and methodologies for research objectives in the next ten years. The set of issues and future directions discussed at the workshop is condensed as follows [Foley 85]:

a. System design tools

   (1) Automated interface design
   (2) Knowledge-based tools and design aids
   (3) Procedures, tools, and techniques
   (4) User tasks, expert systems, and metrics

b. Models of user

   (1) Models and metrics
   (2) Cognitive models
   (3) User conceptual models
   (4) User mental models

c. System development process

(1) Standards
(2) Guidelines
(3) Development methodologies

d. Task Taxonomy

(1) Reuse models done for other systems
(2) Reuse guidelines derived from other systems

e. Sociological Impact

(1) Health and safety
(2) Positive and negative effects

f. Professional qualification

g. Public relations

h. Aid for novices

i. Information retrieval

j. Workstation design

6.3.2.5 Performers and Funders

Research in human factors is being conducted at several academic institutions including the University of Maryland (Shneiderman), Yale (Soloway), University of California San Diego (Norman), University of Toronto (Buxton), Georgia Institute of Technology (Rouse), and Virginia Polytechnic Institute (Williges). Furthermore, research in human factors that has relevance to computer systems is being conducted by human factors scientists, psychologists, and social scientists around the world.

Research is being funded in several departments of the federal government. These departments include the U. S. Army Research Institute, the Office of Naval Research, the Electronics Systems Division of the U. S. Air Force, the National Academy of Sciences, and the National Aeronautics and Space Administration. Furthermore, the investigation of human factors issues are planned in particular computer system programs like Software Technology for Adaptable, Reliable Systems (STARS).

The SDI Command and Control Human Interfaces is being investiaged at the Naval Research Laboratories. NRL is attempting to determine appropriate roles for human SDI commanders under the severe time constraints of an SDI scenario. In addition to establishing the feasibility of man-in-the-loop tasking, NRL is investigating alternative MMI mechanizations.

An SDI Human Factors Program is being developed at the Electronics System Division of the U. S. Air Force. A phased multi-year effort is proposed that systematically examines human factors issues and conducts research related to SDI man-machine interfaces. Funding for the program was discontinued in FY87. Their four objectives are to:

a. Identify SDI human performance requirements

(1) Decompose SDI architectures with emphasis on human factors functions
(2) Identify time and information flow constraints
(3) Identify human vs. equipment trade-off areas

b. Acquire SDI human factors expertise

(1) Establish human factors resource data bank
(2) Establish human factors advisory system

c. Acquire human factors technical support

(1) Support preparation of competitive RFPs/SOWs
(2) Management support of government and contractor HF activities

d. Conduct human performance TVEs

(1) Support selection of candidate architectures in near term
(2) Support SDI system-level experiment in long term

A forum for the exchange of ideas between the participants mentioned above is provided by special interest groups of professional organizations. These groups typically sponsor technical conferences and publish technical journals that monitor and advance the state of the art. This category includes the Association for Computing Machinery's Special Interest Group on Computer and Human Interaction (ACM SIGCHI) and Computer Graphics (ACM SIGGRAPH), the IEEE Computer Society, the Human Factors Society, the Cognitive Science Society, the Technical Advisory Group on Human Factors for the Armed Forces, the American Psychological Association, the Ergonomics Society, and International Federation for Information Processing (IFIP) Working Group (WG) 4.2.

6.3.3 Interaction Techniques

6.3.3.1 Issues

Interaction techniques implement interaction tasks using interaction devices [Foley 85]. Interaction tasks are low-level device-independent operations such as selecting, positioning, keying, etc. Interaction devices are the hardware for input and output such as the display screen, mouse, keyboard, etc.

The choice of a technique or set of techniques to implement a given task should be based upon the degree to which human factors objectives are achieved (Section 6.3.2). The man-machine interface for a specific application defines an associated set of interaction tasks. For example, battle management may require selection and text entry. A given interaction task may be implemented by a variety of interaction techniques. For example, text entry may be performed through menu selection, command language, etc. Through experimentation, a set of interaction techniques can be analyzed empirically to best satisfy requirements for the SDI man-machine interface.

6.3.3.2 State of the Art

Software technology for interaction techniques has progressed from the cumbersome command languages of the 1960's. In the 1970's, more natural command languages, question and answer dialogue, and form fill in were introduced. Menu selection and direct manipulation techniques were incorporated into commercial computer systems in the first half of the 1980's. Modern high level concepts in dialogue management include the

desktop metaphor, user tailoring or profiling, the option of browsing, multiple views, and transparent dialogue throughout a network of workstations [Cooper 86]. Graphical techniques include iconics, graphical viewing, graphical programming, and animation [Cooper 86].

Recently, artificial intelligence techniques have supplemented traditional modes of interactions. Limited success has been achieved in expert system interfaces, intelligent user aids, error-tolerant interfaces, and stroked character recognition. Research-oriented systems in the mid-1980's are attempting to develop natural language dialogues. Such languages are attractive but pose significant technological obstacles in their semantic complexity, lack of conciseness, large vocabulary, etc. [Rich 84]. Such problems are being attacked through linguistic tools provided by artificial intelligence.

Two related technologies, speech understanding and speech synthesis, address vastly different issues. Speech understanding involves computer recognition of vocal commands generated by humans. Advances in this area await the development of acoustic/phonetic devices for recognizing continuous speech in real-time [Cotellessa 84]. Speech synthesis involves computer generation of auditory communication of meaningful speech. Most speech synthesizers are based on mathematical models of the human vocal tract, but suffer from poor voice quality and vocabulary restrictions [Badre 82].

6.3.3.3 State of Practice

Commercially available computer systems incorporate combinations of the above interaction techniques. However, the primary media for interactions have been visual and manual with little development for oral/auditory and multimodal communications. Although such state-of-the-art interaction technology has not fully matured, user experience has demonstration that small details have a profound effect on the quality of the man-machine interface. One such example is name completion, i.e., the automatic generation of the rest of a command, filename, etc., when only partially entered. MMI designers should not lose sight of simple, yet effective, state-of-practice interaction techniques when planning for a state-of-the-art interface.

Due to its current technological intractability, a full natural language interface has not as yet been implemented. However, menu-based systems have been built that guide the user in the formation of commands with a subset of natural language. In particular, NLMenu on Lisp machines and NaturalLink on Texas Instrument PC's provide such an interface to databases [Thompson 85]. The WWMCCS Information System (WIS) project has analyzed requirements for software prototype development of this foundation technology in Ada [WIS/DBMS 86].

6.3.3.4 Future Prospects

As our understanding of user-computer interactions grows, a more effective utilization of existing interaction techniques will be realized. In the near term, we can expect a more widespread integration of sophisticated interaction techniques into commercial workstations. In the foreseeable future, experimental techniques such as circumstantial indexing, multimodal interaction, self-disclosure, eye tracking, and gesture may be implemented [Bolt 84].

Advances in the field of artificial intelligence will be forthcoming in the next decade. Application of future AI technology to natural language, speech understanding, and stroked character recognition should lead to better man-machine interfaces in the 1990's. For natural language understanding, the most promising breakthroughs may be in recognizing

the intent of speakers and the relationships between sentences in continuous discourse, taking into account the structure of the proceeding discourse, the non-linguistic aspects of the situation of utterance, and models of the beliefs and goals of the communication agents [Cotellessa 84]. However, due to its inherent ambiguity and imprecision, natural language may never become a suitable mode for human-computer interactions in the SDI environment.

6.3.3.5 Performers and Funders

Research into interaction techniques is widespread. Current research efforts at George Washington University are directed towards constructing a model of user-computer interaction that can predict the performance of both new and skilled users of various interaction techniques [Foley 84]. Experimental techniques are under investigation in MIT's Media Room [Bolt 84] and the multimedia project at Microelectronics and Computer Technology Corporation. Several innovative input devices have been developed at the University of Toronto by W. Buxton. Interaction techniques using AI are being developed for the DARPA Strategic Computing Initiative program under Simpson and Sears.

Industrial research labs are studying fundamental issues of the man-machine interface to formulate a complete system concept. The key players include IBM Thomas J. Watson Research Labs, Xerox Palo Alto Research Center (PARC), Bell Communications Research, and Microelectronics and Computer Technology Corporation.

Other active participants in the study of interaction techniques include workstation manufacturers. Here the research direction is more practical - testing alternative MMI designs with sets of existing interaction techniques for the development of commercial workstations. Notable participants include Computer Corporation of America (CCA), Tektronix, Hewlett-Packard, Apollo, and Sun Microsystems.

Research into the application of artificial intelligence to natural language understanding is being conducted at Yale, University of Pennsylvania, University of Illinois Urbana, Carnegie-Mellon, University of California Berkeley, University of Massachusetts, Stanford, University of Texas Austin, University of Delaware, NYU, USC Information Sciences Institute, Brown, University of Rochester, University of Connecticut, and Columbia [Cotellessa 84]. Development of natural language systems is underway at AI Corporation, Cognitive Systems, Symantec, Hewlett-Packard, and Texas Instruments [Cotellessa 84]. Industrial research labs working in this area include Microelectronics and Computer Technology Corporation, Bolt Beranek and Newman, SRI International, Xerox PARC, IBM Thomas J. Watson Research Labs, and Fairchild Advanced Research Laboratories [Cotellessa 84].

6.3.4 Workstation Managers

6.3.4.1 Issues

A software module called the workstation manager resides between the graphics package and the physical input/output devices. This low-level module processes device-independent commands from a graphics package and generates device-dependent commands to control graphics devices. Its functionality includes [Bono 85]:

a. Generating graphic primitives

b. Controlling the appearance of graphical primitives with attributes

c. Inquiring graphics device capabilities, characteristics, and states

d. Controlling graphics devices

e. Generating and controlling groups of primitives called segments

f. Obtaining graphical input

Workstation managers provide powerful functionality at a high level of abstraction through device independence without sacrificing system performance.

Such functionality can be expanded to include window management. In the context of computer display terminals, windows are areas on the display surface used for output. The screen can be partitioned into a set of windows representing logical groupings of information. Thus we shall assess window management technology, i.e., the apportionment and control of screen real estate, under the heading of workstation management.

6.3.4.2 State of the Art

The International Standards Organization (ISO) and the American National Standards Institute (ANSI) are drafting a standard called the Computer Graphics - Virtual Device Interface (CG-VDI). CG-VDI is a standard funcitonal and syntactical specification of the control and data exchange between device-independent graphics software and one or more device-dependent graphic device drivers. A working draft of the CG-VDI standard is available and final approval is expected in 1987 [Bono 86].

6.3.4.3 State of Practice

Most commercial implementations of workstation managers are embedded in graphics packages and use a proprietary scheme. However, without open access to the VDI, many of the benefits of standardization are lost [Bono 85]. Some workstation managers have been built based on the CG-VDI standard in its current draft form. Upgrades are expected for conformance with the official CG-VDI upon its release in 1987.

A window management scheme was first implemented at PARC for DLisp and Smalltalk on the Xerox Star computer in the late 1970's [Myers 84]. Since then there have been many commercial instantiations. ANSI is examining whether there is sufficient consensus in the software community to standardize a window management interface [Foley 85].

6.3.4.4 Future Prospects

As yet, no commercial developer has integrated window management functionality into a workstation manager that conforms to the CG-VDI draft standard. Such an integration is expected upon final approval of the CG-VDI standard and through standardization of a window management interface.

6.3.4.5 Performers and Funders

IBM and AT&T have implemented the ANSI CG-VDI standard as it was drafted in 1983 [Bono 85]. These workstation managers have become the industry's de facto standard. Graphic Software Systems, Inc. (GSS) has implemented a PC version of the ANSI CG-VDI standard.

There have been many commercial implementations of window managers [Foley 85]:

a. Apollo
b. Explorer (Texas Instruments)
c. Gem
d. Lisa
e. LMI
f. Macintosh
g. Microsoft
h. Sapphire
i. Sun
j. Symbolics
k. Top View
l. Visi On
m. Xerox 1108

Government is also funding such development efforts. Sponsored by the Department of Defense, a window/workstation manager has been designed for the WIS project [WIS/Graphics 86]. Window managers are under development at academic institutions. Two key projects are Athena at MIT and Andrew at Carnegie-Mellon.

6.3.5 Graphics Packages

6.3.5.1 Issues

"Computer graphics is the most versatile and most powerful means of communication between a computer and a human being" [Enderle 84]. The potential contributions of computer graphics to human factors are well-founded [Enderle 84]:

a. Vision is sometimes the most natural way for humans to access information.

b. More information can be perceived in less time and with fewer errors by visual means.

c. Complex structures and models of abstract concepts are often most easily comprehended by visual means.

Given to its potential utility in a wide variety of applications, computer graphics will form the cornerstone of the man-machine interface in the SDI environment. Graphics will play key roles in the presentation of application-specific information and the interaction with visually oriented techniques. Graphical software helps alleviate some of the current technological limitations in hardware display space and resolution.

Graphics packages facilitate the development of graphical applications and interfaces. A graphics package encapsulates a complete set of subroutines for performing graphical tasks from manipulating visual descriptions to driving graphics hardware. A graphics package typically provides general-purpose facilities that can be used in diverse applications. Furthermore, a graphics package is at a higher level of abstraction than a programming language. Hence programmer efficiency is enhanced through device independence and powerful utilities.

The integration of a standard graphics package into the foundation of SDI software technology has several benefits:

a. Portability

(1) Programs and data between different computers
(2) Experience of programmers and end users

b. Interoperability

(1) System/subsystem integration
(2) Commercial off-the-shelf software

c. Performance

(1) Implement some graphics functions in hardware
(2) Tailor performance to an application

d. Reliability

(1) Validated once
(2) Changes are controlled

A commitment to a graphics standard for all SDI application software would:

a. Increase the operational life of the system

b. Reduce life cycle costs

c. Increase operational confidence and ease of use

6.3.5.2 State of the Art

Current graphics standards and standards projects for the application programmer's interface include the Graphical Kernel System (GKS), its three-dimensional counterpart (GKS-3D), and the Programmer's Hierarchical Interactive Graphics System (PHIGS). PHIGS is an emerging standard to support highly dynamic, highly interactive applications requiring rapid screen update of hierarchically structured 2-D and 3-D images. The standardization efforts are in various stages of development [Bono 86]:

a. GKS - ANSI standard published October 1985.

b. GKS-3D - public review Fall 1986.

c. PHIGS - first public review closed on March 22, 1986.

The standard itself contains only a language independent nucleus of a graphics system. A language binding maps the nucleus into functional declarations of a given programming language. The standardization efforts for the Ada language bindings are in various stages of development [Bono 86]:

a. GKS - public review Fall 1986.

b. GKS-3D - public reviews in 1986-87.

c. PHIGS - working draft review 1986.

6.3.5.3 State of Practice

Although supporting many standard features, all commercial graphics packages do not strictly adhere to the conventions of existing graphics standards. Several commercial graphics packages abide by the draft specifications of graphics standards current at the time of development. These packages are expected to be upgraded to conform to the official

standard upon its release. Access to their subroutine library is provided via older high-level programming languages such as Fortran, Pascal and C. None as yet support invocation by Ada applications.

6.3.5.4 Future Prospects

As Ada gains a foothold in private industry, the introduction of commercial graphics packages with Ada language bindings can be expected. Furthermore, given the strong incentive of conforming to graphics standards, full implementations of the Ada language bindings to GKS and PHIGS are expected to be available commercially in several years.

Current standards for graphics packages are based on the paradigm of the subroutine library. While satisfactory for the older, procedure-oriented programming languages, this approach is not natural to the Ada programmer and does not use the full richness and capabilities of Ada. A standard graphics package unique to Ada, utilizing such facilities as packaging, generics, user-defined data types, operator overloading and data abstraction, may prove to be of great utility to the Ada graphics application programmer. No formal efforts are underway, but the approach has been studied with positive results [Thalmann 79; Airchinnigh 84; Kappel 85].

6.3.5.5 Performers and Funders

Private industry has implemented graphics packages that abide by the current GKS standard. Some graphics packages in this category that contain bindings to programming languages other than Ada include [Bono 85]:

a. Advanced Technology Center
b. Cybervision
c. GIXI
d. Graphic Software Systems
e. IBM
f. Megatek
g. Nova Graphics
h. Precision Visuals
i. Prior Data Sciences
j. Visual Engineering
k. Uniras

One company, Megatek, is marketing a product called Figaro, an implementation of the PHIGS Kernel in its draft form [Brown 85].

The GKS language binding to Ada is currently being implemented by the Harris Corporation under the sponsorship of the Department of Defense. In combination with validated Ada compilers, the GKS-Ada implementation should provide a solid foundation for the future development of graphics application software in Ada. The Ada language bindings to GKS-3D and PHIGS may also be sponsored by the Department of Defense.

The Department of Defense has also funded the design of a graphics package for the WIS project. This package extends the functionality of the GKS-Ada implementation [WIS/Graphics 86].

6.3.6 User Interface Management Systems

6.3.6.1 Issues

A User Interface Management System (UIMS) is a software package that provides the basic support and framework for user interaction [WIS/Command 86]. A UIMS provides a set of high-level utilities to relieve the applications programmer from recoding interactive

operations common to all applications. This approach can provide higher quality interfaces at lower construction costs [Hayes 85].

By separating the application's user interface from its computational processes, the programmer is free to concentrate on the task at hand. Furthermore, this separation supports later changes to the user interface without affecting the application's internal processing [WIS/Command 86].

A UIMS serves to achieve one of the stated objectives of human factors -- consistency. Since different programmers employ only those interaction techniques supported by the UIMS, the user interface is consistent across all applications.

Several additional benefits are afforded by a UIMS [Olsen 84]:

a. MMI specifications can be represented, validated, and evaluated easily.

b. Designs can be rapidly prototyped and implemented.

c. Distribution of functionality across systems and processors is facilitated.

d. The proper roles of those involved in interface development are represented and supported throughout the evolution of the interface.

6.3.6.2 State of the Art

The set of tools that constitute a complete UIMS include [Foley 85; Olsen 84]:

a. Sequence control specification tool

b. Icon editor

c. Menu builder/screen formatter

d. Help/prompt message editor

e. Interaction techniques library

f. Evaluation tool

6.3.6.3 State of Practice

Most UIMSs constructed so far provide only a subset of the total functionality delineated in the previous section [Olsen 84]. Commercially available UIMSs include [Foley 85]:

a. UIMS - Unicad, Boulder, CO

b. Tiger/UIMS - Boeing Computer Services, Seattle, WA

c. Flair - TRW, Redondo Beach, CA

d. Blox - Rubel Software, Cambridge, MA

e. ADM - Apollo Computer, Chelmsford, MA

f. UIMS - Cadlinc, Elk Grove, IL

g. ENTER/ACT - Precision Visuals, Boulder, CO

6.3.6.4 Future Prospects

Given the immaturity of the science of human-computer interactions in general and User Interface Management Systems in particular, significant enhancements can be expected in future implementations of UIMSs. Advances in our understanding of human factors (Section 6.3.2) and in our technology for interaction (Section 6.3.3) will be integrated into future UIMSs.

6.3.6.5 Performers and Funders

The commercial construction of UIMSs is increasing in intensity. The demand for such systems is escalating due to the growing realization by software developers that a UIMS can provide higher quality interfaces at lower development costs. UIMS builders are responding by providing more sophisticated tools and more complete sets of tools. A list of the commercially available UIMSs is given in Section 6.3.6.3.

Research into the composition and effective presentation of UIMS tools is being conducted at a variety of institutions. Research-oriented UIMSs include [Foley 85; Olsen 84]:

a. SYNGRAPH/GRINS - Olsen, Brigham Young University

b. Menulay - Baecker and Buxton, University of Toronto

c. DMS/SUPERMAN - Hartsen and Ehrlich, Virginia Polytechnic Institution

d. Abstract Interaction Handler - Sibert, George Washington University

e. UIMS - Green, University of Alberta

f. REXX/FSX

g. COUSIN - Hayes, Carnegie-Mellon University.

6.3.7 Decision Aids

6.3.7.1 Issues

The list of functions to be performed by humans in the SDI environment (see Section 6.2) indicates that people will be involved in many command and control decisions. Large volumes of data would have to be assimilated and interpreted to make an informed response under time constraints. Decision aids may be employed to facilitate this decision process.

Decision aids do not replace human judgement. They aid judgement by providing a framework to organize the decision making process, thereby ensuring that decisions are logically consistent with values and beliefs. Decision aids are a collection of tools to enhance human decision making performance by [Smith 86]:

a. Identifying key factors

b. Structuring the decision process

c. Estimating observables

d. Weighing alternatives

e. Predicting outcomes

f. Managing information

g. Presenting data effectively

6.3.7.2 Future Prospects

The Defense Communications Agency was tasked to assess and project the impact of technology on C3 systems. In support of this effort, the Office of Naval Technology and the Office of Advanced Technology sponsored a workshop on Decision Support Technology on May 20-21, 1986 [Smith 86]. A panel of experts identified and ranked critical, high-leverage technologies to meet perceived operational needs and application opportunities.

6.3.7.3 Performers and Funders

The U. S. Army Strategic Defense Command is developing a Command and Control Decision Aids Test Environment. The program has three main objectives [Southern 86]:

a. Determine where decision aids are feasible in an SDI BM/C3 subsystem.

b. Apply decision aids in a test bed environment.

c. Provide for technology transfer to BM/C3 TVEs.

The Rome Air Development Center has a program underway to demonstrate the feasibility and utility of advanced computer technology decision aids and establish a technical baseline to support transaction to the operation world [Smith 86]. Their approach includes:

a. Define, design, and develop decision aids for Air Force battle management applications.

b. Create a decision and development support environment.

c. Utilize a battle management laboratory (fixed/mobile) to support evaluation, enhancement and transition.

RADC along with several other government Research and Development organizations are involved in an Adaptive Expert Decision Aiding Program. Their objective is to demonstrate techniques for providing aids which accomodate different user skills, styles and strategies [Smith 86].

The Office of Naval Research is developing graphic techniques to aid in situation assessment for SDI command and control decisions.

6.4 Recommendations

The foregoing assessment of software technology for the man-machine interface serves as a basis for the following recommendations. Their purpose is to assist the SDIO in making decisions about future directions for technical efforts. The recommendations suggest certain actions for the SDIO to acquire the relevant foundations of software technology for the SDI man-machine interface.

6.4.1 Human Factors

Recommendation 1: SDIO should continue its support of the Command and Control Human Interface Program at the Naval Research Laboratories.

The program should emphasize (1) monitoring human factors research, (2) identifying human performance requirements, (3) initiating a human factors TVE, (4) studying decision-making abilities under stress, (5) developing human factors standards for SDI, and (6) formulating an MMI evaluation methodology.

The human interface program should monitor research in human factors being conducted at numerous institutions and in various disciplines. General principles of human factors engineering that are relevant in the SDI environment may be utilized by tracking the state of the art without intervention.

Given the criticality of error susceptibility in SDI command decisions, the human interface program should study decision-making abilities under stressful circumstances. Research should be directed towards the unique SDI environment in which commanders are interacting during a simulated attack. This study should be based upon the results of similar research such as the NASA stress project.

The human interface program should develop standards for human factors in SDI computer systems. Such standards should be documented in a user interface style guide to facilitate the design and implementation of a user interface consistent across all SDI applications. The quality of alternative designs or of a proposed design may then be measured empirically to gauge how well it satisfies SDI requirements.

The human interface program should formulate a rigorous methodology for the empirical evaluation of an MMI design. An MMI should be accepted only after prototyping, simulation, experimentation, and redesign in an iterative manner. This entails early and continued involvement of the end users working under realistic (i.e., stressed) conditions.

6.4.2 Interaction Techniques

Given the uncertain nature of the man-machine interface in SDI computer systems, SDIO should postpone the selection of interaction techniques. As the requirements of SDI applications become more clearly defined, so too will their interfaces. At such time, a more informed decision can be made as to the most promising interaction techniques for the SDI MMI.

Given the enormous investment in the development of commercial workstations, SDIO should monitor the state of practice in interaction techniques. Then when required, commercial computer hardware and software that integrate the most sophisticated and useful interaction techniques may be acquired.

Recommendation 2: Because of the potential utility of a user interface that employs artificial intelligence techniques, SDIO should

monitor its state of the art and, in particular, the work sponsored by DARPA.

When the requirements of the SDI MMI are to be defined, AI interaction techniques including natural language interfaces, stroked character recognition, and speech understanding may be examined for suitability and utility in the SDI environment.

6.4.3 Workstation Managers

Recommendation 3: SDIO should monitor government development efforts for workstation managers, window managers, graphics packages, and user interface management systems.

Foundation technology for the MMI may be borrowed or developed jointly where applicable to the SDI environment. Projects such as WIS, STARS, and Automated System-level Computer Engineering Technology (ASCENT) should be studied for assimilation.

Once the Computer Graphics-Virtual Device Interface (CG-VDI) standard is officially released, commercial instantiations should be available for integration into the SDI environment. If extensions to the CG-VDI workstation manager standard are not forthcoming for window management, SDIO should consider funding such a project.

6.4.4 Graphics Packages

Recommendation 4: SDIO should monitor current standardization efforts for graphics packages.

Once PHIGS is standardized, the Department of Defense can be expected to sponsor the implementation of the Ada language binding. Such software technology is expected to be available for integration into the SDI environment.

6.4.5 User Interface Management Systems

Recommendation 5: SDIO should survey commercially developed and research-oriented UIMSs for adaptation and integration into the strategic defense system.

The most suitable and useful UIMS may then require enhancement to support functionality unique to the SDI man-machine interface.

6.4.6 Decision Aids

Recommendation 6: SDIO should continue to fund the U. S. Army Strategic Defense Command's program for a Command and Control Decision Aids Test Environment.

Their program is on track for the timely integration of decision aids into SDI foundation technology.

6.5 References

[Airchinnigh 85] Airchinnigh, M., "The Specification and Implementation of GKS Application Software in Ada," *Computer Graphics Forum* 3, 2 (June 1984), pp. 153-167.

[Atwood 84] Atwood, Michael E., "A Report on the Vail Workshop on Human Factors in Computer Systems," *Computer Graphics and Applications*, 4 *IEEE Computer Society* (December 1984), pp. 48-66.

[Badre 82] Badre, Albert and Ben Shneiderman, *Directions in Human/Compter Interaction*, Ablex Publishing Corporation, Norwood, NJ, 1982.

[Bailey 85] Bailey, Elizabeth K., *Human Engineering Impact on the STARS SEE Architecture*, IDA Paper P-1818, Prepared for the Office of the Under Secretary of Defense for Research and Engineering, Institute for Defense Analyses, Alexandria, VA, (April 1985).

[Bolt 84] Bolt, Richard A., *The Human Interface* , Massachusetts Institute of Technology, Lifetime Learning Publications, Belmont, CA, 1984.

[Bono 85] Bono, Peter R., "A Survey of Graphics Standards and Their Role in Information Interchange,"*Computer*, (October 1985), pp. 63-75.

[Bono 86] Bono, Peter R., "Guest Editor's Introduction, Graphics Standards," *IEEE Computer Graphics and Applications*, (August 1986), pp.12-16.

[Brown 85] Brown, Maxine D., *Understanding PHIGS*, Megatek Corporation, San Diego, CA, 1985.

[Card 85] Card, Stuart K., Thomas P. Moran, and Allen Newell, *The Psychology of Human - Computer Interaction*, Lawrence Erlbaum Associates, Hillsdale, NJ, 1983.

[Carlin 86] Carlin, Robert J., "Battle Management C^3 Information Flow," SDI Update '86 Conference, Washington, D.C., (April 11, 1986).

[Cooper 86] Cooper, Eric C., et.al., *Technology Identification and Asessment*, Carnegie-Mellon University, Pittsburgh, PA, (February 28, 1986).

[Cotellessa 84] Cotellessa, Robert F., *Identifying Research Areas in the Computer Industry to 1995*, Stevens Institute of Technology, Hoboken, NJ, 1984.

[DOD 82] *Strategy for a DoD Software Initiative*, Department of Defense, (October 1982).

[Eastport 85] *A Report to the Director Strategic Defense Initiative Organization*, Eastport Study Group, Summer Study 1985, (December 1985).

[Enderle 84] Enderle, G., K. Kansy, and G. Pfaff, *Computer Graphics Programming GKS - the Graphics Standard*, Springer-Verlag, Berlin, 1984.

[Foley 82] Foley, James D. and Andries Van Dam, *Fundamentals of Interactive Computer Graphics*, Addison-Wesley Publishing Company, Reading, MA, 1982.

[Foley 84] Foley, James D., Victor L. Wallace, and Peggy Chan, "The Human Factors of Computer Graphics Interaction Techniques," *Computer Graphics and Applications* (November 1984), pp. 13-48.

[Foley 85] Foley, J. and J. Grimes, Course notes on "How to Design Computer-User Interfaces" and "Advanced Topics - Human Factors in Computer Graphics Systems," Association for Computing Machinery's Special Interest Group on Computer Graphics, Twelfth Annual Conference and Exhibition on Computer Graphics and Interaction Techniques, San Francisco, (July 22-26, 1985).

[Hayes 85] Hayes, Phillip J., Pedro A. Szekely, and Richard A. Terner, "Design Alternatives for User Interface Management Systems Based on Experience with COUSIN," *Human Factors in Computing Systems*, Proceedings of ACM SIGCHI Confrence, San Francisco, (April 14-18, 1985).

[IEEE 85] "Special Report: The Strategic Defense Initiative," *IEEE Spectrum* 22, 9 (September 1985).

[Kappel 85] Kappel, Michael R., "An Ada Graphics Package," George Washington University Technical Report, Washington, DC, 1985.

[McCormick 84] McCormick, K.A. and Bleser, T.W., *Developing a User Interface Styleguide*, Computer Graphics Consultants, Inc., Washington, D.C., (December 1984).

[Moran 81] Moran, Thomas P., "An Applied Psychology of the User," *ACM Computing Surveys* 13, 1 (March 1981), pp. 1-2.

[Myers 84] Myers, Brad A., "The User Interface for Sapphire," *Computer Graphics and Applications* (December 1984), pp. 12-23.

[NASA 83] *NASA Computer Science Research Program Plan*, NASA Intercenter Planning Committee for Computer Science, NASA Technical Memorandum 84631, (March 1983).

[Norman 84] Norman, Donald A., "Four Stages of User Activities," *Interact '84*, IFIP, (September, 1984), pp. 81-85.

[Olsen 84] Olsen, Dan R., Jr., W. Buxton, R. Ehrich, D. Kasik, J. Rhyne, and J. Sibert. "A Context for User Interface Management," *Computer Graphics and Applications* (December 1984), pp. 33-42.

[Parnas 85] Parnas, David L., *Software Aspects of Strategic Defense Systems*, University of Victoria, Victoria, BC, 1985.

[Proceedings 85] *Proceedings of the Human Factors Society 29th Annual Meeting*, 1, 2, Baltimore, MD, September 29-October 3, 1985.

[RADC 85] *RADC Software Engineering for Strategic Defense Initiative*, Rome Air Defense Center, Rome, NY, 1985.

[Rich 84] Rich, Elaine, "Natural-Language Interfaces," *Computer* 17, 9 (September 1984), pp. 39-47.

[Smith 83] Smith, S.L. and A.F. Aucella, *Design Guidelines for the User Interface to Computer-based Information Systems*, Technical Report ESD-TR-83-122, Electronic Systems Division, Hanscom Air Force Base, 1983.

[Smith 86] Smith, Yale, "Report to the SDI BM/C3 Working Group," Rome Air Development Center, (July 8, 1986).

[Southern 86] Southern, John, "Report to the SDI BM/C3 Working Group," U. S. Army Strategic Defense Command, (July 8, 1986).

[Thalmann 79] Thalmann, D. and N. Magnenat-Thalmann, "Design and Implementation of Abstract Graphical Data Types," *Proceedings of the Third IEEE Computer Software and Applications Conference*, Chicago, IL, (November 1979), pp. 519-524.

[Thompson 85] Thompson, Craig W., "Menu-based Natural Language Interfaces to Databases," *Quarterly Bulletin of the IEEE Computer Society Technical Committee on Database Engineering* 8, 3 (September 1985), pp. 64-70.

[Turner et al 85] Turner, Robert D., David M. Yanni, and Harlow Freitag, *Functional Analysis of SDI Battle Management Processes*, BATMAN Strategic Defense Initiative Battle MAnagement and C^3 Project, Institute for Defense Analyses, Alexandria, VA, 1986.

[WIS/Command86] *Softare Requirements for the WIS Command Language Prototypes*, Paper P-1983, Vol. II, Institute for Defense Analyses, Alexandria, VA, 1986.

[WIS/DBMS 86] *Software Requirements for WIS Database Management System Prototypes*, WWMCCS Information System Task Force on Database Technology, IDA Paper P-1983, Vol. V, Institute for Defense Analyses, Alexandria, VA, 1986.

[WIS/Graphics 86] *Software Requirements for WIS Graphics Systems Prototypes*, WWMCCS Information System Task Force on Computer Graphics, IDA Paper P-1983, Vol. VIII, Institute for Defense Analyses, Alexandria, VA, 1986.

SECTION B7

Parallel Processing

Prepared by John Chludzinski and Cathy Jo Linn

Topics covered in Section B7:

7.1 Introduction
7.1.1 Background

7.2 SDI's Requirements

7.3 Current Status
7.3.1 Introduction
7.3.2 Architectural Survey
7.3.3 Languages
7.3.3.1 Implicit Parallelism
7.3.3.2 Explicit Parallelism
7.3.4 Language Processors
7.3.4.1 Detecting Parallelism
7.3.4.2 Static Scheduling of Resources
7.3.5 Retargetable Code Generation
7.3.6 Operating Systems
7.3.7 Algorithms

7.4 Recommendations
7.4.1 Introductions
7.4.2 Languages
7.4.3 Ada Compilers for Parallel Processing Engines
7.4.4 Retargetable Code Generation
7.4.5 Operating Systems
7.4.6 Research Algorithms for Parallel Processing

7.5 References

7.6 Bibliography

List of Tables

Table	Title	Page
I	Summary of Architectures	148

7.0 PARALLEL PROCESSING

7.1 Introduction

The purpose of this document is to discuss parallel processing and its relation to SDI's needs. This technology is just beginning to emerge and is consequently represented by many differing and competing design philosophies. The situation is further complicated by the fact that many of the machines competing for attention are merely paper machines, hence their touted advantages are difficult to evaluate. As many of these machines move from paper to realization, it will be possible to evaluate their architectures and choose those which offer the greatest promise.

7.1.1 Background

It is possible to partition computer programs into individual tasks, of which data independent subsets may be executed in parallel on seperate processing elements. Early examples of parallel processing machines include the Cray 1 and CDC's Cyber. Since these early efforts, there have been many advances in the techniques used to analyze programs to detect parallelism and in the architectural support for parallel execution of tasks.

7.2 SDI's Requirements

It is widely believed that SDI's needs for signal processing, image processing and missile tracking will be extensive. These needs require extremely fast computational engines to process the vast quantities of data supplied by sensors. As stated in the report *Elimination of the Threat Posed by Nuclear Ballistic Missiles* [Fletcher 84], the computational requirements could be as large as 600×10^6 computer operations/second (600 MOPS). Later studies have pointed to even higher levels of computation power that may be required for the SDI effort.

It is now apparent that the next generation of chips will be facing physical limitations if used in single CPU architectures. To achieve any higher performance it will be necessary to look past this limiting view of processing to parallel processing architectures. This form of processing has great potential but opens a whole range of problems not encountered at the single CPU level.

To make effective use of parallel processing technology, the following must be accomplished:

a. Languages that express inherent parallelism and compilers that discover parallelism must be developed.

b. Operating systems must be developed that effectively manage the resources of parallel architectures.

c. New algorithms must be developed that make effective use of parallel architectures.

A caveat, at this point, is in order. Parallel processing within the context of the SDI effort should be viewed as a "necessary evil" and consequently approached with a dose of caution. The difficulties arise due to SDI's necessarily large emphasis on fault-tolerance.

The support of parallelism potentially complicates both the hardware and the software of the system and hence may make the system, as a whole, less fault-tolerant.

7.3 Current Status

7.3.1 Introduction

Parallel processing is an immature technology. Only recently has research been initiated into the problems presented by this approach to processing. However, these architectures have received a great deal of attention in recent years as demonstrated by NSF's Supercomputing Centers and DARPA's Supercomputing Research Projects. The former represents an effort to make this technology more widely available and the later represents an effort to push the development of these machines. The Rome Air Defense Command (RADC) is currently funding work to compare data processing architectures. Additionally, the Naval Research Laboratory is funding an effort to combine numerical and symbolic processing on a hybrid parallel architecture.

In this section the current status of the field of software for parallel processing is reviewed. The software issues of parallel processing are divided into four major categories: languages, compilers, operating systems, and algorithms. The sections covering these areas will develop the concerns unique to parallel processing and present current technologies used to solve these problems. To aid in discerning the relevant software issues and to provide a basis for addressing current technology, a brief survey of the wide variety of parallel architectures is presented first.

7.3.2 Architectural Survey

The following survey of machine architectures is used to discern the issues relevent to parallel processing. The machines in this survey represent a broad spectrum of parallel architectures; each represents a solution to the problems of parallel processing and each has a particular theater of applications within which it strives to succeed. These architectures exploit parallelism at different levels of granularity, ranging from the fine grain (single instruction) to the coarse grain (task). In this paper a task is an entity with its own independent thread of control, such as a UNIX process or an Ada task. Table I, presented at the end of this section, provides a more detailed outline of these twelve different architectures. To assist the reader in drawing connections between these machines, a simple categorization scheme is provided:

a. Switched shared-memory -- machines which access a common memory through a switching network,

b. Networked distributed-memory -- machines which distribute their memory throughout the network of processing elements as local memory to each of the processing elements,

c. Hierarchical-memory -- machines which provide both local and global memory to the processing elements,

d. Single processor -- machines which have a single control of flow during program execution,

e. Data flow -- machines designed to behave as Petri nets, where the availability of arguments initiates ("fires") an operation, and

f. Machine building blocks -- processing engines that may be configured in many possible architectures.

7.3.2.1 Switched Shared-Memory

The NYU Ultracomputer has N identical processing elements and N shared memory modules. The synchronization primitive of this machine is built into the interconnection network and provides support for parallelism among tasks [Gottlieb 83].

The Butterfly Parallel Processor also consists of multiple interconnected processors with shared memory. The synchronization primitive of this machine, however, is built into the memory control units; thus, its interconnection network is faster than that of the Ultracomputer [Crowther 85].

The Delencor HEP machine is a hardware-multi-programming machine. Each (of 16) processing execution modules (PEM) selects a task status word from its queue and fetches the next instruction to be executed in that task. The instruction is then assigned to one of several functional units available to that PEM. Similar queues exist for memory access and I/O operations. This unique approach supports parallelism at the task level not only between multiple PEMs but also within each PEM as tasks are switched at each instruction. In addition, the functional units of each PEM are pipelined, providing another level of parallelism [Gajski 85].

The Cray X-MP is a representative of vector architectures. It allows parallelism at the instruction level for individual tasks through the use of multiple function units, data pipelining, and multiple processor-to-memory data paths. The X-MP also allows parallelism using 2 or 4 CPUs along with a shared memory and hardware semaphores [Cray 84].

7.3.2.2 Networked Distributed-Memory

The Goodyear Massively Parallel Processor (MPP) is an array of 16K single bit processing elements. Each processing element has 1Kbits of local memory. The Array Control Unit (ACU) broadcasts both instructions and addresses to all processing elements resulting in the same operation being performed by each processing element on the same local memory. Each single bit processing element performs the specified operation on n-bit values by the execution of multiple single bit operations. A staging memory, in the I/O path, acts as a buffer between the array unit and outside world, reformating data so that both the the array unit and the outside world can transfer data in an optimal format [Batcher 82].

The Caltech Cosmic Cube has 26 processing elements connected by a 6-dimensional hypercube network. Each of the nodes is comprised of an Intel 8086 with an Intel 8087 floating point coprocessor and 128K of random access memory along with 8K of read only memory. Processing elements communicate with nearest neighbors via queued message passing along the arcs of the cube at a rate of 2Mbits per second [Seitz 85].

The Connection Machine®, produced by Thinking Machines Corporation, has a network of 64K processing elements (cells). Each processing element is composed of a one bit CPU and a local memory of 4K bits. The processing elements are connected by a 16-dimensional hypercube network. The Connection Machine's 64K processing elements are divided into four sections, each controlled by a seperate microcontroller, allowing four separate front end machines to concurrently use the Connection Machine [TMC 86].

7.3.2.3 Hierarchical Memory

The Cedar machine consists of a global control unit for task synchronization and scheduling, global memory modules, and clusters of identical processing elements with local memory for each cluster. A task force (group of related tasks) is assigned to a cluster by the global control unit. Each cluster assigns and schedules the task to the processing elements of the cluster, providing parallelism at the task level [Emrath 85].

7.3.2.4 Single Processor Machine

The Yale ELI (Enormously Long Instruction) was designed to exploit the high performance of horizontally programmed engines. In this machine, parallelism is provided by compacting multiple register-to-register level instructions into very long (horizontal) instructions [Fisher 82].

7.3.2.5 Data Flow Machine

In dataflow machines, such as the Tagged-Token Dataflow Machine, a node of the program flow graph is executed on an available functional unit when the data needed for the execution becomes available. Multiple functional units provide parallelism at the level of granularity specified by these nodes. If the functional units are pipelined, then another level of parallelism is provided [Gajski 85].

7.3.2.6 Machine Building Blocks

The CMU Programmable Systolic Chip and the INMOS Transputer were both designed to be the building block for large arrays of processors. These types of systems are designed for special purpose problems, such as matrix operations, that can be done by a large number of processors all performing the same operation. The level of parallelism supported by these systems is specified by the level of the operations performed by each processor of the array. The operations performed by an INMOS Transputer are typically at a larger level of granularity (task level) than those performed by the Programmable Systolic Chip (instruction level) [Treleaven 85].

From this survey it is apparent that different architectures support differing granularities of parallelism. Each attempts to exploit parallelism at some level, but each has a unique set of costs associated with exploiting this parallelism. The networked distributed-memory architectures are usually cheaper to construct than the switched shared-memory architectures. However, because the shared-memory architectures support the efficient sharing of both programs and data, they are better suited to tightly-coupled, highly-parallel processing than are the distributed-memory architectures. Hierarchical-memory architectures, hybrids between distributed-memory and shared-memory architectures, allow processors and/or processor clusters to cache memory, thereby compromising between the ability to share programs and data and the speed of accessing local memory. The cost of this compromise is increased complexity of memory management. The data-flow architectures attempt to achieve tightly-coupled, highly-parallel processing with an architecture that departs radically from the von Neumann model. The ELI, which is the sole representative of the single processor class, represents an attempt to exploit parallelism at the micro-architecture level by using a radical compiler technology. Four points should be apparent from this survey:

a. Language mechanisms for expressing parallelism, both implicitly and explicitly, will be needed,

b. Compilers, which target these architectures, will be essential to the broad application of this technology,

c. Unique problems will arise in operating systems design, e.g., memory management, and

d. Machine architectures will dictate which algorithms will run efficiently.

These issues are examined in the following sections.

7.3.3 Languages

The success of the parallel approach to processing will inevitably depend upon the amount of parallelism that can be exploited. All too often the language of implementation hides potential parallelisms or even worse, introduces data dependencies into the program that are not inherent in the algorithm being implemented.

These concerns have not escaped the attention of the parallel processing community. Differing strategies have arisen to exploit as much parallelism as possible based on different languages. The efforts fall into two basic groups: (1) languages which express parallelism at an implicit level and (2) languages which express parallelism at an explicit level. Imperative languages, such as FORTRAN, will be included in the former. Even though they were not designed with parallel architectures in mind, these languages do not require a sequential execution of statements to preserve program semantics.

7.3.3.1 Implicit Parallelism

Implicit parallelism within a language may be detected by the compiler, using sophisticated data flow techniques, or it can be expressed in a manner where the parallelism is more easily detectable. FORTRAN, ALGOL, and Pascal are examples of the former; they have an explicit total ordering of statements, where the data dependencies only impose a partial ordering. FORTRAN programmers, in an effort to be space economical, are also prone to introduce data dependencies not inherent in the algorithm. A clever compiler, in most cases, can detect the proper partial ordering of statements and remove unnecessary data dependencies.

Functional languages and the data flow languages also express parallelism implicitly but in a way that is hoped to be easily detectable. The properties of data flow languages include:

a. Freedom from side effects

b. Locality of effects

c. Equivalence of instruction scheduling constraints with data dependancies

d. "Single assignment"

e. No static variables within the language

f. Referential transparency

The data flow community hopes to exploit these properties to gain more parallelism than what they believe is possible with imperative languages, e.g., FORTRAN, ALGOL, Pascal. To date there is little evidence to support this thesis. The functional semantics,

where a variable represents a value and not a storage address, combined with the "single assignment" aspect of these languages, makes storage management, i.e., garbage collection, a major problem. An additional issue is that detecting the maximum amount of parallelism possible within these languages seems to employ most of the same techniques used with imperative languages; thus the hypothesis that the parallelism would be easy to detect may not be true.

Functional programming languages are touted by some in the parallel processing community as offering greater opportunities for exploiting parallelism than conventional languages. With these languages, programming consists of defining and applying functions to objects (which may themselves be functions) to produce other objects.

In addition to the functional properties listed above for data flow languages, this genre of programming offers the opportunity to use demand driven application and an extension of this concept, lazy evaluation. With demand driven application, an expression, which is bound to free occurrences of a variable, delays evaluation until its value is required. Going one step further, lazy evaluation evaluates only as much of the result as is needed, the remainder of the result is packaged in a "recipe" for further evaluation as needed. Both of these strategies are only possible due to lack of side effects in functional programming. This evaluation scheme is conducive to parallel processing, allowing processors, as they become available, to evaluate recipes when needed. A further benefit is that with lazy evaluation, infinite objects are realistic data objects.

Two principal representatives of functional programming languages are LISP, based on Church's lambda calculus, and FP, based on Curry's combinatory logic. Although LISP, which has a wide following in the artificial intelligence community, is not a pure functional language (containing serial constructs, e.g., COND), it has most of the attributes of this class. FP, created by John Backus, one of the original authors of FORTRAN, was designed with parallel processing in mind. It incorporates such properties as:

a. Easy transformation of recursion into iteration

b. Clear hierachical structure of programs

c. No explicit statement sequencing or explicit changing of specific memory locations

7.3.3.2 Explicit Parallelism

Languages which provide for explicit parallelism usually do so at a coarse level of granularity, the task level. Languages in this category include Ada, Concurrent Pascal and PL/1. Ada is unique, in that tasks (concurrent processes) are declared objects, whose parent task cannot exit their scope of visibility without the declared tasks first terminating. This feature of tasking provides for more predictable behavior but it limits the behavior of Ada tasks.

The greatest difficulty for this level of concurrency is walking the tight rope between cooperation and interference. Two or more tasks cooperate when they interact (communicate) to achieve a common goal. Interference occurs when two or more tasks contend for the same resource. Usually this contention problem is resolved by some arbitration mechanism. It is the communication mechanisms, used to achieve cooperation, and the arbitration mechanisms, used to resolve contention, that are the subject of continuing research.

The ability to specify parallelism at a finer level of granularity, usually at the statement level, is provided by a few languages. FORTRAN 8X, APL and Occam are examples of this class. FORTRAN 8X allows the programmer to explicitly operate on arrays without the usual DO loops. Other FORTRAN dialects include the DOALL construct for parallel execution of the loop iterates. With Occam, the programmer has at his disposal the constructs to explicitly state the order of execution of statements:

SEQ for sequential execution of statements,
PAR for parallel execution of statements, and
ALT chooses an arbitrary statement from a list for execution.

Although the Cray X-MP provides both high speed semaphores and limited scalar communication between CPUs, vectors must be passed through shared memory. Hence, the ability of the Cray X-MP to support small-grain/tightly-coupled parallelism has been questioned. To overcome these difficulties microtasking was introduced in [Calahan 83] and [Calahan 85]. Calahan expresses the view that small-grain, tightly-coupled (microtasking) parallelism on the X-MP will require low level (assembly) language programs to maintain the concurrency between the control and numeric functions, which places a barrier to the wide spread use of the X-MP for this purpose.

7.3.4 Language Processors

The ability to program in a high-level language and produce efficient machine code from the high level source is essential to the long term success of parallel processing. This places the onus on compiler writers to construct a tool that produces code that effectively uses the machines resources. From the point of view of the compiler writer, there are two unique concerns for producing efficient code for a parallel architecture: (1) detecting parallelism and (2) static scheduling of resources (functional units, registers, etc.).

7.3.4.1 Detecting Parallelism

Exploiting parallelism requires (1) realizing (detecting) the inherent data dependencies, (2) avoiding the introduction of any additional data dependencies, and (3) removing data dependencies not inherent to the program. Machine architecture greatly influences the granularity of parallelism that a language processor seeks to reveal. The Parafrase compiler, developed at Illinois is a source-to-source FORTRAN compiler. It seeks to reveal a fine level of parallelism by performing machine independent transformations. These include scalar renaming, loop normalization, forward substitution, and subscript expansion. The granularity of parallelism that is actually used depends on the architecture of the target engine.

7.3.4.2 Static Scheduling of Resources

Assignment of machine resources (functional units, etc.) may be done (1) staticly, at compile time or development time (using special constructs within the language), or (2) dynamically, at run time by the operating system. In this section, the issues of static assignment and scheduling are reviewed. Static assignment and scheduling have the advantage of being a one time expense, but the disadvantage of being far from optimal in the case of a poor prognosis of the run-time behavior. The granularity of the units to be scheduled influences some of the details of scheduling but the general problem applies across all level of granularity. Scheduling at different levels of granularity is referred to in various ways including microcode compaction, code generation, and task scheduling. The following are examples of these three types.

a. Microcode compaction

Bulldog, the compiler designed for ELI at Yale, exploits the massive amount of parallelism in the data path of the Very Long Instruction Word (VLIW) architecture by clustering sequential register-to-register level instructions into horizonal microinstructions (> 500 bits). This is accomplished by a trace scheduling algorithm that performs code compaction (scheduling) on long loop-free streams of code (traces), using such techniques as code motion, software pipelining, and anti-aliasing memory references.

b. Code generation

With proper scheduling of instructions, the Cray X-MP's fourteen functional units can process many instructions in parallel. The compiler should generate code (schedule instructions) in a manner which will preserve data dependencies between instructions while maximizing the parallel execution of instructions.

c. Static task scheduling

The compiler for the UC Berkeley MIDAS architecture [Maples 85] schedules tasks by scheduling memory modules within a cluster to switch between different classes of processing elements within the same cluster. Classes of processing elements are defined by the programmer. Each class is programmed to perform a subtask best suited to that classs of processing elements.

7.3.5 Retargetable Code Generation

With the current plethora of machine architectures, it is desirable to have some mechanism for retargeting code generation. This would provide a vehicle for experimentation with differing architectures resulting in a more objective evalution of machine capabilities and liabilities. It would also be useful for examining the potential bottlenecks of different algorithms executing on different machine architectures.

The difficulty with retargetable code generation is seperating the machine specific considerations from the generation of code. This becomes exceedingly difficult when code optimization is added as a requirement. This problem is more difficult to overcome with parallel architectures than with single threaded architectures, because parallel architectures varying more wildly. These differences may be significant enough to invalidate the very assumptions used in the optimizers. Hence, the optimization phases, which are already complex, will need to be retargetable, resulting in a quantum leap in the difficulty of producing retargetable code generators for parallel architectures.

7.3.6 Operating Systems

The operating system considerations for a parallel architecture present some unique challenges: (1) dynamic task assignment, to make more optimal assignments of tasks to processing elements; (2) synchronization, to coordinate the sharing of resources in a parallel environment; and (3) memory management, to coordinate memory accesses and prevent a bottleneck.

Task Assignment and Scheduling

Whereas static task assignment and scheduling, as discussed in the previous section, are once incurred expenses, the overhead for dynamic task assignment and scheduling is constantly being incurred at run time. The dynamic methods can be centralized or distributed. A centralized strategy has a single scheduler making assignments, while the distributed assignment strategy uses many schedulers distributed across machine resources to make task assignments.

Denelcor's HEP uses a centralized assignment scheme, which assigns each task to a PEM. Each PEM then schedules the individual instructions of tasks assigned to it in a FIFO fashion. The problem with the centralized assignment strategy is the possible bottleneck that can occur when the number of processing elements gets large.

Task scheduling on the Cedar machine is a two layer event. A simple low-level, round-robin scheduler runs on each processor. A high level scheduler is evoked when a processor goes idle. An array of busy/idle processor flags is maintained along with a queue of ready tasks. The high-level scheduler checks to see if enough processors can be allocated to a related group of tasks and assigns the first such group.

Synchronization

Synchronization is the means by which multi-programming/parallel-processing environments coordinate and manage the sharing of machine resources. One method for synchronization is the use of shared variables. A shared variable may be used to indicate a state, which would either allow or disallow a subtask to continue, an example being semaphores. Shared variables may also be used to pass messages between subtasks to coordinate activities. Shared variables can be a bottleneck because of simultaneous accesses to shared memory. The NYU Ultracomputer circumvents this dilemma by *distributing the synchronization primative (Fetch&Add) throughout the entire* interprocessor network, including the memory modules.

Message passing is another mechanism for synchronization. The Cosmic Cube uses message passing to synchronize the sharing of machine resources (64 processing elements) between the various active processes. A consequence of this mechanism is that messages must be routed through the interconnection network resulting in time penalties in the case of long paths between processing elements. Thus, there is an incentive to minimizing the length of paths between any two processing elements within the network.

The Cray X-MP uses a special technique called chaining to provide the data synchronization required for vectorization of such operations as:

$$\Sigma_{i=1,n} ((a_i + b_i) * c_i)$$

In this operation the multiplication of vector elements can overlap, in time, the addition operation, thereby speeding up the operation as a whole.

Data flow machines use special techniques for control synchronization. Tokens flow along the arcs of the control graph, tagged by their corresponding input value. A node is only executed when all incoming arcs have tokens.

Memory Management

Many parallel processing architectures have developed memory hierarchies to avoid the bottleneck of accessing into a global memory. In the Cedar project there are two levels of

memory, memory local to a given cluster and memory global to all clusters. Memory management on the Cedar is assisted by hardware support in the form of a page mapping mechanism for each processing element. The problems incurred by this architecture include: (1) the requirement of notifying all processing elements when one processing element writes a virtual page back to global memory and (2) the confusion that will occur if multiple programs share the same memory map, hence the same virtual address space.

The Cedar project's operating system is an enhanced version of Berkeley Unix 4.2, implemented in C++. Within this system, the address space of a task is divided into pages. Each page is marked by an attribute indicating whether the page resides in global or local memory and whether the page is read-only or read-write. Global pages may be private to a single task, shared amongst a proper subset of tasks or shared amongst all the tasks of a task force. Local pages can be shared between tasks only if those tasks are executing on processing elements within the same cluster.

Global pages may be cached in local memory. The cost of swapping in a page from global memory is balanced against the gain of reduced access times. To guarantee the coherency of the global memory, a system call is used to uncache the page when a global cached page is to be updated.

7.3.7 Algorithms

The need exists to research algorithms that exploit parallelism. In particular, the need exists to look at data dependencies inherent in particular algorithms and examine how bottlenecks might develop. Additional emphasis must be placed on research into algorithms which minimize synchronization, recognizing both the difficulties and the performance gains inherent to this approach. To exploiting the full capacity of these architectures, it is essential to tailor the communication requirements of the algorithm to the specific target architecture. Similarly, reconfigurable architectures, that can accommidate the communication needs of a broad spectrum of algorithms, need to be researched.

The work to date has emphasize the theory of the subject, which all too often does little to advance the practical state-of-the-art. Much of the work has concentrated on theoretical machines, which possess such unrealistic properties as a polynomial number of processing elements. These assumptions are used in obtaining equally unrealistic asymptotic performance estimates. A greater emphasis should be placed on practical results modeled on real(istic) machines with their inherent liabilities (costs). The Strategic Defense Command is currently funding work to develop candidate set of algorithms for weapon to target pairing for late midcourse and terminal phases, including work to implement and evaluate selected algorithms in realistics environments. Appendix B, Section 2 covers the R&D related to BM/C3 Algorithms/Processors at greater length.

7.4 Recommendations

7.4.1 Introduction

The recommendations for this subject are general in nature due to the immaturity of the subject. An additional problem arrises due to the difficulty of extricating purely software issues from hardware (architectural) concerns.

7.4.2 Languages

Ada has been choosen by the DoD as the required implementation language for embedded applications. The inevitable consequence of this will be a broad range of support tools and a growing body of programmers with Ada expertice.

Recommendation 1: In the short term, Ada should be the principal language for the development of software for parallel architectures. In the long term, basic research to develop of languages better suited to the expression of parallelism, both explicitly and implicitly, should be pursued.

7.4.3 Ada Compilers for Parallel Processing Engines

Recommendation 2: With the current emphasis on the development of Ada compilers, SDI should piggy-back on such efforts and develop Ada compilers targetted to parallel processing engines.

A warning should accompany this recommendation. Whereas Ada and the environments developed to support Ada offer great promise for the implementation of large scale software projects, complications will be introduced by exception handling (side-effects that are impossible to predict). The problem is not unique to Ada, but is common to all languages which allow user defined exceptions and exception handlers.

7.4.4 Retargetable Code Generation

With the plethora of parallel machine architectures it would be advantageous to be able to retarget code generation for different architectures without developing a new back-end to a compiler for each new architecture. Another potential and more immediate benefit of such an effort would be the development of an architectural description language to be used as input to a "code optimizer generator", which would automatically build optimizers for various architectures. Such a language would aid both the development and classification of parallel processing engines.

Recommendation 3: SDI should pursue the development of retargetable parallel code generators, with a particular emphasis on the development of architectural description languages for parallel architectures.

7.4.5 Operating Systems

Recommendation 4: In the short term, SDI should pursue the development of operating systems to support the development of code for parallel processing engines.

To study the concerns of the operating systems designer, i.e., scheduling and task assignment, memory management, and synchronization, it will be necessary to support a facility where parallel architectures are made available (see recommendation 5).

7.4.6 Research Algorithms for Parallel Processing Systems

It is important that SDI support the investigation into algorithms that exploit parallelism. The performance of an algorithm is directly influenced by the architecture of the system. It is possible that the system's data communication paths may become bottlenecks for one algorithm, whereas another algorithm for the same purpose would not produce a

bottleneck. Tailoring an algorithm so that its resource requirements can be well supported by the system's architecture will be paramount to exploiting the full potential of these systems. It is important that software developers have the ability to investigate the performance of particular algorithms on particular architectures. This need could be supported by simulation or by use of an actual instance of an architecture.

Recommendation 5: SDI should support the development of facilities to allow investigation into parallel algorithms. These facilities should be coordinated with those of the National Test Bed and those of the NSF and DARPA's Super Computing Centers, as well as the Algorithm Test Center recommended in Section 2, Appendix B.

7.5 References

[Batcher 82] Batcher, K.E., "Bit-Serial Parallel Processing Systems," *IEEE Transactions on Computers*, (May 1982).

[Calahan 83] Calahan, D.A., *Task Studies Solving Linear Algebra Problems on a Cray-class Multiprocessor*, Report SARL #2, Dept. of Electrical Engineering and Computer Engineering, University of Michigan , (December 1983)

[Calahan 85] Calahan, D.A., "Task Granularity Studied on a Many-Processor Cray X-MP," *Parallel Computing*, (June 1985)

[Cray 84] Cray Research Inc., Cray X-MP Computer Systems, *Four-Processor Mainframe Reference Manual,* HR-0097, 1984.

[Crowther 85] Crowther, W., Goodhue, J., Gurwitz, R., Rettberg, R., Thomas, R., "The Butterfly™ Parallel Processor," *IEEE Computer Architecture Technical Committee*, (September/December 1985).

[Emrath 85] Emrath, Perry, "Xylem: An Architecture for the Cedar Multi-processor," *IEEE Software*, (July 1985).

[Fisher 82] Fisher, Joseph A., *Very Long Instruction Word Architectures and the ELI-512*, Yale University, Department of Computer Science, Research Report #253, (December 1982).

[Fletcher 84] Fletcher, James C. et al., Report of the Study on Eliminating the Threat Posed By Nuclear Ballistic Missiles, (February 1984).

[Gajski 85] Gajski, D. D. and Peir, Jih-Kwon, "Comparison of Five Multi-processor Systems," *Parallel Computing*, (November 1985).

[Gottlieb 83] Gottlieb, A., Grishman, R., Kruskal, C., McAuliffe, K., Randolph, L., Snir, M., "The NYU Ultracomputer -- Designing an MIMD Shared Memory Parallel Computer," *IEEE Transactions on Computers*, (February 1983).

[Maples 85] Maples, Creve, "Analyzing Software Performance in a Multi-processor Environment," *IEEE Software*, (July 1985).

[Seitz 85] Charles L. Seitz, "The COSMIC Cube," *Communications of the ACM* 28, 1 (January 1985).

[Treleaven 85] Treleaven, Philip C., and Refenes, Apostolos N., "Fifth Generation and VLSI Architectures," *Future Generation Computing Systems*, (December 1985).

[TMC 86] Thinking Machines Corp., Introduction to Data Level Parallelism, Thinking Machines Technical Report 86.14, (April 1986)

7.6 Bibliography

Ackerman, William B. "Data Flow Languages," *IEEE Computer*, (February 1982).

Arya, S., "An Optimal Instruction-Scheduling Model for a Class of Vector Processors,"*IEEE Transactions on Computers*, (November 1985).

Blazewicz, J., Drabowski, M., and Weglarz, J., "Scheduling Multi-processor Tasks to Minimize Schedule Length," *IEEE Transactions on Computers*, (May 1986).

Gajski, D. D., Kuck, D. J., Padua, D., Kuhn, R., "A Second Opinion on Data Flow Machines and Languages," *IEEE Computer*, (February 1982).

Ganapathi, M., Fisher, C., Hennessy, J., "Retargetable Compiler Code Generation," *ACM Computer Surveys*, (December 1982).

Ghezzi, Carlo "Concurrency in Programming Languages: A Survey," *Parallel Computing*, (November 1985).

Harrison, P., and Khoshnevisan, H., "Functional Programming Using FP," *BYTE*, (August 1985).

Haynes, Leonard S., Lau, R., Siewiorek, D., Mizell, D., "A Survey of Highly Parallel Computing," *IEEE Computer*, (January 1982).

Hockney, R.W., "MIMD computing in the USA - 1984," Parallel Computing, (June 1985).

Kogge, Peter M., "Function-Based Computing and Parallelism: A Review," *Parallel Computing*, (November 1985).

Mehlhorn, Kurt, *Graph Algorithms and NP-Completeness*, Springer-Verlag, 1984.

Szalas, Andrzej and Szczepanska, Danuta, "Exception Handling in Parallel Computations," *SIGPLAN Notices* 20, 10 (October 1985).

Tremblay, Jean-Paul and Sorenson, Paul G., *The Theory and Practice of Compiler Writing*, McGraw-Hill, 1985.

Table I. Summary of Architectures

NYU Ultracomputer

- N identical processing elements and N memory modules with shared memory
- Omega/message passing switch interprocessor network connecting processing elements to memory modules
- O(N log(N)) switching elements with access time logarithmic in N
- Fetch&Add atomic primative used for synchonization of operating system primatives and applications built into interconnection network
- simultaneous memory references to same memory cell results in a small degradation of performance

---

Butterfly™ Parallel Processor

- composed of a series of identical processor modules connected by a high performance switching network
- each processor module contains a processor and local memory
- the system is the union of all local memory within the individual modules
- synchronization primatives in the memory central unit
- each processor module contains:
  - MC68000 processor
  - at least 1 Mbyte of memory
  - message co-processor (Processor Node Control)
  - memory management hardware
  - I/O bus
  - interface to the Butterfly switch
- each Butterfly switch is a 4 input - 4 output custom VLSI chip
- the Butterfly switch network uses packet switching

Cedar/ University of Illinois at Urbana-Champaign

- processing divided-up into processor clusters
- each cluster consists of a number (currently 8) of identical processing elements, local memory modules, a cluster control unit and a local switch network linking local memory to the processing elements
- a global control unit exists to assign groups of processes to clusters and to perform process synchronization
- global memory modules are linked to the individual cluster via a global network (packet switched)

Goodyear MPP

- 16 K single bit processing elements
- network topology is North-South-East-West

Table I. (Continued)

- an Array Control Unit to control processing in the individual PE's in the array, manage the inflow/outflow of data through the array and run the main applications program
- Staging Memory to act as the buffer between the array unit and the external world and to format (incoming and outgoing) data for optimal use
- Program & Data Management Unit (DEC PDP-11) controls the overall flow of the program and the data to the MPP

Delencor HEP

- hardware-multi-programming machine designed for high speed performance on scientific calculations
- 16 processing execution modules (PEM) connected to 128 memory modules
- each PEM contains:
  - a process queue
  - a memory queue
  - I/O queue
  - several pipelined functional units
- process synchronization done by hardware
- each PEM picks up a process status word from its process queue, fetches the next instruction to be executed in that process, and assigns the instruction to the appropriate functional unit

---

Cray X-MP

- vector processor
- 1, 2 or 4 CPUs, depending upon the particular configuration
- each CPU has 5 vector functional units, 3 floating point functional units, 4 scalar functional unit and 2 address functional units
- functional units have long pipelines
- multiple processor to memory paths
- word length = 64 bits
- each CPU has 8 vector registers with 64 words per register
- each CPU has 8 scalar registers
- each CPU has 4 instruction buffers; each buffer can hold 128 consecutive16-bit "instruction parcels"
- the CPUs communicate and synchronize via five shared cluster of registers

---

Tagged-Token Dataflow Machine

- identical processing elements consisting of an ALU, a waiting-matching element and an I-structure storage (input storage for the next "code block")
- tokens (messages) sent asychronously
- manager coordinates execution of program

Table I. (Continued)

- program flow graph clustered into "code blocks", each of which is part of the flow graph representing the job to be performed
- code block initiated by sending all necessary tokens to assigned physical domain(s)
- code blocks are executed in parallel on same or different "physical domains" (sets of processors)

Yale ELI

- designed to exploit the high performance of horizonally programmed engines
- compiler technique called trace scheduling used to compact register-to-register level instructions into very long horizonal microinstructions
- 500+ bit instruction word
- executes 10-30 RISC level instructions per clock cycle
- single central control unit
- 8 M-clusters -- each with: local memory, integer ALU, multiport integer register bank limited cluster bar
- 8 F-clusters -- each with: floating point ALU, multiport floating register bank, limited cluster bar

---

CMU Programmable Systolic Chip

- 64 x 60-bit micocode dynamic RAM and microsequencer
- 64 x 9-bit words DRAM register file
- ALU
- multiplier-accumulator
- three input and three output ports
- communication between the above mentioned units transpires via three buses
- the building block of programmable systolic arrays
- arrays of these chips allow multiple operations on a single piece of data, with communication paths optimized to coincide with the data paths inherent in a particular algorithm

---

INMOS Transputer

- the building block for large multi-processor arrays
- engines cooperate in a fashion made explicit by the programmer
- 32 bit words
- each processor element (PE) has N-S-E-W links
- each PE has 2K of local memory
- six registers:
  - three-register evaluation stack
  - instruction pointer register
  - workspace pointer register
  - operand register

Table I. (Continued)

- RISC level instructions
- 32 bit multiplexed memory interface

CalTech Cosmic Cube

- 64 processing elements, each with:
  - Intel 8086
  - Intel 8087 floating point coprocessor
  - 128K bytes random access memory
  - 8K bytes read only memory
- processing elements connect with a 6-dimensional hypercube

Connection Machine by Thinking Machines Corp.

- 64K processing elements, each with:
  - 1 bit CPU
  - 4K of bit memory
- processing elements connect with a 16-dimensional hypercube
- 64K processing elements divided into four section each with a seperate microcontroller, allowing four separate front end machines to use the Connection Machine concurrently

SECTION B8

Computer Systems Engineering Technology

Prepared by Deborah Heystek

Topics covered in Section B8:

8.0 Computer Systems Engineering Technology

8.1 Introduction
8.1.1 Purpose and Scope
8.1.2 Relationships to Other Areas
8.1.3 Organization of Section

8.2 SDI Requirements
8.2.1 Functionality and the Required Facilities
8.2.2 SDI Overall Goals and Objectives for
Computer Systems Engineering Technology
8.2.2.1 Adaptability
8.2.2.2 Performance
8.2.2.3 Testability
8.2.3 Summary of SDI Requirements

8.3 Current Status
8.3.1 Introduction
8.3.2 Requirements Specifications
8.3.3 Design Specifications
8.3.4 A Paradigm for System Design
8.3.5 Current Work in Computer System Design
8.3.5.1 ADAS
8.3.5.2 EIS
8.3.5.3 TSD
8.3.5.4 SARA
8.3.5.5 SADT
8.3.5.6 TAGS
8.3.5.7 SREM
8.3.5.8 NRL
8.3.5.9 RADC
8.3.5.10 DARPA
8.3.6 Summary of Current Status

8.4 Recommendations
8.4.1 Summary of Recommendations

8.5 References

List of Figures

Figure	Title	Page
1	Binding Option	161

8.0 COMPUTER SYSTEMS ENGINEERING TECHNOLOGY

8.1 Introduction

8.1.1 Purpose and Scope

This section examines the technology for Computer Systems Engineering that will be needed during the design of computer systems for the Strategic Defense Initiative (SDI). The benefits to be gained by varying the allocation of the system functions between hardware, microcode, and software are discussed to illustrate that performance, reliability and fault tolerance, integrity and security, flexibility and cost may all be impacted by considering how functions are implemented. The technology addressed would support system design, tradeoff analyses among hardware, microcode, and software, and provide specifications for implementations in the resulting allocated manner.

8.1.2 Relationships to Other Areas

The Computer Systems Engineering Technology area is closely related to a number of the other areas. Section 9 of Appendix B, Software Engineering Environments, contains a discussion on software development methodologies. This information, while relevant to this section, will not be repeated. A discussion on software reliability and survivability is provided in Section 11, Appendix B. Section 7, Appendix B, Parallel Processing, discusses various parallel processing architectures and the software implied by these architectures.

8.1.3 Organization of Section 8

Section 8.2.1 discusses the types of computer capabilities needed by the SDI. An explanation of the expected functionality of each type is provided. Section 8.2.2 outlines the required or desired characteristics for SDI computer systems. Section 8.3 reviews the current status of computer systems engineering as it applies to SDI needs. Section 8.4 provides funding recommendations for further work in computer systems engineering.

8.2 SDI Requirements

8.2.1 Functionality and the Required Facilities

The computer systems needed to provide the functionality required by SDI may be partitioned into two separate systems. This partitioning is based upon which aspect of the SDI project the computer system is designed to support. The project will have a development support system and an operational system [Eastport 85]. The computer systems needed to support each of these will have different requirements.

The type of computer system supporting the development of the SDI will be a ground-based, distributed system. The sites will be interconnected via communication networks. The system will consist of computing facilities for the development of both software and hardware. A software/hardware development environment consisting of an integrated set of tools and design management facilities, such as databases, should be provided. In addition, the computer systems supporting SDI development will provide simulation facilities consisting of both the computer hardware and simulation software. The simulation facility, which will provide multi-level simulation capabilities, will be used to debug and verify the design of the operational computer system. It will also be used to

evaluate the operational computer system under different loads and under circumstances of system faults or component failure.

The second type of computer system, supporting the operational phase of the SDI may be distributed on ground, air and space based platforms. To meet performance requirements, functions will be partitioned and allocated to specific platforms. The varied nature of the platform environments will impose different performance, reliability and design constraints on the computer systems. Thus, the individual computer systems will be designed for the specific types of platforms, based on their environment and processing requirements [Kenny 86].

The operational SDI computer system may be viewed conceptually as two distinct systems. These are a) platform systems, and b) platform management systems. The platform systems are ground, air and space based hardware, microcode, and software to support battle management processing, signal processing and communications processing. The platform management system will consist of space- and ground-based hardware, microcode, and software to control and coordinate the platform systems. This division corresponds roughly to the division between battle management, and command and control.

8.2.2 SDI Overall Goals and Objectives for Computer Systems Engineering Technology

Three goals and objectives that have been identified -- adaptability, performance, and testability -- for the SDI computer systems are particularly relevant to the technology being discussed. As has been stated already, the specific requirements of adaptability, performance and testability will vary among the platforms and subsystems. Generally, system requirements are more critical in the case of the operational systems than in the case of the developmental systems. For example, space-based computer systems will require tighter constraints on size, computing power, reliability and fault-tolerance [Kenney 86].

8.2.2.1 Adaptability

The research being considered for the SDI is directed toward developing a defense system that is:

a. Potentially long lived,

b. Based on exploiting advancing technologies, and

c. Beyond the existing military experience and doctrine [Eastport 85].

Based on these characteristics, adaptability of the SDI computer system is not only necessary, but one of the most fundamental requirements. The SDI computer system must be capable of continuous evolution while remaining in service. This evolution is projected to occur over a longer period of time than for most other defense systems. Some of the architectural features which contribute to the adaptability of a system will now be discussed.

In order to allow the insertion of unanticipated new components, the computer system must be extensible. Extensions to the system may take the form of changes in the deployments, such as replicating a component or enhancing the capabilities of components. An "open system" concept of design is envisioned to provide extensibility [Eastport 85]. A modular design scheme, applied to both the hardware and software components, will facilitate the

goal of designing and building an extensible system. In such a scheme, a system is composed of well-defined building blocks with standard interfaces. This will facilitate the addition, removal or replacement of modules.

As a result of the longevity of the system, the ability of the system to adapt to changing needs is vital. Advances in technology may lead to improvements in performance, reliability, cost effectiveness or testability. System requirements may change due to upgraded countermeasures or changes in treaties and threats. Allowing the system to respond to advancing technologies and changing requirements may require the capability to migrate functions between hardware, microcode, and software.

8.2.2.2 Performance

Performance requirements of the SDI computer system are difficult to quantify. The Eastport Group Study Report states that, "the number and fraction of warheads intercepted in an all-out attack is a rough measure of the systems performance" [Eastport 85]. Typically, however, performance will be inferred from the results of small scale tests. Characteristics which contribute to system performance include speed, reliability, durability, hardware/software interaction and fault tolerance.

8.2.2.3 Testability

Testability is measured by the confidence with which one can determine the correctness of functionality and the performance of a deployed system [Eastport 85]. Short of war, full scale testing of the SDI System will not be possible. Confidence in the system may still be possible by structuring the system so that correctness and performance may be inferred from less than full scale tests. These tests may take the form of technical tests or military exercises, and may be accomplished through high- or low-level simulations.

8.2.3 Summary of SDI Requirements

In order to best meet the functional requirements of the SDI, two types of computer systems, a developmental support system and an operational system, will be needed. The operational system may be viewed conceptually as consisting of platform systems and platform management systems.

The attributes of adaptability, performance and testability need to be considered for each computer system in the SDI separately, as each is unique. The requirements and constraints of each system will vary, based upon what aspect of the SDI the computer system is designed to support, the environment within which the computer system is designed to operate, and the functions performed by the computer system.

8.3 Current Status

8.3.1 Introduction

Until recently, computer system design was typically driven by the high cost of hardware. Software was used to help reduce the functional requirements of the hardware, thus minimizing the use of hardware in the design. This design philosophy and the increasing size of systems and applications programs has lead to increasingly complex software. This complex software tends to be poorly supported by the hardware with which it interacts because the software designers were being forced to compensate for inadequacies in the hardware [Toy 84]. This has lead to systems which have suboptimal performance, are expensive to produce, and even more expensive to maintain [Turn 78].

Such results and the increasing cost of software development have led to an increasing emphasis on software. Some computer system designs have been constrained by the need not to obsolete large investments in existing application software. This has resulted in software-driven system designs.

It has become apparent that the feasibility of designing, testing and maintaining software depends on the system architecture -- that, in fact, the design of the software should be an integral part of the entire design process and should not simply be considered after the rest of the design is complete. In this section, the process of specifying system requirements and formulating design specifications is discussed. This is followed by a presentation of a system design paradigm. This is followed by a review of current development efforts and results directed at supporting the total system design process.

8.3.2 Requirements Specifications

The process of system or component design begins with the definition of a set of requirements. Requirements are precise statements of desired functional and performance characteristics, independent of any implementation details. Requirements engineering involves systematic, iterative analysis of needs, documentation, and verification and validation of the resulting requirements. In order to specify requirements, a conceptual model of the environment within which the system or component is to operate is constructed. The internal state and behavior (functional requirements) of the proposed system are then described in relation to the environmental model. Non-functional requirements (constraints) are then added to the description. Non-functional requirements are typically difficult or impossible to specify formally. These cover a wide range of issues including interface, performance, operating, life-cycle, economic and political constraints [Roman 85].

Requirement specifications may be classified according to the following criteria [Roman 85]:

a. Formal foundation - the theoretical basis for the specification (data flow, finite-state-machine, stimulus-response path, communicating concurrent processes).

b. Scope - the type of requirements (functional or non-functional) addressed.

c. Level of formality - machine interpretability of the specifications.

d. Degree of specialization - the size of the problem domain for which the technique is appropriate.

e. Specialization area - the type of entities that are specifiable.

f. Development method - the approach used to construct the requirements.

Some of the desirable properties in requirement specifications will be discussed. The appropriateness of requirement specifications refers to their ability to capture, in a straightforward, implementation independent manner, the concepts relevant to the proposed system or component. Requirement specifications should be precise, complete, consistent, and lack ambiguities. Highly formalized specifications enhance analyzability. Testability, the verification of the design, is the most important characteristic of specifications. However, testability is the most difficult property to achieve. Traceability of the requirements to the design is necessary, as system design is not necessarily an implicit

statement of the specified requirements. Traceability indicates the ability to cross-reference entities in the requirements specifications with entities in the design specifications. Executability refers to the extent to which functional simulation of the specifications is possible. Finally, since the definition of requirements specifications is an iterative process, the ability of the specifications to permit modifications, adapt to these changes and tolerate incompleteness is important [Roman 85].

It would be inappropriate to state requirement specifications in terms of the implementation technology used to realize any portion of the design since the implementation decisions are only made after the design specifications are generated from the requirement specifications. Thus, requirement specifications should be stated at a system level, rather than as hardware, microcode or software requirement specifications.

8.3.3 Design Specifications

Many methods and languages/notations for the specification of software designs exist. Some of these are discussed in Section 9, Software Engineering Environments. Similarly, the methods and languages/notations for the specification of hardware designs are equally as numerous. However, these methods tend to be different than the ones used for software design specification. This is due to the specialized nature of the methods and languages/notations. Hardware specification methods and languages/notations will not be discussed in this section however, as the seperate treatment of software and hardware specifications through the use of different methods and languages/notations is inappropriate from the standpoint of integrated system engineering. A system-level specification method and language/notation, independent of implementation technologies, is needed so that design allocation decisions are not made prematurely.

The goal of an integrated approach to system engineering is to consider and design the potential hardware, microcode , and software components which constitute the system. To achieve this integrated design approach, the design specifications must be at a system level, independent of any eventual implementation. System-level design specifications allow the designer to begin early implementation technology (hardware, microcode and software) tradeoff analysis. Design specifications are generated, either automatically or manually, from requirement specifications. Design specifications should be traceable to the requirement specifications from which they were produced.

Some of the desirable characteristics in a system-level design specification method and language include: support for design description at multiple levels of abstraction, support for description of timing, behavior, structure and function, support for simulation of the design for verification, unambiguous specification of designs of VHSIC-level complexity, readability, ease of learning and use, support of varied design methodologies (top-down, bottom-up and mixed), extensibility and hardware and software independence.

The degree to which requirements and design specification languages are standardized will impact the design, development and maintenance of computer systems. Standard specification languages would facilitate the design process in many ways, including increasing communication capabilities among designers and between designers and manufacturers, increasing sharing and transfer of information, increasing portability and reusability, increasing productivity as a result of decreased designer retraining efforts, and better tool and design environment development.

8.3.4 A Paradigm for System Design

The design of a computer system may be characterized as consisting of two phases: a "system decomposition" phase resulting in a collection of modules; and a "module to implementation technology binding" phase. During the system decomposition phase, the computer system to be designed is decomposed into subsystems or modules. These subsystems may then be partitioned further. This process is shown on the left side of Figure 1.

The way in which the system is decomposed imposes a solution to module binding on the eventual design. Techniques to aid in this decomposition have been addressed in various software engineering environments and are discussed in Section 9, Appendix B.

The issue of concern here is the second step in the system design phase; the binding of the resulting functional modules to a particular implementation technology, as shown in the right side of Figure 1. This binding can take place early in the design decomposition, late in the design decomposition or even after the system is developed (during system execution). In the remainder of this section the choice of implementation technology available for binding, as well as the time at which the binding can take place, is discussed further.

Implementation Technology

A hierarchy of implementation technologies exists, including a range of implementations within the three level -- software, microcode, hardware -- hierarchy. For instance, software implementations include high level languages (HLL), assembly languages and machine languages. Microcode may be implemented as vertical, two-level or horizontal microcode. Hardware options range from gate arrays to uncommitted transistors.

A basic design input would be the technology base options available to realize the design. Power requirements, heat dissipation, I/O pin count, communication delays and layout all affect the implementation technology which may be used [Allison 77, Fairbairn 82]. SDI computer systems have stringent radiation hardening requirements. This requirement may only be satisfied by either a narrow range of possible implementation technologies, or the use of shielding.

A second basic design input is whether the available hardware and software will be off-the-shelf or reusable, customized or custom-designed. Tradeoffs to be considered include, for example, that custom hardware and software may be potentially faster, but may be a higher risk due to a shorter history of use [Roman 84].

Design Binding Time

Two fundamental approaches exist to the question of when it is appropriate in the design stage to bind to a particular implementation technology. These approaches are referred to as early and late binding strategies.

The difference between an early and a late binding strategy is the degree of system decomposition that occurs. In a late binding strategy, the system to be designed is decomposed further before binding occurs than in the early binding strategy. The modules resulting from the successive partitioning are at the level of functional primitives.

In an early binding strategy, the binding to an implementation technology occurs early in the design decomposition process. This typically occurs when a designer recognizes an implementation technique-specific solution. An early binding is less flexible than a late binding.

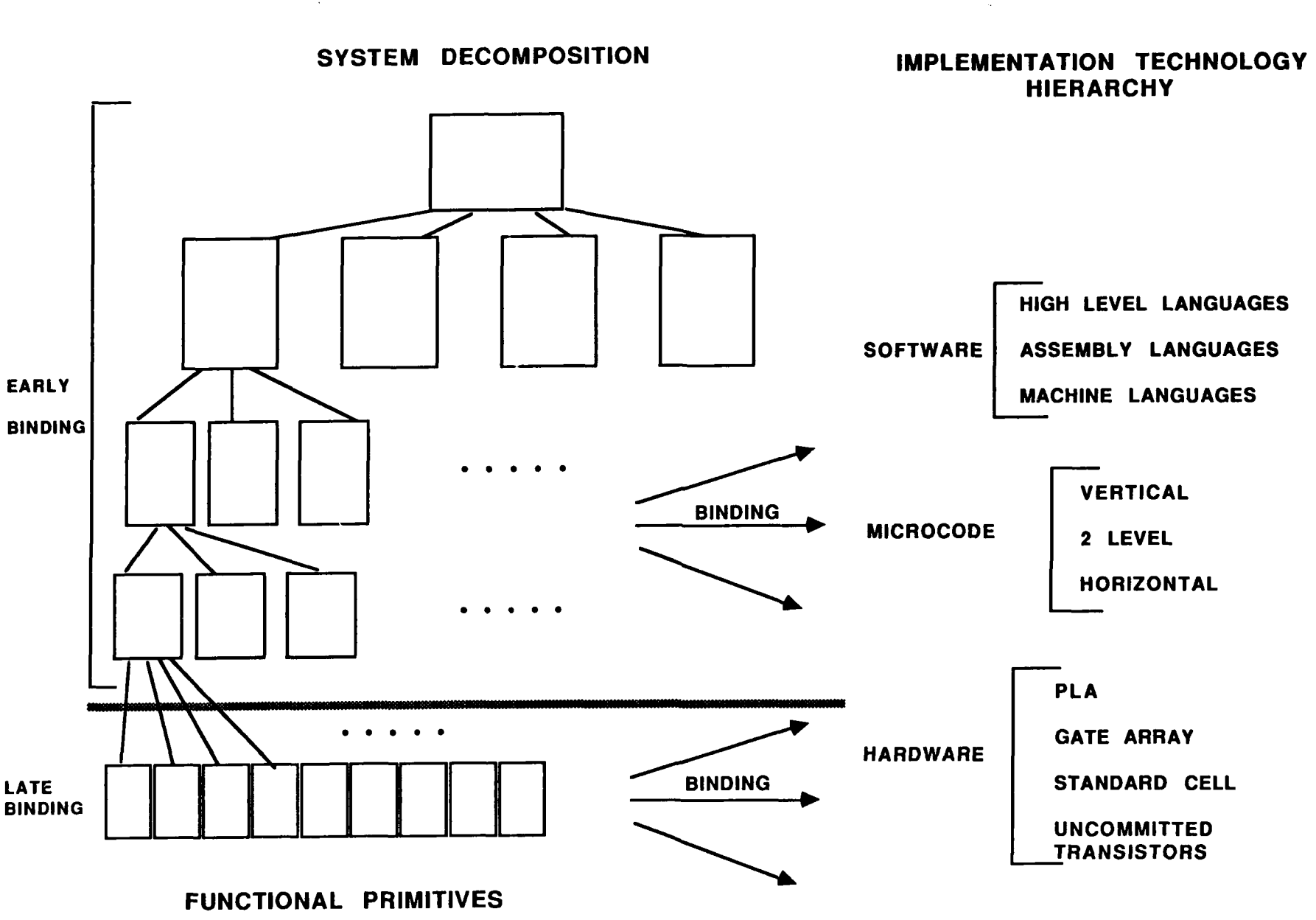


FIGURE 1. BINDING OPTION

Execution Binding Time

Decisions of how to allocate functional primitives may be deferred until after the system is built. Microprogrammable processors are an example of such a system. Functions can be migrated between software and microcode by reloading the microcode. This binding can be done in a maintenance time frame, under the control of the system designer or it can be done automatically during execution, as the system determines that certain functions should be relocated for optimum performance [Winner 86]. Reconfigurable networks are another example of systems that delay binding until after the system is developed. Processes may be moved from one node in a network to another and links between nodes may be changed.

8.3.5 Current Work in Computer System Design

Generally, design systems attempt to provide the designer with a design methodology and a design environment. A design environment may be characterized as consisting of a project management facility, a data management facility and a toolset [Katz 83].

The design environment determines, to a certain extent, the design methods which may be used. The spectrum of design methods ranges from computer-aided design, to expert system based design, to automatic generation of a design from a high level problem description [Gajski 83]. Computer-aided design, providing support for design evaluation, refinement, optimization and constraint propagation, offers the potential for high quality, reliable designs. Although expert system based design has been demonstrated, the technology does not yet support the design of entire systems. Thus, it is not possible, within the time constraints, to expect this design method to provide complete solutions to the SDI design problem. However, expert system based design may be appropriate for the design of selected components of the system. Automatic design synthesis (generation) is well beyond the current state-of-the-art, and therefore will not be considered in the design of SDI systems.

The deficiencies in existing system design methods, that is, the independent design of system hardware and software components, and the need for an integrated (codesign) approach to the design of computer systems have been recognized. As a result, several improved design systems are now being developed. This section reviews seven systems that have attempted to provide facilities needed for an integrated approach to the design of computer systems. Software design methodologies, languages and toolsets are discussed in the Software Engineering Environments (SEE) section. Included in that section is an explanation of Yourdon's methodology, SARA, SADT and SREM. Some of these systems have been either partially or completely extended to include support for the concept of codesign of hardware and software, and for this reason have been included in this section. However, a complete repetition of material covered in the section on Software Engineering Environments is not necessary or desirable.

8.3.5.1 ADAS

The Architecture Design and Assessment System (ADAS), developed at Research Triangle Institute, is a tool to support the codesign of the hardware and software components of a system. ADAS allows the modeling of both algorithms and architectures as directed graphs.

The ADAS system is based on a top-down, iterative refinement design methodology consisting of a description, verification and an assessment phase. A high-level design is successively decomposed, and at each level of decomposition the resulting components are

refined. The description, verification and assessment methodology is applied to each level of refinement. The nature of the ADAS system methodology is to encourage design tradeoffs early in the design process even if these tradeoffs are based only on approximate information. As more information is gathered, analyses are repeated.

The ADAS Computer Aided Design (CAD) system consists of a set of visually oriented tools which support the design methodology. The graph editor, EDIGRAF, allows the designer to represent the software and hardware as a hierarchy of directed graphs. A consistency checker, CONCH, is invoked to evaluate the graph for possible inconsistencies. The petri net analyzer, XPETRI, or the interactive petri net simulator, GIPSEM, are provided to the designer for design verification and performance analyses. ADAS also provides a hardware description language (HDL) interface between the system-level hardware /software architecture and the VLSI chip design. The HDL interface program generates an HDL description of the system from descriptions of the nodes on the hardware graph and other information in the graph. An HDL simulator may then simulate this description. Interfaces to ISPS and HELIX also exist [Smith 85].

8.3.5.2 EIS

The VHSIC (Very High Speed Integrated Circuit) Program Office has initiated an effort to develop an EIS (Engineering Information System). The goal of the EIS program is to (1) provide a set of reference specifications which will form the basis of a standardization effort, and (2) produce a prototype system to demonstrate the requirements.

The proposed EIS is a set of services and specifications which will provide a framework for supporting engineering functions. The EIS will allow tool integration, thus facilitating efficient, uniform use of tools with differing data and hardware requirements. Communication and translation of data among different hosts and tools will be supported. The EIS will allow the engineering process to be monitored and controlled. All types of engineering information will be managed through the capabilities supported by the EIS. Fundamental characteristics of the EIS are that it be tailorable, evolvable and portable [Linn 86].

8.3.5.3 TSD

The Total System Design (TSD) framework supports the development of integrated system design methodologies. These methodologies emphasize the importance of considering both the hardware and software during the design of a system. The underlying goal of the TSD is to decrease the number of possible binding options available to the designer in order to simplify the component selection process. The TSD framework is based on a series of predefined design stages. These stages are organized such that they represent a natural structuring of the design process and present the designer with a top-down approach to the problem. The framework may be tailored to a specific application and used to develop finely tuned methodologies [Roman 84].

8.3.5.4 SARA

The System Architects Apprentice (SARA), developed at UCLA, is a system to support the development of concurrent systems. SARA is currently a prototype design environment. Increasingly complex computer systems, particularly concurrent processing and distributed systems, have necessitated the development of a design philosophy and tool framework for the construction of formal models supporting automated design. This design approach is intended to reduce the discrepancy between the designer's intent and the actual system behavior.

The UCLA design methodology is based on modeling the behavior of the design within its assumed environment. Thus, input to the design framework is a set of system requirements and assumptions about the environment within which the system will operate. System requirements are specified at an abstract level in order to defer the design implementation decisions. The design methodology explicitly supports several principles thought to contribute to the development of improved designs. In order to allow software-hardware design tradeoffs, the design methodology supports the concurrent design of software and hardware systems. The design and reuse of embedded systems and modules is enhanced through the support for modeling the environment within which the system or module is assumed to operate. Alternative subsystem designs are permitted through the support of separate structure and behavior modeling. Top-down design, which contributes to high-level design verifiability, is supported. Building block composition, or bottom-up design is also supported. This method of design permits smaller units of the design to be created and tested, and later combined to form the complete system, or subsequently reused. Finally, the design methodology supports the designer in the use of tools.

Predefined subsystem components, are examined to determine if they fulfill the requirements of the system. If a set of these components does exist, they are selected and interconnected. This process is called composition. Decisions are made at this time about how the components are to be implemented. The behavior of the individual components is used to verify the design. If a set of components which will fulfill the requirements does not exist, the system is partitioned further. Each of the subsystems is then examined to insure that together they meet the overall requirements. If all requirements are met, composition or partitioning of these subsystems continues. If the requirements are not met, the design process backs up a step, and the system is partitioned again. Thus, this methodology allows both top-down and bottom-up design.

SARA, the system that supports this design methodology is composed of two fundamental types of tools, system support tools and methodology tools. The system support tools include system libraries and utilities. The methodology tools include the structural and behavioral modeling tools. The structural modeling tool allows the designer to interactively describe multi-level system structure. The primary behavioral modeling tool, Graph Model of Behavior (GMB), allows designers to create graphs modeling the flow-of-control and the flow-of-data. The topologies of these graphs are fixed, and allow static behavior modeling. Dynamic behavior modeling is also possible. A simulation environment is provided to allow timing analysis, deadlock detection analysis and predictions of results in the GMB [Estrin 78].

8.3.5.5 SADT

SofTech has developed a methodology to allow designers to describe, manipulate, document and check elements in the problem domain. This methodology, called SofTech's Structured Analysis and Design Technique (SADT), supports expression of design concepts. SADT is based on the principle of limitation as a means by which a designer may cope with the complexity of the proposed system. Thus, the methodology implies a hierarchical, top-down approach to the structured analysis of systems.

SADT consists of two fundamental tools; a "box-and-arrow" graphic notation language of structured analysis, and a design technique. The structured analysis language provides a mechanism to express the thought process which constitutes the structural analysis. This language is designed to be simple, readable and general enough to be applicable to any problem. However, the SA language has yet to be formalized.

Using the notational language, a model of the proposed system may be constructed. The design technique provides a disciplined approach to design , that results in the effective use of the structured analysis language. SADT may be coupled with other design methodologies as part of a total system development environment. By coupling SADT with other methodologies, all life-cycle phases of system development and maintenance including, design, performance analysis, rapid prototyping, simulation and optimization may be supported. Version control, project libraries, project organization and modeling archiving may be supported. SADT has been shown to be particularly effective in early and late stages of system development [Ross 85].

8.3.5.6 TAGS

Technology for the Automated Generation of Systems, developed by Teledyne Brown Engineering, Inc., is a computer-aided software development and system specification methodology and toolset. The TAGS system consists of an Input/Output Requirements Language (IORL), a system/software computer-based tool system and a TAGS methodology.

The Input/Output Requirements Language is a graphics and tabular specification language used to identify and model each software, hardware, embedded or non-embedded component of the system being designed. These components are modeled using graphic symbols borrowed from systems theory. Using this notation, the control and data flow of each component is indicated separately. Sievert and Mizell state that the IORL enforces a methodology for system development, is applicable to the specification of all types of systems (not just computer systems), is easy to use, and supports the expression of performance characteristics and algorithms using mathematical notation [Sievert 85].

The TAGS system provides four software tools to support the system development process; a Diagnostic Analyzer, Simulation Compiler, Configuration Manager and a Storage and Retrieval package. The Diagnostic Analyzer may be invoked to check diagrams created by IORL for static errors, such as syntax errors. The Simulation Compiler accepts Ada templates generated by the Diagnostic Analyzer, produces additional Ada code to link these templates into Ada packages, and allows simulation of the proposed system through the execution of the Ada packages. Thus, simulations provide a method by which dynamic errors in the IORL diagrams may be detected. Simulation also permits system performance evaluation and experimentation with design optimization. The Storage and Retrieval tool is a package which provides archiving and recall capabilities to the on-line TAGS tools.

The TAGS methodology, which is designed to implement the automation-based paradigm, consists of four fundamental activities; conceptualization, definition, analysis and allocation. During the conceptualization stage, the user requirements are developed into a conceptual model of the system to be built. The conceptual model is then defined in terms of the functional and performance requirements. The model is analyzed to determine if it is an accurate description of the proposed system. Finally, during the allocation phase, the functional and physical requirements are partitioned and assigned to appropriate subsystems [Sievert 85].

8.3.5.7 SREM

Software Requirements Engineering Methodology, has been developed by TRW for the Army Strategic Defense Command to accurately capture requirements specifications. The motivation for the development of SREM was that the hierarchy of functions model

typically used to define specifications led to inaccurate, ambiguous requirement specifications.

SREM consists of three fundamental components: RSL, a language in which to state requirements; REVS, a requirements validation system; and the SREM methodology. RSL, originally developed to state software requirements, is based on a set of elements, attributes, relationships and structures. REVS is a tool set used to translate the requirements language, perform automated consistency and completeness analyses, produce documentation and generate simulators. The tools have six categories of functions:

a. Extension - to tailor the RSL to a particular application.

b. Translation - of the RSL to an automated database.

c. Analysis - of the database for consistency and completeness.

d. Extraction - of information from the database.

e. Generation - of functional analytical simulators from requirement specifications.

f. Generation and display - of graphical descriptions of the requirements networks.

The SREM methodology facilitates the generation of requirements through the use of the RLS and the tools contained in REVS.

SREM is based on a model of a highly structured finite-state-machine. SREM has been extended into SYSREM, System Requirements Engineering Methodologies to permit the definition of system requirements. The basic approach taken to support this has been the extension of SREMS finite-state-machine model to allow the representation of decomposition and concurrency. System requirements may be defined in terms of actions occurring over an interval of time, which are iteratively decomposed and the resulting fragments allocated to system components. All activities are decomposed to the stimulus-response level. SYSREM has been used in a project to develop a distributed computing design system (DCDS). The aim of the project was to develop and integrate design language and tools with the SYSREM system. This has resulted in the ability to detect faults and allocate functions to distributed systems, thus permitting the design of concurrent processing. The resulting system provides a methodology to trace requirements to design specifications [Alford 85]. The Army Strategic Defense Command plans to establish the requirements in February 1987 for a system to succeed DCDS that explicitly emphasizes the inclusion of both software and hardware design called CoDES (Computing Development System).

8.3.5.8 NRL

NRL is currently funding research in the area of system integration and engineering. This work includes experiment design effors to establish the feasibility of autonomous battle management systems.

8.3.5.9 RADC

RADC is currently funding research in the area of formal verification technology. As part of this work, RADC is funding the designing, developing and demonstrating tools, techniques and methodologies for formal verification systems.

RADC is also funding research in the area of trusted system design. This work emphasizes exploiting state of the art technologies in the development of tools and techniques tobe used int he design of secure systems.

8.3.5.10 DARPA

DARPA is funding work on an Ada Engineering environment. This research will result in the delivery of an Ada Engineering Environment based on an Object Management System Running on MACH.

8.3.6 Summary of Current Status

For many years, the approach to computer systems design has been *ad hoc*. Components were designed in isolation and later integrated to form a complete system. Software, in particular, has almost always been designed on top of the hardware, and not as an integral part of the entire system. With the increasing size and complexity of computer systems, this approach is neither desirable nor feasible. A method in which all system components are considered and designed together is necessary.

In an integrated system design paradigm, system design proceeds in two stages. Initially, the system to be designed is decomposed into subsystems or modules. At some point during the decomposition decisions are made about how to implement the resulting modules. A hierarchy of implementation technology options exists. Deferring implementation decisions until late in the decomposition stage or even until after deployment gives the system increased flexibility. Furthermore, a late binding strategy provides the potential for improving performance. Fault tolerance may be enhanced through the use of a late binding strategy which allows vertical function migration and reconfigurability.

Several systems which support an integrated approach to the design of computer systems have been discussed. A single illustration of work in the area of system-level requirements capture, the SYSREM extension of SREM, was given. A great deal of research in this area is still necessary. A discussion of TAGS was used to illustrate some of the issues being addressed in the area of system-level design specification. Other work in this area, for example, VHDL and its environment, is also occurring. A great deal of effort is being directed at the problem of inadequate design environments. Of the systems being developed to solve this problem, a small number are addressing the issue of codesign. These include TSD, ADAS, EIS, SARA, and CoDES.

8.4 Recommendations

System requirements, which specify the desired functionality in terms of performance, reliability, and constraints such as size, power and weight, are the starting point in the system design process. Although a great deal of work in analysis techniques is occurring, a method leading to well formed requirements still does not exist. Thus, the SDIO should support research to develop a method of system-level requirements capture.

Recommendation 1: In the near-term, the SDIO should examine existing and ongoing work in system-level requirements capture. Methods specifically for either software requirements or

hardware requirements capture should be examined to determine whether this work may be extended to accommodate the other aspect of the system. Based upon the quality and applicability of existing or ongoing work to SDIO needs, the SDIO should either provide continued funding for these projects, or initiate a project to construct prototypes for system-level requirements capture.

A design specification language serves as a means of precise communication between designers, influences the approach to the problem, and determines the design and analysis tools to be used. Thus, a specification language plays an important role in system design. Since the hardware/software design tradeoffs are made from the system level down, a system-level specification language is appropriate. Due to the complexity, size, and requirement to verify the design of the SDI system, the specifications must be traceable to the requirements. The SDIO should support research leading to a system-level specification language that is neutral to eventual design implementations.

Recommendation 2: In the near-term, the SDIO should examine existing and ongoing work in system-level specification languages/notations. Methods specifically for either software specification or hardware specification languages/notations should be examined to determine whether this work may be extended to accommodate the other aspect of the system. Based upon the quality and applicability of existing or ongoing work to SDIO needs, the SDIO should either provide continued funding for these projects, or initiate a project to construct prototypes for system-level specification languages/notations.

Recommendation 3: In the far term, following the development of a reasonable set of requirements capture and specification language prototypes, the SDIO should support research aimed at beginning to develop automatic verification (design meets requirements and specification) capabilities.

An integrated design environment consisting of project management facilities, design data management facilities and design tools is necessary to support the system development process.

Recommendation 4: In conjunction with the work outlined in Recommendations (1) and (2), the SDIO should examine the tools and engineering environments provided for using the requirements capture and design specification languages/notations. Based upon the quality and applicability of these tools and environments to the SDIO needs, the SDIO should either provide funding for continued work or support the construction of engineering environment and tool prototypes.

In particular, the design environment must provide support for multiple design alternatives, multiple function to implementation assignments, multiple levels of design representation and allow propagation of design changes across representations. Design tools must support automated system decomposition, automated assignment of functions to levels within the implementation technology hierarchy, simulators to assist in tradeoff analysis,

and transformation of representations such as microcode compilers and gate array compilers.

Recommendation 5: Based on the above recommendations, the SDIO should make policy decisions on the level of requirements specification language standardization, design specification language standardization, and engineering environment standardization.

8.4.1 Summary of Recommendations

An integrated approach to the design of computer systems is needed. This will require the development of design support environments and integrated toolsets. Input to the design system should be a system-level, implementation independent specification generated from and traceable to a set of system requirements. Thus, in addition to an integrated design system, a method to derive this input is required.

8.5 References

[Alford 85] Alford, M., "SREM at the Age of Eight; the Distributed Computing Design System," *IEEE Computer* 17, 4 (April 1985), pp. 36-45.

[Allison 77] Allison, D., "A Design Philosophy for Microcomputer Architecture," *IEEE Computer* 10, 2 (February 1977), pp. 16-22.

[Eastport 85] "A Report to the Director Strategic Defense Initiative Organization," Eastport Study Group, (December 1985).

[Estrin 86] Estrin, G., Fenchel, R., Razouk, R., and Vernon, M., "SARA (System Architects Apprentice): Modeling, Analysis, and Simulation Support for Design of Concurrent Systems," *IEEE Transactions on Software Engineering* SE-12, 2 (February 1986), pp. 293-311.

[Fairbairn 82] Fairbairn, D. "VLSI: A New Frontier for Systems Design," *IEEE Computer* 15, 1 (January 1982), pp. 87-96.

[Gajski 83] Gajski, Daniel and Kuhn, R., "New VLSI Tools," *IEEE Computer* 16, 12 (December 1983), pp. 11-14.

[Kenny 86] Kenny, W., "Computer Systems" in *SDI Large-Scale Systems Technology Study*, Final Report, Army SDC, (March 28, 1986).

[Katz 83] Katz, R. "Managing the Chip Design Database," *IEEE Computer* 16, 12 (December 1983), pp. 26-36.

[Linn 86] Linn, J., Winner, R., et al., *The Department of Defense Requirements for Engineering Information Systems (EIS)*, Volume I and II, IDA Paper P-1953, (July 1986).

[Roman 84] Roman, G., Stucki, M., Ball, W. and Gillett, W. "A Total System Design Framework," *IEEE Computer* 17, 5 (May 1984), pp. 15-26.

[Roman 85] Roman, G., "A Taxonomy of Current Issues in Requirements Engineering," *IEEE Computer* 18, 4 (April 1985), pp. 14-22.

[Ross 85] Ross, D., "Applications and Extensions of SADT," *IEEE Computer* 18, 4 (April 1985), pp. 25-33.

[Sievert 85] Sievert , G. and Mizell, T., "Specifications-Based Software Engineering with TAGS," *IEEE Computer* 18, 4 (April 1985), pp. 56-65.

[Smith 85] Smith, C., Frank, G., and Cuadrado, J., "An Architecture Design and Assessment System for Software/Hardware Codesign," *IEEE 22nd Design Automation Conference*, (1985), pp. 417-424.

[Toy 84] Toy, W., "Hardware/Software Tradeoffs," *Handbook of Software Engineering*, Van Nostrand Reinhold Publishers, (1984), pp. 149-183.

[Turn 78] Turn, R., "Hardware-Software Tradeoffs in Reliable Software Development," *11th Annual Asilomar Conference on Circuits, Systems and Computers*, (1978), pp. 282-288.

[Winner 86] Winner, R., and Carter, E. "Automated Veritcal Migration to Dynamic Microcode: An Overview and Example," (In publication 1986).

SECTION B9

Software Engineering Environments

Prepared by Bill Brykczynski, John Chludzinski, Samuel T. Redwine, Jr.

Topics covered in Section B9:

9.0 SOFTWARE ENGINNERING ENVIRONMENTS

9.1 Introduction
9.1.1 Purpose and Scope

9.2 SDI Requirements for Software Engineering Environments
9.2.1 Functionality
9.2.2 Required Software Qualities
9.2.3 Required Qualities of an SDI SEE
9.2.4 Summary

9.3 Current Status of R&D
9.3.1 Major Software Engineering Environment Efforts
9.3.1.1 Toolpack/Odin
9.3.1.2 Arcturus
9.3.1.3 Gandalf
9.3.1.4 UNIX
9.3.1.5 Smalltalk-80
9.3.1.6 DCDS
9.3.1.7 NASA Software Support Environment
9.3.1.8 WIS SDME
9.3.1.9 FASP
9.3.1.10 Arcadia
9.3.1.11 ALS
9.3.1.12 KBSA
9.3.1.13 Alvey IPSE
9.3.1.14 Sigma
9.3.1.15 PCTE
9.3.1.16 CEDAR
9.3.1.17 STARS
9.3.2 Management
9.3.3 Development and Maintenance
9.3.3.1 Requirements and Design Development
9.3.3.1.1 Methodologies
9.3.3.1.1.1 DSSD
9.3.3.1.1.2 HDM
9.3.3.1.1.3 SADT

9.3.3.1.1.4 SA/SD
9.3.3.1.1.5 SCR
9.3.3.1.1.6 SREM
9.3.3.1.1.7 JSD
9.3.3.1.1.8 PAISLey
9.3.3.1.1.9 SARA
9.3.3.1.1.10 USE
9.3.3.1.2 Requirements Development
9.3.3.1.2.1 State of the Art
9.3.3.1.2.2 State of Practice
9.3.3.1.3 Rapid Prototyping
9.3.3.1.3.1 State of the Art
9.3.3.1.3.2 State of Practice
9.3.3.1.4 Artificial Intelligence
9.3.3.1.5 Simulation and Modeling
9.3.3.1.6 Human Engineering
9.3.3.2 Coding
9.3.3.2.1 Formal Code Specification Languages
9.3.3.2.1.1 State of the Art
9.3.3.2.1.2 State of Practice
9.3.3.2.2 Formal Verification
9.3.3.2.2.1 State of the Art and State of the Practice
9.3.3.2.3 Structured Editors
9.3.3.2.3.1 State of the Art
9.3.3.2.3.2 State of Practice
9.3.3.2.4 Visual Programming
9.3.3.2.4.1 State of the Art
9.3.3.2.4.2 State of Practice
9.3.3.2.5 Code Generation/Compiling
9.3.3.2.5.1 State of the Art
9.3.3.2.5.2 State of the Practice
9.3.3.3 Testing
9.3.4 Ancillary Factors
9.3.4.1 Metrics
9.3.4.2 Quality Assurance
9.3.4.3 Instrumentation of the Environment
9.3.4.4 Reuse and Conversion
9.3.4.5 Validation and Verification
9.3.5 Design Factors for SEEs
9.3.5.1 Structure and Interfaces
9.3.5.2 Database Support
9.3.5.3 Human Interface
9.3.5.4 Security
9.3.5.5 Operating Systems
9.3.5.6 Tool Building Tools

9.4 Recommendations
9.4.1 General Recommendations
9.4.2 Specific Areas
9.4.2.1 Requirements and Design Development
9.4.2.1.1 Methodologies
9.4.2.1.2 Requirements Development
9.4.2.1.3 Rapid Prototyping
9.4.2.2 Coding

9.4.2.2.1 Visual Programming
9.4.2.2.2 Formal Verification
9.4.2.2.3 Code Generation/Compiling
9.4.2.3 Testing
9.4.2.4 Metrics
9.4.2.5 other Considerations
9.4.2.5.1 Reuse
9.4.2.5.2 Database Support
9.4.3 Summary

9.5 References

9.0 SOFTWARE ENGINEERING ENVIRONMENTS

9.1 Introduction

Software has increasingly become an essential component of military systems for the DoD. The current paradigm for the software life cycle begins with requirements definition and design specification, and ends with maintenance and finally retirement. Each phase of the life cycle is associated with a great many tools and procedures, some automated and some performed manually. There are currently many initiatives based on incremental improvement of this life cycle and its respective tools and procedures. These incremental improvements have a high probability of advancing the quality of today's current software life cycle products.

A software engineering environment (SEE) is defined as a computer-based system to assist in the activities associated with the creation and evolution of software systems. It includes procedures, methods, automated and un-automated aids, supporting software creation and evolution. The functional capabilities of a software engineering environment are fairly numerous. These capabilities involve supporting managers, software analysts, programmers, quality assurance personnel, training staff, and other software specialists.

There are currently underway a great many research efforts aimed at improving the tools and methodologies comprising the functional capabilities of a software engineering environment. Some are based on refining tools that have been in existence for a number of years. Others are aimed at defining new tools and methodologies that have, as yet, not been used within the software life cycle. Other efforts are aimed at integrating the tools and methodologies together into a uniform environment. With the decreasing cost of hardware, and the corresponding increasing relative cost of software, these tools and methodologies are becoming more and more critical to the deployment of a military system such as the Strategic Defense Initiative system.

9.1.1 Purpose and Scope

This portion of the appendix characterizes and assesses the current state of the art and state of practice in software engineering environments, identifies areas where research in the long and short run will yield useful payoffs, and recommends actions to the SDIO in order to facilitate the advancement of the elements of SEE's suitable for the SDI software.

This report is intended to give the SDIO a foundation for discussions of what has been carried out in areas involving SEE(s). It presents an overview of many of these efforts but is not intended to be a complete enumeration of all the efforts. This report benefits from several recent related reports prepared by the Software Technology for Adaptable, Reliable Systems (STARS) Program, the DoD Software Engineering Institute (SEI), and the Rocky Mountain Institute for Software Engineering [STARS 85a, STARS 86, Nestor 86, Riddle 85]. This report also presents pointers to the literature for additional information, as well as references to appropriate DoD, industry, and academic organizations involved in software engineering environment related research.

This section is organized as follows: Section 9.2 addresses certain aspects of a SEE relevant/critical to the SDIO effort. Functional capabilities of a generic SEE are given first. Overall requirements of a SDI SEE are then enumerated, and then implications of the architecture are discussed relevant to the SEE. Section 9.3 provides an analysis of areas of research pertaining to SEEs. Emphasis is placed on issues that impact these areas, and the state of the art, and state of practice of each of these areas. Section 9.4 presents

recommendations to the SDIO for research and development in areas perceived as not being at an acceptable level given SDI's requirements. It also identifies other DOD (and external) efforts with which to coordinate. Section 9.5 presents a list of references used in compiling this portion of the appendix.

9.2 SDI Requirements for Software Engineering Environments

This section of the report describes the SDI requirements for SEEs. The specific system functions of a SEE are explicitly outlined. These functions are generic to most SEEs. The descriptions are drawn heavily from the STARS Operational Concept Document [STARS 85a].

A discussion of overall requirements for a SEE is included. Issues such as integration, compatibility, portability and extensibility are explored.

Finally, implications of the SDI architecture for software engineering environments is presented. These implications are general in nature, based on observations that will most probably be applicable to any SDI architecture.

9.2.1 Functionality

There are many functional elements in a software engineering environment. Each of these elements may have associated with it many different procedures, methods, automated and un-automated aids to support software creation and evolution. This section of the report discusses roles played by each of these functional elements in a software engineering environment. This section is not intended to provide full functional descriptions, but to simply provide the reader with a general overview of the various activities a SEE must support.

Project Management

Project management involves both project planning and project control. Project planning involves generation and maintenance of work breakdown structures, estimation and scheduling, planning of resource use, and adjusting to reflect changes that may occur as a result of technical decisions made during the development effort. Project control involves measuring resource use, assessment of project and procurement actions, access control, and support for DoD review and audits.

Requirements Specification and Analysis

Requirements specification and analysis involves requirements formulation and evaluation. Evaluation with regard to consistency and completeness, testability, interdependencies, real-timeness, and compatibility of interfacing systems needs to be supported.

Design Specification and Analysis

Design specification and analysis involves design notation, analysis, and procedures. Design notations must support development and reuse, and express rationale. Design analysis involves analysis for consistency, completeness, traceability, testability, realizability, reuse and other purposes. Design procedures involve decomposition, composition, evaluation, reuse, and recording of rationale.

Software Prototyping and Modeling

Software prototyping and modeling involves evaluating requirements and designs, predicting performance and evaluating the impacts of system changes.

Reuse

Reuse involves acquisition, selection and reuse of work products and software components including possible conversion, adaptation, re-generation, and compostion into new arrangements.

Coding and Unit Testing

Coding and unit testing involves entry and modification, translation, integration, assessment of the status evaluation and debugging, and unit testing of programs and data.

Integration Testing

Integration testing involves identification, design, development, and performance of tests; the analysis and reporting of test results; and the development of test support.

Maintenance

Maintenance involves assessment of the impact of proposed system changes, performance of changes to existing software products, requirements evaluation for consistency and completeness, and performance of site/unit specific adaptations of released systems.

Quality Assurance

Quality assurance involves the selection of development techniques, tools, and methodologies; specification, measurement and assessment of product quality; status of and traceability to product baselines; and performance of quality assurance audits.

Verification and Validation

Verification and validation involves planning and specification of the verification and validation process, verification of the product at each life cycle activity against its predecessors, and validation of the product as meeting user needs.

Configuration Management

Configuration management involves identification of baselines, control and tracking of software access and change, assessment of status of the project, and performance/control of software releases.

Software/Hardware Integration

Software/hardware integration involves identification, design and development of test procedures; integration of programs and data into systems; performance and monitoring of tests; data extraction, reduction and display; and analysis of test results.

Project Communications

Project communications involves the communications of information among the people of the project, between project personnel and external support or control functions, and to and from project database systems and other environments.

Document Generation

Generation of documents involves document development, update, analysis, output, and distribution.

Data Collection, Performance, and Project Measurement

The software engineering environment should support project measurement which involves data collection, comparison of individual tools/methodologies, measurement/assessment of productivity and quality improvements resulting from the use of the environment, and analysis of the environment's performance.

Transition to SEE

The environment must support the transition of SDI software products to its domain. This involves integration of existing software into the environment and possibly insertion of environment products into other existing systems.

SEE Tailoring

The environment should be tailorable from many aspects including tailoring of the following: decision aids, tool building tools, the database, documentation, and the addition and enforcement of project/installation specific methods, procedures, and tools.

Training Tools

Training tools must be provided which include on-line instruction, user courses and tutorials, and documentation.

Knowledge Engineering

The environment should support knowledge-based expert system development as applied to itself and its intended application area.

Required Software Qualities

To meet the software development needs of the SDI, it is important to examine the requirements for the software of the deployed system. Certainly it is possible to identify "tent-pole" properties with regards to SDI: size, performance, security, interoperability with existing DoD systems, reliability and survivability, and maintainability and expandability.

These properties are essential in the deployed software system and therefore should be a high priority for consideration within a SEE. The SDI software engineering environment(s) should be designed with the objective of supporting the development software with these properties.

The large size and real-time performance requirements of the SDI software will factor importantly in the quality of the developed code. Code developed for portions of the system concerned with engagement will need to be extremely efficient. High performance algorithms will need to be developed for these applications. The environment must provide capabilities for designing, implementing, and testing software with such performance requirements.

Security of the system must be placed at as low a level in the system as possible, recognizing that is impossible to build a secure layer upon an insecure layer. Maintainability of the system is addressed by such tools as structured editors, to promote code readability and configuration managers to structure large development efforts. Maintainability is also facillitated by tools that map the relationship between different requirements, design, and code units.

A number of tools impact the reliability of the developed code: formal program verifiers may be used to guarantee the correctness of the source code; metrics may be used to measure properties that reflect software quality; machine code generators may be used to produce optimized code; etc. Still, ensuring software reliability is dependent upon rigorous review and testing procedures.

If formally verified code is an objective for portions of the SDI software system, then certainly the environment must support tools to assist in this effort. Formal verification as a coding requirement, however, is still a long range prospect when applied to large segments of code. Currently, these techniques are only able to formally verify relatively small segments of software.

There are other software qualities to be supported in the development of the SDI system and should therefore be reflected in the software engineering environment(s). The system should be expandible to meet new requirements and to better accommodate existing ones. There is also the issue of usability, which addresses such concerns as the human interface and the flexibility of the software. All of these issues should be addressed by the tools within the environment.

9.2.3 Required Qualities of an SDI SEE

The principle concern of the SDI SEE(s) should be to support the development of large distributed software systems and the maintenance of those systems. When the size of the SDI's software needs are taken into account, it becomes apparent that the development of the software will be a diversified effort. This effort will probably span across a broad spectrum of organizations. Therefore should SDI standardize on one software engineering environment or is it sufficient to require interoperability between environments? If a single environment is deemed too restrictive of a requirement, is a single environment framework a more managable requirement? These are currently open issues, but are given more consideration in 9.4.1 of this report.

Although security itself is a high-level issue, the implementation of security in a software engineering environment begins at the hardware, communication and operating system kernel levels. It is at these levels and lower that security must be guaranteed. Tools must be developed to provide the user with an interface to the security mechanisms, allowing the user to label files (top secret, secret, etc.), share information, etc.

An important facet of an environment is the user interface. Without a user interface that is both easy to learn and easy to use, the environment may well be more of a liability to productivity than an asset. The integration of tools and consistency of tool interfaces are two of the most important measures of quality. The principle measure of usability is typically user performance and satisfaction, which also depends heavily on the quality of the tools (e.g., a fast compiler with meaningful diagnostics, etc.).

Environments in the past have been notoriously non-portable. Too many operating system and machine level dependencies leave the environment "married" to at least the same

operating system and a similar hardware configuration. With the exception of a few ongoing efforts (CAIS and PCTE), portability of environments is still a task with much work to be done.

The tools of an environment are the same as any piece of software -- subject to bugs, changing requirements, the desire for enhancements, and therefore revisions. An environment must be capable of accommodating these modifications in a manner that is transparent to the users. Software tools are also subject to obsolescence and new technology is always developing. Consequently, an environment must be capable of accommodating completely new tools. Such evolution and interoperability can be achieved by standardizing the representation of design entities.

With the advent of networks of heterogeneous machines (mainframes, minis, micros, workstations and special purpose engines) there now arises the issue of developing distributed environments. It would be unrealistic to attempt to construct a single environment across such a network. The level of access to the distributed resources must be determined. The issue of network security must not be overlooked. Inter-environment communication protocols must be agreed upon early in the design of a SEE. Translators between environments' differing representations becomes a necessity.

9.2.4 Summary

In summary, there are many functional elements within a software engineering environment. These elements may be tied to software development phase-related activities (requirements development, coding and unit testing, etc.), may be related to aspects of the SEE (training tools, SEE tailoring, etc.) or may be concerned with management (project communications, configuration management, etc.).

With respect to the SDI, there are several SEE requirements imposed due to the implications of the SDI architecture. Properties such as size, performance, reliability, maintainability, etc., relate to the developed software, the SEE itself, or both. These properties impact the design and implementation of the SDI SEE(s). In addition, the effect of distribution will factor into the design of the SEE(s). Finally, the SEE(s) developed will be long-lived and evolving. Many decisions will have long-lasting ramifications. The SEE itself will be an entity that must be maintained, managed, and supported.

9.3 Current Status of R&D

In this section, the current status of research and development within software engineering environments is discussed. An overview of existing/planned SEE efforts is presented first. Next a discussion of management issues relevant to the SEE is discussed. Design and development issues are covered, followed by auxiliary and other considerations.

9.3.1 Major Software Engineering Environment Efforts

In this section of the report a number of existing/planned environment efforts are overviewed. Some of these environments have been operational for quite some time, while others are just beginning to be implemented. These environments exist mainly in DoD and industry, although several are products of foreign endeavors. The descriptions of these environments draws heavily from [McDonald 85]. Although many of these environments have a methodology associated with them, we treat the discussion of environments separately from methodologies. In 9.3.3.1.1 a survey of existing methodologies is presented, many of them closely associated with the environments presented below.

In the last couple of years there have been several conferences devoted to areas involving software engineering environments. They are an excellent source of information regarding current state of the art in technology for SEE's. The following is a partial list of some of the more prominent ones:

a. Softfair 1985, San Francisco, California, December 2-5, 1985.

b. Rocky Mountain Institute of Software Engineering's Software Environments Workshop, November 2-5, 1985.

c. Workshop on Software Engineering Environments for Programming-in-the-Large, Harwichport, Massachusetts, June 9-12, 1985.

d. IEEE Computer Society 1984 Conference on Ada Applications and Environments, St. Paul, Minnesota, October 15-18, 1984.

e. ACM SIGSOFT/SIGPLAN Software Engineering Symposium on Practical Software Development Environments, Pittsburgh, Pennsylvania, April 23-25, 1984.

f. Symposium on Application and Assessment of Automated Tools for Software Development, San Francisco, California, November 1-3 1983.

9.3.1.1 Toolpack/Odin

Created by a consortium of university, Government and industrial research labs, the Toolpack environment is an experimental environment created to support the development of numerical analysis software in FORTRAN. Toolpack provides its users with a product-oriented interface. Rather that specify what tools should be invoked, a Toolpack user indicates what object (an object file, a test report, etc.) is desired and, assuming that the object can in fact be produced given the current state (no necessary inputs are missing), Toolpack carries out the necessary steps to produce the desired object.

Odin, developed at the University of Colorado, is intended to serve as the basis for an extensible program development environment. Odin is an outgrowth of the Toolpack project and an extension of the Integrated System of Tools (IST) subsystem found in Toolpack.

9.3.1.2 Arcturus

During the last five years the University of California, Irvine, has been involved with research in an experimental environment named Arcturus. The Arcturus prototype offers execution with both compiled and interpreted Ada, template-assisted Ada text editing, tools for measuring the performance of Ada programs and displaying the data in easily interpretable form, formatted listings of Ada programs that can be easily controlled to match individual preferences, a program design language compatible with Ada, and assistance for refining designs into executable code. This list of capabilities emphasizes that the focus has been on the user interface and the facilities provided from within the Ada Programming Support Environment (APSE) virtual machine layer.

9.3.1.3 Gandalf

Gandalf is a family of environment efforts, each intended for different audiences, that share a common paradigm as to how software systems should be developed. The paradigm

emphasizes the incremental preparation of software by teams of people with the extensive use of automated support stemming from the environment-processable description of project team and software systems structures in addition to the description of the software itself.

The Gandalf project was begun at Carnegie-Mellon University in 1978 with the intent of developing an Ada programming support environment which did not necessarily use the layered architectural structure espoused in Stoneman. The focus was on providing support for the programming of software systems by teams and so the emphasis was on supporting programming, configuration management and team interaction activities. Over the years, the emphasis on Ada has lessened, but the environment's scope of activity coverage has remained essentially the same.

9.3.1.4 UNIX

While not really an environment in the sense of the other efforts mentioned here, the UNIX operating system is used or has influenced a number of them. It was initially developed in the late 1960's by Bell Telephone Laboratories to provide similiar services on small computers found in modern operating systems for large mainframe computers. Over the years, its popularity has spread, particularly within the academic research community, and it is currently available on a wide variety of host computers.

Viewed as an environment, UNIX is a loosely integrated collection of tools supporting a wide variety of approaches to software creation and evolution. Many of the tools are simple ones that perform text transformations or other processing of general utility. There are also many more powerful, high level tools for specific tasks, such as document preparation, found during software development and maintenance. Few, if any, of the tools are wedded to a particular way of performing software creation and evolution---the tools are, for the most part, general purpose ones that can be used in a variety of ways to support a variety of methodologies.

9.3.1.5 Smalltalk-80

The Smalltalk-80 system was developed by the Learning Research Group at Xerox Palo Alto Research Center. Smalltalk-80 combines a programming language and a programming environment, and the two are highly intertwined [Krasner 83].

The fundamental viewpoint of Smalltalk-80 is object-oriented. Smalltalk-80 programs, and similarly the Smalltalk-80 system, are composed entirely of objects which send messages to one another. A message may ask an object to perform one of its methods, which is an operation that the object knows how to carry out. Each object is an instance of some class of objects, and all objects in a class have exactly the same set of methods defined for them. Thus the set of methods defined for a class of objects strictly circumscribes the ways in which any instance of the class may be manipulated, since objects ignore any messages asking them to perform any method that is not defined for the class to which they belong. An added richness in this object class structure is achieved through the notion of subclass and inheritance. If a class of objects is defined to be a subclass of some other class, then the subclass inherits all of the methods defined for that other class.

9.3.1.6 DCDS

The Distributed Computing Development System (DCDS) has been developed over the last decade by TRW Defense Systems Group, Huntsville, Alabama, under contract to the Army Ballistic Missile Defense Advanced Technology Center. In its original form, called

Software Requirements Engineering Methodology (SREM), it was intended to support the definition and analysis of software requirements. This original system has been extended over time to cover the activities of system requirements definition, software high-level and detailed design, and allocation of modules in a software system to the physical processors in a distributed computing facility. The system is compatible with a variety of programming languages although work has concentrated on Pascal and Ada.

DCDS is, therefore, an environment supporting a wide spectrum of life cycle activities during the creation and evolution of distributed, concurrent software systems. DCDS provides a variety of languages for describing software during the various life cycle phases. It also provides a variety of analyzers for reasoning about software specifications and the system they describe. Some support is provided for moving between phases by deriving new descriptions from old or tracing back a sequence of descriptions, but this is largely left to developer intuition, experience, and expertise.

9.3.1.7 NASA Software Support Environment

NASA is in the process of developing a Software Support Environment (SSE), for use with the Space Station program. NASA envisions a uniform, NASA-furnished SEE, and will mandate compatibility with delivered software. The SSE will use a modular, layered architecture, and will likely utilize UNIX as the host operating system. Ada has been chosen as the NASA Space Station programming language. The STARS SEE Operational Concept Document [STARS 85a] was used as one starting point in defining the operations concept for the SSE.

9.3.1.8 WIS SDME

GTE is currently the integration contractor for building the Software Development and Maintenance Environment (SDME) for the World Wide Military Command and Control System (WWMCCS) Information System (WIS). This environment will support the development and maintenance of WIS operational software. The environment will be written in, and designed for the production of, Ada software. The SDME is planned to be distributed and heterogeneous, and able to eventually provide a secure operational system.

9.3.1.9 FASP

The Facility for Automated Software Production (FASP) has been developed by the Advanced Software Technology Division of the Naval Air Development Center, Warminster, Pennsylvania. Its original release was in 1975 and provided a batch capability supporting development for the Advanced Signal Processor. Since this initial release, it has evolved to provide extensive interactive and batch, life cycle programming and management support for projects producing Navy standard software written in a variety of languages and intended to run on one of a variety of target computers.

9.3.1.10 Arcadia

A software development environment, named Arcadia, is being developed by a consortium of researchers from the University of California at Irvine, the University of Colorado at Boulder, the University of Massachusetts at Amherst, TRW, Incremental Systems Corporation, and The Aerospace Corporation.

The research objectives of the Arcadia project are two-fold: discovery and development of environment architecture principles and creation of novel software development tools, which will function within an environment built upon these architectural principles.

Work in the architecture area is concerned with providing the framework to support integration while also supporting the sometimes conflicting goal of extensibility. Thus, this area of research is directed toward achieving external integration by providing a consistent, uniform user interface, while admitting customization and addition of new tools and interface functions. In an effort to also attain internal integration, research is aimed at developing mechanisms for structuring and managing the tools and data objects that populate a software development environment, while facilitating the insertion of new kinds of tools and new classes of objects.

The initial focus of Arcadia research is on creating a prototype environment, embodying the architectural principles, which supports Ada software development. This prototype environment is itself being developed in Ada.

9.3.1.11 ALS

SofTech, under contract to the US Army, has built the Ada Language System (ALS). The ALS is an environment based on the Ada Programming Support Environment (APSE) model of environments. These environments provide automated support for the full spectrum of life cycle activities for large, embedded software systems. Currently, the ALS implementation directs most of its attention in environment capabilities to supporting the programming phase of software development.

9.3.1.12 KBSA

The Knowledge-Based Software Assistant (KBSA) is a research project aimed at producing a software development environment supporting a knowledge-based perspective on the software development and maintenance process. The intent is to "introduce a fundamental change in the software life cycle maintenance and evolution by modifying the specifications and then re-deriving the implementation rather than attempting to directly modify the optimized implementation." The goal is to have all software development and maintenance activities carried out at the specifications and requirements level, not the implementation level. "The transformation from requirements to specification to implementation will be carried out with automated, knowledge-based assistance." KBSA was defined in a report produced by Kestrel Institute for Rome Air Development Center in 1983 [Green 83].

The KBSA itself is intended to "put the machine in the loop", i.e., have all activities carried out during software development and maintenance mediated by the KBSA. The result would be an environment that would monitor, capture and reason about those activities and serve as a knowledgeable assistant to the human software developer.

9.3.1.13 Alvey IPSE

In the United Kingdom, government, industry and academia have formed a program named Alvey, which has as one of its goals the construction of two generations of Integrated Project Support Environments (IPSE). The first generation will be a file-based UNIX system, and the second will be database-oriented with a distributed operating system and include artificial intelligence aspects.

9.3.1.14 Sigma

The Japanese Ministry of International Trade and Industry (MITI) and the Japan Software Industry Association (JSIA) have begun work to create a standard, Japan-wide software

development environment. The system, known as Sigma, seeks to attain basic goals such as:

a. Improvement of the quality and productivity of software

b. Prevention of overlapping development of software

c. Perfection of software development facilities, accumulation of know-how, and improvement of technological capabilities

d. Increased efficiency in the training of technical experts

The project will extend from 1985-1989. A wide selection of target machine independent development tools will be built upon a UNIX-like operating system [Kishida 84].

9.3.1.15 PCTE

ESPRIT, a European cooperative research and development program devoted to multinational computing development, is developing an environment named Portable Common Tool Environment (PCTE). PCTE will consist of an infra-structure which will provide the basic framework and a set of tools and components for projects developed for ESPRIT. The operating system of PCTE is an enhancement of UNIX.

9.3.1.16 CEDAR

Developed in 1978 at the Xerox Palo Alto Research Center, CEDAR is one of the first fully integrated programming environments designed and implemented for a personal computer. CEDAR combines high-quality graphics, a sophisticated editor and document preparation facility together to achieve a very user-friendly interactive interface. Users interact with CEDAR with a mouse, manipulating icons on a high-resolution bit mapped display. Excellent on-line HELP information is available, as well as interpreters, compilers and an electronic mail facility [Tietelman 84].

9.3.1.17 STARS

The STARS goal is to decrease costs and increase the reliability of DoD software systems and subsystems. In the spring of 1983, the Department of Defense launched a DoD-wide effort named Software Technology for Adaptable, Reliable Systems (STARS). The initial areas of the STARS efforts of relevance here have been in three areas: methodologies, measurement, and software engineering environments.

The methodology effort was concerned with the development of methodology classification, evaluation, and selection technologies and a framework of characteristics that can be used to support these technologies [McDonald 85].

The measurement effort was concerned with two issues. An interim compendium of collection forms was produced. These forms provide information on lines of code, cost of used resources, characteristics of personnel, etc. In addition, a working guidebook has been developed on how to select and apply particular measurements [IITRI 85, Cavano 85].

The Software Engineering Environment (SEE) effort began in the summer of 1983. It started with a relatively low budget, with three primary products anticipated:

a. A computer-based set of integrated software engineering methods, tools, and procedures to perform the development and life cycle support of software that meets DoD mission needs.

b. The supporting standards upon which SEEs can be interrelated.

c. The architectural framework and supporting standards upon which SEEs can be built.

A proposed Operational Concept Document has been produced [STAFS 85a], along with a preliminary system specification document [STARS 86].

STARS has produced many documents relating to issues of environments. Issues such as distribution, interfaces, extensibility, and architectures have been discussed. Studies of DoD-funded environments, such as ALS, ALS/N, Ada Integrated Environment (AIE), FASP, and DCDS, have been performed. Studies of non-DoD-funded environments such as APSE, CSE, BASE, etc., have also been performed.

For the future, the STARS program has five major categories of activities. They are: (1) Service Initiatives, (2) Common Ada Foundations, (3) Prototype Environments, (4) Shadow Projects, and (5) the Repository. Most of the work will be accomplished through industry contracts. The services will continue to support the goal and objectives of STARS by evolving independent applications. The STARS Joint Program Office will, with the assistane of developers, migrate appropriate software to the repository where it will be made available for widespread use.

The Service Initiatives are seven old STARS projects that will be carried over to the new STARS program. They are:

- Common Ada Missile Packages (CAMP)
- Ada Based Signal Processing
- Ada Based Integrated Control System (ABICS)
- Automation of User Requirements
- Convert-to-Ada Tools
- Computer Aided Instruction (CAI)
- Distributed Computer Development

The projects will be restructured to fit into the new STARS program and will be completed under this new program.

The Common Ada Foundations include tools and parts from nine areas. They are:

- Command Language
- Software design, description and analysis tools
- Text processing

- Database management system
- Operating system
- Planning and optimization tools
- Graphics
- Network protocols
- Others

The Prototype Environments will be built by integrating the adaptable collections of methods and tools from the nine foundations areas. The Shadow projects will be developed as a proof of these concepts. The Repository is composed of the R&D projects from both the services and industry, catalogued for reuse by all interested parties.

9.3.2 Management

A software engineering environment must support several different forms of management. This report will discuss two of these forms: project management and configuration management. Although other forms of management exist (e.g., contracting management, risk/benefit evaluation management, etc.), project management and configuration management are the two most prominent.

Project management, in part, is concerned with work breakdown structures, software cost evaluation, scheduling, planning and tracking. Many of these functions are provided in user-friendly formats on widely used microcomputers (e.g., Apple Macintosh, IBM PC, etc.). It would be essential for any software engineering environment to provide these and other tools for the software project manager.

Any large software effort will contain hundreds if not thousands of separate sections of software code. These sections undergo revisions and modifications constantly. An effective means on tracking these sections so that current and previous versions of software are controlled is an important part of managing a software project. This aspect of management is typically called configuration management.

The functions that a configuration management system must perform are relatively well understood. Functions such as historical data, release information, dependency information, rebuild information, etc., have been successfully employed in several research and commercial products. The state of the practice in configuration management is probably suitable for the SDI's needs, with the possible exception of management of rapidly prototyped AI Systems. Few, if any, of todays configuration management systems are explicitly prepared for use in rapid prototyping environments. Care must be taken, though, to ensure that the SEE is properly integrated with configuration functions in mind.

9.3.3 Development and Maintenance

Three elements exist within the development and maintenance portions of the software life cycle. They are requirements and design definition, coding, and testing and evaluation. This section of the report discusses issues within these areas, and addresses the state of the art and state of practice in areas where applicable.

9.3.3.1 Requirements and Design Development

There are many issues within requirements and design definition. This section of the report describes the state of practice in software methodologies, state of the art and state of practice in requirements engineering, issues in rapid prototyping, issues in knowledge engineering, the importance of simulation and modeling, and the use of human engineering techniques.

9.3.3.1.1 Methodologies

In this section of the report various methodologies are described which constitute todays state of practice. These methodologies support three portions of the software life cycle: requirements analysis, architectural design, and detailed design. Descriptions of these methodologies draws heavily from [Addleman 85]. Although many of these methodologies have an environment associated with them, we treat the discussion of methodologies separately from environments. A survey of existing environments, many of them closely associated with the methodologies presented below, is presented in 9.3.1.1.

9.3.3.1.1.1 DSSD

Data Structured System Design (DSSD) was developed by Ken Orr and Associates, Incorporated. It is a data-structured development methodology. The fundamental idea is to define outputs and their structure and then to work backwards to inputs.

The basic technique is to construct hierarchically structured diagrams called assembly line diagrams that read left to right instead of top to bottom. The diagrams can be structured to represent a hierarchy of processing steps, events in time, data flow, or data structures. The methodology uses entity diagrams to model the software system and its environment and to model functional flow. Detailed design for processes (transformations) is done through a variant of Warnier-Orr diagrams in which the process is always found at the leftmost bottom edge of the diagram.

9.3.3.1.1.2 HDM

The Hierarchical Development Methodology (HDM) was developed by the Computer Science Laboratory at SRI International. It combines the data structured and algorithmic refinement approaches to design. A specification written in HDM is a hierarchy of abstract machines. The methodology assumes that the requirements for the software system have been captured in a model and can be written as a top-level specification known as a TLS.

The TLS describes the systems's external (observable) behavior and it is written in a nonprocedural language called SPECIAL. Beginning with the TLS, a hierarchy of abstract machines is produced by providing mappings from the representations in the higher level machine to the representations in the lower level machine. Each lower level machine provides more and more concrete detail. Eventually, the lowest level can be translated into an implementation language.

The system was primarily developed for the purpose of verifying security properties of software. The automated tools support formal verification of design.

9.3.3.1.1.3 SADT

Structured Analysis and Design Techniques (SADT) was developed by SofTech Incorporated. It is a disciplined decomposition approach to modeling complex problems

and systems. The language of SADT (SA) combines a blueprint-like graphics language with the nouns and verbs which describe the problem domain of the system to be modeled.

A SADT diagram, known as an actigram, is composed of no more than six basic blocks. Each basic building block is drawn as a box with four sides called INPUT, CONTROL, OUTPUT, and MECHANISM. Arrows coming in and out of the basic box represent flows of data or control. The box represents a transformation from a before state to an after state. Thus, an actigram can represent a decomposition of data or activity. SADT is also discussed in 9.3.3.1.2.1.

9.3.3.1.1.4 SA/SD

Structured Analysis/Structured Design (SA/SD) is a real-time methodology developed by Yourdon, Inc. SA/SD actually refers to two distinct methodologies: real-time and classic. This description covers the real-time version. The classic version is very similar, but it does not address real-time issues such as concurrency/synchronization or mapping to a distributed hardware architecture.

The methodology blends the three major requirements analysis techniques of state, flow, and object modeling with the design techniques of decomposition. Not only does SA/SD provide description of the techniques to use for requirements and design specification, but it also gives rules of thumb for applying the techniques in a reasonable and consistent manner both within and across life cycle phases.

The specific steps in software development as recommended by SA/SD are to construct a model:

a. Of the context or environment of the system

b. Of the internal behavior of the system

c. That shows the processor utilization of the system

d. That shows the software architecture utilization

e. That shows the coding architecture utilization

Each model has both a physical and logical organization. The first two steps are requirements specification and analysis activities; the remaining three are design activities. Each model is differentiated by the type of behavior which it describes and its constraining effect on the final implementation. As one proceeds through the steps, the final implementation becomes more constrained.

9.3.3.1.1.5 SCR

The Software Cost Reduction (SCR) project was developed by the Naval Research Lab. The SCR requirements and design specification methodology is purely textual. It is based on the principles of information hiding and separation of concerns. Separation of concerns requires that information be divided into clearly distinct and relatively independent documents. Information hiding guides the architectural design of the software and leads to software that is easy to change.

The basic approach is data abstraction. Data items and the functions needed to create, store, retrieve, or manipulate them are identified. Event lists are used to document how the

abstractions change relative to changing conditions as the software executes. These conditions can be nothing more complicated than passage of time.

The methodology includes procedural guidelines and suggested documentation formats that help keep the specifications complete and consistent.

9.3.3.1.1.6 SREM

The Software Requirements Engineering Methodology (SREM) was developed by TRW through BMD. SREM is based on a graph model of software requirements. The basic concept is that design-free functional software requirements should specify the required processing in terms of all possible responses (and the conditions for each response) to each input message across each interface. A message may contain input data or represent stimuli generated from an external event.

The methodology is based on a stimulus/response approach as opposed to a purely hierarchical decomposition approach. The required actions of the software are expressible in terms of R_NETS (requirements networks) of processing steps. Each processing step is defined in terms of input data, output data, and the associated data transformation. The input interfaces (the system stimuli) are defined and the R_NETS trace the inputs through the various functional transformations to their associated system outputs (responses).

Environment aspects of SREM are discussed in 9.2.1.6 and 9.3.3.1.2.1. DCDS is an outgrowth of the SREM methodology providing for a distributed environment. Environment aspects of DCDS are discussed in 9.3.1.6 and 9.3.3.1.2.1.

9.3.3.1.1.7 JSD

The Jackson System Development (JSD) methodology was developed by Michael Jackson Systems Limited. JSD divides the process of software development into three main phases:

a. Modeling - an explicit examination of the external world with which the system will be concerned resulting in a diagrammatic description that clearly isolates and defines those aspects of the external world that are of interest. The model serves as an aid to understanding the subject matter of the system and provides the core for the formal specification of the systems's functions.

b. Function - concerns the outputs of the system, what they should be and how they should be generated, resulting in diagrammatic descriptions attached to the functions in the previously defined model. The completed specification consists of a system specification diagram which defines the system as a set of logical processes communicating by data transfer and a set of structure diagrams which define the internal logic of each process.

c. Implementation - the system specifications is converted into a form suitable for running on the chosen hardware by applying standard JSD procedures (called TRANSFORMATIONS) to package and realize the logical processes previously defined.

9.3.3.1.1.8 PAISLey

The Process-Oriented, Applicative, Interpretable Specification Language (PAISLey) was developed by AT&T Bell Laboratories. PAISLey was developed explicitly for

requirements specification of embedded real-time systems and takes an "operational" approach to requirements specification. That is, PAISLey allows the specifier to construct an executable model of the software as it would function in its environment.

The primary unit of specification is the process -- a simple, abstract representation of autonomous digital computation. Each process is specified by supplying a "state space" (set of all possible states) and a "successor function" on that state space which defines the successor state for each state. A process is cyclic and goes through an infinite sequence of states (a distinguished "halted" state can be defined) asynchronously with other processes.

Because requirements are executable (by simulation), it is possible to attach and test timing constraints to processes. The constraints can be defined as maximum, minimum, mean, or constent evaluation time for the process.

Since PAISLey is an applicative language, a process can only access information in the state of another process by an explicit request. These requests are formulated as exchange functions and allow every form of synchronization to be defined.

9.3.3.1.1.9 SARA

The System Architect's Apprentice (SARA) was developed by the University of California at Los Angeles. SARA is a set of modeling and evaluation tools that support a requirements-driven design methodology for concurrent systems. SARA encourages partition of a design or analysis universe into a system and its environment, with explicit models of their behaviors. The system includes tools to model behaviors (GMB) , to model structures (SL1), and to model the structure of code-modules (MID).

GMB produces a graphics-based model of behavior in three domains: flow of control, flow of data, and interpretation. The model described in GMB may be interactively simulated and the control flow information can be formally analyzed for inconsistency, completeness, liveness (freedom from deadlock), and termination.

SL1 describes hierarchically related structures. A designer can specify a nested space of identifiers which partition a design universe and encapsulate behavioral models. A module encapsulates part of a behavioral model; a socket encapsulates behavior related to the interface between a module and its environment; an interconnection connects modules at their sockets and represents potential flow of data or control.

GMB models are mapped to SL1 structures, providing the connection between the behavior of objects in the actual environment and objects that will be instantiated during execution of the software system.

9.3.3.1.1.10 USE

The User Software Engineering methodology (USE) was developed by Anthony Wasserman at the University of California at San Francisco. USE is a methodology to support the development of specifications, designs, a transition diagram interpreter, an interface to a relational database system, a utility for constructing rapid prototypes of interactive dialogue, a programming language for int. raction, and the USE control system for management support.

The steps for requirements analysis are:

a. Identify system objectives and constraints, including conflicts of interest among user groups.

b. Model the existing system using a requirements analysis method (Structured Systems Analysis for instance).

c. Construct a conceptual model of the database, using the Semantic Hierarchy model of Smith and Smith.

d. Produce a system dictionary containing the names of all operations, all data items, and all data flows.

e. Review the analysis results within the development group and with the users and customers.

Architectural design constructs a structure chart of the overall design, following the general structure already outlined during the requirements phase. Detailed design is done via a program design language. Both design phases are reviewed with walkthroughs.

9.3.3.1.2 Requirements Development

The DoD projects currently underway utilize a phased approach to the software life cycle. This approach is sometimes referred to as the waterfall cycle. It seems that fundamental flaws in this life cycle result in severe problems for pioneering systems.

Within the software development life cycle, requirements are converted into design specifications, which are then converted into an implementation. Unfortunately, the first two portions of this process are frequently informal, labor intensive, mostly undocumented, and largely un-automated. During the maintenance phase, information leading to the production of these steps is difficult to find, incorrect, or non-existant. Compounding this problem is the fact that when maintenance is performed, it is performed directly to the source code. This not only makes information pertaining to the source code more disbursed, but typically the specifications are not updated to reflect this modification. This makes the software much harder to understand.

What is needed is a method directly of maintaining design and requirements specifications. These revised specifications could then be reimplemented utilizing tools that would aid the maintainer. Even more helpful would be the ability to modify the requirements specification, which in turn would automatically be reflected in the design or requirements specifications. During the last five years, research has been undertaken to formalize and more fully automate portions of the software life cycle pertaining to requirements engineering. The potential payoffs in this area of research are promising.

9.3.3.1.2.1 State of the Art

It is difficult to draw the line in requirements engineering between state of the art and state of the practice. There is work being undertaken in the academic arena which is clearly state of the art. However, other efforts have been undertaken resulting in full implementation and use. The latter is discussed first.

In 1981, part of the Army's Strategic Defense Command then known as the Ballistic Missile Defense Advanced Technology Center (BMDATC) initiated research for what has become the Distributed Computing Design System (DCDS) [Alford 85]. DCDS is an outgrowth of the Software Requirements Engineering Methodology (SREM) [Rzepka 82],

also sponsored by the BMDATC, which began in 1973. SREM attempts to formalize and automate techniques for specifying software requirements. Components of SREM include a software requirements language, a set of tools to translate this language and perform automated consistency analysis, and a methodology for generating the requirements through the use of the language and its tools. SREM has matured to a point where they are being used on a variety of projects at a number of installations.

One of the most significant limitations of SREM was the transition from requirements to design. This part of the life cycle was completely left to the imagination of the developers. DCDS research evolved to compensate for this this and other limitations. DCDS seeks to develop and integrate design languages and tools into the SREM model. DCDS provides a variety of languages for describing software during the various life cycle phases. Automated analyzers are provided for reasoning about software specifications and the system they describe. In addition, support is provided between these phases for deriving new specifications from old ones. The goal of DCDS is to increase reliability and productivity by creating a software engineering environment to automatically monitor the requirements, design, code, and tests involved in life cycle management.

The Rome Air Development Center [RADC 85] has several projects aimed at implementing automated tools and methodologies:

a. Requirements Engineering Workstation Design. The objective of this project is to reduce the number of errors introduced into C3I systems/software during the requirements definition phase of the life cycle by improving communication of the requirements between the user, acquisition engineer and developer through the technology of rapid prototyping.

b. Natural Language Requirements Translation. The objective of this project is to develop a translator which will accept as input a software requirements specification written in a formal specification language and produce as output a natural language paraphrase of the requirements.

c. Knowledge Based Requirements Assistant. The objective of this project is to begin the process of machine mediation of requirements analysis in order to increase productivity and reduce the number of errors occurring in this phase.

9.3.3.1.2.2 State of the Practice

SA/SD, the Structured Analysis Structured Design methodology, is probably the most widely used approach and is supported by a wide variety of tools. In 1977, SofTech, Inc. published several papers providing an overview of their in-house methodology called Structured Analysis and Design Technique (SADT) [Ross 77, Schoman 77]. SADT permits requirements and functional specifications to be expressed at a very high level, thus capturing valuable knowledge early in the life cycle process. SADT comprised a diagramming language used for structured analysis, and a complete design methodology. SADT has seen successful use in a broad range of application areas.

SREM, the Structured Requirements Engineering Methodology outlined above as the precursor to DCDS, has also seen use in a variety of application areas.

9.3.3.1.3 Rapid Prototyping

During the software life cycle, activities such as requirements analysis, specification, design and coding occur in a cyclic sequence. Each of these activities are directly based

upon its predecessor. Often, requirements shift or are found to be incorrect. Unfortunately, when these findings occur, they are usually late in the software life cycle, and are very expensive to correct.

When systems are built with novel capabilities, requirements are usually not completely known at the onset of a project. A learning process is helpful, in which introducing the user to a semi-functional working model provides feedback between the user and the developer occurs. If this model can be created cheaply and quickly, much more can be learned about the system earlier on, thereby providing stabler and accurate requirements. This capability is referred to as definition exploration. Alternative experimentation is other aspects of rapid prototyping. This refers to the capability of quickly determining the value of a particular implementation of software.

9.3.3.1.3.1 State of the Art

At this point in time, there are few products dedicated exclusively to the techniques of rapid prototyping. The areas of research are directed to describing general purpose techniques, formal specification languages, methods, and criteria needed for successful rapid prototyping. Work is being done, however, in describing features and characteristics of a rapid prototyping language.

Taylor and Standish [Taylor 82], through DARPA, have identified technical approaches for rapid prototyping. Techniques such as heavily parameterized models, reuse of software, restricting the functionality of models, and reconfigurable test harnesses characterize successful approaches to rapid prototyping. Rapid prototyping languages have also been proposed as a way of creating a well-rounded approach to rapid prototyping techniques. Methods of incorporating rapid prototyping into the software life cycle have also been researched.

Balzer and collegues [Balzer 82], through DARPA, have been exploring the uses of operational specification languages to aid in rapid prototyping. They have developed an operational specification language, Gist, that allows specifications to be transformed into an implementation which can then be evaluated to assess their behavior.

GTE [Davis 82] has been developing a software tool which can execute requirements specifications of real-time systems. RPS (Requirements Processing System) is a set of software tools whose combined purpose is to automate the requirements specification task. Another tool, the Feature Simulator, can interpretively execute the requirements specifications that have been processed through RPS. Functional characteristics of the system are thus simulated, providing an opportunity for review of the specified system.

The Smalltalk-80 [Goldberg 83] system was developed by the Learning Research Group at the Xerox Palo Alto Research Center. Smalltalk-80 provides an object-oriented programming environment and a programming language for the user. The Smalltalk-80 system defines objects which may send messages to one another. Each object is strictly typed, thereby limiting manipulations that may be performed upon them. As Smalltalk-80 is an entirely object-oriented system, modifications of the environment and its components are straightforward. This encourages prototyping, as results of possible modifications are readily observed.

RADC [RADC 85] currently has a project being planned for developing an environment designed for rapid prototyping. The objective is to develop a methodology and supporting tool environment for rapidly configuring software models which can be used to validate user requirements for mission critical computer systems.

9.3.3.1.3.2 State of Practice

The term rapid prototyping has been used frequently within the software engineering circles within the last several years. The idea of rapid prototyping is not new. However, the use of a methodology and supporting tool environment specifically designed for use with rapid prototyping is new [Konchan 83].The use of formal executable specification languages is not widespread. It may be several years before the use of these techniques becomes prevalent [Budde 84].

9.3.3.1.4 Artificial Intelligence

The current state of the art in software engineering environments reflects few, if any, direct contributions from artificial intelligence. It is unlikely that this situation will change in the near to mid-term. There have, however, been several indirect contributions [McDonald 85]. A recent JASON workshop explored a number of aspects and possibilities in this area [JASON 85]. The areas with the greatest potential for high payoff are knowledge representation and design support. It seems probable that computer-aided capture of software artifact-related information and the computer interpretable encoding of software process-related information will advance the state of the art in the software life cycle.

Techniques and approaches developed within the AI arena supporting the development of a software system's rough logic and structure prior to its implementation in a programming language seem to be the main contribution to be gained from the AI community [McDonald 85].

9.3.3.1.5 Simulation and Modeling

Simulation and modeling will play an important part in the testing, evaluation, and validation of SDI software. It seems that thorough and effective simulation will be one of the foundations upon which reliance of the system will be based. These aspects are discussed in Appendix B, Section 10, Simulation.

9.3.3.1.6 Human Engineering

Human engineering can be supported by a number of tools that can be incorporated into a software engineering environment. It is a body of thought that concerns itself with how humans interact with equipment. Human engineering, with regards to SEEs, is mainly concerned with software tool interactions (e.g., command languages, graphics, menu selection, etc.). These aspects and others are discussed in Appendix B, Section 6.0, Man-Machine Interfaces.

9.3.3.2 Coding

This section is concerned with some of the tools which address programming-in-the-small. It is not the intent of this section to address all the tools which apply to programming-in-the-small, but it is intented to give a sense of which tools are most relevant to SDI needs. It should also be stressed that what is meant here by coding is the actual process of producing (quality) program text.

9.3.3.2.1 Formal Code Specification Languages

There are two classifications of specification languages, the specification of structural elements used for programming-in-the-large and the specification of functional behavior for

programming-in-the-small. Functional specifications may be used to generate executable code, to generate test data sets, to do spot checking, or to actually verify the program text. Structural specifications, which give a rigorous description of the inter-relationships between program units, may be used to ensure that these units fit together correctly and to aid configuration management.

9.3.3.2.1.1 State of the Art

ANNA (ANNotated Ada) is a Stanford University product developed under the leadership of David Luckham [Luckham 85]. It is Ada specific and provides specifications for functional behavior (programming-in-the-small). Work is continuing with ANNA at Stanford to develop a set of tools to support ANNA as a debugging tool. This work is being funded by the Advanced Research Projects Agency, DoD.

PIC (Precise Interface Control) language, is an ongoing effort at the University of Massachusetts. PIC provides a precise specification of requested and provided visibility of declared subprograms, objects, and types. During the entire life cycle of the software, the PIC family of languages (PIC/graphics, PIC/PDL, PIC/Ada) supports development, management and maintenance of the system.

GYPSY is an integrated language system consisting of a specification language and an implementation language. The purpose of the GYPSY system of languages is to produce verifiable code. For this purpose the GYPSY environment supports a range of tools. This effort is ongoing at the University of Texas Institute of Computing Science and being funded by NSF and NSA.

9.3.3.2.1.2 State of Practice

No language is in wide use. Ada as a specification language is used in some instances. Finite state machine models have been standardized in the communications industry. The most highly developed and experienced system using specification languages is the University of Texas GYPSY effort.

9.3.3.2.2 Formal Verification

Formal verification of a software product, which could possibly include firmware and hardware, is the process of mathematically proving the software performs the functional behavior as specified by the entry and exit assertions. The entry and exit assertions are written in a high level language (usually higher than the implementation language), which rigorously specifies what is true about the object(s) being operated upon, before and after the execution of that segment of the program.

Program verification is a subject which raises a myriad of questions:

a. If the programmer cannot program his loop correctly, how can he be expected to provide the correct loop invariant to the verifier?

b. How can the verifier be proven correct?

c. Is it reasonable to expect a large body of programmers to become proficient in first order predicate calculus and set theory in order to provide a complete set of specifications (verification conditions) necessary for verification?

d. How are the entry and exit conditions checked for correctness?

e. Once the high level source code is verified, what guarantees the correctness of the machine code produced by the compiler?

f. Has the microcode been verified?

g. Has the silicon been verified?

These questions summarize the difficulties in realizing formal verification as a practical method for achieving reliability in software. The lack of completely satisfactory answers to any of these questions would seem to indicate that verification is more a long term prospect.

9.3.3.2.2.1 State of the Art and State of the Practice

The state of the art and the state of the practice can be represented by the GYPSY project at the University of Texas. Specifications are broken up into external and internal specifications. External specifications are further divided into environmental and operational. The environmental specifications provided a mechanism to precisely define visibility of defined data objects for a procedure. These specifications also give the type of an object and whether or not it is a variable or constant. These specification play a very important role in GYPSY, considering it is an unnested (flat) language. The operational specifications are used to precisely define the results on the external environment caused by the execution of the procedure. Finally, the internal specifications provide a mechanism to constrain the internal behavior of the procedure.

Formal specification and verification of Ada was a topic of a workshop held recently at the Institute for Defense Analyses [Roby 85]. Issues such as verification needs as perceived by the Ada community, near term solutions to Ada verification, standards as they apply to formal verification, advanced theoretical verification results applied to Ada, the interface between verification and other software engineering concerns, etc., were discussed. This workshop, and others that are planned, illustrates a growing interest in the formal specification and validation of Ada programs.

9.3.3.2.3 Structured Editors

Structured editors are one of the most developed programming-in-the-small tools. These tools allow the programmer to create at least syntactically and possibly semantically correct programs. Basically structured editors are an integration of the front end of the compiling process with the editoring process. Most often the only representation of the program being created is an abstract syntax tree (possibly attributed), with the editor supplying templates to the programmer to be "filled in". There exists a great deal of variation in their capabilities, reflecting a number of differing philosophies about just what a structured editor should be:

a. Should it provide static semantic analysis or merely guarantee syntactic correctness?

b. Should incremental compilation be concurrent with the editing process?

c. At what point does the editor allow the user to start typing -- at the statement level, at the expression level, etc.?

d. Should the editor, if it does incremental semantic analysis, force the user to immediately correct an error upon detection or issue a warning and allow the editing session to continue?

e. Should the editor allow the user to edit an abstract intermediate form (e.g., DIANA trees)?

9.3.3.2.3.1 State of the Art

The Cornell Synthesizer Generator [Reps 84], funded by NSF, makes use of attribute grammars to specify the static semantics of the language. The editing process is actually a process of pruning and grafting from an attributed tree.

CMU's ALOE [Garland 84], funded by CENTACS/CORADCOM, uses a BNF description of the language to generate an abstract syntax tree. Action routines, which triggered at node creation and deletion, provide for such things as incremental semantic analyses.

Rational Machines' Ada Environment [Caruso 85] provides an Ada specific editor with incremental semantic analysis and incremental compilation.

9.3.3.2.3.2 State of Practice

With the exception of Rational's Ada environment, which provides both semantic analysis and incremental compilation, and Digital Equipment Corporation (DEC) Language Sensitive Editor and a few other syntax driven editors, there are not yet many of these tools commercially available.

9.3.3.2.4 Visual Programming

The use of interactive graphics to improve computer programming is a relatively new field. Current programming languages and environments are typically rather difficult to use because designers have not come very far in adapting them to the user. It is hoped that bringing the more abstract properties of programs out on the computer screen would aid the programmer in designing and maintaining programs.

9.3.3.2.4.1 State of the Art

There are several completed graphical programming systems which reflect the state of the art in this field.

The Program Visualization environment [Kramlich 83] [RADC 85], developed at the Computer Corporation of America under contract through RADC and BMD, is mainly devoted to displaying graphics that represent code and data structures during execution. This is helpful to the programmer in that it enables him to monitor directly the programs control sequence and any data modifications. The user can create graphical pictures and link them to code and data, thus establishing high-level icons that can be used during execution monitoring. Graphical representation of both top-level drawings of the whole system being developed, and depictions of bottom level language code can be displayed.

Pecan [Reiss 84], developed at Brown University, is a program development system which utilizes high-resolution graphics that show a large amount of information about a program's syntax, semantics, and execution as the program is being developed. Program listings, NS-diagrams, data-type-schemas, parse trees, symbol tables, flow graphs, and execution stacks are available to the Pecan user simultaneously on the screen. The programmer may

edit the program either as text or by diagrams. Text edited programs utilize a syntax directed editor. Programs are incrementally compiled, and can be executed both forward and backward. This system provides the programmer with an excellent view of his program in a variety of manners.

PegaSys [Morconi 85], developed at SRI, is a documentation and design aid. Pictures in PegaSys describe how algorithms and data structures fit together to form the design of a larger program. High level graphical pictures are mapped into mathematical notations, which are then checked to enforce design rules, and to determine whether a program meets its pictorial documentation.

9.3.3.2.4.2 State of Practice

Due to graphical programming's relative immaturity, there is not yet a current state of practice. By far, the majority of computer programming is done modifying text, with few graphical aids available.

9.3.3.2.5 Code Generation/Compiling

One final product of any software engineering environment is the code produced for a given target. Concern exists for both space and time efficiency. The development cycle (compile, link and load or interpret) process should be acceptably fast and provide useful/informative diagnostics of both syntactic and static semantic errors. To debug the dynamic semantics and logical errors of the program, a highly interactive environment (symbolic or graphical) must be provided to interact with the program during execution.

Of increasing importance will be the issue of code generation for distributed processing. With the advent of tightly-coupled/highly-parallel architectures the need for code generators which exploit this computational power by searching for potential parallelisms in the code to be compiled is definitely of primary importance. This work is in its infancy and will need constant nurturing to reach its full promise.

9.3.3.2.5.1 State of the Art

Work on some of the more exercised aspects of compilation has and will continue: code optimization (time optimization), compiler-compilers (compiler generators), highly interactive graphical and symbolic debuggers, and structured editors which integrate the front end of a compiler with the editoring process to provide an highly interactive environment for producing syntactically correct and possibly (static) semantically correct code. But the most important work is that going on in code generation for highly parallelled processing. It is this work which will present the greatest challenge in the arena of compiler design.

9.3.3.2.5.2 State-of-the-Practice

With the preponderance of evidence in support of the need for highly parallelled and highly distributed processing in SDI, the need is for code generators capable of exploiting these new architectures. What is needed is to bring to the state of practice what is now just beginning to emerge as the state of the art.

9.3.3.3 Testing

The reliability concerns for software are a direct function of the size and complexity of the software system. Currently, testing is the usual means of assuring the reliability of large

software systems. With testing accounting for 50% to 80% of software development costs, the development of effective testing procedures is of particular concern to SDI.

The development of testing tools that are both reusable and general purpose is required in order to pursue the establishment of systematic testing methodologies. These tools can be divided into two broad categories: static analysis tools and dynamic analysis tools. An environment that addresses reliability through testing is one with a well integrated set of both static and dynamic testing tools.

Static analysis testing tools analyze the properties of software (programs) without execution. A partial list of the tools included in this category are:

a. Code Auditors measure compliance with certain coding standards.

b. Consistency Checkers check for adherence of the code to formal design specification.

c. Cross Referencers are dictionaries relating objects by their logical names and are commonly used by compilers in debugging tools.

d. Interface Analyzers check for adherence to agreed upon interfaces of program units (e.g., actual parameter lists checked against formal parameter lists).

e. Data Flow Analyzers analyze the flow of data through control structures within the program to determine such things as:

   (1) Variables that remain constant at certain points in program text

   (2) Sections of code which can be moved outside of a loop

   (3) Common sub-expressions

   (4) Definitions of a variable which may be bound to a particular usage, etc.

f. Type Analyzers check for the correct use of types used within the program text (e.g., check whether or not the types of the actual parameters correspond to those of the formal parameters).

g. Units Analyzers serve to analyze whether or not the dimensions or units assigned to an object are properly used and maintained.

Some of these static analysis tools have been incorporated into existing compilers (e.g., data flow analyzers, type checkers, etc.).

Dynamic Analysis Tools are designed to test a program during execution. The information provided by these tools is very broad. Dynamic analysis tools can generally be divided into four basic categories:

a. Symbolic Execution involves software tools designed to accept and evaluate symbolic input as the value of variables. The tool's basic operations are syntax analysis, path selection, evaluation of path constraints, constraint analysis and inequality solving. Symbolic evaluators may be used to support such efforts as: test data generation, assertion checking, path analysis and detection of data flow anomalies.

b. Test Data Generators are tools which aid the user in the development of effective test data. One of the many advantages of such tools is the development of large sets of unbiased test data. These tools have three general classifications:

(1) Pathwise test data generators

(2) Data specification systems

(3) Random test data generators

Some of the problems associated with the use of these tools are the computational efforts wasted to compute impossible paths, in checking array bounds and the generation of much test data that is duplicative.

c. Program Instrumentors are tools designed to insert investigative code segments into a program, in order to collect certain relevant statistics during code execution. These tools have three general classifications:

(1) Dynamic execution verifiers

(2) Self-metric instrumentors

(3) Dynamic assertion processors

The principle applications of these tools are coverage analysis, assertion checking and detection of data flow anomalies. The basic implementation of the tools is a preprocessing phase followed by a compilation and execution phase and finally a post-processing phase.

d. Program Mutation Analyzers are tools designed to evaluate test data based on the mutation of program text. An underlying assumption of such testing is that the program was written by experts and is consequently almost correct. These tools usually maintain a mutation "score" that is a measure of test data adequacy.

Listed above are a broad classification of testing tools, a classification of testing strategies would include:

a. Unit Testing is an effort to divide code into segments (units) which meet the following constraints:

(1) A unit is the effort of a single programmer.

(2) A unit's functional behavior is well defined by specification.

(3) A unit is intended to be integrated into the software system of which it is a part.

(4) A unit can be separately compiled and tested.

(5) A unit is essential to the functioning of the system as a whole.

It is the intention of unit testing to decompose testing to this atomic level in order to remove any errors with regards to a unit's specifications.

System Testing is an effort to test the integration of all the parts to determine whether or not the behavior of subsystems or the whole system meets specification requirements.

Integration Testing is the point at which the units of the system are tested to determine whether or not they interface as designed. Another aspect of this issue is testing how well the hardware specifications are matched to the software specifications, giving particular concern to real-time constraints. Integration of hardware to software is as vital a concern as the integration of the individual software units of the system.

9.3.4 Ancillary Factors

It is the purpose of this section to address issues that may not be directly involved in the coding process, but which certainly impact on the development of software. The issues of metrics, quality assurance, instrumentation of an environment, and reuse are covered within this section.

9.3.4.1 Metrics

Software metrics are an effort to obtain some quantitative measure of software quality. There four basic categories of product metrics:

a. Design metrics

b. Specification metrics

c. Code metrics

d. Test data metrics

Each of these categories is represented by a very broad list of metrics with differing approaches, each attempting to measure some aspect of quality.

Design metrics, that attempt to quantify design quality, have been proposed. Most of these are concerned with such issues as:

a. Does the system deign allow for easy maintenance?

b. Is the design robust?

c. How complex is the design?

These metrics are very much an emerging technology and will require further development to reach their potential.

Specification metrics measure the functions provided by a software product against the functions called for in the specifications. Most often, these metrics are the result of a thorough examination of the software, measuring the level of compliance to the specifications from the user point of view:

a. To what degree does the software comply with the specifications?

b. How critical are the specifications that are not complied with fully?

c. Has the testing procedure been a superset of the conditions that could reasonably be expected in the "field"?

d. Has the software been tested for robustness with random testing?

Code metrics attempt to give a measure of the quality of code. Syntactic complexity metrics are one measure of code quality. These metrics are an attempt to measure complexity by examining such things as:

a. Volume: the number of lines of code, the number of executable statements, the number of operators, etc.

b. Control Organization: the number of control structures.

c. Data Organization: the use and visibility of data and the manipulation of data within procedures.

Code metrics have failed to gain general acceptance; there is still a great deal of debate as to just what they measure. What is generally agreed upon is the need for more experimentation to discover how well these metrics correlate to debugging time, number of errors, etc.

Test data metrics attempt to measure how thoroughly a given set of test data exercises the software. Many such metrics have been proposed: all-uses, branch coverage, statement coverage, etc. Some of these metrics are based on data flow analysis techniques developed by compiler designers for code optimization. The majority of the work in this area is focused on the automated generation of test data to satisfy a given metric.

Types of metrics not directly measuring the products include:

a. Resources and time used on a project

b. Personnel characteristics

c. Environments and tools

d. Methodologies used

e. Performance

f. Application type and characteristics.

These metrics attempt to give some measure of issues that are perhaps not well defined, but none the less relevant to the productivity of the entire environment. Examples of data that need to be collected include are resource expenditure, software testing, software characteristics, software evaluation, etc. [Cavano 85, STARS 85b].

9.3.4.2 Quality Assurance

Unlike the manufacturing of other products, little of the software production process has been automated. The consequences of this handcrafted production process are the inevitable logic and semantic errors, which places the task of assuring functional quality on

the testing procedure. But this is a limiting view of software quality and it needs broadening to encompass such areas maintainability, robustness, flexibility and usability. These too are the hallmarks of software quality. These qualities are designed into a software product and consequently the task of assuring these qualities cannot wait until testing.

The quality assurance function is concerned with such issues as standards and methods, the quality of resources used, and reviewing intermediate products. Among the techniques used are the establishment of standards and guidelines and the performance of audits and reviews.

One approach to quality assurance that is gaining support in the software engineering community is to mandate a set of minimal metric scores that software products must achieve. Where as this might make the decision process as to what is a quality product and what is not easier, no general agreement exists on which metrics and values, if any, guarantee quality.

9.3.4.3 Instrumentation of the Environment

An environment should be instrumented to measure its own workload and its own performance (e.g., responsibility) and to help measure the projects using it (e.g., productivity). Additionally, the individual tools should be monitored to determine such things as user productivity on the tool and time required to achieve reasonable productivity.

9.3.4.4 Reuse and Conversion

With the explosive costs of software development, the reusability of software is increasingly attractive, especially when faced with the task of developing and maintaining large systems. Reusability must be a design criteria from the very beginning; if is not, very little of what is designed will be reusable. These facts have not escaped the attention of language designers and their efforts are reflected in the recent development of polymorphic languages (Russell developed at Cornell by Allen Demurs and to a lesser extent Ada). The principal strategy of this family (polymorphic) of languages is to allow the programmer to implement an algorithm without regard for the structural details of the object it is intended to operate upon. An obvious example is a sorting routine; it is not important what data structures were chosen to implement the object, just that the object have a total (or linear) ordering defined on it (in this case: two). The polymorphic sort will allow the user to sort any object passed to it on which this total ordering is defined. These languages are just now beginning to appear and will require both time and effort to mature.

A more immediate prospect is the development of reusable libraries of code. An example of this is the run time libraries offered in some language environments, most notable in FORTRAN environments. An effort in the beginning to identify areas of common need in the development of a large system is the first step in the development of reusable libraries. Then, with thoughtful effort, it will be possible to develop code general enough to meet these common needs without sacrificing efficiency.

Conversion of existing software to meet a new need is a viable alterative to beginning anew. What is needed in such an effort is well documented, well understood code in order to avoid any unforeseen pit falls. Without an excellent understanding of the software to be converted, the advantages of converting may be outweighed by the risks involved. But, as in the case of reusablity, it is possible to apply some farsighted design principles and look for areas in which there are common but not identical needs. The software for these areas may the designed with conversion as one criteria.

9.3.4.5 Validation and Verification

Verification and validation is a process of review, analysis and testing designed to ensure quality software products throughout their software life cycle. It is an integrated approach to examine (test) product performance against specification requirements.

A rigorously defined set of testing procedures, along with necessary test data sets, is essential for the verification and validation of a software product. Procedures must be established to evalute the results and determine what is acceptable behavior and what violates acceptable behavior. There must also be a procedure to guarantee standards for testing conditions to ensure reliable results.

Provisions must be made to give software developers a detailed analysis of the results of testing, especially if the software fails. This is necessary for the developer to make constructive use of such testing and to assist the developer, whose product fell short by some small measure, to make rapid corrections.

9.3.5 Design Factors for SEEs

There are many considerations which should influence the design decisions of a software engineering environment. These include overall structure and interfaces between software layers, security, database support for the environment, etc. These issues are not intended to be taken as separate and unrelated but are intended to be understood as inter-related. It is the purpose of this section to address these issues and explain their relevance to software engineering environments.

9.3.5.1 Structure and Interfaces

Once the minimum set of functions to be provided by a software engineering environment has been determined, along with the interfaces, it may then be reasonable to standardize on these functions and interfaces. This will allow both programmers and projects to migrate between systems without losing time transitioning to a new environment. These standards should be flexible enough to allow new functions (tools) to be incorporated easily into the environment. It should also allow for existing tools to be updated (enhanced) without invalidating the standard.

Another important level for standardization, besides the tool level, is the environment kernal interface level. Standardization at this level will allow for portability of tool sets and greatly decrease the time required to develop environments for new systems. Examples of this type of effort can be found in the AJPO CAIS effort and the ESPRIT PCTE effort. These efforts address the issue of defining a common (standard) set of kernal interfaces for the tool designer.

The information manager of a software engineering environment is a management layer, hosted upon the environment kernal, to provide the tool designer with primitives to create, maintain, and operate upon information. The common interfaces to the environment kernal allow the tool designer to manipulate information, in manners not provided by the information manager, without sacrificing portability. The information manager should also manage interface level resources. This will provide for a common user interface to tool features (e.g., windows) [Morton 86]. There is an ongoing effort at VHSIC to define such an information management system for an engineering environment.

9.3.5.2 Database Support

In the past, database support was added to software engineering environments as an afterthought. This has changed somewhat in the last few years. Databases are now being considered an integral activity in the design of SEEs. There are several issues of concern with respect to database support that are important to SEEs.

Choosing the appropriate data model seems to be an open question. Most recent databases are designed around the relational model. However, the needs of a software engineering environment do not seem best served by this model. Instances of entities identified within SEEs are often as many as one, several, or many. In addition, the relationships between these entities is often extremely complex. Current data models do not seem to be able to handle requirements posed by SEEs.

Another issue of concern is the level of integration the database will achieve within the SEE. If tight integration is employed, techniques that constrain, and are themselves constrained by, the software process are required. If loose integration is employed, too low level of support by the database may result [McDonald 85].

Issues such as performance, hardware support, protection, and security also need be considered. Issues such as these are fully discussed in Appendix B, Section 4, Data Management Systems.

9.3.5.3 Human Interface

The human interface to the tools of a software engineering environment is essential in the acceptance and productivity of the tools. The user should be presented with a uniform environment in which all the tools that provide the same features do so in the same manner. There should be an underlying methodology common across the entire tool set to give guidance to the tool designers. The functions of the tool set should be as orthogonal as possible to guarantee a common development thread. An environment should be tailorable to the individual user's needs and tastes.

9.3.5.4 Security

It is best to place the enforcement of security at as low a level in the system design as possible (possibly hardware) in order to guarantee that the security cannot be defeated. In a distributed environment, the operating system and the database management system are the principle managers of information, with a local area network to provide communication between nodes. It will be necessary to address the security of each of these seperately to guarantee the security of the tools that are built upon the operating system, the integrity of the data in the database and the confidentiality of messages sent over the network. An expanded explanation of these issues can be found in Sections 3, 4 & 5, Appendix B.

9.3.5.5 Operating Systems

The SDI software engineering environment(s) will be hosted on a secure and distributed operating system and must provide development and maintenance support for a secure and distributed operating system(s) hosted on a parallel architecture(s). Part of the maintenance supported by the environment must be the remote programming of the platform systems.

9.3.5.6 Tool Building Tools

Tool building tools can greatly expedite the development of tools and allow for rapid prototyping of ideas. Numerous categories of these tools already exist: structured editor

ge- rators (e.g., CMU's ALOE and the Cornell Snthesizer Generator), parser generators (e.g., UNIX-YACC) and lexical analyzer generators (e.g., UNIX-LEX). One great promise of these tools is to allow the user to customize the environment with tools tailored to his needs, instead of the current practice of making do with what is available. Another promise of these tools is to reverse the current practice of letting hardware considerations drive software design; software considerations should drive hardware design. An example of this would be a compiler that designed the optimal architecture for the program being compiled and produced both the microcode (if needed) and VLSI design of the chip. Research in hardware/software synergy is addressed in Appendix B, Section 8.

9.4 Recommendations

This section makes recommendations for the funding of research and development projects related to the software engineering environment needs of the SDI. Not all areas covered will contain specific funding recommendations. Some areas will contain general recommendations where research in that area is either not unified, or is a relatively new area.

One way to organize recommendations for funding in areas relating to software engineering environments is on an area by area basis (e.g., specification languages, verification techniques, configuration management techniques, etc.). An alternative method of organization would be to place each research area within that segment of the traditional software life cycle where it is most prevalent (e.g., requirements, design, coding, etc.). The former method will be utilized.

A very important concern in the design and implementation of any SEE would be the properties listed in 9.2.3.1, such as extensibility, portability, etc. These properties are of paramount concern to the success of the SDI SEE(s). We will address this issue separately as a recommendation, and stress their consideration in the design and implementation of any SDI SEE.

9.4.1 General Recommendations

Software engineering environments must support all phases of the software lifecycle. This is a complex task, involving many phases (from requirements to maintenance), and many functional capabilities (see 9.2.1). Any SEE must be designed and integrated with other system components such as database management systems and operating systems. A SEE is a long-lived entity, conceptually lasting from the beginning to the end of a large scale software effort. A SEE itself is an entity that experiences the phases comprising the software development life cycle. As with almost all software projects, the maintenance of a SEE is the major endeavor within it's life cycle. In addition, the SDI SEE(s) must support a diverse set of application domains covering all aspects of Battle Management C3I.

All of these complicating factors indicate that the SDI SEE(s) will be an integral element of the large-scale software project.

The strategy adopted in the SEE area needs to recognize that development SEEs (used by contractors) need not be treated the same as SEE(s) used for test, evaluation, or maintenance and that several generations of SEE technology (what the SEI calls "waves") will occur over the SDI effort's planned lifespan.

Besides generational concerns, the SDI has several options with regards to the development and use of SEE. The first option would be to simply design and implement a single, standard SDI SEE. This option has several major disadvantages. With an effort as large as

the SDI, many contractors and subcontractors will be involved. If a single SEE was implemented and all code was required to be written using this SEE, this may require all software to be developed on a specific family of hardware. In addition, potential leverage in other government SEE efforts would not be taken advantage of.

Another option would be for the SDI to specify only SEE work products and software standards (e.g., Ada, Graphical Kernel System (GKS), SQL, etc.) and have the contractors produce software conforming to these standards. This would mean that each contractor would develop code utilizing the most efficient (hopefully) means possible. There are a number of disadvantages with this approach, also. Many functions generic to software construction would be duplicated across contractors. Also, much information gathered from the construction of the software might be lost.

Another approach lies between the previous two approaches. It would be possible to define work products and software standards, tool interfaces, and supply "portable" code for many of the tools comprising the SEE. Implementations of the SEE would be provided for the most common hardware architectures. This would allow many contractors access to a SEE and other contractors aids for their SEEs.

Within the next few years a base SEE will need to be specified for maintenance of SDI software. All SDI software, when completed, would be placed under control of the base SEE. This would mandate some common tool interfaces, common software standards, configuration management and limited portability. SDI should fund studies to determine the many ramifications of choosing this particular approach.

Note that work product interchange interface standards are needed but still allow flexibility and internal variation for SEE intervals.

Recommendation 1: Place early emphasis on standardizing software work product interfaces for interchange among SEEs. Use Ada and Common LISP (for AI) as the standard programming languages.

These would apply to work products delivered for test (e.g., to NTB/NTF), evaluation, or maintenance. An approach of prototyping combined with trial-use standards with measurement and evaluation should be taken.

A basic decision, or awareness, should be made regarding the short term vs the long term SEE research and development balance. Research should be concentrating on developing technology for today's use, in addition to developing technology for tomorrow's use. The successful outcomes of advanced research must be anticipated for the SEE. The initial SEE development should provide for extensible incorporation of new tools and methodologies.

Bearing in mind that early decisions may ease or hinder migration to later SEEs or versions, the SDI needs to:

a. Shortly, make decisions on initial NTB SEE and possibly other immediate needs.

b. Within a few years, decide on a base SEE for maintenance.

c. Within a few years or less, decide on interfaces in base SEE important for extensibility and tool integration (in part to guide on technology efforts).

d. In the longer term, be able to gracefully incorporate new results.

Recommendation 2: Apply only mild constraints to SEE(s) needed immediately (e.g., initial NTF) that must be composed from what is currently available.

These constraints should include Ada and Common LISP and Internet compatibility plus possibly UNIX word/document formats, and other straightforward items.

Concurrent near and long term SEE R&D are required. The initial SDI SEE(s) must be fielded shortly. If the SDI software development program lasts fifteen years, many advances in SEE technology will surface during that time.

Many government agencies exist with which to leverage current SEE technology (e.g., RADC, SDC, NASA, DARPA, WIS, STARS, etc.). An immediate concern of the SDI is to assess and evaluate the ongoing SEE efforts in greater detail. Many of these efforts have a great many ideas/innovations that would be desirable for the SDI SEE. It may be able to "pick and choose" those innovations found, and incorporate them into the SDI SEE. For instance, NASA has drafted a preliminary statement of work for the planned SEE for the Space Station [NASA 86]. Efforts such as these should allow the SDI to "piggy-back" some of the necessary technology needed for fielding its SEE(s).

There will be need for much coordination between the various government agencies that will be able to provide leverage for SEE developement. Such coordination is needed for determining what is needed for the SEE, and determining exactly how to get there compatibly.

Recommendation 3: Position the SDI SEE effort to benefit over the next few years from several SEE efforts. These include the NASA Space Station, WIS, SEI, and DARPA efforts.

Indeed, SDI should end up obtaining much of its basic SEE from the more successful of these efforts.

Recommendation 4: SDI should continue to fund research in developing effective, efficient, and extensible methods of integrating software tools into an environment.

Work such as the Arcadia environment provides a good framework to begin developing integrated environments. This may be the most important issue confronting the SDI SEE(s), and should be given a high priority.

The properties in 9.2.3.1 cannot be stressed enough. It is imperative that the SEE(s) be tightly integrated, with goals such as extensibility, portability, etc. There is insufficient understanding of how these tools can be effectively knitted together into something that can support users now and that will become increasingly powerful to users as time goes on -- and without disrupting their work from time to time as bright new ideas emerge. The Arcadia project seems to be the best example of state of the art research striving to achieve these goals. An underlying object management system should be considered a must for the SDI environment. The Arcadia environment is referenced in several other areas in this report, and should be studied to determine the extend of its applicability for developing software systems. DARPA is currently developing an Ada engineering environment

utilizing an object management system running on the MACH operating system. This system is being funded by the SDIO, and may prove very suitable for the software development needs of the SDI

Interfaces internal to SEE are important for tool compatability and integration. Compatibility with target systems, simulation facilities, systems and hardware engineering environments, and other interacting systems also deserve attention.

9.4.2 Specific Areas

It is recommended that SDI fund research into specific areas that would result in prototype tools that may be incorporated into the environment discussed in Recommendation 4.

> Recommendation 5: SDI should fund tools to incorporate into the standard environment framework of recommendation 4. These include tools for methodology, requirements development, rapid prototyping, support, visual programming formal verification, code generation/compiling, testing, metrics and data base support. The state of practice in most of these areas is not suitable for an SDI SEE.

Details of these tools are given in 9.4.2.1 through 9.4.2.5.

9.4.2.1 Requirements and Design Development

This section of the report discusses recommendations with regards to requirements and design development concerns of software engineering environments. Areas discussed are (1) methodologies, (2) requirements development, and (3) artificial intelligence.

9.4.2.1.1 Methodologies

As seen in 9.3.3.1.1, a great many methodologies exist for constructing software. Many of the methodologies discussed have existed for a number of years, and have experienced success when dealing with large-scale projects. In the short-term, SDI must determine which methodology, or methodologies, are suitable for initial software development.

Currently, SDC is incorporating modern software engineering principles into DCDS specifically for SDI requirements. It is not yet clear whether this work will provide SDIO with a methodology that will satisfy all of SDI's requirements.

In the mid- to long-term, SDI should monitor advances in software methodology development, and fund research and development of advanced methodologies in the light of future SDI technical requirements. There are currently few research directions being aimed at design methodologies for combining distributed, ultra-reliable, highly parallel, fault-tolerant or secure software. Each of these properties of software is extremely important to the SDI, yet the state of the practice is far away from implementing them with design methodologies. SDI should consider strongly funding research in these areas.

9.4.2.1.2 Requirements Development

Requirements development is perhaps the least automated and least understood phase of the software life cycle. In addition, errors in this phase are the most costly to repair later in other phases. SDI's needs for requirements development will be demanding. A high priority must be given to ensure confidence that the system properly reflects its

specifications. Complicating this is the fact that the requirements will be changing frequently. While other techniques, such as rapid prototyping, will enable modeling and simulation of the system requirements, means must be provided to ensure, at each step of the way, that current requirements reflect the current implementation. In the short term, SDI should sponsor research into developing means of validating requirements (e.g., through specification languages, simulators, animation, etc.).

This is an area within software development that is of paramount concern to the SDI SEE(s), yet one that is relatively weak.

In the long term, much more formal methods should be developed into practical tools. If the whole design and implementation process were completely and consistently automated, much more confidence could be held in a system.

9.4.2.1.3 Rapid Prototyping

Major new developments in rapid prototyping methodologies have not yet reached maturity within the research world. However, rapid prototyping seems to be an area of research that could yield high payoffs for the SDIO in both the short and long term. In the short term, methodologies and supporting tool environments that can be used for validating user requirements quickly need to be explored and developed. In the long-term, efficient methods of rapidly prototyping implementations should be studied. Basic research should be funded in areas relating to prototyping (e.g., executable requirements and design specification languages).

Significant payoff can be expected from gains in the area of rapid prototyping.

9.4.2.2 Coding

This section addresses the recommendations regarding coding (programming), and tools/issues that will be of importance to the SDI program. Two issues, visual programming and formal verification; and one tool, code generators, is covered.

9.4.2.2.1 Visual Programming

The state of the art in visual programming has progressed dramatically in the last few years. However, it seems that few developments, useful to the SDI, will occur in the short term. One possible area where visual programming techniques may play an important role is in distributed systems. Pictorial representation of state and other data in a distributed environment may be useful to the SDI. In both the short and long-term, SDI should monitor the developments in this area.

9.4.2.2.2 Formal Verification

While it is desirable to formally verify as much of a large system as possible, a realistic approach is to discriminate critical segments of code for formal verification. In the short term, formal verification of critical segments of software is the advisable strategy for SDI's software needs. As verification of larger pieces of code becomes feasible, this added capability can be incorporated at that time.

The development of verification technologies and tools should be centered around Ada and Common LISP. It should be pointed out that a critical component of any verification effort is the need to develop formal specification languages. It is with these languages that the specific makes explicit the functional requirements of the program. The subsetting of these

languages (Ada and Common LISP) for the explicit purpose of verifiability may, at some point, become an attractive prospect for the SDI program.

An outstanding program concerned with formal verification is the University of Texas Gypsy program. It would be beneficial to exploit the existing expertice at Texas and guide the effort there to better accommodate the needs of SDI for verification technologies and tools. Other verification efforts should not be overlooked. The principal emphasis behind any SDI effort should be the development of practical tools. These would relate rigorous specification and design notations (see prior subsection) to Ada and possibly Common LISP.

9.4.2.2.3 Code Generation/Compiling

The principal unique SDI need in this area will be for generators of highly optimized code for highly-parallel and highly-distributed architectures. The development of code generators for pipeline and vector architectures has been an ongoing research effort at numerous universities (Rice University, University of Illinois at Urbana-Champaign, etc.) and several commercial establishments (Cray Research, IBM, Control Data, etc.). The same effort is lacking with regard to the MIMD architectures. With the exception of a few academic and commercial efforts (University of Illinois' PARAFRASE and Denelcor's HEP), the issue of code generators for MIMD architectures is still very much an emerging technology. SDI should fund the development of prototype code generators for highly parallel and highly distributed architectures.

9.4.2.3 Testing

Testing is still the principal means for ensuring reliability within a software product and the quality of a given testing procedure is directly related to how well the test data exercises the software. Futher studies in automated test data generation, from formal specifications and program text, and in which test data metrics provide some measure of quality, is recommended.

Work in this area is ongoing under the auspices of the Air Force Office of Scientific Research and others.

The best combinations of types of testing and other quality assurance methods and measurements for practical use need to be discovered and then systematically improved. Methods for validating sets of test data as reflecting the real-world situation and requirements need to be explored. Testing of special kinds of software (distributed, highly parallel) and for special properties (security, reliability, survivability, real-time performance) also need to be explored.

Data collection should be performed to arrive at a sound basis for using all the evidence and test results to predict the merit of a software system, subsystems, or version and make decisions on its acceptability. The National Test Bed facility should be utilized for these, and other, purposes.

It would also be worthwhile for the SDI effort to support studies in the theoretical (mathematical) and empirical foundations of testing to better understand both its limitations and its potential. This is an area of obvious neglect. These studies will also be useful in directing work in practical testing methods.

9.4.2.4 Metrics

An important need for the SDI program is to have some method to measure the performance of both its software engineering environment and the users of that environment and the products of the environment. Metrics provide one vehicle for such measurements, as well as measuring software atttributes such as portability, readability, etc. SDI should examine both product (software) metrics and project metrics to determine just how well these metrics measure what they purport to measure and their usefulness.

Best would be a comprehensive measurement program combining a required minimum set of measures with special in-depth measurements, facillitated by the software engineering environment. This would provide a common ground from which all SDI software could be evaluated.

9.4.2.5 Other Considerations

9.4.2.5.1 Reuse

Even if one accepts the more modest estimates of the SDI's software needs, the size of the project warrants futher study in the area of software reusability. With the advent of polymorphic languages (e.g., languages supporting multiple methodologies, such as Ada), the development of large libraries of reusable code is possible. The extent to which reusable code will be of value to SDI is an open question and certainly should be pursued.

9.4.2.5.2 Database Support

There are many research activities which need to be undertaken with regards to database support in software engineering environments [Nestor et al. 86, p. 151-2]. Although in the past database management systems research technology has received much attention, little attention has been focused on software engineering environments. Therefore, this area is less understood than other areas.

Activities such as integrating database facilities into SEEs, the development of database tools for software development, and studies to determine what data models may be better suited for use in SEE's should be addressed. In the long term, more complex activities should be undertaken. Implementation of better data models should be pursued. More tools will need to be integrated into the SEE. Research should be undertaken on how multiple user views can be supported in a distributed SEE.

9.4.3 Summary

The SDI needs to have an overall effort to develop SEE interface standards, a SEE for post-delivery software support, and SEE capability usable by developers. The effort should build on other efforts such as the WIS or the NASA Space Station efforts.

There are many areas of technology related to software engineering environments that should be researched and developed. An object management system, standard software development methodologies, specification languages, metrics standards, etc., will all need to be developed for the SDI SEE(s). In addition, prototype tools and applications need to be developed in the areas of database support, software reuse, testing, and other fields.

9.5 References

[Addleman 85] Addleman, D., Davis, M., and Presson, P., *Specification Technology Guidebook*, Rome Air Development Center, (August 1985).

[Alford 85] Alford, M., *DCDS Environment Architecture Description*, 4567G950-002R1, TRW Defense Systems Group, Huntsville, AL, (December 1985). (JSSEE Report Number JSSEE-ARCH-005).

[Balzer 82] Balzer, R., Goldman, N., Wile, D., "Operational Specifications As the Basis for Rapid Prototyping," *ACM Software Engineering Notes* 7,5, (December 1982), pp. 3-16.

[Beizer 84] Beizer, B., *System Testing and Quality Assurance*, Van Nostrand Reinhold, (1984).

[Budde 84] Budde, R. et al., *Approaches to Prototyping, Proceedings of the Working Group on Prototyping*, Springer-Verlag, (1984).

[Caruso 85] Caruso, Denise, "Development System Breaks Productivity Barrier, *Electronic* (July 8, 1985), pp. 36-40.

[Cavano 85] Cavano, J., "Towards High Confidence Software," *IEEE Transaction on Software Engineering* SE-11, 32, (December 1985), pp. 1449-1455.

[Davis 82] Davis, A., "Rapid Prototyping Using Executable Requirements Specifications," *ACM Software Engineering Notes* 7, 5, (December 1982), pp. 39-44.

[Garland 84] Garland, David B., and Miller, Philip L., "GNOME: An Introductory Programming Environment Based on a Family of Structured Editors," *Proceedings of the ACM SIGSOFT/SIGPLAN Software Engineering Symposium on Practical Software Development Environments*, Carnegie-Mellon University, (May 1984).

[Goldberg 83] Goldberg, A., and Robson, D., *Smalltalk-80: The Language and its Implementation*, Addison Wesley Press, 1983.

[Green 83] Green, C., Luckham, D., Balzer, R., Cheatham, T. and Rich, C., *Report on a Knowledge-Based Software Assistant*, Technical Report RADCTR-83-1985, Rome Air Development Center, Griffis Air Force Base, New York, (August 1983).

[IITRI 85] *IIT Research Institute, Interim Software Data Collection Forms Package*, IIT, Rome, New York, (June 1985).

[JASON 85] *Report on the Workshop for Automated Software Programming*, JASON Report Study, (December 1985).

[Kishida 84] Kishida, K. "E: A Note on the Sigma Project," Software Technology Committee, Japan Information Services Industries Association, (December 4, 1984).

[Konchan 83] Konchan, T., and Klausner, A., *Rapid Prototyping and Requirements Specification Using PDS*, Harvard University, (1983).

[Krasner 83] Krasner, G., *SmallTalk-80: Bits of History, Words of Advice*, Addison-Wesley, Reading, Massachussetts, (1983).

[Kramlich 83] Kramlich, D., Brown, G. P., Carling, R. T. and Herot, C. F., "Program Visualization: Graphics Support for Software Development," *Proceedings ACM/IEEE 20th Design Automation Conference* (June 1983), IEEE-CS Press, Los Alamitos, Calif., pp. 143-149.

[Luckham 85] Luckham , David C. and Von Henke, Friedrich W., "An Overview of ANNA, a Specification Language for Ada," *IEEE Software* (March 1985), Stanford University.

[McDonald 85] C. McDonald, W. Riddle and J. Wileden, *Environment Extensibility Impact on the STARS SEE Architecture*, Institure for Defense Analyses, Paper P-1828, (April 1985) (draft), pp. 20-21.

[Morton 86] Morton , R. and Redwine, S., Information Interface Standards for Software Engineering Environments, *1986 Computer Standards Conference*, (May 13-15 1986).

[Morconi 85] Moriconi , M. and Hare, D.F., "Visualizing Program Designs Through PegaSys," *IEEE Computer* 18, 8, (August 1985).

[NASA 86] "Space Station Software Support Environment," Phase C/D, Statement of Work, NASA, (Draft May 1, 1986).

[Nestor 86] Nestor , John et al. *Technology Identification and Assessment: A Survey of Software Development Environment Technology*, Software Engineering Institute (SEI-86-MR-4), (February 28, 1986).

[RADC 85] RADC briefing papers.

[Reiss 84] Reiss, S. P., "Graphical Program Development with PECAN Program Development Systems," Proceedings ACM Sigsoft-Sigplan Software Engineering Symposium Practical Software Development Environments, April 1984, printed as SIGPLAN Notices 19, 5, (May 1984), pp. 30-41.

[Reps 84] Reps , Thomas and Teitelbaum, Tim, "The Synthesizer Generator," *Proceeding of the ACM SIGSOFT/SIGPLAN Software Engineering Symposium on Practical Software*

Development Environments, Cornell University, (May 1984).

[Riddle 85] Riddle , W. and Williams, L., *Software Environments Workshop Report*, Rocky Mountain Institute of Software Engineering, (December 1985).

[Roby 85] C. Roby, editor, *Proceedings of the First IDA Workshop on Formal Specification and Verification of Ada*, IDA Memorandum Report M-146, (March 18-20, 1985).

[Ross 77] Ross, D., Structured Analysis(SA): A Language for Communicating Ideas, *IEEE Transactions on Software Engineering* SE-3, 1, (January 1977), pp. 16-34.

[Rzepka 82] Rzepka, W., *Using SREM to Specify Command and Control Software Requirements*, Technical Report RADC-TR-82-319, Rome Air Development Center, Griffiss AFB, NY, (December 1982).

[Schoman 77] Schoman, K.E., and Ross, D.T., Structured Analysis for Requirements Definition, *IEEE Transactions on Software Engineering* SE-3, 1 (January 1977), pp. 6-15.

[STARS 85a] *STARS Joint Service Team for Software Engineering Environments, STARS-SEE Operational Concept Document*, proposed version 001.0, (October 1985).

[STARS 85b] "STARS Interim Software Data Collection Forms," DACS Rep., (1985).

[STARS 86] *STARS Joint Service Team for Software Engineering Environments, STARS-SEE Preliminary System Specification*, (January 1986).

[Taylor 82] Taylor, T., Standish, T., "Initial Thoughts on Rapid Prototyping Techniques," *ACM Software Engineering Notes* 7, 5, (December 1982), pp. 160-166.

[Tietelman 84] Tieltelman, W., "A Tour Through CEDAR," *IEEE Software*, (April 1984).

SECTION B10

Simulation

Prepared by Michael R. Kappel

Topics covered in Section B10:

10.0 SIMULATION

10.1 Introduction
10.1.1 Purpose and Scope
10.1.2 Background
10.1.3 Organization of Document

10.2 Requirements
10.2.1 Goals
10.2.2 Simulations
10.2.3 Models

10.3 Current Status
10.3.1 Introduction
10.3.2 Simulation Projects
10.3.2.1 Issues
10.3.2.2 SDI Simulation Facilities
10.3.2.3 Existing SDI Simulations
10.3.2.4 SDI-Related Models
10.3.3 Simulation Programming
10.3.3.1 Issues
10.3.3.2 State of the Art
10.3.3.3 State of Practice
10.3.4 Simulation Support
10.3.4.1 Model Description Aids
10.3.4.2 Presentation Aids
10.3.4.3 Integration Aids
10.3.5 Model Validation
10.3.6 Data Analysis
10.3.7 Hardware/Software Integration

10.4 Recommendations

10.5 References

10.0 SIMULATION

10.1 Introduction

10.1.1 Purpose and Scope

This section assesses the software engineering process of simulation for evaluating the various components of the Strategic Defense Initiative (SDI) System. The results of a preliminary investigation into the current theory and practice in simulation are presented. Those technical aspects of simulation that pertain to unique SDI software requirements are identified and assessed. This section provides broad guidelines and direction for research efforts into fundamental simulation technology that is promising for SDI software.

10.1.2 Background

The task area of simulation should not be considered in isolation. It relates to other task areas addressed in this Software Technology Integration Plan. Simulation is closely associated with the Reliability task area (see Appendix B, Section 11) which addresses verification and validation (V&V). Simulation plays a V&V role in the testing of both hardware and software components of the SDI system. Furthermore, the simulator, as a software component itself, must be deemed reliable through V&V methodology.

Simulation is also associated with the application-specific task area of Battle Management/Command, Control, Communications (BM/C3) (see Appendix B, Section 2). Engagement scenarios will be simulated to evaluate the performance of battle managers. Input to the BM/C3 module will be simulated for sensor and weapon systems. Moreover, the step-wise development of the BM/C3 module may rely on simulating as yet undeveloped BM/C3 components to test developed software and hardware.

Simulation also relates to the software foundation technology of the SDI man-machine interface (MMI) (see Appendix B, Section 6). The simulation of real scenarios not only supports user training but provides feedback to assess the effectiveness of the man-machine interface. Experimental analyses of user-system interactions on a simulator will provide insight into the design and integration of tasks to optimize human performance.

Simulation relates to the task area of software engineering environments (SEE) (see Appendix B, Section 9). Since simulators may be large and complex software projects, it is important that sound engineering practices be employed during their development. Thus the results of the SEE task area assessment should be applied to the construction of SDI simulators.

10.1.3 Organization of Document

Section 10.2 proposes generic requirements for simulation in the SDI environment. Section 10.3 details various technological subareas of simulation. For each subarea, issues relevant to SDI are delineated, the state of the art and the state-of-practice are documented, prospects for future developments are conjectured, and key participants in ongoing research and development efforts are listed. Section 10.4 tenders recommendations for the most auspicious course for SDI development and acquisition of simulation technology. Section 10.5 contains bibliographic references.

10.2 Requirements

10.2.1 Goals

Simulation in the SDI environment will serve many purposes:

a. Supplementing the testing and debugging of battle management software

b. Comparing the performance of alternative component designs

c. Evaluating the performance of SDI components and subsystems

d. Evaluating and comparing various designs for BM/C3 architectures

e. Evaluating SDI system improvements and updates

f. Exercising the SDI system in simulated attack scenarios to supplement traditional testing techniques

g. Training users of the SDI system to ensure efficient operation during a real engagement

h. Evaluating battle management strategies and tactics

i. Analyzing the offensive, defense-suppression, and responsive Soviet threats.

j. Evaluating algorithms for tracking, targeting, kill assessment, etc.

k. Evaluating sensitivities of SDS performance with regard to underlying assumptions of threat, vulnerability, lethality, etc.

l. Evaluating computer resource and performance requirements.

10.2.2 Simulations

To achieve the goals listed above, a variety of simulations will be executed:

Real time vs. Non-Real time: In real-time simulations, SDI components are operating and interacting in real time. Non-real time implies either slower or faster than real time. Simulations may be conducted slower than real time to allow for processing delays associated with high fidelity models, for example. Simulations may be conducted faster than real time to contract the duration of a lengthy engagement.

Simulated vs. Emulation: Simulations may be conducted in which all SDI components are simulated such as in a preliminary evaluation of a candidate BM/C3 architecture. Alternatively, actual hardware may be substituted for its simulated counterpart for in-line testing purposes.

Man-in-the-loop: Since humans will play a critical role in the strategic defense system (refer to Section B6, Man-Machine Interface), simulations will be run which incorporate human interaction. In early SDI BM/C3 architecture evaluations, such simulations will help draw the line between automated and manual components by evaluating the trade-off between timely response and positive control. Man-in-the-loop simulations will also be used to train operators and commanders to maximize efficiency and accuracy during a real engagement.

End-to-end vs. Phase: The defensive engagement of offensive ballistic missiles may be partitioned into distinct phases (boost, post-boost, early midcourse, late midcourse, and terminal) with different operational and performance requirements. An end-to-end simulation would be performed to evaluate the performance of a subsystem or architecture over all phases of the battle. Alternatively, simulations could be run for a specific phase or phases.

Engagement vs. Support: Engagement simulations will be run to evaluate the performance of SDI components, subsystems, architectures, strategies, etc. Support simulations will be used to configure computer systems at simulation facilities. For example, simulation of computer resources at the National Test Facility will help optimize their configuration to support engagement simulations at the National Test Bed (see Section 10.3.2.2).

Levels of Fidelity: Level of fidelity measures how closely the real behavior of a system is modeled. Early engagement simulations may be of low fidelity. As operational characteristics become more clearly defined, models would reflect increased realism. In-line testing of an actual component may require high fidelity models of units that interface closely with the one being tested. High fidelity, real-time, end-to-end engagement simulations may not be achievable in the near future due to limits of computer power.

Distributed: Simulations may involve the coordinated operation of geographically distributed facilities. For example, a technology validation experiment (TVE) may require the interaction of the Advanced Research Center, Los Alamos National Laboratory, and the National Test Facility (see Section 10.3.2.2).

10.2.3 Models

To achieve the simulation goals listed above, most SDI components will require modeling. A first level decomposition of BM/C3 model components is as follows [Titan 86]:

1.0 Battle Management

1.1 Architecture

Automated BM
Intermediate BM level requirements
Inter-tier interfaces
Intra-tier interfaces
Alternate BM architectures

1.2 Algorithm

Optimization
Data integration
BM space partition/dynamic allocation
Weapon allocation/engagement control
Weapons control
Sensor/sensor correlation
Target discrimination
Kill assessment

1.3 Computer Resources

Computer processing loads
Information networking techniques
Large scale information management
Spaced-based platforms
Computer security/data integrity

2.0 Command and Control

C2 display
Response time
C2 data requirements
Human decision making
Decision aids
Distributed data base requirement
C2 functional assignment
C2 architecture definition
C2 interface analysis
System loading

3.0 Communications

Connectivity/reliability
Message transit time
Data switching
Beam pointing/tracking
Error control
Remote COMSEC control
Channel availability
Communications quality
Multiplexing
Communications protocol
Network synchronization

Detailed analyses of the functional characteristics of these models are beyond the scope of this report, but can be found in [Frenkel 86], [Martin 86] and [Turner 86].

10.3 Current Status

10.3.1 Introduction

Simulation has been defined as "the process of designing a computerized model of a system (or process) and conducting experiments with this model for the purpose either of understanding the behavior of the system or of evaluating various strategies for the operation of the system" [Shannon 75]. Simulation modeling offers five advantages over other problem solving methods [Adkins 77]:

a. Controlled experimentation

b. Sensitivity analysis

c. Does not disturb the real system

d. Effective training tool

e. Time compression

The simulation approach, on the other hand, has four disadvantages [Adkins 77]:

a. Large costs may be incurred in manpower and computer time.

b. Development time may be extensive.

c. Model may diverge from reality.

d. Model parameters may be difficult to initialize.

These drawbacks are expected to impact the SDI project considering its magnitude. The first two, however, may be minimized through the use of software development tools and sound engineering practices. The latter two drawbacks may be reduced through the application of model validation techniques and through careful design of simulation experiments.

In this section, six simulation subareas are examined - simulation projects, simulation programming, simulation support, model validation, data analysis and hardware/software integration. For each subarea, we will review relevant issues, the current state of the art, the state of practice, future prospects, and performers and funders.

10.3.2 Simulation Projects

10.3.2.1 Issues

Simulation has been applied to a wide variety of problems in diverse disciplines. Past experiences with simulation projects provide a wealth of information for their effective design, implementation, and application. SDI contractors must draw upon this knowledge base to avoid reinventing the wheel. General principles and methods in the art and science of simulation should be understood and applied.

Specific knowledge gained in related simulation projects may be useful in the SDI environment. While no one project compares in scope or addresses the same problems as SDI, selected aspects of a number of simulation projects may be relevant. To save on the development costs of entirely new simulation software, it may be possible to adapt and integrate existing computer programs that may satisfy at least some of SDI simulation requirements [Kaufman 84].

10.3.2.2 SDI Simulation Facilities

Comprehensive facilities for the simulation and evaluation of SDI components and subsystems and BM/C3 architectures have been proposed and are being analyzed by the Armed Services and government contractors.

a. National Test Bed (NTB)

The NTB will provide a "comprehensive capability to demonstrate and independently evaluate alternative Strategic Defense Initiative System and Battle Management/Command, Control, Communications (BM/C3) architectures and key defensive technologies" [NTB 86]. These goals are to be accomplished

through end-to-end engagement simulations at varying levels of granularity with hardware in the loop and man in the loop. Technology Validation Experiments (TVEs) will be conducted to test key SDI components and subsystems in a real-time, distributed environment.

The NTB comprises a National Test Facility (NTF) and a dynamic membership of remote facilities such as the U. S. Army Advanced Research Center (ARC), U. S. Air Force Electronics Systems Division (ESD), Los Alamos National Laboratories, etc. The NTF will be a major computing facility forming the hub of the NTB network.

The NTB development will be accomplished in three phases:

(1) Concept Definition

(2) Preliminary Design, and

(3) Implementation and Integration.

Four contractors -- Martin-Marietta, Rockwell International, TRW, and Boeing -- competed in Phase 1. Martin-Marietta and Rockwell International are currently working on Phase 2. The Request for Proposal (RFP) for the open competition for Phase 3 is currently being drafted. An initial operating capability (IOC) is expected in 1988.

b. Architecture Simulation Center (ASC)

The ASC will support the evaluation of candidate BM/C3 architectures by performing end-to-end engagement simulations. The ASC will serve two primary functions:

(1) Be a testing ground for designing and revising battle management architectures, and

(2) Be a high level screen so that only robust architectures move to the NTB for more comprehensive assessments.

SDI components (sensors, weapons, BM/C3, environment, threat) will be modeled at an appropriate granularity to support architecture evaluations. The definitions of battle management architectures will plug into the models to drive the simulations. A comprehensive series of experiments can then be conducted via parametric excursions.

c. BM/C3 Experimental Version (EV)

The EV is a prototype of the BM/C3 subsystem, chartered to perform analyses of its tactical configurations and to demonstrate achievement of required technical performance. The U. S. Army Strategic Defense Command has proposed an EV program that extends the functionality of the Advanced Research Center (ARC) to run distributed battle management experiments. An initial capability to validate BM/C3 algorithms and architectures is to be developed by 1988 and is named EV88. An EV will be demonstrated through the National Test Bed in a system-wide simulation against operational threat

levels. The goal is to demonstrate end-to-end BM prototype architectures by 1990 to a support full-scale development decision for strategic defense.

d. Simulation Display Facility (SDF)

As part of the National Test Bed, the SDF is a proposed facility for the visual presentation of SDI simulation experiments to engineers and politicians. Such graphic evidence is envisioned to chart progress on key defensive technologies and to support a full-scale development decision for SDI. The SDF will incorporate state-of-the-art computer graphics and display technologies.

10.3.2.3 Existing SDI Simulations

SDI simulators have undergone intensive development for the past couple of years by the Armed Services and by government contractors. Existing simulators model a wide variety of SDI components and typically generate graphic renditions of the engagement. However, these simulators suffer from limitations in computing power resulting in unrealistic simplifying assumptions, severe reductions in the engaged threat, slower than real-time operation, or limited scope of engagement. Even so, valuable experience has been gained from their implementations. A partial list of SDI simulation projects include:

a. Strategic Defense Initiative Simulation [McDonnell 1986]

The Strategic Defense Initiative Simulation (SDISIM) was employed by the McDonnell Douglas Astronautics Company for the BM/C3 System Architecture Study. SDISIM includes all tiers of the SDI system and models the threat, BM/C3, sensor and weapon elements. The simulation was used to determine system performance and key BM/C3 timeline sensitivities.

b. Midcourse Battle Management Experiment [Shaw 86]

The Midcourse Battle Management Experiment was developed at the Advanced Research Center (ARC) of the U.S. Army Strategic Defense Command. It was designed to be a driver for high-fidelity communication routing and weapon-target assignment algorithms. This simulation collects low-level message traffic and target assignment data for analysis and algorithm refinement.

c. Layered Defense Performance Evaluator [Frenkel 86]

The Layered Defense Performance Evaluator (LPDE) is an end-to-end simulation of the SDI environment to evaluate the performance of candidate SDI defensive architectures for comparison. The comprehensive engagement model comprises three major elements - boost/post-boost, early midcourse, and late midcourse/terminal - that could be run separately or as a whole in the face of flexible threat parameters. The simulator was developed by the Institute for Defense Analyses, coded in FORTRAN and executed on a VAX computer.

d. Architecture Effectiveness Model [Shaw 86]

The Architecture Effectiveness Model (AEM) is a generic simulation model being developed by Alphatech, Inc. under the auspices of the Naval Air Development Center. AEM is to be a set of non-proprietary, high fidelity, end-to-end simulation models that could be distributed among cognizant government

organizations and to the SDI BM/C3 community at large for the purpose of quantifying the performance of alternative SDI BM/C3 architectures.

e. Battle Management Engagement Simulator [Harris 86]

The Battle Mangement Engagement Simulator was developed by General Research Corporation (GRC) and used to evaluate the performance of candidate BM/C3 architectures. As part of the BM/C3 System Architecture Study, the Government furnished twelve SDI weapon/sensor configurations, time phased from near-term configurations with boost defense only and minimal kinetic weapon inventory to far-term configurations with boost, post-boost and midcourse systems with a full complement of kinetic and directed energy weapons and discriminators. The various configurations were run against a baseline threat and threat excursions.

f. SATIN

Development of the SATIN project began several years ago for detailed simulation of probe sensors. Its purpose has since been redirected to simulations of full midcourse engagements to include target intercept and kill assessment. SATIN was running at the ARC until project development was discontinued for lack of funds at the beginning of FY 87.

g. Defense-in-Depth Simulator (Teledyne Brown Engineering)

Development of the Teledyne Brown version of the Defense-in-Depth Simulator (DIDSIM) began about three years ago. It was designed as an end-to-end engagement simulation of all defensive functions. Models are of medium fidelity with the threat assessed on an individual basis. DIDSIM was running at the USA SDC Simulation Center until project development was discontinued for lack of funds at the beginning of FY 87.

h. Defense-in-Depth Simulator (Sparta)

The Sparta version of DIDSIM comprises separate simulations for the boost, midcourse, and terminal tiers. Though not fully integrated, output from the previous tier is input to the subsequent tier for a complete end-to-end simulation. This low fidelity simulation is fully operational on a VAX 11/780.

i. Tactical Warning/Attack Assessment

The Tactical Warning/Attack Assessment (TW/AA) project is a high-fidelity simulation of existing sensors. TW/AA was developed by Teledyne Brown to run at the ARC.

j. SIDSIM [Titan 86]

SIDSIM was designed to measure and evaluate interactions between BM/C3 tiers and regions, analyze pre-commit/post-commit activities of various sub-system concepts that may include any combination of boost, post-boost, mid-course, or terminal sub-systems. SIDSIM is currently operational at Teledyne Brown Engineering and is continually being enhanced to meet SDI analysis requirements.

10.3.2.4 SDI-Related Models

Numerous models have been implemented in other military simulation projects that can be employed in SDI simulations. Existing models of engagement, sensors, weapons, environment, communications, etc. may be incorporated into an SDI simultion portfolio. A survey of such models is beyond the scope of this report, but can be found in the BM/C3 Concept Definitions submitted to the U.S. Army Strategic Defense Command -- [Ford 85], [GRC 85], [Hughes 85] and [TRW 85], and in other evaluations for the USA SDC -- [Titan 84], [AT&T 85] and [Titan 86].

10.3.3 Simulation Programming

10.3.3.1 Issues

Many simulation programs perform similar functions [Graybeal 80]:

a. Generate random variables.

b. Manage simulation time.

c. Handle routines to simulate event executions.

d. Manage queues.

e. Collect data.

f. Summarize and analyze data.

g. Formulate and print output.

These functions are not inherently supported by a general-purpose programming language like Ada. It is possible to build this functionality in such a language as is evidenced by the fact that the majority of simulation applications are still coded in FORTRAN. However, the redundant effort of recoding common functions for each SDI simulation would increase development costs and decrease program portability.

There are two alternative approaches to eliminate these potential problems. *First,* simulation programs may be coded in a special-purpose simulation language. Languages designed specifically for the purpose of computer simulation provide certain useful features [Shannon 75]:

a. Reduce the programming task.

b. Provide conceptual guidance.

c. Aid in defining the classes of entities within the system.

d. Provide flexibility for change.

e. Provide a means of differentiating between entities of the same class by characteristics or properties.

f. Describe the relationship of the entities to one another and to their common environment.

g. Adjust the number of entities as conditions vary within the system.

The second approach is to package standard simulation functionality in Ada. Common functions may be coded once to build a package to be accessed by all simulation application programs. Furthermore, advanced Ada features such as operator overloading, user-defined data types, and generics may be utilized to support processing requirements unique to simulation. Simulation programs may then be written in Ada and invoke a standard set of high-level simulation tools.

10.3.3.2 State of the Art

A handful of researchers at academic institutions are developing simulation facilities for Ada applications. The following demonstrate the state of the art in Ada simulation programming:

a. Ada Simulation Support Environment (ASSE) [Adelsberger 82]

   ASSE provides simulation facilities to cover all forms of combined continuous and discrete modeling techniques. Its design is similar to the Ada Programming Support Environment (APSE) concept in which support tools are grouped around the central Ada language. Different packages are organized in a hierarchical and parallel manner to support simulation model design and verification, the software development process, and data analysis.

b. Simulation and Modeling on Ada (SAMOA) [Lomow 82; Inkster 84]

   SAMOA is a fully integrated, general purpose, discrete event simulation package. It uses the tasking and data abstraction constructs of Ada to provide process-oriented simulation facilities similar to those of the Discrete Event Modeling on Simula (DEMOS) package. Extensions to SAMOA provide general and unrestricted facilities for combined discrete and continuous simulation in Ada. Other features include statistics collection, report generation, and event tracing.

c. Discrete Event Simulation [Bruno 84]

   Simulation facilities for a discrete event environment in Ada were proposed. A process interaction approach was employed to test the correctness of programs for real-time applications. The standard simulation package includes facilities for (1) process scheduling, (2) definition of entities, (3) random number generation, and (4) statistics collection.

d. Ada Environment for Simulation Studies [Friel 85]

   Ada packages to support event and process oriented simulation have been developed at Texas A&M University. These packages include facilities for queue handling, random number generation, automatic statistics collection, and simulation control.

10.3.3.3 State of Practice

The Ada programming language was used to implement several of the simulators surveyed in 10.3.2. Those simulators did not, however, develop and/or employ an Ada package of

basic simulation functions. Such functionality was embedded within the source code of the overall system. Although proving the suitability of Ada for simulation, these projects suffered from increased development costs and decreased program portability.

The coding of simple, non-embedded applications in Ada was evaluated in *An Assessment of Ada's Suitability in General Purpose Programming Applications* [Cavitt 85]. FORTRAN and Pascal programs for numerical computation, simulation, and file processing were translated into Ada. The number of lines of source code, transportability, maintainability, readability, and execution time were then compared. The study revealed that while further research is needed, Ada is a powerful programming language suitable for use in these non-embedded applications [Cavitt 85]. However, commercial developers have experienced real problems as a result of the inexperience of Ada programmers and the immaturity of Ada compilers and software engineering environments [AMC 86a].

While a multitude of simulation languages exist, only a few have gained some degree of acceptance [Law 82]:

a. GASP: an event-oriented simulation language consisting of more than 30 FORTRAN subroutines and functions for discrete simulations.

b. SLAM (Simulation Language for Alternative Modeling): an event-oriented or process-oriented simulation language for discrete, continuous, or combined discrete-continuous simulations.

c. SIMSCRIPT: an event-oriented or process-oriented simulation language for discrete, continuous, or combined discrete-continuous simulations.

d. GPSS (General Purpose Simulation System): a process-oriented simulation language that is well-suited for queuing systems.

10.3.4 Simulation Support

The following list of commercial packages and research projects for simulation support software was compiled after a preliminary survey. It is provided as a representative sampling of support tools. The list is partitioned into aids for model description, presentation, and integration.

10.3.4.1 Model Description Aids

Software to help the designer describe models and provide control information, producing output that can be simulated directly [Standridge 85].

a. Simulation nets [Torn 85]

   Simulation nets is a tool based on Petri nets with extensions for easy modeling of simulation designs. The simulation program is described in a non-procedural, graphical manner. The computerized tool is capable of interpreting the simulation net and performing the implied simulation.

b. Technology for the Automated Generation of Systems (TAGS) [Teledyne 84]

   TAGS is a system developed by Teledyne to convert graphical specification languages into Ada code. TAGS has four modules: Storage and Retrieval, Diagnostic and Analyzer, Configuration Management, and Simulation

Compiler. TAGS has been used in banking, airline reservations, power station control, and defense.

c. Simulation Display Generator (SDG) [Patterson 83]

SDG supports the creation, display, modification, storage, and retrieval of components of simulation models via interactive menu selections. A common high-level library of interactive display routines allows the simulation designer to easily and quickly generate graphics displays without having a knowledge of the graphics software.

10.3.4.2 Presentation Aids

Software to present results independent of simulation runs including report generators and graphics generators [Standridge 85].

a. DYNAMO Report Generator and Graphics Package [Catalog 85]

Extensions to the DYNAMO special-purpose simulation language to enable users to create reports in any format. Marketed by the Software Group.

b. GPSS Graphics Package [Hoover 83]

The GPSS Graphics Package supports the graphical display of simulations written in the special-purpose simulation language GPSS. Seven pre-defined GPSS macros comprise the device independent, easily used graphics package which can be incorporated into any GPSS simulation model, permitting analysts to view the behavior of the system as it unfolds.

c. GRAPHIC_SIMULATION [Dewar 84]

GRAPHIC_SIMULATION makes simulation programs easier to trace and debug by using animated graphics. The dynamic nature of the display, changing to reflect the state of the world as the simulation progresses, provides an intuitively clear representation of the interaction between different processes at different points in simulation time. Interactive control provides for arbitrarily complex detail, zooming, and breakpoints.

d. The Cinema System [Catalog 85]

A hardware/software system for creating and displaying computer simulation with animation. Based on the SIMAN simulation language. Marketed by Systems Modeling Corporation.

10.3.4.3. Integration Aids

Software to support the integration of the components of a simulation including model descriptions, executable modules, data for analysis, reports, and graphics [Standridge 85].

a. Jade [Lomow 85]

Jade is a distributed software prototyping and simulation tool. Its facilities include multi-lingual distributed programs, flexible monitoring of inter-process communication, multiple views into executing programs via a window system,

graphical animation of executing programs, and the prototyping of distributed software.

b. TESS [Catalog 85]

TESS is an integrated simulation support system with database and graphics capabilities. Marketed by Pritsker and Associates, Inc.

10.3.5 Model Validation

While verification and validation of SDI software is addressed by the Reliability task area (Appendix B, Section 11), model validation is a special requirement in simulated systems. Validation is determining whether a simulation model is an accurate representation of the real-world system under study [Law 82]. The model should be exercised using a broad variety of realistic initial conditions, control inputs, disturbance inputs, parameter variations, etc., to demonstrate that the responses are indeed good and reliable approximations to the real system [Huntsville 86].

A higher level of confidence in the results of a simulator is obtained after validation using statistical techniques. Techniques to validate models of small-scale systems are well-founded. However, these techniques cannot easily be applied or fall short in the validation of large-scale system models. Given the relative newness of the field of large-scale system simulation, there has been little or no research and development activity for techniques and tools to validate such models.

One current project, conducted by the Rome Air Development Center in conjunction with Syracuse University, is investigating a model validation and analysis technique for large-scale simulations. Their objective is to reduce the number of experiments required to validate and analyze large-scale models using a frequency domain approach to determine the sensitivities of critical parameters.

10.3.6 Data Analysis

Once a simulator is implemented and validated, a series of experiments may be conducted to assess the performance of an SDI component, architecture, strategy, etc. The tests should be designed to maximize the information gained from each run of the simulator. A suite of assessment exercises may then be planned "to (1) achieve acceptable levels of statistical confidence, and (2) cover a sufficiently comprehensive and diverse range of situations and parameters to assure that any weaknesses and vulnerabilities have been identified" [Turner 85]. Statistical methods used for testing the assumptions of a model and also the conclusions or inferences drawn from experiments run with the model are, in general, those that deal with [Shannon 75]:

a. Hypothesis testing

(1) Tests on estimates of population parameters assuming an underlying probability distribution (parametric tests such as F, t, and Z).

(2) Tests on estimates of population parameters not dependent on the assumption of an underlying population distribution (nonparametric tests, such as Mann-Whitney test of means).

(3) Tests to establish the probability distribution from which a sample comes (goodness of fit tests, such as chi square or Kolmogorov-Smirnov).

(4) Test on the degree of relationship among two or more variables (correlation analysis).

b. Estimation

(1) Calculation of point and interval estimates of population parameters.

(2) Determination of quantitative equations relating two or more variables (regression analysis).

Statistical methods used for hypothesis testing and estimating are, therefore, mainly [Shannon 75]:

a. Tests of means

b. Analysis of variance and covariance

c. Goodness of fit tests

d. Regression and correlation analysis

Such analytical techniques are objective and well-founded. They may be employed to help identify the critical factors that more heavily determine the behavior of the system. However, given the size and complexity of the strategic defense system, output from SDI simulators may be voluminous and not sufficiently reduced by statistical compilations. Human intelligence is still required to sift through the output data in order to isolate critical factors. Unfortunately, application of human intelligence is more an art than a science.

10.3.7 Hardware/Software Integration

The integration of simulation software and hardware components is not a trivial task. Programs developed by various contractors to model different components may require integration to analyze the performance of a subsystem. Hardware components may replace their simulated counterparts for in-line tests. Geographically distributed testbeds may interface for large-scale simulations. A module modeling a component at a given level of granularity may replace its counterpart from a model set of varying realism. Hardware and software will be integrated in a distributed environment, thereby compounding the problem.

Technical issues that affect the integration process include:

a. A global mechanism to control the integrated simulators.

b. Time management to synchronize events.

c. Coordination of disparate world views (process, event, object, etc.).

d. Reconciliation of simulator output of varying accuracy.

e. Transfer of knowledge between integrated simulators.

10.4 Recommendations

The foregoing assessment of simulation software technology serves as a basis for the following recommendations. Their purpose is to assist the SDIO in making decisions about future directions for technical efforts. The recommendations suggest certain actions for the SDIO to acquire the relevant foundations of software technology for SDI simulations.

Virtually all of the functionality in SDI simulators will be embodied in computer software. To achieve strategic defense objectives for simulation, the software must be developed and experiments designed, performed, and analyzed with modern and capable tools, techniques, and concepts. Application of such software technology is essential for the coherent, efficient, and timely development of simulation systems, and thereby of the SDI system as a whole [Turner 85].

Valuable experience has been gained from the initial development of SDI simulators. However, the efforts were conducted by the Armed Services and government contractors without a coordinated plan. As a result, the simulators suffer from redundancy, increased development costs, inflexibility, and lack of interoperability and portability. Furthermore, several large simulation facilites have been proposed for SDI without a coordinated plan. The National Test Bed, the Architecture Simulation Center, and the BM/C3 Experimental Version have some overlap in functionality and similar requirements for hardware and software technology.

Recommendation 1: SDIO should examine the operational roles of the NTB, ASC, and EV programs for possible redirection to plug gaps and eliminate redundancy.

The development and insertion of hardware and software technology into the NTB, ASC, and EV programs should be coordinated to optimize resource use.

Two important benefits would be reaped from committing to a simulation language standard for all SDI simulation applications:

(1) Increase program and programmer portability.

(2) Enhance interoperability and interchangeability of simulators.

The first one reduces software development costs and enhances the robustness of the overall system. The second benefit ensures that different simulators can talk to one another and modules modeling a given component at varying levels of granuality can be interchanged.

The committment to Ada as the standard simulation would have additional advantages. An Ada simulation support environment supplements the Ada language kernel with high-level simulation functions and analytical facilities. Ada then becomes as robust and powerful as any special-purpose simulation language. Ada compilers will also be more widely available than those of special purpose simulation language. Availability, which implies program portability, is an important consideration for a program as large and diverse as SDI. Finally, since Ada was designed to interface with other programming languages, existing simulations can be easily integrated into an Ada simulation portfolio.

Recommendation 2: SDIO should choose Ada as the standard simulation programming language. All new simulation software should be coded in Ada; exceptions require government approval.

Section 10.3.4 surveyed a variety of high-level simulation tools to support activities in all phases of simulation development and application.

Recommendation 3: SDIO should standardize an Ada Simulation Support Environment for SDI simulation and evaluation.

A set of software tools that are most useful for SDI simulation and evaluation should first be identified. A research-oriented Ada simulation support environment should then be selected, enhanced and integrated into the SDI Ada Programming Support Environment.

Section 10.3.5 stated the scarcity of automated tools to validate models of large-scale systems.

Recommendation 4: SDIO should fund basic research into techniques for validating models of large-scale systems.

The objective is to suggest or devise techniques to be developed into automated tools for model validation. This analysis should be based upon and extend the results documented in SDI Large-Scale System Technology Study [Huntsville 86]. The model validation and analysis project for large-scale simulations at RADC should be examined for useful results. This research project should commence as soon as possible.

The behavior of a simulator is governed by the values of the system variables. Thus it is imperative that the values of input parameters be chosen to enhance realism and the confidence in the statistical analysis. SDIO should insist that each software delivery be accompanied by a plan for testing to verify that it meets its specifications [Eastport 85].

Recommendation 5: SDIO should standardize a testing methodology for SDI software and hardware components.

Conformance to the test plan specification in form and content would promote consistency across SDI contractors, thereby increasing confidence and ensuring objectivity in performance assessments. The production of a specification for a standard test methodology should commence as soon as possible.

10.5 References

[Adelsberger 82] Adelsberger, Heimo H., "ASSE - Ada Simulation Support Environment," in *Conference 1982*, pp. 89-101.

[Adkins 77] Adkins, Gerald and Pooch, Udo W., "Computer Simulation: A Tutorial," *IEE Computer* 10, 4 (April 1977), pp. 12-17.

[AMC 86a] "Concerns Accompany Success for Ada," *Advanced Military Computing* 2, 7 (April 7, 1986).

[AT&T 85] *Survey of Simulations Applicable to BMD and SDI System Engineering/Architecture Effort*, AT&T (February 1985).

[Bruno 84] Bruno, Giorgio, "Using Ada for Discrete Event Simulation," *Software Practices and Experience* 14, 7 (July 1984), pp. 685-695.

[Catalog 85] "Catalog of Simulation Software," *Simulation* 45, 4 (October 1985), pp. 196-209.

[Cavitt 85] Cavitt, Larry D. and Panek, Anthony A., *An Assessment of Ada's Suitability in General Purpose Programming Applications*, Department of the Air Force, Air Force Institute of Technology, (September 1985).

[Dewar 84] Dewar, Alan and Brian Unger, "Graphical Tracing and Debugging of Simulations," *Simulation* 13, 2 (February 1984), pp. 68-73.

[Eastport 85] *A Report to the Director Strategic Defense Initiative Organization*, Eastport Study Group, Summer Study 1985, (December 1985).

[Ford 85] *Battle Management/Command, Control and Communications Concept Definition*, Model Data Sheets, Ford Aerospace and Communications Corporation (September 30, 1985).

[Frenkel 86] Frenkel, G., Smith, M., and Garcia, K. *Layered Defense Performance Evaluator*, Institute for Defense Analyses, Alexandria, VA (April 15, 1986).

[Friel 85] Friel, P. and Sheppard, S., "Implications of the Ada Environment for Simulation Studies," *Proceedings of the 1984 Winter Simulation Conference*, (November 28-30, 1984), pp. 477-489.

[Graybeal 80] Graybeal, Wayne J. and Pooch, Udo W.; Simulation: Principles and Methods, Little Brown and Company, Boston, 1980.

[GRC 85] *Battle Management/Command, Control and Communications Concept Definition*, Model Data Sheets, General Research Corporation (1985).

[Harris 86] Harris Corporation, *Strategic Defense Initiative Battle Management/Comand, Control, and Communication (BM/C3) Architecture Study*, Melbourne, FL (January 1986).

[Hoover 83] Hoover, S.V. and Perry, R.F., "Concurrent Graphical Display of GPSS Simulations," *Proceedings of the 1983 Summer Computer Simulation Conference*, pp. 12-135.

[Hughes 85] *Battle Management/Command, Control and Communications Concept Definition*, Compendium of Models and Simulations, Hughes Aircraft Company, Ground Systems Group, Fullerton, CA (1985).

[Huntsville 86] *SDI Large-Scale System Technology Study*, Final Report, Army BMD, Huntsville, AL, (March 28, 1986).

[Inkster 84] Inkster, James, Lomow, Greg, and Unger, Brian, "Combined Discrete and Continuous Simulation in Ada," *Simulation* 13, 2 (February 1984), pp. 16-21.

[Kaufman 84] Kaufman, J.A., Durrenberger, J. A., Parkin, E.S. and Schwartz, R.E. *Survey of Models for the Warrior Preparation Center*, Institute for Defense Analyses, Alexandria, VA (March 84).

[Law 82] Law, A.M. and Kelton, W.D., *Simulation Modeling and Analysis*, McGraw-Hill, New York, 1982.

[Lomow 82] Lomow, Greg and Unger, Brian, "The Process View of Simulation in Ada," *1982 Winter Simulation Conference.*

[Lomow 85] Lomow, Greg and Unger, Brian, "Distributed Software Prototyping and Simulation in Jade," *INFOR Journal* 23, 1, (February 1985), pp. 69-89.

[Martin 86] *SDI National Test Bed Concept Definition*, Martin-Marietta, NTB Phase I Competition, CDRL A004, Vol. 5, (July 1986).

[McDonnell 86] McDonnell Douglas Astronautics Company, *SDI BM/C3 Architecture Study*, Huntington Beach, CA (February 1986).

[NTB 86] *Statement of Work Strategic Defense Initiative (SDI) National Test Bed Concept Definition and Preliminary Design*, Attachment 1 to RFP SDIO84-86-0001, (January 1986).

[Patterson 83] Patterson, D.R. and Rhodes, J.J., "Design and Use of an Interactive Simulation Display Generator," *Proceedings of the 1983 Summer Computer Simulation Conference*, pp. 145-148.

[Shannon 75] Shannon, Robert E., "Simulation: A Survey with Research Suggestions," *AIIE Transactions* 7, 3 (September 1975), pp. 289-301.

[Shaw 86] Shaw, John J., "Development of a BM/C3 Architecture Effectiveness Model (AEM)," Presentation, Alphatech, Inc., Burlington, MA (October 1986).

[Standridge 85] Standridge, Charles R., "Software Aids for Simulation," *Simulation*.

[Teledyne 84] "TAGS Executes Code from Spec Language," Teledyne, *Datalink* (December 3, 1984), p. 4.

[Titan 84] *Survey of Existing C3I Battle Management Models and Methodologies with Applications to the Strategic Defense Initiative*, Titan Systems, Inc., Huntsville, AL (April 1984).

[Titan 86] *Battle Management/C3 Simulation Use Plan,* Titan Systems, Inc., Huntsville, AL (September 12, 1986).

[Torn 85] Torn, Aimo A., "Simulation Nets, a Simulation Modeling and Validation Tool," *Simulation* 45, 2 (August 1985), pp. 71-75.

[Turner 85] Turner, Robert D., *Assessment of C3 Capabilities for Strategic Defense Initiative*, Institute for Defense Analyses, Alexandria, VA, (July 1985).

[Turner 86] Turner, Robert D., *National Test Bed Post-Award Conference, Institute for Defense Analyses*, Alexandria, VA, Memorandum for the Record, (March 6, 1986).

[TRW 85] *Battle Management/Command, Control and Communications Concept Definition,* Model Data Sheets, TRW Defense Systems Group, Redondo Beach, CA (September 30, 1985).

SECTION B11

Software Dependability

Prepared by Samuel T. Redwine Jr., Jeffrey L. Grover, and R.J. Martin

---

Topics covered in Section B11:

11.0 Software Dependability

11.1 Introduction
11.1.1 Purpose and Scope
11.1.2 Background
11.1.3 Organization of Section

11.2 SDI Requirements
11.2.1 Reliability Requirements
11.2.2 Survivability Requirements

11.3 Current Status
11.3.1 State of Failure Definition Technology Research and Development
11.3.2 State of Fault Prevention Technology Research and Development
11.3.2.1 Fault Avoidance Technology
11.3.2.1.1 Computing-Oriented Specification Technology
11.3.2.1.2 Design Technology
11.3.2.1.3 Programming Technology
11.3.2.2 Fault Location and Removal
11.3.2.2.1 Static Analysis
11.3.2.2.2 Dynamic Analysis
11.3.2.2.3 Formal Verification
11.3.2.2.4 Quality Assurance and Verification and Validation
11.3.2.2.5 Problem Reporting and Tracking
11.3.2.2.6 Regression Testing
11.3.2.3 Quality Measurement/Modeling
11.3.2.4 Acquisition/Management Practices
11.3.2.4.1 Life Cycle Models
11.3.2.4.2 Organizational Approaches
11.3.2.4.3 Effect of Standards
11.3.2.4.4 Effect of Quality Resources
11.3.3 State of Fault Tolerance Technology Research and Development
11.3.3.1 Fault-Tolerant System Structures
11.3.3.1.1 Autonomous Decentralization
11.3.3.1.2 Recovery Blocks
11.3.3.1.3 N-Version Mechanism
11.3.3.1.4 Exception Handling

11.3.3.1.5 Deadline Mechanisms
11.3.3.1.6 Man in the Loop
11.3.3.1.7 Data Diversity
11.3.3.2 Error Detection
11.3.3.2.1 N-Version Programming
11.3.3.2.2 Majority Voting
11.3.3.2.3 Assertions
11.3.3.3 Error Isolation
11.3.3.3.1 Damage Confinement and Assessment
11.3.3.3.2 Modularity
11.3.3.3.3 Compartmentalization
11.3.3.3.4 Data Integrity
11.3.4 Inter-relationships

11.4 Recommendations
11.4.1 Introduction
11.4.2 Comprehensive Approach
11.4.3 General Recommendations
11.4.4 Recommendations for Failure Definition Technology
11.4.5 Recommendations for Fault Prevention Technology
11.4.6 Recommendations for Fault Tolerance Technology

11.5 References

List of Tables

Table	Title	Page
1	Software Dependability Quality Definitions Partial Sources.....	244
2	Software Failure-Related Goals and Special Knowledge Needed to Predict (Partial Source)	247
3	Types of Actions.....	266
4	Impact of Recommendations.....	273

11.0 SOFTWARE DEPENDABILITY

11.1 Introduction

Dependability is a general term encompassing the many qualities of a system that relate to the ability to justifiably rely on its service [Laprie 85]. These qualities can relate to failure -- availability, maintainability, reliability, security, and survivability. Or they can relate to the need to change the definition for a system's failure -- specification faults and system evolvability.

Software dependability covers the dependability of the software portion of the system. Software dependability requirements derive from system dependability requirements, but techniques for achieving software dependability are more immature than those for hardware. The critical, high dependability requirements for the SDI system combined with this immature status of software technology leads to the special attention provided by this section.

11.1.1 Purpose and Scope

This section addresses the dependability enhancements which can be provided through the software engineering process. The enhancements that can be provided by the distributed nature of the system are emphasized elsewhere in the task areas on Distributed Operating Systems, Network Communication and Distributed Data Management. Security is mentioned but is likewise addressed primarily in the three sections related to distributed systems.

This section reviews the software dependability needs, status of technology, and recommended R&D related to the Strategic Defense Initiative's BM/C3 software. The SDI-BM/C3 Software Technology R&D purpose is two-fold: to provide information to make deployment and other programmatic decisions, and to develop the capabilities for producing and supporting a BM/C3 system.

The production of a software system as complex as that envisioned for the SDI-BM/C3 is a significant undertaking. The requirements of high reliability, survivability, and other failure related qualities under adverse conditions further complicate the development process. The software development activities are viewed as pursuing three primary goals:

a. Ensuring the existance of valid specifications,

b. Avoiding the introduction of faults during the development process, and,

c. Structuring the operational system to tolerate any faults which could not be avoided.

This separation of activities is carried into the organization of this section as three major technology status sub-sections on failure definition, fault prevention, and fault tolerance.

11.1.2 Background

Initial studies examining the task of relieving the United States from the threat of nuclear ballistic missiles recognized the challenge of engineering the Battle Management/Command, Control, and Communication (BM/C3) software:

> . . . Specifying, generating, testing, and maintaining the software for a battle management system will be a task that far exceeds in complexity and difficulty any that has yet been accomplished in the production of civil or military software systems. [Fletcher 83]

Further investigations centering on the problem of the BM/C3 System advised the Strategic Defense Initiative Organization (SDIO) "not to expect any single software technology to be the solution - the "magic bullet" - for the SDI battle management system" [Eastport 85].

As these and later studies point out, the SDI software development activities will be directed toward providing a system capable of surviving a harsh environment. The system must survive a variety of natural and man made phenomena and continue operating reliably. This requirement includes the continued service of the system for a period in which a portion is inoperable. The successful operation of the SDI-BM/C3 will require that the development process employ techniques which produce a highly fault-tolerant system.

The traditional approach to achieving software reliability (the probability that the software will perform as specified when used under stated conditions for a specified period of time) has largely been based on fault-prevention. Fault-prevention technologies can be divided into two groupings, fault-avoidance, and fault location and removal. Fault-avoidance is the selection and use of technologies and resources intended to prevent the introduction of faults during system design and implementation. Fault location and removal efforts usually inspect the system before use to determine the correctness of the system implementation; correctness is the degree to which software satisfies its specifications.

Recognizing that some faults may be present in the final system implementation, one goal of software development is to include fault-tolerance mechanisms as a part of the operational system. Fault-tolerance requires that the system evaluate its own behavior. Capabilities must exist to detect the occurrence of an error, limit the extent of the error's effect, recover from the condition, and reconfigure to prevent recurrence and continue service. The mechanisms to provide these functions are executed along with the other parts of the operational system.

11.1.3 Organization of Section

Section 11.1 has provided an introduction to the purpose of this study, a brief overview of the concepts associated with dependable computation, and the definition of a few terms used throughout the document. Following this introduction, 11.2 summarizes the requirements of the SDI-BM/C3 software to support system dependability. Section 11.3 details the state of technology in the fields of software failure definition, fault prevention and fault tolerance. Technical requirements are noted along with promising research efforts in each area. Section 11.4 provides recommendations for SDIO involvement with R&D areas critical to SDI-BM/C3 software dependability requirements.

11.2 SDI Requirements

The SDI system is proposed for a critical role in future defenses. Therefore, a high level of dependability is a requirement of its implementation. The SDI-BM/C3 software will provide a significant role in the overall system operation. For those reasons, the dependability of the SDI-BM/C3 software is the object of this section.

The already difficult problems of battle management, and command and control will be amplified in the SDI environment by:

a. Critical short time duration to accomplish surveillance, acquisition, tracking, target destruction and kill assessment.

b. Need to keep human decision makers in the loop.

c. Advanced hardware and software technologies necessary to meet operational requirements.

d. Difficulty of determining the requirements for such a large and novel system operating in an unprecedented environment.

e. High reliability and survivability requirements.

f. Security requirements.

g. Large processing needs distributed over space and ground based networks.

New issues that arise in the design of distributed systems include concurrency control, asynchronous processing, scheduling, interprocessor/intertask communications, and deadlock detection.

Any BM/C3 system must be designed to tolerate a large number of error classes and possess a number of qualities. In addition to reliability, several other software failure-related characteristics are of interest. These include safety, availability, maintainability, correctness, and utility. Survivability is only rarely related to software failure, but it is strongly related to software fault tolerance.

Table 1 lists a set of key software dependability qualities and their definitions. Due to its mission the SDI BM/C3 subsystem will have requirements related to utility, safety, evolvability, security, and survivability. These, when the architecture and approach are established, will imply reliability, fault tolerance, availability, and maintainability requirements. Probable correctness -- the probability that no faults exist in a piece of software [Hamlet 86] -- is an interesting but immature concept whose applicability to SDI is still an open question. Fault avoidance and removal are developmental goals that will devolve from the approach chosen.

Altogether these qualities (including Table 1) provide a view of the variety of SDI BM/C3 dependability issues. Before reviewing the current states of the art and practice in software dependability, SDI BM/C3 software dependability requirements will be discussed at greater length.

11.2.1 Reliability Requirements

Reliability is defined under stated environmental and usage conditions as either (1) the probability that the system will perform as specified for a given period of time or (2) its mean time before failure to perform as specified. Given the mission of the SDI-BM/C3 system and the circumstances under which its correct operation is of most interest, the motivation for extreme reliability requirements being levied on the system is apparent. Since the majority of the SDI-BM/C3 system capabilities will be provided by software, the high system reliability requirements have been translated into high failure-free software performance goals.

Given the evolutionary approach to the SDI system and the potentially rapid change of threats, technology, and user desires, changing the software to conform to new specifications will also be required for reliability. Such changes must result in reliable conformance to the new specifications. Thus a software failure can occur either as a result of not changing the software to conform to a specification in time or as the result of faults introduced during a change.

The existence of a specification for the software is a prerequisite for software reliability to be meaningful. While some software technology is relevant to the definition of BM/C3 software specifications, many political, systems, platform, sensor, and weapons technologies will also be relevant and these are not addressed here. The analysis and design involved will be substantial while defining the following failure-related conditions:

a. Unacceptable world situations

b. Unacceptable SDI system effect (on reality or perceptions)

c. Failure of SDI system to perform to its specifications

d. Failure of SDI BM/C3 subsystem to perform to its specifications

e. Failure of SDI BM/C3 software to perform to its specifications

If this requirements capturing and specification is done improperly, even perfect software reliability will not deliver what is truly desired.

11.2.2 Survivability Requirements

There is an ever present need to implement the system in a manner that enhances its ability to continue operating regardless of the environmental stimuli. The SDI platforms are expected to operate with limited physical support for a period in excess of 10 years. During that period of time the exposure to damaging stimuli is considered to be great. This ambient level of damaging stimuli can come in the form of solar induced radiation or electromagnetic interference, cosmic dust particles, space debris, or any combination of these influenced by the earths magnetic and gravitational fields. Single event upsets that are caused by background cosmic radiation can be handled by hardware techniques. Recent advances in hardware technologies such as the use of the less radiation vulnerable material Gallium Arsenide (GaAs), and electro-magnetic shielding are directed toward these goals. There are some naturally occurring phenomena that will nevertheless induce hardware failures. The recovery mechanisms to support this class of failure are intended to operate at some low level of frequency.

The most frequently discussed SDI-BM failure mode involves the tactics which would be employed during an intercontinental ballistic interchange. Such a scenario is characterized by electronic countermeasures, disruptive electromagnetic impulses, and physical attacks. It is likely that an enemy attack would be preceded by a defense suppression effort. This surpression could include direct interception of the space-based SDI elements and nuclear detonations to disturb the communications environment. The goal of such activities is to damage or destroy susceptible SDI system components. In addition, enemy re-entry vehicles may be designed to self-detonate when intercepted, thereby increasing the possibility of an intense radiation environment for the SDI system. This will include electro-magnetic pulses (EMP), radiation, thermal shock, and blast effects.The anticipated battle dynamics indicate that this damage will occur simultaneously among elements

throughout the BM/C3 system. Unfortunately, this will probably occur at a peak in processing workload.

The survivability requirement dictates that, regardless of the threat, the system shall be capable of functioning continuously. The software support of such a requirement entails evasive actions to avert the disrupting events, detection of failures, continued service in the presence of failures, containments of damage, recovery and the reconfiguration of available components in the event that more permanent damage does occur.

Survivability is related to fault tolerance especially in its distributed aspects discussed in the task areas of Distributed Operating Systems, Distributed Database Management Systems, and Network Communications.

11.3 Current Status

Consistent with the Eastport recommendation, this section reviews the field of software engineering with emphasis on determining the combination of techniques whose use would maximize the dependability of the BM/C3 software.

11.3.1 State of Failure Definition Technology Research and Develoment

A prerequisite to the development of reliable software is a precise understanding of its intended performance. In other words, a fault, error, or failure cannot be addressed until it is defined. Software should not be attempted to satisfy unknown requirements. This statement holds for functional requirements and for quality requirements. If the software is to achieve certain levels of quality,[1] the associated requirements must be specified prior to the software's design and construction. The SDI BM/C3 software must be safe, secure, reliable, survivable, and evolvable. *One problem is how to adequately specify these quality* requirements; 11.3.1.1 discusses this further.

One way to determine and validate requirements is to build prototypes of key portions of the system for use by application specialists. Prototypes facilitate communication between system/software users and system/software developers [Henderson 86]. The software development process consists primarily of the software developer documenting and constructing his interpretation of the users' system concept, however, seldom do the backgrounds and terminologies of the system user and the software developer coincide. Therefore, tools and techniques that aid in the unambiguous transfer of information and requirements from user to developer are of prime importance. As well as allowing the discovery and revision of requirements, prototypes can demonstrate the feasibility of system requirements and provide a basis for experimentation during trade-off analyses to determine the appropriate allocation of functions to hardware vs. software [Aho 85]. The National Test Bed is intended as an arena for this type of work in SDI-BM/C3.

Another vehicle for user/developer communication is application-oriented specifications. (See 11.3.3.2.1.2). The redo of the A-7 aircraft operational flight software used an interesting tabular approach that was both application inspired and moderately general [Heninger 80].

Executable specifications and the creation of multiple specifications are other techniques for

[1]Quality attributes of interest may include: efficiency, integrity, reliability, survivability, usability, correctness, maintainability, verifiability, expandability, flexibility, interoperability, portability, and reusability [Bowen 85].

QUALITY	DEFINITION
Reliability	the probability of an item performing as specified under stated conditions for a stated period of time
Safety	measure of the time to castrophic failure from a reference initial instant
Probable correctness	the probability that no faults exist in the software
Fault tolerance	service complying with the specification in spite of faults
Utility	worth of services provided
Availability	the probability that a system is operating to specifications at any point in time, when used under stated conditions
Maintainability	the effort or time required to locate and correct a fault in the software
Evolvability	the effort or time required to modify software products and provide service conforming to change(s) in requirements including changes in technology
Security	protection from accidental or mailicious unauthorized access, use, modification, destruction, disclosure, or denial of service
Survivability	the extent to which the software will perform and support critical functions without failures within a specified time when a portion of the system is inoperable

Table 1. Some Software Dependability Quality Definitions
Partial Sources [McDonald 86, Laprie 85]

uncovering errors in the initial documentation of system requirements. Executable specifications allow testing and validation of the requirements prior to detailed design and implementation [Zave 86]. Multiple specifications allow cross-comparison for variations. When inconsistencies exist, the users are queried for the information necessary to settle the dispute. When variations represent alternate conceptualizations of the system, they can form the basis for multi-use software, future trade off studies, or redundant development efforts (see 11.3.3.2.1).

Requirements-oriented or applications-oriented specification activity is often thought of in the embedded systems community as a systems task. In a large, complex software intensive subsystem such as SDI BM/C3, however, it must form a mutually understood contract between the software's developer and the software users be they the total system's users, hardware engineers or other components of the system. For software development and fault tolerance to readily achieve high reliability, one representation of the specifications must be formal [Cohen 86] although other representations intended for communicating aspects to users with limited ability to understand the formalisms may be quite useful.

Implications of Software Failure-Related Goals

Reliability is not the only software failure-related goal, and it and the other goals can only be quantitatively addressed if certain types of information are available. Other goals and measures include safety, correctness, fault tolerance, availability, utility, and evolvability. To predict and engineer toward values for each of these requires varying special knowledge about requirements, usage distributions, and other items (see Table 2). This is called "special" knowledge because it is in addition to the functional, capacity and environmental conditions specifications whose existence is prerequisite for all of them.

The need for information characterizing distribution of use -- especially during conflict -- is critical to predicting software reliability and therefore to system creditability. The goals of safety, utility, availability, and evolvability also call for predicting use or future requirements before these goals can be calculated. These prerequisite predictions, and their representation and validation, are difficult issues that should be faced.

The concept of probable correctness [Hamlet 86] might seem to provide a way to avoid the necessity of some of these prerequisite predictions of distributions since it offers an approach based on minimaxing -- minimize the maximum probability of component incorrectness across all software components. This is not entirely the case, however, since estimating the probability of correctness is presently not fully understood (especially if one wants to combine other evidence with testing) and its relationship to reliability is in general arbitrary. Nevertheless, minimaxing is a suggestive concept for use while distributions are unknown.

Fault removal and fault tolerance require knowledge of the development and support system's error characteristics. Components such as types of tools, methods, and people (or even individuals) have error patterns that need to be discovered if fault removal and fault tolerance are to be predicted. And, of course, predicting fault tolerance is necessary for analytically predicting software system reliability where fault tolerance is involved.

Security more than the other goals is directly concerned with maleficent attempts to degrade or overcome it making models using probabalistic distributions much less appropriate. Models of security requirements for operating systems, networks, database management, and possibly applications all suitable for the SDI BM/C3 subsystem will be needed plus the means to verify the design and possibly the code against them. The National Computer

Security Center has efforts aimed at models or criteria in each of these areas except applications [Castro 86, Jordan 86].

Some of these qualities requirements imply the need to conform to DoD standards or regulations. Security and nuclear safety are good examples of this.

Note that the goal of (total certain) correctness is not on the list in Table 2. This reflects the present state of the art and practice for large systems as well as concerns such as those raised by DeMillo and colleagues [DeMillo 79].

However, when a piece of software indeed has no faults, the values for software reliability, availability, etc. become dramatically high and the impact on the persons and organizations using and supporting the software can likewise be dramatically favorable. Such a goal makes significant sense within the working culture of the software developers and supporters -- indeed today the best groups appear to be achieving it for systems in the tens of thousands of source lines in part because they really believe it can be done [Mills 86]. But the government must have solid evidence allowing quantification plus software contingency handling if it is incorrect despite all previous evidence.

Finally, it should be noted that the SDI software is unlikely to be considered adequate even with fault tolerance unless it is correct almost everywhere. This is true not only because the severe technical (including safety) requirements but also because of the planned openness of the decision making process and the importance of non-technical decision makers.

11.3.2 State of Fault Prevention Technology Research and Development

Computing systems such as envisioned for the SDI-BM/C3 are very complex, consequently there are many things that can go wrong. Fault-prevention techniques attempt to ensure that the operational system is free from faults.

11.3.2.1 Fault Avoidance Technology

The first step in the development of reliable software is to avoid the initial introduction of faults into the software. This section will review software engineering technology that addresses the specification, design, and construction of reliable software.

11.3.2.1.1 Computing-Oriented Specification Technology

Each of the techniques described previously concentrates on the system requirements as represented by the specifications. In one sense, this is a look backward in the system development process. Specification technology also encompasses a variety of computing-oriented specification techniques that emphasize the forward look into the software development process.

Formal computing-oriented specification notations can be either state-oriented or relational [Fairley 85]. State-oriented notations include decision tables, finite state machines, and Petri nets. Petri nets are of special interest for the SDI BM/C3 system since they are used to specify concurrency [Peterson 77, Bruno 86]. Relational notations include formal languages (both graphical and textual) which allow automatic analysis for completeness, consistency, and satisfaction of other specification rules. Examples of formal languages/processors include PSL/PSA [Teichrow 77], RSL/REVS [Alford 77], and Gist [Balzer 81]. Cohen, Harwood, and Jackson provide an overview of formal specification methods that was originally produced for ESPIRIT [Cohen 86].

GOAL	NEED TO KNOW
Reliability	Use distribution
Safety	Castastrophes/Hazards/Mishaps plus use distribution
Probable Correctness	Ignorance/indifference (minimax)
Fault Tolerance	Correlated errors
Fault Removal	Development and support error committing distributions and error detecting distributions
Utility	Values of outputs and failure rates plus use distribution
Availability	Reliability in MTTF and MTTR for repair or recovery, i.e., mean time to resume service
Maintainability	Distribution of fault types and time to repair each type
Evolvability	Distribution of future specification changes and ease of corresponding software changes
Security	Formal models of security requirements
Survivability	Types of component loss and distributions

Table 2. Software Failure-Related Goals and Special Knowledge Needed to Predict (Partial Source [Redwine 86a])

In contrast to many of the formal specification techniques described above, fuzzy specification techniques can be used to avoid the over-specification of requirements. This technique incorporates imprecision in situations where additional accuracy does not contribute to the ultimate solution of the problem [Rine 80]. Given the size of the BM/C3 system, this technique could be quite promising as a method for simplification and more effective utilization of the scarce resources allocated for requirements specification.

The use of an Ada-based process description language (PDL) for specifying the information elements of SDI BM/C3 architectures has been explored by an Institute for Defense Analyses workshop and by an IBM Federal Systems Division feasibility study. The results of both efforts were positive and an SDI Ada PDL (SA/PDL) will be used in Phase 2C of the Architectural Structures [IDA 87].

The anticipated life time of the BM/C3 system and the certainty of changes in its operational environment make software engineering technologies that support evolution very important. The evolutionary process pivots on the continuing evaluation and improvement of the specifications over time. Essential to the control of this process is a mechanism for retaining information about the specifications for past, present, and future instantiations of the software [NSIA 83].

Other aspects of the BM/C3 system that demand advanced specification technology include the requirements that it be real-time, distributed, fault-tolerant, and secure. Real-time and distributed system requirements necessitate the ability to represent timing, synchronization, and ordering information in the specifications [Liskov 79b]. One example of work in this area is CSP, Communicating Sequential Processes [Hoare 78]. Fault-tolerant and security requirements necessitate the ability of the specifications to represent information about the fault model being employed [Trivedi 82], as well as security model properties [DeMillo 83a, DeMillo 83b]. Finally, the anticipated use of the Ada language for the BM/C3 system compounds the need for a specification technique that supports concurrency and tasking.

11.3.2.1.2 Design Technology

Once the requirements for the BM/C3 system/software have been determined and appropriately specified, software engineering design technology exerts its influence on the ultimate reliability of the application. The fundamental concepts of software design include abstraction, information hiding, structure, modularity, concurrency, and verification [Fairley 85]. An assortment of design techniques and notations are currently available. They can be distinguished by the levels of emphasis placed on each of the fundamental concepts. Examples of design techniques include: stepwise refinement [Wirth 71], structured design [Yourdon 79], object-oriented design [Booch 83], and Jackson System Development [Jackson 82]. These techniques employ design notations such as data flow diagrams, structure charts, procedure templates, and design languages.

Selections of design technology for application to specific projects are often made subjectively due to the lack of objective methods for differentiating the effectiveness of the techniques. In fact, it appears that the major benefit derived from the use of modern design techniques stems from the enforcement of a systematic approach rather than inherent merits of the techniques. Surprisingly, evidence of the benefits of adhering to popular design concepts such as modularity and information hiding is also lacking [Card 86].

As with the specification technology outlined previously, the selection of technology for use during the design of the BM/C3 software must be guided by the requirements of the application. Again, any techniques selected must be able to represent timing, synchronization, and ordering information. Appropriateness of the design technique for

use with the Ada and Common LISP languages (the intended implementation languages) is also important. Finally, the evolution of the BM/C3 software requires that chosen design techniques facilitate the process of change.

11.3.2.1.3 Programming Technology

Current programming technology, in many cases, parallels current design technology. Modern programming language features support the fundamental concepts of design. Data abstraction and separate compilation support design abstraction. Scope and visibility rules support information hiding. Coding standards support structured design and modularity, and concurrency supports parallelism in designs. Further, many of the design techniques reviewed in the previous subsection are implemented by currently available programming techniques: structured programming [Linger 79], object-oriented programming [Booch 83], and Jackson Structured Programming [Jackson 82]. And, as with design, it is not apparent that any of the modern programming techniques is clearly superior to the others with respect to the ultimate reliability of the application software.

The application of artificial intelligence to the realm of software development has resulted in a variety of attempts to either automate a portion of the programming process or assist human beings with the tasks associated with programming. The term applied to this area of research is automatic programming. Programming techniques that reduce the dependence on humans are attractive since human error is thought to be the cause of low reliability in software systems. Automatic programming systems currently under investigation can be distinguished by their specification methods and approaches [Barr 82]. Specification methods include the use of formal specifications, specification by example, and natural language specification. Approaches to program generation include theorem proving, program transformation, knowledge engineering, and traditional problem solving. Examples of experimental systems include PSI [Green 76], SAFE [Balzer 76], the Programmer's Apprentice [Waters 82] and Protosystem I [Ruth 78].

11.3.2.2 Fault Location and Removal

As software products become available, the emphasis of the software development process shifts to locating and removing the faults that were introduced in spite of the use of fault avoidance techniques. Three primary types of activities undertaken are test and evaluation, quality assurance, and verification and validation. Testing techniques are usually classified as either static or dynamic analysis, where the former does not require the execution of the code and the latter does. See [STEP 83] for a recent survey of software testing technology.

11.3.2.2.1 Static Analysis

Static analysis techniques as applied to both design documents and code have benefitted from extensive examination and application in recent years. These techniques include data flow analysis, control flow analysis, consistency checking, complexity metrics, and code inspections and walkthroughs. Although tools implementing static analysis techniques are not abundant for application with the Ada and Common LISP languages, the underlying technology is mature and it is unlikely that the automation of desired capabilities for these languages would pose many problems.

Much of the error detection done during software development today is done by humans. Reviews of one kind or another are the common approach particularly for software products other than code, e.g., specifications and designs. While the quality of reviews appears to make a big difference, relatively little research exists in this area. Harlan Mills

has suggested that two reviewers, if their error finding could be treated as statistically independent, would provide much improved detection over a single reviewer [Mills 86].

Such independence is not probably strictly obtainable, but perhaps complementarity is. Several review techniques call for different persons to play different roles or concentrate on detecting different types of errors [e.g., Yourdon 77, Parnas 86]. Complementarity of techniques has been a concern of testing researchers although definitive results have not yet been obtained. For example, Selby has performed an interesting experiment and analyzed it for using pairs of error detection techniques including reviewing [Selby 86].

Symbolic execution lies in the gray area between static analysis and dynamic analysis. This technique assigns symbolic values to program variables and produces an execution tree which characterizes the possible execution paths. If each path can be shown to be symbolically correct (i.e., the symbolic expression associated with the execution of the path corresponds to the specifications), then the program is correct [Darringer 78]. The primary uses of symbolic execution are test data generation and proving program correctness.

11.3.2.2.2 Dynamic Analysis

The application of dynamic analysis techniques is essential for the BM/C3 software. Determination of the software's satisfaction of extreme reliability requirements can only be accomplished by systematically exercising the software. Principal dynamic analysis activities are the design of tests to establish certain characteristics of the software, the execution of the tests according to plans, and the analysis of results as allowed by the underlying theory of the chosen testing technique. Promising testing techniques for the SDI BM/C3 software include functional testing, program instrumentation on both microscopic and macroscopic (e.g., module) levels, input space partitioning, mutation analysis, and random testing.

Functional testing [Howden 80] is a design-based approach to testing where the program is decomposed into functional units and test data is generated to test each of the functional units independently. The application of functional and data abstraction techniques during the design process is important to the use of this technique. Therefore, the modularity of the Ada language and the functional nature of the Common LISP language should facilitate the use of this technique.

Microscopic program instrumentation is accomplished by inserting recording, or history collecting, processes into the software. These "probes" do not affect the functional behavior of the system, but provide key information about software performance and the thoroughness of testing for examination during test analysis and debugging activities [Fairley 78]. Although automated tools may not be available to support microscopic program instrumentation in conjunction with the Ada or Common LISP languages, the underlying technology is mature for non-performance oriented testing and should be easily extended to this arena.

The extreme reliability requirements of the BM/C3 system result in the requirement to design and build testable software. This implies more than the need for modular, simple code. It implies that rather than depend on the insertion of probes at test time, ports must be designed and built into the software to allow visibility, as needed, during software execution. Macroscopic program instrumentation provides visibility to the tester in a form that allows the assimilation of information about the execution of large pieces of software. This capability is essential for software the size of the BM/C3 application. One approach to the display of massive amounts of software execution information involves the use of animation [Brown 85; London 85].

Input space partitioning consists of dividing the possible program inputs into sets corresponding to distinct execution paths. One instance of this technique, partition analysis, requires the existence of a formal specification for the software and assumes that it is correct. Using this technique, the input domain is then divided into sets that are treated uniformly by both the specification and the software as implemented [Richardson 85]. Research to date in this area appears promising; however, much work remains to be done if this is to be appropriate for application to software as large as that expected for the BM/C3 system.

Program mutation is a technique that measures test data adequacy, where adequacy refers to the ability of the test data to ensure the absence of certain types of errors in the software [DeMillo 78]. This is accomplished by inserting likely errors into the software one at a time and determining whether or not the test data was able to distinguish the mutant program from the original program. Walsh has addressed simulating virtually all modern software testing strategies by selectively applying mutant operators during the testing process [Walsh 85]. Efforts are currently underway to investigate the feasibility of extending this technique for application to the Ada language and large software systems as anticipated in the BM/C3 system.

Random testing selects test data from all possible inputs randomly according to the expected distribution of inputs during actual system operation [Duran 81, Currit 86]. The effectiveness of the technique is directly related to the correctness of knowledge concerning operational inputs. It is important to note that the technique is not concerned with ensuring the absence of all faults in the software, rather only those that would be likely to cause failures during system operation. It is this characteristic of random testing that allows its direct use in estimating the reliability of the software.

The exact applicability of each dynamic analysis technique reviewed above to testing distributed, real-time, fault-tolerant software in its native environment has not been established. The National Test Bed is planned to provide the experimental environment; the form and source of its specific software testing and support tools needs to be defined.

Dynamic analysis has suffered in the past due to a reluctance to apply the requisite computing power. It is encouraging that recommendations are now being made to "exploit the advanced computer technologies to simplify the more difficult tasks such as software development and testing" [Eastport 85]. Testing is an area that will benefit greatly from a strategy combining modern testing techniques and advanced computer technology. At present there are no coordinated testing facilities available to reasonably test software as large as that embedded in the BM/C3 system.

11.3.2.2.3 Formal Verification

The goal of formal verification is to rigorously demonstrate, using mathematical logic, that software is consistent with its specifications. Techniques that have been applied to this problem include the use of input-output assertions and mathematical induction. Although the quest for a formal proof of the correctness of software may seem appealing, the technique is not universally endorsed.

The guarantee that software conforms to the design specification does not assure the specifications conformance to the original software requirements. It must not be forgotten that the original software requirement, by its very nature, is informal. Thus, although formal verification may enhance confidence in an item, we still do not have complete

assurance that the software will perform satisfactorily in its operational environment [DeMillo 79].

Recognizing the limitations of this analysis method, formal verification research is being actively conducted. Notably, efforts to evaluate trusted computer systems strive for formal verification of designs and source code with respect to its specification [DoD 83]. Examples of technology efforts include the GYPSY project at the University of Texas and the Enhanced HDM effort at SRI International. The Vienna Development Method (VDM) has significant support in Europe.

Formal verification using Ada is receiving increasing attention and has been the subject of several recent workshops organized by the Institute for Defense Analyses [Roby 85, Mayfield 86]. ANNA (ANNotated Ada) developed at Stanford University provides Ada specific specifications for functional behavior mainly at the more detailed levels [Luckham 84].

Formal verification is related to transformational programming and wide spectrum languages that use formal transformations or relationships to transform or check for consistency. The formal verification of software is greatly aided by co-creation of design or code and proof whether or not automated aids are used. Manual programming and reviewing using the concepts of formal verification, even "informally", appear to provide benefit [Mills 86].

11.3.2.2.4 Quality Assurance and Verification and Validation

Quality assurance and verification and validation are performed throughout the software development life cycle with the ultimate objective of building quality, and hence reliable, software. When properly applied, each activity has a distinct but complementary purpose [Fujii 78]. Quality assurance is concentrated on the definition of an appropriate approach to the creation of the software followed by continued monitoring of the development process for adherence. Verification and validation, on the other hand, is a continuous technical review of the software development products. Verification is the process of reviewing the products of each step of the life cycle for consistency with the products of the previous phase. Validation is the process of testing the software for satisfaction of the original requirements. Though each of these processes could benefit from additional automated support, the basic concepts are well established in practice.

11.3.2.2.5 Problem Reporting and Tracking

Approaches to problem reporting and automated systems for tracking the status of problem corrections are readily available. However, the SDI BM/C3 software does present a unique need in this area due to the requirement for fault-tolerance. If a fault is detected in the software and fault-tolerance mechanisms are exercised, it would be desirable for the fault to be reported even though a system failure was averted. In this manner, faults may be corrected as they are first encountered, thereby reducing future stress on the fault-tolerance mechanisms.

11.3.2.2.6 Regression Testing

Regression testing is testing performed, after a change is made to the software, to ensure that previously existing capabilities have not been compromised. Capabilities for effective and efficient regression testing are essential to the reliable operation of the BM/C3 software. Although somewhat primitive systems do exist to control test information and

support limited automated regression testing, the expected size of the BM/C3 software makes currently available alternatives inadequate for the task at hand.

Limitations in current technology are based upon the computational power available in the testing facilities. Current efforts directed toward removing this limitation are applying super computing power to development environments. The present levels of activity in sales of commercial multi-processor systems is seen as encouragement that advances will be made.

11.3.2.3 Quality Measurement/Modeling

Control of a process implies the ability to specify reliability and other requirements, predict success, track progress toward goals, and assess achievements. Today, software development is not a controlled process. However, this problem has been under investigation by NASA, the Naval Weapons Center, the Rome Air Development Center, the Software Technology for Adaptable, Reliable Systems (STARS) Program and others [Cavano 85, Conte 86].

Software measurement and evaluation has been criticized for its lack of ties to basic scientific principles [Browne 81]. Key omissions include the identification of invariant principles or relationships between basic measures, and hypothesis formulation and validation. Efforts to date have centered on tailoring standard definitions into measurable software-related entities that bear little resemblance to those used throughout the remainder of the scientific community [Bowen 85].

An important but elusive goal is to faithfully model software reliability. A number of software reliability models have been proposed over the years. Most of these models borrow heavily from the field of hardware reliability and, as a result, the basic assumptions of the underlying probability distributions are to be questioned and carefully assessed prior to their use [Goel 85]. Experience at Bell Laboratories, however, indicates that the best current software reliability models can be very useful [Musa 87 and Currit 86]. Musa believes the models are now well thought out and data exists but the models predictive power could be improved.

While some models can now combine old and new testing results [Currit 86], the ability to use a variety of evidence in making predictions is still needed

As opposed to efforts to model software reliability independent of the system, research is beginning to consider software reliability in the context of the system [Hecht 86] and some interesting empirical data has been collected [e.g., McCluskey 85]. International standards efforts have arrived at uniform terminology, and model builders such as Tirvedi at Duke have been making progress [Robinson 86].

11.3.2.4 Acquisition/Management Practices

All of the software technology reviewed previously is employed in the context of some organization's approach to doing business. On large software system developments, the approach is usually a combination of contractually imposed customer standards and the contractor's internal standards and procedures. This section will examine the influence that management practices exert on technology and vice versa.

11.3.2.4.1 Life Cycle Models

The relationships between the software development activities of prototyping, specifying, designing, programming, testing, evaluating, and maintaining the software are defined by a variety of life cycle models. It is clear that the BM/C3 software will evolve. However, there is no single approach to the evolution of software. The choice remains as to whether the evolution will occur as the waterfall model repeated sequentially, for example, or continuous multiple parallel developments. Some models cycle through the specification phase until satisfactory results are achieved, then design and build pieces of the system. Another variation is to completely specify and design the system, and then build and test in smaller pieces. Redwine has suggested that the model might follow the fault tolerant system paradigm as covered in 11.3.3 and 11.4.2 [Redwine 86b].

A number of views were evident at the Third International Workshop on the Software Process held in November 1986. As with the majority of the software technology discussed, there is little to no objective evidence available to aid in the choice of a development model whose use will result in reliable software for a given project. Today, the selection must be guided by engineering judgment and the expected timing of the availability of information about the system requirements.

11.3.2.4.2 Organizational Approaches

The involvement of multiple independent groups is important to the development of reliable software. This is necessary if misconceptions about the operational environment and misinterpretations of requirements are to be avoided. Therefore, commonly employed techniques include the use of independent test teams, independent quality assurance groups, and independent verification and validation organizations, in addition to involving the ultimate users and maintainers of the system in the development and testing process to the maximum extent possible.

In cases where N-version programming (see 11.3.2.2.1) is being used to build fault-tolerance into the system, multiple independent development groups are employed. The basic principle being followed is to minimize the harmful effects of individuals or single organizations on the ultimate reliability of the system.

In addition to the organizational independence, the evolution of the BM/C3 system may force the use of parallel development groups. Multiple concurrent developments are not uncommon in evolutionary systems.

The consequence of the required independence and parallelism in the development organizations, as well as the size of the development, is the severe requirement for effective management control and carefully planned management, communication, and coordination.

11.3.2.4.3 Effect of Standards

Examination of past programs has determined that the primary influence on the choice of software development technology has been the standards and requirements as defined by contractual agreements [Redwine 84]. This implies that conscious, careful acquisition and management decisions must be made with respect to the desired software technology during the tailoring process. If a diversity of approaches is desired [Eastport 85], conscious decisions must define which approaches will be applied to which portions of the development.

DoD-STD-2167 on Defense System Software Development is a Tri-Service standard that is applied to the majority of weapons systems being procured today. Issues have been raised as to its appropriate use in concert with the Ada language and evolutionary developments.

These issues are being addressed by the Joint Logistic Commanders, and a draft of revision A of 2167 is currently out for public comment and is expected to be in effect by the fall of 1987. In addition, DOD-STD-2168 on software quality is under development.

11.3.3 State of Fault Tolerance Technology Research and Development

Fault-tolerant computing accepts that an implementation will not be fault free. Activities in the field are devoted toward devising measures and methods to enable the system to continue operation in the presence of faults. The faults may remain in the design or develop in the implementation. Hardware fault-tolerant techniques are well established and form a vital part of any reliable computing system. The fault-tolerance strategies generally employed for hardware components rarely consider design faults. For software, however, design faults are a major concern.

Research activities in the field can be grouped according to a few major goals:

a. Identify components requiring fault-tolerance.

b. Develop low risk fault-tolerant system structures.

c. Assess fault-tolerant techniques.

d. Improve individual fault-tolerant techniques.

The identification of high risk components is typically based on two primary concerns; the criticality of the function performed with regard to the mission needs and the degree of difficulty associated with the fault free production of that component. Although references to these concerns do appear in this section, the selection of functions to protect with fault-tolerant mechanisms is based upon the analyses addressed in 11.3.2.

The field of fault tolerance is still actively developing with new principles, techniques, and empirical evidence. It already has, however, a substantial set of results and practices [Anderson 81, Anderson 83, Campell 85, and Kim 86]. These involve such elements as systematic error detection, damage confinement and assessment, error recovery, and fault treatment and continued service. Some typical characteristics of fault-tolerant approaches are

a. Redundancy - to avoid single fault failures.

b. Dissimilarity - to avoid common internal faults.

c. Partitioning - for damage confinement and recovery.

d. Dispersion - to avoid multiple faults from a common external cause.

e. Explicit engineering for reliability - cost vs. reliability tradeoffs allows one to avoid failures by paying added design, component and operations costs.

Error detection is typically done by either assertions or by the comparison of outputs -- the latter often by majority voting among multiple versions (called n-version programming). Special techniques are also sometimes used such as error correcting codes. Error detection and recovery are often easier if the potential types of faults can be anticipated. Failure mode analysis is often used during design in an attempt to anticipate errors/failures.

Recovery can be by rolling back to a prior error-free point and retrying with an alternative process (called recovery block), by using the majority result, or by using the designer's knowledge of the task to attempt to remove damage and continue on (called fail forward).

Not all use of fault tolerance is straightforward or positive. Concurrent, asyncronous systems can complicate fault tolerance mightily. Some attempts at fault tolerance with components without sufficient individual reliability can actually decrease overall reliability. An additional complication is the need to avoid the introduction of faults when adding software for fault-tolerance mechanisms.

Reliability is not the only possible goal of fault tolerance. Any property that is a function of failures, such as safety [Leveson 86] or survivability can be a goal.

There is an acute need for the SDI-BM/C3 system to continue service in the presence of failures. The cause of failures may be due to natural causes or due to man-made phenomena. The failure causing events may appear as system wear, an overload in processing, a nearby electro-magnetic event, or the physical disruption of elements. The effects on the system may range from a temporary saturation of sensor inputs to the permanent disruption of processing capability. In all cases, it is necessary to maintain some minimal level of functionality. Continued service in such a situation is contingent on a capability to work around the failure condition. The work around for each class of failure may be different. During this degraded mode of operation, available resources are allocated in the most beneficial manner identifiable. The system resources will be further divided, this time between mission functions, implementation of a failure specific work around, and the recovery from or repair of lost functions.

One concern centers on the ability to detect and continue operation when a critical processing element is no longer available. The large physical areas which must be observed, and over which operations must be coordinated presents it's own unique complexities.

The system level structure in support of a reliable/survivable system will be further discussed in 11.3.3.1. The ongoing research work concerned with assessment and improvement of techniques will be addressed throughout the remainder of 11.3.3.

11.3.3.1 Fault-Tolerant System Structures

Hardware components are mounted in a physical structure. Software components are similarly mounted in a logical system structure. This logical structure is referred to as the system level organization. This organization is devised with several considerations in mind: evolutionary development, secure processing, and concurrent operation. When considering a system such as that envisioned for the SDI-BM/C3, there is an additional need to structure the system to enhance system reliability and survivability [Neumann 86]. The system structure to support these requirements must undoubtedly provide a mechanism which is intended to provide continuously reliable operation in the presence of potential failure in both hardware and software components [Fletcher 83].

The suggestion of such a structure implies that the system resources are divided between mission functions, fault detection, and the handling of anomalous conditions. The support of such a requirement with respect to software failures should include provisions for the following generic activities:

a. Transition from normal to anomaly handling

(1) Detect error

(2) Activate anomaly handling

b. Confinement of damage

(1) Contain error propagation

(2) Propagate failure reporting

c. Continuation of service

(1) Expedient operation

(2) Workaround until recovery

(3) Workaround until repair

d. Recovery from damage

(1) Determine extent of damage

(2) Roll-back of cooperating activities

e. Prevention of recurrence

(1) Trace cause of failure

(2) Select proper repair action

(3) Reconfigure system components.

Consistent with current work in the field, the remainder of this section will consider the presence of a decentralized flow control entity as a part of a fault tolerance harness. The purpose of a fault tolerance harness is to provide a stable foundation from which error detection and recovery activities can be conducted with a high degree of dependability. This fault tolerance harness would provide the framework and services to execute n-versions in parallel, detect errors by assertions or majority voting, recover through recovery blocks or fail-forward, reconfigure, issue warnings, and record fault tolerance related events.

The control of such a system requires that atomic synchronization techniques be used due to the distributed nature of the application. Atomic actions provide the designer with a useful conceptual tool for factoring and organizing the interactions between system components [Moss 85]. Formal treatments of a systems atomic activity are based on precise notions of information flow and dependency between events [Cristan 85]. This treatment is particularly effective when atomic actions can be enforced by imposing constraints on component interactions, thereby imposing structure on the flow of information in the system. Techniques for enforcing the atomic structure range from physical separation to controls on facilities for component interaction.

The use of a fault-tolerance harness identifies a division between normal processing with error detection, and abnormal event handling. The transition into the abnormal mode of operation is based on the recommendation of a detection mechanism. The concept of this

harness is further described in 11.3.2.1.3. Referencing a harness of this type facilitates the discussion of the following activity areas.

The remainder of this section concentrates on the system structure and the operational transition mechanisms. The following section will address the detection mechanisms.

11.3.3.1.1 Autonomous Decentralization

The concept of autonomus decentralization has been developed to structure and control a distributed system with fault-tolerance as an objective [Ihara 79; Mori 81]. This concept considers the presence of a faulty subsystem to be a normal condition. Therefore, the fault tolerance of a system could be attained by dividing the entire system state in such a way that each subsystem is responsible for and able to control its own allocated part of the states even if other subsystems fail. That is, each subsystem has autonomy to control the part of the system for which it is responsible.

The envisioned SDI-BM/C3 system is considered to not have a global clock, any shared global memory, or any a priori designated central controller. The absence of common global memory implies that the "current state" will not be directly observable by any process.

The mechanism for achieving this capability for control requires that the processing elements be logically grouped with a number of other processing elements in a fault-tolerant mesh structure [Miyamoto 83]. Each processing element may be a member of more then one such grouping. These groupings serve two purposes, they form the set of redundent voters in a consensus arrangement, and they serve as watch-dog, or observer, tasks to ensure that the group controller remains active. In the event that an element failure is detected, the remainder of the group can continue to provide function with one less member; if the lost member is the control element, a new controller will have to be designated. A basic requirement for such a system is that it be composed of uniform subsystems, and each subsystem should control his own part based only on the local information so there should be neither supervisory parts nor common resources [Laksham 86].

11.3.3.1.2 Recovery Blocks

The recovery block scheme provides a framework for fault-tolerant software. The technique incorporates error detection, backward error recovery, and fault treatment. The implementation of a backward recovery mechanism, as a part of this framework, precludes the need for a damage assessment strategy. The backward error recovery mechanism assumes the elimination of all damage caused by a faulty module.

The use of recovery blocks for fault-tolerant applications was first introduced a little more than a decade ago. However, it does appear that the recovery block scheme provides a coherent framework for encapsulating redundancy in a software system [Leveson 83a; Avizienis 86].

11.3.3.1.3 N-Version Mechanism

The software N-Version mechanism for fault tolerance shares distinct similarities with the hardware technique of active redundancy. These techniques employ multiple units performing identical functions in parallel. In hardware systems redundancy arbitration is provided by parallel electrical paths. In software the system structure provides a software

arbitrator to: (1) determine if agreement has been reached, and (2) select the result to honor in the event of no agreement. The current SDI-BM/C3 system concepts rely heavily upon distributed processing with decentralized control. To support such a structure it may be necessary to provide a distributed agreement mechanism as well [Butler 85]. The details of the voting mechanism will be addressed in 11.3.3.2.2, (addresses only the control mechanism to manage the voting and possible transition to fault handling).

The previously mentioned fault-tolerance harness provides a structure within which specific functions may be executed. One class of function is an error detecting module, in this example multiple software versions performing identical functions. Another class of function is the arbitration module, in this case the voting mechanism. The harness initiates the error detecting modules, upon completion the results are made available to the arbitration module and its execution is initiated. At the completion of execution by the arbitration module an indication of results is provided to the harness. At this point the harness determines the next appropriate action: forward an agreed-upon result to the next stage, select a specific result for use in a subsequent stage, or reinitiate the error detecting modules.

11.3.3.1.4 Exception Handling

Exceptions and exception handling provide a mechanism for transitioning within a fault-tolerant framework. This framework provides a clear separation between normal and abnormal activities. The detection of an error results in the raising of an exception. The handler associated with that exception will be evoked and some error handling activity appropriate to the situation will be initiated.

Some research [Goodenough 76] has suggested the use of exceptions for general processing, others [Liskov 79a; Cristan 80] have proposed reserving the use of exceptions to the mechanisms of fault-tolerance. However, the Ada exception handling mechanism may not be satisfactory for this latter approach [Knight 84].

11.3.3.1.5 Deadline Mechanisms

The deadline mechanism is a variation of the recovery block scheme. This technique is generally utilized to monitor the interaction of cooperating concurrent processes. In this variation, the mechanism is more concerned with the timely completion of the operation rather than a post comparison of the results. It is undesirable to delay subsequent activities based on the expected availability of a previous calculation. In this situation, a voting mechanism could merely use the intermediate result in its assessment process. It is not sufficient to know that the result is inaccurate, some alternative action based on that result must be performed in a timely fashion.

The deadline mechanism provides an advance alert that the function will not be completed in a timely fashion. This advance notice permits the harness control to initiate an exception activity with ample time to complete prior to the primary deadline. The scheduling algorithms used for these purposes are not uniformly accepted in the community at this time. This lack of acceptance is especially true in the case of distributed systems such as that proposed for the SDI-BM/C3. Work is currently underway to better understand the benefits and drawbacks of each [Campbell 79].

11.3.3.1.6 Man in the Loop

There are many arguments for retaining a human operator in the processing loop. These arguments include control of command level decisions, verification of operational activities,

and last resort exception handling. The human operational traits desired are the superior pattern matching capabilities and the subliminal intuitive deduction capability. The error performance of humans can be helped by careful design [Bailey 82].

The debate frequently purports advances in AI which provide many of the capabilities currently provided by humans. At this time the AI community does not claim to have adequate capability to reliably provide the low level functions. There is a low expectation of supplanting the higher level cognitive processes completely in the near-term. The required basis for this approach centers on the concept of multi-agent interactions. The loosely coupled architectural concept [Eastport 85] can be likened to a hierarchy present in the military chain of command. In this scenario there is a need to coordinate the various system components, cooperative and adversative agents must be considered. Current work in the area of Order of Battle Management (OBM) and anti-satellite (ASAT) technologies appear to be the best avenues for advancement in the area of completely autonomous operation.

11.3.3.1.7 Data Diversity

Researchers have suggested that when an output error is detected one can reasonably try again with slightly different data. Since (1) in highly tested and almost correct software the failure regions are often small and (2) the error tolerance of input such as sensor input is usually significant, slight random changes in the input data may often move it out of the failure region without loss of real accuracy. Two schemes are suggested "retry block" modeled on recovery blocks and "n-copy programming" modeled on n-version programming [Ammann 86].

11.3.3.2 Error Detection

When in a normal operational mode, the first step toward active fault-tolerance is the detection of errors. The fault tolerance harness control mechanism, as described in 11.3.3.1, directs a set of software module interface checks during execution. The checks which are performed are constructed according to identifiable constraints and goals. The specific error detection requirements suggested by the system implementation must be considered by the system developers. This understanding of the unique operational situation will assist in the selection of an appropriate combination of the following fault-detection techniques.

11.3.3.2.1 N-version Programming

The N-version programming approach to fault-detection requires that several independent versions of a piece of software be produced for a specific application, usually from the same specification. These versions are then executed in parallel and the outputs are subjected to a majority voting process. The voting process determines the results which will be carried forward for use by the remaining components of the system.

Work in this area has primarily been sponsored by NASA. Results suggest that N programmers are capable of misinterpreting the programming specifications in very similar ways [Knight 86]. Specifically, the success of this technique depends greatly on the application specifications. Maximum utility is provided when identical requirements can be stated in highly dissimilar terms [Knight 85; Anderson 81]. Researchers have suggested the need for additional research to identify the common source of errors which result in software faults [Aviziens 85], and some interesting results showing common errors are often related to boundary and special values have been achieved [Vouk 86]. European

efforts have also been active in this area [Anderson 85, Anderson 85a, Anderson 85b, Bishop 86].

Some studies have suggested a possible advantage to multi-version programming which actually aids the development process. One such report cited reduced effort in detecting errors [Panzl 81] by performing a preliminary test between different versions developed. This technique is credited with an average 23% reduction in resources necessary for the testing of a second or successive version.

11.3.3.2.2 Majority Voting

One of the most important aspects of N-version programming is the majority voting check performed at the direction of the fault tolerance harness control mechanism. Some applications do not require an exact match of results to indicate agreement. In those instances, there is a need for an "inexact voting" mechanism, that is, a check that can indicate a consensus even though small discrepancies occur between sets of result compared. The need for such an "inexact voting" mechanism is typically great when dealing with digitized sensor data. The high degree of sensor fusion envisioned for BM/C3 applications indicates that this capability may be necessary.

Past efforts to resolve the "inexact voting" problem with range checks resulted in indications that difficulties are associated with defining the allowable range of agreement for specific application functions. Furthermore, once it has been determined that there is no consensus, it becomes a non-trivial matter to determine the value to use.

The detailed characteristics of the voting algorithm are highly application dependent. Guidelines for selection of voting algorithms for specific applications do not appear to be well defined [Barbara 85; Anderson 81]. The most promising work in this area research into the use of distributed Byzantine agreement [Perry 85; Cristan 85], and clock-driven synchronization mechanisms [Butler 85; Haseganas 85].

11.3.3.2.3 Assertions

An assertion is used to provide an explicit check for the plausibility of some result or system state. This check is performed in real-time and is utilized by a reliability harness to determine the need for corrective actions. The advantage of this method over a multi-version voting arrangement is based on two factors: potential for improved run-time performance, and immunity from the difficulties of developing truly independent versions. The success of this method relies upon the developers' ability to provide reasonable checkpoints with respect to the system's operation. The checkpoints must be selected to provide access at a point where a reasonability check can be implemented and from which a recovery can be affected. The criteria for such a point is an understanding of the form that the data or system state should be in at that point, the criteria for acceptability, and the identification of the entities which must be rolled back in order to overcome this failed check [Leveson 83b].

11.3.3.3 Error Isolation

11.3.3.3.1 Damage Confinement and Assessment

Once it has been determined that a error has occurred, it is imperative to confine the propagation of the anomalies at that level of operation. The propagation of subsequent errors will further degrade the performance, reliability, and survivability of the system. There have been studies into the forward tracking of failures in software systems which

provide a "coasting" mode of operation [Shin 84; Mili 84]. Coasting allows tasks downstream of the failure to continue operation based on partial information. These methods rely heavily upon the use of Petri nets to identify companion tasks which must be notified in the event of a detected failure [Leveson 83b]. Interchanged data blocks are generally provided with identifying information such as time stamps, calibration coefficients, and data slew rate information. The incorporation of this information to determine confidence in the use of data for a particular calculation provides a basis for individual decision points.

This is one area in which advances of fault-tree analysis would provide additional capabilities.

11.3.3.3.2 Modularity

A system such as the SDI-BM/C3 can be described as large, complex, and requiring parallel development, parallel execution, and a planned phased upgrading implementation. These constraints logically lead to the consideration of some structure which allows for the easy modification and interchange of functional entities. The concept of modular design is not new to the field. Recent developments in object-oriented system designs and the use of packages in Ada have served to more firmly set these concepts in the state of the practice. Layers of objects and trustworthiness also appear to be useful [Neumann 86].

The modular organization of the software is a demonstrated asset to fault-recovery [Scott 84]. This is particularly beneficial in limiting the extent of failure propagation [Krishna 84].

11.3.3.3.3 Compartmentalization

The potential for error propagation in a complex, concurrent, real-time system is great. Damage-containment provides an ability to detect and contain an error at a point where it will have minimal impact on the system performance. By compartmentalizing the system developers may enhance the system's damage-containment capabilities. The compartmentalization of a system to support damage containment is defined according to the data and control structure of the application. The transfer of data and control across the compartmental boundaries can be regulated based upon the operating conditions of the system. Reconfiguration techniques will be useful to reconfigure the system based upon the availability of various compartmentalized capabilities.

The modular structure to support compartmentalization should provide a fabric defined by the following criteria; localization of data usage, interface guarding of critical functional units at the lowest level at which an error may be detected, and a reasonably even coverage of all system dependencies with no threads totally devoid of checks. The fault analysis methods to support such a technique are being pursued as fault-tree analysis [Leveson 83a] and the use of Petri nets [Leveson 83b].

11.3.3.3.4 Data Integrity

The system structures proposed for use in the SDI BM/C3 require a great deal of dependence on data and its informational content. The internal data structures currently envisioned for use by the system will undoubtedly be large, dynamic, multi-threaded, and complex in nature. A single logical data item may in fact be utilized by multiple processing elements at any one time. The need to arbitrate the access, manipulation, and purging of these records will pose a significant organizational database problem [Fernandez 81].

Recent work in the field of robust data structures [Seth 85], and reliable database recovery [Griffith 85] show promise in alleviating the consequences of a simple fault.

The possibility of BM/C3 activities based upon the introduction of erroneous information is of concern as well. The validation of input data [Kopetz 80] and the use of data encryption techniques to ensure the integrity of the intra-system data transfers must be supported.

11.3.4 Inter-Relationships

The actual dealing with software faults, errors, and failures frequently crosses into more than one of the subareas of failure definition (requirements), fault prevention (development and support), and fault tolerance (execution). This can be clearly seen if one considers the various steps that might be taken as the result of detecting a run-time error:

a. Invoke run-time fault tolerance mechanism.

b. Establish temporary workaround (temporary change in effective requirements).

c. Repair fault by changing software products.

d. Revise tests and retest.

e. Change design so specification of the component no longer has behavior as error.

f. Change development and support process to avoid causing similar faults in future.

g. Change requirements so behavior not defined as failure.

While not all of these actions would ever occur together, they illustrate the interactions across the different subareas addressed in separate subsections above. Note that, in general, the lower an action is on this list the longer it takes.

The potential efforts to reduce or eliminate software failures involve actions spanning from requirements through technology R&D, software development and support to execution. A number of types of actions have been mentioned many of which require efforts that span subareas. These include those shown in Table 3. The list in Table 3 is very generic (avoiding for example saying each act applies to only faults, errors, or failures) and does not include much about when (e.g., possible, probable or certain error) and how (e.g., atomically, locally, safely) to perform the actions or their impact (e.g., criticality, safety, payoff). The list, however, does show many of the reliability-related types of actions that span subareas -- for example specifying interfaces is both a requirements activity for the system and a design activity for the components.

The recognition of the generic nature of these actions and that they can apply to requirements, development and support, execution, and other activities is the basis for the comprehensive approach that is central to the recommendations provided in the next section.

11.4 Recommendations

11.4.1 Introduction

This section makes recommendations for funding research and development projects related to software engineering advances required to build a dependable SDI BM/C3 system. Some of the recommendations are general and do not identify any particular candidates for funding. In the cases where candidates or funding paths are identified outside of DoD, however, further analysis of their related research may need to be undertaken.

Each recommendation relates to one or more of five types of issues or technology timespans.

a. Technology policy or standards

b. Technology for use in early experimental prototypes during 1988-90

c. Technology for use in experimental prototypes during 1990-93

d. Technology for longer term use

e. Acquisition practices or contents

This recommendation section will (1) cover aspects of a general approach to achieving reliability and related goals (to provide a context and additional rationale for the recommendations) and (2) provide recommendations -- both general ones and ones related to failure definition (requirements), fault prevention (development and support, and the means used for them), and fault tolerance (execution).

11.4.2 Comprehensive Approach

In order to make an integrated set of recommendations for the technology, experimental, and policy efforts related to software failure and provide them with a coherent rationale, the approaches that might lead to SDI BM/C3 success in this area need to be identified and outlined. Because many subareas exist and none provides the complete answer (as seen in 11.3) and the BM/C3 requirements are unprecedently severe, the approach used here will be a maximal one combining the many actions, techniques, and principles into a multi-tiered defense against software failure. Taking such a maximal approach also ensures coverage over the full breath of alternatives.

The main features of this maximal, synergistic, multi-tiered approach are:

a. Establish a multi-tiered defense with four tiers related as shown in Figure 1:

   (1) Requirements
   (2) Means
   (3) Development and Support
   (4) Execution

b. Use technology and principles traditionally used in only a subset of the tiers in all of them -- especially the application of fault tolerance to all four tiers.

c. Synergistically use aspects across tiers; for example, exploiting the existence of multiple versions intended for execution fault tolerance during development testing.

d. Provide cumulativeness, stability, and evolvability through an object-oriented open system structure with a growing set of invariant or upwards compatible interfaces.

e. Learn and improve all aspects through the planned series of experiments plus technology efforts.

Conceptually the first feature parallels that of the SDI itself with the idea that each tier will defend against failures leaked from prior tiers as well as errors within its tier. With this thought in mind, the order of the tiers becomes clear -- failures in establishing requirements specifications or in developing the means for software development and support can result in errors during software development and support that, if they remain, can become faults in the operational software. Fault tolerance during execution (and in some cases human concurrence) is the final, terminal defense against software failures.

The second feature also deserves some further discussion. One can, for example, apply the concepts and techniques used in designing fault-tolerant systems to the system that develops and supports software and thereby address its use of faulty components, particularly humans (Redwine 86b). The application of this idea to requirements, development and support, and to the R&D of technology provides a useful organizing framework for a variety of known facts and practices and for making recommendations.

A number of facts and practices regarding software development can be seen in a new light when viewed from the perspective of fault tolerance. These include "error" data, organizational practices, and human characteristics.

It has long been known that the cost to fix a mistake rises rapidly the longer it remains undiscovered (Boehm 81). This corresponds to larger recovery blocks with less damage confinement. Indeed, frequent (and often local) error detection activities such as reviews, analyses, and tests seem to be characteristic of the more successful software developers.

Many technolgists have tended to emphasize dividing the system being developed into parts with fixed interfaces and then assigning these parts to separate work groups. Clearly this is also a means of confining damage and easing error recovery. This, however, places a premium on dividing the system and defining the interfaces well. But this is exactly what the open system approach requires (the fourth feature).

Software development organizations must routinely tolerate certain types of organizational events. A good example is personnel turnover. Some practices -- such as always having at least two persons knowledgeable about each element and the use of teams with systematically shared knowledge -- exist in part to help provide this tolerance.

Software methodologists have argued that the human as a cognitive mechanism has basic limitations that would be to try to exceed. These include short-term memory capacity (7 plus or minus 2), amount of complexity, and mental distance between representations (e.g., preliminary and detailed designs). This view of people as processors with failure modes matches the fault-tolerant systems model.

All these examples of existing data, thought, and practices indicate that the fault-tolerant system view of the software development process has the power to relate a variety of disparate facts about software development in a consistent way. Fault-tolerance concepts

Understand larger goals.	Detect bad conditions.
Manage expectations.	Avert bad consequences.
Plan for change.	Confine bad consequences.
Specify interfaces.	Recover from bad consequences.
Reduce or eliminate uncertainty.	Continue service despite or after bad conditions or consequences.
Reduce or eliminate complexity.	Repair cause of bad condition or consequence.
Reduce or eliminate undesirable sensitivities.	Prevent similar bad conditions or consequences.
Use quality means.	Upgrade supportability.
Maintain consistency.	Retire/replace before obsolete.
Limit stress.	Observe, measure, and record.
Exploit alternatives.	Predict and evaluate.
Accelerate experience/learning.	Issue warnings.
Certify before use and recertify as appropriate.	Analyze, learn , and improve.
Authenticate rights before their exercise.	

Table 3. Types of Actions

are also a fruitful source of ideas concerning the software development process. These ideas relate to such aspects as the amount and organization of the work, improving error detection, and the analysis and treatment of humans [Redwine 86b].

The third feature in the approach is to exploit opportunities for synergy across the tiers. For example, redundancy is one of the key features of most fault tolerant systems. The suggestion that parts of the development process be done two or more times -- possibly in dissimilar, dispersed fashions -- comes immediately to mind. This is not a new thought when applied to programming (e.g., Panzl 81), and it fits nicely with any system being developed with software fault tolerance since this already would call for multiple different versions. Likewise, using old versions of the software as fallback versions in a recovery block scheme is a similar but not new thought.

The fourth feature -- using an evolving open systems approach -- is central to the solution of several SDI problems such as evolvability, but from a reliability and survivability viewpoint, it also needs to provide the applications interfaces to a stable, survivable environment with operating system, data management services, etc.; to supply an application fault tolerance harness service; and to decompose the system and the applications software with software dependability needs in mind.

The fifth feature on learning and improving through a series of experiments fits current SDI BM/C3 plans and is necessary to reduce uncertainties and to evaluate and improve the means used such as algorithms, tools, and people.

In addition the SDI contractors must have the openness, initiative, and discipline to use the best including obvious good principles such as planning ahead, cyclic problem solving, awareness of larger picture, striving for excellance and improvement, concern for utility, comprehensive and consistent measurement, dividing and structuring, concern for human and organizational factors, proper use of randomness, and systematic cutting of cost/time/variance, as well as exploiting automated support, fault tolerance and logical rigor.

This using of the best concepts, principles and practices is non-trivial to accomplish within the DoD acquisition process but essential to the chance for success of SDI software. This relates particularly to the first three tiers of the defense against software failure.

11.4.3 General Recommendations

The SDI BM/C3 subsystem software-failure requirements are unprecedentally severe in a system of this size and complexity, and the full scale engineering development decision is likely to require substantial evidence that these software-failure requirements can be met. Together this means that the series of prototyping experimental validation efforts should make every effort to achieve success, learn all that can be learned from the attempts, and go all out again. To learn the most, any legitimate alternative candidates should be tried and comprehensive but focused measurement and analysis should be employed. Thus, the maximal approach described in 11.4.2 and variations should be tried in the experiments.

Recommendation 1: Adopt a maximal, multi-tiered, fault-tolerant approach covering requirements, development and support (including the means used), and run-time to defend against software failures for the experimental prototypes including diversity and systematic measurement and improvement.

This will not only maximize the chances that the experiments will be convincingly successful but also help build the capability for software development and support that will be required for efforts beyond the experiments.

11.4.4 Recommendations for Failure Definition Technology

Recommendations related to failure definition or requirements cover three areas (1) requirements analysis and system design and their resulting software requirements specifications, (2) notations and related technology, and (3) validating the requirements specifications.

A dilemma exists with rigorous, accurate specifications being an essential prerequisite to the development of reliable software and the SDI requirements only becoming gradually firm and known over a long period of time. This is compounded by the changing nature of threats, technology, and desires over time. To deal with these, software requirements specifications must be explicit about the ranges of possibilities and evolution that may need to be accommodated by the software.

The requirements specifications need to establish the specifications of quality requirements in operational terms. This includes the specification of the prerequisite knowledge listed in Table 2 that is systems requirements knowledge.

Some qualities, such as fault tolerance and survivability, have characteristics and potential solutions that are strongly related to the systems and the hardware design.

All these points indicate the need for awareness of software concerns during requirements analysis and system design (including architecture studies) and the need for sound resulting specifications.

> Recommendation 2: Systems requirements and engineering activities should include a strong awareness of software concerns and result in -- initially multiple -- software requirements specifications that are formal, comparable, machine analyzible and manipulatable, explicitly address the range of possible systems implementations and evolutions, operationally define software-failure-related quality requirements and include prerequisite knowledge such as usage distributions.

Technology efforts in this subarea should concentrate on notations and their related characteristics and capabilities such as fit to the application, usability, analyzibility, tools, fit with other (e.g., design, simulation, and programming) notations, and ease of validating. Since suitability for large complex systems such as the SDI is a key concern, these should be given large scale trials as part of the experimental validation series.

> Recommendation 3: Invest in multiple efforts toward the development or improvement of specification technologies potentially well suited for use in accurately representing and communicating SDI BM/C3 system/software requirements and try them in the experimental version efforts.

Examples of promising specification technologies include man-machine interface prototyping, executable specifications, formal application-oriented specifications, formal computing-oriented specifications, wide-spectrum languages, and fuzzy specifications. Different types or styles of notations may be needed for representing different parts of the

specifications -- for example, communications vs. target assignment, or functionality vs. qualities.

Validation of the requirements specifications is a key concern. Initially, if multiple, redundant versions are prepared, they will need to be compared for the purpose of discovering differences that are potential faults. Individual versions will need to be analyzed for internal consistency. In addition validation involves the need to have views of the specifications that communicate to key validating humans (e.g., graphics, animation, English, interactive prototypes) and interface with the National Test Bed and with threat, reality, and systems models. Concern should be given to the types of errors different validation techniques find and the combined effectiveness of sets of techniques. In practice, some entity will need to be the authoritative system reference with the final authority to control and interpret the specifications.

While some small efforts on particular parts of the problem can contribute to advancement in limited areas, the top priority should go to larger scale efforts that are tied to the BM/C3 experiments.

11.4.5 Recommendations for Fault Prevention Technology

The development and support process needs to be explicitly organized including fault tolerance with respect to the requirements specifications and the means used. The means used need to be up to the severe challange. The resulting product needs to meet the many requirements. And the gaining of convincing evidence needs to be facilitated. In addition, particularly during the early years lessons need to be learned and improvements made. Recommendation 1 covered the need to organize for a maximal defense against errors.

Concern for the means used has three facets. The first is that the means development process be organized appropriately including concern for risk management and quality. This is covered in the task assessments on Technology Transition, and People and Organizations. The second concern is that the means be properly specified and certified -- both technology and people. The third is that acquisition practices and contents actually result in the desired contractor behavior.

> Recommendation 4: Ensure the proper organization of the processes of technology development and use, and personnel selection and development including characterization of their error patterns and their certification for use.

Verification technology for error detection during development and support is an important area for SDI. The potential for error avoidance through automation should be disclosed by Recommendation 4's characterizations, but from a software dependability viewpoint, its extreme form, automatic programming, does not appear to be ready for profitable SDI investment.

> Recommendation 5: Identify the error detection power of verification techniques; support the enhancement for use on SDI BM/C3 of technology for software testing, simulating faults and recording error propagation, and formal verification with Ada; and monitor automatic programming research.

Error detection during software development and support is a key concern under the maximal, fault tolerant approach. Multiple, comparble versions of all types of technical products and their comparison is one technique. Additionally, several important techniques

need technology efforts, and the many techniques available need their error detection power characterized and compared so that complementary sets of error detection techniques can be identified and used. Therefore SDIO should identify the individual error detection power during development and support of candidate techniques and sets of complimentary techniques.

Testing is one of the error detection techniques of key importance to SDI. Existing testing techniques need work to make them suitable for SDI BM/C3 size and programming language. SDIO should upgrade existing software testing technology so that it can be effectively applied to software for the SDI BM/C3 system, and invest in proposals of merit directed toward the development of newer testing technology that emphasizes systematically exercising the software in concurrent systems.

Selected software testing techniques are mature enough that extensive investment in research activities is not needed to gain the benefits of their utilization for the SDI BM/C3 system. The needs of these techniques lie in the area of technology transition: the principal motivation for modification to existing capabilities is to allow application to the Ada and/or Common LISP languages. In some cases, further modifications may be necessary to scale-up the capabilities for application to large software systems. Examples of testing techniques to be considered for transition include functional testing, random testing, static analysis, and regression testing.

Some of the promising newer techniques stress the observation of software performance under varied conditions and for varied goals. These include both microscopic and macroscopic program instrumentation, and mutation analysis. Other techniques emphasize the determination of a minimal, adequate set of test data. Promising software test data generation techniques include symbolic execution, generation from prolog-like specifications, and input space partitioning. Testing (and debugging) of concurrent (and real-time) software is less well understood but essential for SDI and therefore requires emphasis. Cost benefit studies could also be conducted on these techniques to support their selection for application to specific projects.

Formal verification can provide important evidence related to software reliability for small sections of the BM/C3 software that are particularly important. Examples might be security enforcement modules, fault-tolerance mechanisms, and weapons release software. SDIO should accelerate the availability of formal verification capability for Ada comparable to that for some languages in the research community today but pursue formal verification for Common LISP with less urgency.

In highly reliable systems one needs to accelerate the experiencing of errors and the testing of fault tolerant capabilities. Although fault-tolerance mechanisms may avert system failures, it is still desirable for information about errors encountered during operation to be recorded and reported. Information relayed should be adequate to show the strength of the fault tolerance, to allow the location of the fault, and to guide the generation of test data to verify its correction. Therefore, SDI should develop a capability to simulate faults and report information concerning errors encountered during system execution that were averted by the operation of fault-tolerance mechanisms as well as failures.

Finally, the application of artificial intelligence to software development to ease the burden and reliance on human beings is attractive and has benefited from much attention and investment in recent years. It is recommended that SDIO monitor the results of active research projects in this area.

The evidence must be collected and analyzed to make a convincing conclusion.

Recommendation 6: Invest in quality measurement/modeling research. Use measurement and modeling on all BM/C3 software efforts, and learn and change.

Quantitative models and measurement techniques are needed to provide precise methodologies for requirements specification, determining system development status and predicting future performance of the system, including the reliability of the software. Research in this area has almost exclusively centered on simple adaptations of existing statistical models, statistical data analysis, small-scale experimentation, and the compiling of often non-comparable evidence. Research is needed in approaches to measurement and modeling that have deeper scientific roots as well as more practical efforts. The developing of models (e.g., relating software characteristics to reliability) and validating model predictions are included. Of particular interest is the development of methods to combine evidence from a variety of sources into conclusions related to software-failure qualities.

11.4.6 Recommendations for Fault Tolerance Technology

This section recommends research and development projects required to advance the fault tolerance technology described in 11.3.3 to the state needed for application to the development of the SDI-BM/C3 system.

The software research required to support development of the SDI- BM/C3 will demand advances to enhance the system's dependability. Research in these areas has been conducted for similar, but less rigorous, applications for over two decades. Research efforts to date in the area of software fault tolerance remain promising; the availability of sufficient numbers of qualified researchers remains the long-term problem. A research program designed to facilitate widening the base of qualified individuals and applied concepts in the field should be undertaken.

Recommendation 7: Support R&D programs that cover fault tolerance at both the software and systems levels; address Ada definitional shortcomings in failure semantics, autonomy, decentralization, atomic synchronization of distributed processes, application-specific fail-forward and adaptive control mechanisms, and software safety; and result in stable level(s) of machine abstraction and a fault-tolerance harness that applications software can use straightforwardly.

A sound basic understanding in the theory of fault tolerance does exist. Recent efforts to apply it at the system level have begun to generalize the concepts to the point that they should be proven in practice. Large scale testing of the most promising concepts should be undertaken to provide a better reference for making such design decisions.

In theory, the fault-tolerant needs and assets of hardware and software components can complement one another. Little effort has been devoted to identification of special purpose hardware to support software reliability. Exploration of hardware-assisted software reliability should be considered. This is a relatively immature technology; therefore, a series of parallel research efforts should be initiated to explore as diverse a set of solutions as is possible. Continued efforts to perfect the software assistance to hardware reliability and survivability should be sustained. Software fault-tolerant techniques should be investigated and evaluated. The relative resource use, practically, and performance of N-version programming versus recovery blocks and majority voting versus assertions should be determined. As covered in this and the following paragraphs a set of suitable fault-

tolerance abstractions and interfaces need to be derived. Prototypes may also be used to investigate these issues.

A thorough understanding of system dependencies are necessary to identify those sections of the software which are critical to the success of mission goals. Past efforts which are most promising in this area have been conducted in support of nuclear power plant safety. The techniques employed for those applications will provide a basis for assessing critical BM/C3 components. SDIO should support software safety research and use the techniques in experiments and monitor the failure mode analysis efforts currently funded through DoE-Sandia in applying AI techniques to safety monitoring. The SDIO should augment the ongoing efforts in this area of DOE, NASA, Newcastle-upon-Tyne, and University of California, Irvine.

This effort should be directed to provide a system wide mechanism to identify the potential problem areas which will occur in the development and operation of the system. At present, the mechanisms for implementing such techniques are not automated. Research efforts, directed toward transitioning the activity from that of off-line design to an operational capability, should be solicited. The dynamic analysis of the system configuration and stress is necessary to withstand successive waves of damaging stimuli.

The concurrent activities of a real-time system are further complicated by the time delays present in a distributed system. SDIO should provide additional funding to ongoing efforts in autonomy, decentralization, atomic actions, and synchronization of distributed processes. Funding agencies for these efforts should be NASA, DARPA, and UK-MoD (with work performed at Newcastle-upon-Tyne).

Current activities in this area show promise, the extent of research appears to be limited by the available funding rather then supply of researchers. The SDIO should stimulate additional activity in this area by making funds available to agencies currently conduction such efforts.

The ability to detect a failure is required in order to recover from that failure. The present mechanism of rolling back to a point prior to the failure introduces the possibility of repeating the events. The desired approach would be to fail forward (perhaps in a degraded mode) in order to continue service while preventing the recurrence of the failure. SDIO should continue to monitor the NASA funded efforts researching fail forward recovery mechanisms. Initiate an effort to study application-specific fail forward techniques and adaptive control mechanisms as a failure recovery techniques. One or more low activity level studies should be undertaken in the area of adaptive controls. Efforts should strive toward the identification of system indicators of: error states, low resistance paths to non-error states, and the sequence of actions necessary to transition states.

There is a pressing need to provide a stable level of machine abstraction on which the application will be hosted. This abstraction should include the hardware, interconnecting software facilities and possibly the software driven reconfigurability capacity. Therefore, SDIO efforts should establish stable level(s) of machine abstraction and a fault-tolerance harness that applications software can use straightforwardly. This will require the pulling together of results from and coordination with the R&D experimental versions, and architecture efforts.

These recommendations will impact SDI policy and standards, early prototypes, longer-term efforts, and acquisitions. Table 4 shows the main expected impacts of each recommendation. All recommendations are expected to have multiple impacts.

Recommendations Regarding		Policy	1988-1990 Prototyping	1990-1993 Prototyping	Longer term	Acquisition
1:	Maximal, multi-tiered, fault tolerant approach	X	X	X		X
2:	Systems engineering software awareness	X	X	X	X	X
3:	Multiple specifications technology efforts		X	X	X	
4:	Technology and personnel preparation error patterns, and certification	X	X	X	X	X
5a:	Error detection during development		X	X	X	X
5b:	Testing technology		X	X	X	
5c:	Formal verification			X	X	
5d:	Simulate faults and report errors		X	X	X	
5e:	Automatic programming			X	X	
6:	Measurement and modeling	X	X	X	X	X
7a:	Systems and software fault tolerance	X	X	X	X	X
7b:	Ada definition	X		X		X
7c:	Autonomy decentralization		X	X	X	
7d:	Automatic synchronization		X	X	X	
7e:	Fail forward			X	X	
7f:	Software safety	X	X	X		
7g:	Stable virtual machines and fault tolerance harness	X	X	X		X

Table 4. Impact of Recommendations

11.5 References

[Aho 85] Aho, Alfred V., Kernighan, Brian W., and Weinberger Peter J., "Awk - A Pattern Scanning and Processing Language Programmer's Manual," AT&T Bell Laboratories, *Computing Science Technical Report No. 118*, Murray Hill, NJ, (June 1985).

[Alford 77] Alford, M. "A Requirements Engineering Methodology for Real-Time Processing Requirements," *IEEE Transactions on Software Engineering*, SE-3, 1, (1977).

[Amman 86] Amman, Paul E., and Knight, John C., *Data Diversity: An Approach to Software Fault Tolerance*, (Computer Science Report No. TR-86-29), University of Virginia, 2 December 1986.

[Anderson 81] Anderson, T., Lee, P.A., Fault Tolerance, Principles and Practice, Prentice/Hall International,Inc. London, (1981).

[Anderson 83] Anderson, Thomas and Knight, John C., "A Framework for Software Tolerance in Real-Time Systems," *IEEE Trans - ctions on Software Engineering*, SE-9, 3, (May 1983).

[Anderson 85a] Anderson, T. (ed.), Resilient Computing Systems, Vol. I John Wiley & Sons, 1985.

[Anderson 85b] Anderson, T. (ed.), Software Requirements, Specification, and Testing, Blackwell Scientific Publications, 1985.

[Anderson 85] Anderson, T., Barrett, P.A., Halliwell, D.N., and Moulding, M.R., "Software Fault Tolerance: An Evaluation," *IEEE Transactions on Software Engineering*, SE-11, 12, (December 1985), pp. 1502-10.

[Avizienis 76] Avizienis, A., "Fault tolerant systems," *IEEE Trans. Computers*, C-25, December 1976, pp. 1304-1312.

[Bailey 82] Bailey, Robert W., *Human Performance Engineering: A Guide for Systems Designers*, Prentice-Hall, 1982.

[Balzer 76] Balzer, R. M., Goldman, N., and Wile, D., "On the Transformational Implementation to Programming," *Second International Conference On Software Engineering*, (1976), pp. 337-344.

[Balzer 81] Balzer, R. *Gist Final Report* , Information Sciences Institute, University of Southern California, (February 1981).

[Barbara 85] Barbara, Daniel, and Garcia-Molina, Hector, "Evaluating Vote Assignments with a Probabilistic Metric," *Digest of Papers FTCS-15: Fifteenth Annual International*

Symposium on Fault-Tolerant Computing, Ann Arbor, Michigan, (June 19-21,1985), pp.72-77.

[Barr 82] Barr, Avron, and Feigenbaum, Edward A., Eds., *The Handbook of Artificial Intelligence*, 2, William Kaufman, Inc., Los Altos, CA, (1982).

[Bishop 86] Bishop, P.G., Dahll, G., and Lahti, J., "PODS - A Project on Diverse Software," *IEEE Transactions on software Engineering*, SE-12, 9, (September 1986), pp. 929-940.

[Boehm 81] Boehm, B., *Software Engineering Economics* , Prentice-Hall, (1981).

[Booch 83] Booch, Grady. Software Engineering with Ada, Benjamin Cummings, Reading, MA, (1983).

[Bowen 85] Bowen, Thomas P., Wigle, Gary B., and Tsai, Jay T., "Specification of Software Quality Attributes," *RADC-TR-85-37*. (February 1985).

[Brown 85] Brown, Mark H., and Sedgewick, Robert., "Techniques for Algorithm Animation," *IEEE Software*, 2, (1985), pp. 28-39.

[Browne 81] Browne, J. C., and Shaw, Mary, "Toward a Scientific Basis for Software Evaluation," *In Software Metrics* , Perlis, Alan J., Sayward, Frederick G., and Shaw, Mary, Editors, (1981), pp. 19-41.

[Bruno 86] Bruno, Giorgio, and Marchetto, Giuseppe, "Process-Translatable Petri Nets for the Rapid Prototyping of Process Control Systems," *IEEE Transactions on Software Engineering*, SE-12, 2, (1986), pp. 346-357.

[Butler 85] Butler, R.W, Palumbo, D.L., Johnson, S.C., "Application of a Clock Synchronization Validation Methodology of the SIFT," *Digest of Papers FTCS-15: Fifteenth Annual International Symposium on Fault-Tolerant Computing*, Ann Arbor, Michigan, (June 19-21, 1985), pp.194-199.

[Campbell 79] Campbell, R.H., Horton, K.H., and Belford, G.G., "Simulations of a Fault-Tolerant Deadline Mechanism," *Digest of Papers FTCS-9: Ninth Annual International Symposium on Fault-Tolerant Computing*, Madison, (June 1979), pp. 95-101.

[Campell 85] Campell, R.H., and Randell, B., "Error Recovery in Asynchronous Systems," IEEE Transactions on Software Engineering, 1, 8, (August 1986), pp. 811-826.

[Card 86] Card, David N., Church, Victor E., and Agresti, William W., "An Empirical Study of Software Design Practices,"

IEEE Transactions on Software Engineering, SE-12, 2, (1986), pp. 264-271.

[Castro 86] Castro, Lawrence, "An Overview of the DoD Computer Security RDT&E Program," Proceedings National Computer Security Conference, 15 - 18 September 1986, pp. 213-15.

[Cavano 85] Cavano, Joseph P. "Toward High Confidence Software," *IEEE Transactions on Software Engineering*, SE-11, 12, (1985), pp. 1449-1455.

[Chen 78] Chen, L., and Avizienis, A., "N-Version Programming: A Fault-Tolerance Approach to Reliability of Software Operation," *Digest of Papers FTCS-8: Eighth Annual International Conference on Fault Tolerant Computing*, Toulouse, France, (June 1978), pp.3-9.

[Cohen 86] Cohen, B., Harwood, W.T., and Jackson, M.I., "The Specification of Complex Systems, *Addison-Wesley*, 1986.

[Conte 86] Conte, S.D., Dunsmore, H.E., and Shen, V.Y., Software Engineering Metrics and Models, Benjamin Cummints, 1986.

[Cristan 80] Cristan, F., "Exception Handling and Software-Fault Tolerance," *Digest of Papers FTCS-10: 10th International Symposium on Fault-Tolerant Computing Systems*, Kyoto, (October 1980), pp.97-103.

[Cristan 85] Cristan, F., Aghili, H., Strong R., Dolvend, D., "Atomic Broadcast From Simple Message Diffusion to Byzantine Agreement," *Digest of Papers FTCS-15: Fifteenth Annual International Symposium on Fault-Tolerant Computing*, Ann Arbor, Michigan, (June 19- 21, 1985), pp.200-206.

[Currit 86] Currit, A., Dyer, M., and Mills, H.D, "Certifying the Reliability of Software," *IEEE Transactions on Software Engineering*, SE-12, 1, (January 1986), pp.3-11

[Darringer 78] Darringer, J. A., and King, J. C., "Application of Symbolic Execution to Program Testing," *Computer*, 11, 4, (1978), pp. 51-60.

[DeMillo 78] DeMillo, Richard A., Lipton, Richard J., and Sayward, Frederick G., "Hints on Test Data Selection: Help for the Practicing Programmer," *Computer*, 11, 4, (1978), pp. 34-41.

[DeMillo 79] DeMillo, Richard A., Lipton, Richard J., and Lipton, Perlis, Alan J., "Social Processes and Proofs of Theorems and Programs," *Communications of the ACM*, 22, 5, (1979), pp. 271-280.

[DeMillo 83a] DeMillo, Richard A., Davida, George I., Dobkin, David P., Harrison, Michael A., and Lipton, Richard J., "Applied Cryptology, Cryptographic Protocols, and Computer Security Models," American Mathematical Society (PSAM) 29, Providence, RI, (1983).

[DeMillo 83b] DeMillo, Richard A., and Merritt, Michael J., "Protocols for Data Security," *Computer*, 16, 2, (1983), pp. 39-54.

[DoD 83] DoD Computer Security Center. Department of Defense Trusted Computer System Evaluation Criteria. (August 15, 1983).

[Duran 81] Duran, J. W., and Ntafos, S., "A Report on Random Testing," *Proceedings of the Fifth International Conference on Software Engineering*, San Diego, CA, (March 9-12, 1981), pp. 179-183.

[Eastport 85] Eastport Study Group. Summer Study 1985. A Report to the Director Strategic Defense Initiative Organization, (December 1985).

[Fairley 78] Fairley, Richard E., "Tutorial: Static Analysis and Dynamic Testing of Computer Software," *Computer*, 11, 4, (1978), pp. 14-23.

[Fairley 85] Fairley, Richard E., Software Engineering Concepts, McGraw-Hill, (1985).

[Fernandez 81] Fernandez, Edwardo B., Summers, Rita C., and Wood, Christopher, Addison-Wesley, 1981.

[Fletcher 83] Fletcher, James C. (Study Chairman), McMillan, Brockway (Panel Chairman), and Defensive Technologies Study Team, "Report of the Study on Eliminating the Threat Posed by Nuclear Ballistic Missiles," *Volume V: Battle Management, Communications, and Data Processing*, Alexandria, VA, (October 1983).

[Fujii 78] Fujii, Marilyn S., "A Comparison of Software Assurance Methods," *Proceedings of the Software Quality and Assurance Workshop* , San Diego, CA, *ACM*, New York, (November 15-17, 1978), pp. 27- 32.

[Goel 85] Goel, Amrit L., "Software Reliability Models: Assumptions, Limitations, and Applicability," *IEEE Transactions on Software Engineering*, SE-11, 12, (1985), pp. 1411-1423.

[Goodenough 75] Goodenough, J.B., "Exception Handeling: Issues and a Proposed Notion," *Communications of the ACM*, (December 1975), pp.683-696.

[Green 76] Green, C., "The Design of the PSI Program Synthesis System," *Proceedings of the Second International Conference on Software Engineering*, (1976), pp. 4-18.

[Griffith 85] Griffith, N., and Miller, J.A., "Performance modeling of Database Recovery Protocols", *IEEE Transactions on Software Engineering*, SE-11, 6, (November 1985), pp. 564-571.

[Gunningberg 83] Gunningberg, Per, "Voting and Redundancy Management Implemented by Protocols in Distributed Systems."

[Hamlet 86] Hamlet, Richard, "Probable Correctness Proceedings of Workshops on Software Testing," (July 1986).

[Haseganas 85] Haseganas, S., Liu, J.W.S., Lu, C., Railey, M., "Reliable Clock Driven Process Synchronization Algorithms," *Digest of Papers FTCS-15: Fifteenth Annual International Symposium on Fault-Tolerant Computing*, Ann Arbor, Michigan, (June 19-21, 1985), pp.207-213.

[Hecht 86] Hecht, Herbert, and Hecht, Myron, "Software Reliability in the System Context," *IEEE Transactions on Software Engineering*, SE-12, 1, (1986), pp. 51-58.

[Henderson 86] Henderson, Peter, "Functional Programming, Formal Specification, and Rapid Prototyping," *IEEE Transactions on Software Engineering*, SE-12, 2, (1986), pp. 241-250.

[Heninger 80] Heninger, K., "Specifying Software Requirements for Complex Systems: New Techniques and Their Application," *IEEE Transactions on Software Engineering*, SE-6, 1, (January 1980), pp. 2-13.

[Hoare 1978] Hoare, C. A. R., "Communicating Sequential Processes," *Communications of the ACM*, (August 1978), pp. 666-677.

[Howden 1978] Howden, W. E., "An Evaluation of the Effectiveness of Symbolic Testing,"*Software Practice and Experience*, 8, 4, (1978), pp. 381-397.

[Howden 1980] Howden, W. E., "Functional Program Testing," *IEEE Transactions on Software Engineering*, SE-6, 2, (March 1980), pp. 162-169.

[IDA 87] Linn, C., Linn, J., Kappel, M., Salasin, J , *Strategic Defense Initiative Ada Process Descripti : Language, Version* 1.00, P-1953, Institute for Defense Analyses, Alexandria, VA, 1987.

[Ihara 79] Ihara, H., et. al., Fault-Tolerance Through Autonomus Decentralized System, *Working Conference on Reliable Computing and Fault-Tolerance in the 1980's*, (1979), London.

[Jackson 82] Jackson, Michael A., System Development, Prentice-Hall, Englewood Cliffs, NJ, (1982).

[Jordan 86] Jordan, Carole S., "Guidelines and Standards," Proceedings National Computer Security Conference, 15 - 18 September 1986, pp. 231-233.

[Kim 86] Kim, K., Garcia-Melina, H., Kuhl, J., Nelson, V., Ramamoorthy, C.V., and Sarin, S., RADC Workshop on Fault-tolerant and Survivable Distributed Systems, July 29-30, 1986.

[Knight 84] Knight, John C., and Urquhart, John I. A., "On the Implementation and Use of Ada on Fault-Tolerant Distributed Systems," *Ada LETTERS*, (October 1984).

[Knight 85] Knight, John C., Levenson, Nancy G., St.Jean, Louis D., "A Large Scale Experiment in N-Version Programming," *IEEE 1985 Fault Tolerent Computing*.

[Knight 86] Knight, J.C., and Leveson, N.G., "An Experimental Evaluation of the Assupmptions of Independence in Multi-Version Programming," *IEEE Transactions on Software Engineering*, SE-12, 1, (January 1986), pp.96-109.

[Kopetz 80] Kopetz, H., *Software Reliability*, Springer-Verlag, 1980.

[Krishna 84] Krishna, C.M., Shin, K.G., and Butler, R.W., "Synchronization and Fault-Masking in Redundant Real-Time Systems," *Digest of Papers FTCS-14: Fourteenth Annual International Symposium on Fault-Tolerant Computing*, Kissimmee, Fla, (June 20-22 ,1984), pp.152-157.

[Laksham 86] Laksham, T.V., Agrawala, Ashok K., Efficient Decentralized Consensus Protocols, *IEEE Transactions on Software Engineering*, SE-12, 5, (May 1986), pp. 600-607.

[Laprie 84] Laprie, Jean-Claude, "Dependability Evaluation of Software Systems in Operation," *IEEE Transactions on Software Engineering*, SE-10, 6, (November 1984).

[Laprie 85] Laprie, Jean-Claude, "Dependable Computing and Fault-tolerance: Concepts and Terminology, Workshop on Fundamental Concepts Summer 1985 Meeting of the IFIP WG 10.4, Le Chateau Montebello, Canada, June 1985 (a version also appeared in FTCS-15).

[Leveson 83a] Leveson, Nancy G. and Harvey, Peter R., "Analyzing Software Safety," *IEEE Transactions on Software Engineering*, SE-9, 5, (September 1983), pp.569-579.

[Leveson 83b] Leveson, Nancy G. and Shimeall, Timothy J., "Safety Assertion for Process-Control Systems," *IEEE Digest of Papers FTCS-13: Fault-Tolerant Computing 13th Annual International Symposium,* IEEE (1983), pp. 236-240.

[Leveson 84] Leveson, Nancy G., "Software Safety in Computer-Controlled Systems," *IEEE Computer*, (February 1984), pp. 48-55.

[Leveson 86] Leveson, N.G., "Software Safety: Why, What and How," *Univ. Calif. Irvine, Technical Report 86-04*, (February 1986).

[Linger 79] Linger, R. C., Mills, H. D., and Witt, B. I., "Sructured Programming: Theory and Practice," Addison-Wesley, Reading, MA, (1979).

[Liskov 79a] Liskov, B.H., and Snyder, A., "Exception Handeling in CLU," *IEEE Transactions on Software Engineering*, SE-5, 6, (November 1979), pp. 546-558.

[Liskov 79b] Liskov, Barbara H., and Berzins, Valdis, "An Appraisal of Program Specifications," *In Research Directions in Software Technology*. Wegner, Peter, Editor. The MIT Press, Cambridge, MA, (1979), pp. 276-301.

[Liskov 86] Liskov, B. and Guttag, J., Abstraction and Specification in Program Development, MIT Press, 1986.

[London 85] London, Ralph L., and Duisberg, Robert A., "Animating Programs Using Smalltalk," *Computer*, 18, 8, (1985), pp. 61-71.

[Luckham 84] Luckham, D.C., and Von Henke, F.W., "An Overview of ANNA, a Specification Lnaguage for Ada," *IEEE Computer Society 1984 Conference on Ada Applications and Environments*, (October 15-18, 1984), pp. 116-127.

[McCluskey 85] McCluskey, E.J., and Andrews, D.M., "The Measurement and Statistical Modeling of Computer Reliability as Affected by System Activity, Final Report," *Center for Reliable Computing, Stanford University (NTIS AD-A164 249)*, (November 1985)

[Martin 83] Martin, D.J., "Dissimilar Software in High Integrity Applications in Flight Controls," *Software for Avionics, AGARD Conference Proceedings*, no.330, (January 1983).

[Mayfield 86] Mayfield, W.T., Chludzinski, J., McHugh, J., and Welke, S.R. (eds.), Proceedings of the Third IDA Workshop on Formal Specification and Verification of Ada 14-16 May 1986, (IDA Memorandum Report M-241), August 1986.

[Mili 84] Mili, A., "Towards a Theory of Forward Error Recovery," *IEEE Transactions on Software Engineering*, SE-11, 8, (August 1984), pp.735-748.

[Mills 86] Mills, Harlan D., Private Communication, (May 86).

[Miyamoto 83] Miyamoto, Shoji, Nohmi, Makoto, Mori, Kinji, Ihara, Hirokazo, "FMPA: A Fault-Tolerant Multi-Microprocessor System Based on Autonomous Decentralization Concept," *Proceedings of the Fault-Tolerant Computer Conference*, IEEE, 1983, pp. 4-9.

[Mori 81] Mori, K., et. al., Autonomous Controlability of Decentralized Systems Aiming at Fault-Tolerance, *IFAC *th World Congress*, x-129-134, (1981).

[Moss 85] Moss, J.E.B., *Nested Transactions: An Approach to Reliable Distriuted Computing*, MIT Press, (1985)

[Musa 87] Musa, John D., Anthony Iannino, and Kazishira Okanoto, "Software Reliability: Measurement, Prediction, Application," McGraw-Hill to be published in 1987.

[Neumann 86] Neumann, Peter G., "On Hienarchial Design of Computer Systems for Critical Applications," *IEEE Transactions on Software Engineering*, SE-12, 9, (September 1986), pp. 905-920

[NSIA 83] National Security Industrial Association, "C2 Software Development and Acquisition Study," R. J. Martin, Director. Briefing Book, (August 1983).

[O'Neill 86] O'Neill, Don, Ada Process Description Language Feasibility Study for SDI BMC2, IBM Federal Systems, October 1986.

[Panzl 81] Panzl, D.J., "Experience with Automatic Program Testing," *Proceedings Trends and Applications 1981*, IEEE, (May 28, 1981).

[Parnas 86] Parnas, D.L., "When can Software be Trustworthy?" Keynote Address to Compass '86, Washington D.C. July 7-11, 1986.

[Parnas 85] Parnas, D.L., and Weiss, D.M., *Active Design Reviews: Principles and Practices*, NRL Report 8527, (November 18, 1985).

[Perry 85] Perry, K.J., "Randomized Byzentine Agreement," *IEEE Transactions on Software Engineering*, SE- 11, 6, (November 1985), pp. 535-546.

[Peterson 77] Peterson, J., Petri Nets. ACM Computing Surveys, 9, 3, (September 1977).

[Redwine 84] Redwine, Samuel T., Jr., Becker, Louise, Giovane, Marmor-Squires, Ann B., Martin, R J., Nash, Sarah H., and Riddle, William E., "DoD Related Software Technology Requirements, Practices, and Prospects the Future," *IDA Paper P-1788*, (June 1984).

[Redwine 86a] Redwine, Samuel T., Jr., "Presentation at Future Directions in Software Testing Session," *Workshop on Software Testing*, (17 July 1986).

[Redwine 86b] Redwine, Samuel T., Jr., "The Software Development Process as a Fault-Tolerant System, " *Position Paper submitted to 3rd International Workshop on the Software Process*, (17-19 November 1986).

[Richardson 85] Richardson, Debra J., and Clarke, Lori A., "Partition Analysis: A Method Combining Testing and Verification," *IEEE Transactions on Software Engineering*, SE-11, 12, (1985), pp. 1477-1490.

[Rine 80] Rine, David C., "Some Models for Security and Protection Analysis Based on Possibility Theory and Fuzzy Sets," *CYBERSOFT 80, International Symposium on Cybernetics and Software,* Namur, Belgium, (September 9, 1980).

[Robinson 86] Robinson, Arthur S., Private communication, (May 1986).

[Roby 85] Roby, C., (editor), "Proceedings of the First IDA Workshop on Formal Specification and Verification of Ada," *IDA Memorandum Report M-146*, (March 18-20, 1985).

[Ruth 78] Ruth, G., "Protosystem I: An Automatic Programming System Prototype," *Proceedings of the National Computer Conference*, Anaheim, CA, AFIPS 47 (1978), pp. 675-681.

[Scott 84] Scott, R.K, Gault, J.W., McAllister, D.F., and Wiggs, J., "Experimental Validation of Six Fault-Tolerant Software Reliability Models," *Digest of Papers FTCS-14: Fourteenth Annual International Symposium on Fault-Tolerant Computing*, Kissimmee, Fla, (June 20-22, 1984), pp.102-107.

[Selby 86] Selby, R.W., "Combining Software Testing Strategies: An Empirical Evaluation," *Proceedings Workshop on Software Testing, IEEE*, (16-17 July 1986), pp.82-90

[Seth 85] Seth, Shard C. and Muralidhar, R., "Analysis and Design or Robust Data Structures," *Digest of Papers FTCS-15: Fifteenth Annual International Symposium on Fault-Tolerant Computing*, Ann Arbor, Michigan, (June 19-21, 1985), pp.14-19.

[Shin 84] Shin, K.G., and Lee, Y-H, "Evaluation of Error Recovery Blocks for Cooperating Processes," *IEEE Transactions on Software Engineering*, SE-10, 6, (November 1984), pp.692-700.

[STEP 83] OSD/DDT&E Software Test and Evaluation Project, Phases I and II Final Report. *Volume 2: Software Test and Evaluation: State-of-the-Art Overview*, (June 1983).

[Teichrow 77] Teichrow, D. and Hershey E., "PSL/PSA: A Computer Aided Technique for Structured Documentation and Analysis of Information Processing Systems," *IEEE Transactions on Software Engineering*, SE-3, 1, (1977).

[Trivedi 82] Trivedi, Kishor Shridharbhai, "Probability and Statistics with Reliability, Queuing, and Computer Science Applications," Prentice-Hall, Inc., Englewood Cliffs, NJ, (1982).

[Vouk 86] Vouk, M.A., McAllister, D.F., and Tai, K.C., "An Experimental Evaluation of the Effectiveness of Random Testing of Fault-Tolerant Software," *Proceedings Workshop on Software Tesing, IEEE*, (15-17 July 1986), pp.74-81.

[Walsh 85] Walsh, Patrick Joseph, "A Measure of Test Case Effectiveness," *PhD Dissertation, T. J Watson School of Engineering, Applied Science, and Technology*, State University of New York at Binghamton, NY, (1985).

[Waters 82] Waters, Richard C., "The Programmer's Apprentice: Knowledge Based Program Editing," *IEEE Transactions on Software Engineering*, SE-8, 1, (January 1982).

[Wirth 71] Wirth, Niklaus, "Program Development by Stepwise Refinement"," *Communications of the ACM*, 14, 4, (1971).

[Yourdon 77] Yourdon, E., *Structured Walkthoughs*, Yordon Press, (1977).

[Yourdon 79] Yourdon, E., and Constantine, L. "Structured, Design: Fundamentals of a Discipline of Computer Program and Systems Design," Prentice-Hall, Englewood Cliffs, NJ, (1979).

[Zave 86] Zave, Pamela and Schell, William. "Salient Features of an Executable Specification Language and Its Environment," *IEEE Transactions on Software Engineering*, SE-12, 2, (1986), pp. 312- 325.

SECTION B12

People and Organizations

Prepared by Sarah H. Nash and Samuel T. Redwine, Jr.

Topics covered in Section B12:

12.1 Introduction
12.1.1 Purpose and Scope
12.1.2 Background
12.1.2.1 Relationship to Other Areas
12.1.3 Organization of Section

12.2 SDI Requirements
12.2.1 Functionality
12.2.2 Implications of Architecture
12.2.2.1 People
12.2.2.2 Organizations

12.3 Current Status
12.3.1 Individuals
12.3.1.2 Individual Differences
12.3.1.3 Education and Training
12.3.1.3.1 Manpower Shortage
12.3.1.3.2 Undergraduate Curricula
12.3.1.3.3 Graduate Curricula
12.3.1.3.4 Continuing Education and Specialized Training Courses
12.3.1.3.5 Summary
12.3.1.4 Selection
12.3.1.5 Evaluation, Certification, Testing, Measurement
12.3.1.6 Motivation
12.3.2 Organizations
12.3.2.1 Teams, Group Dynamics, Quality Circles
12.3.2.2 Projects
12.3.2.3 Organizations
12.3.2.4 Interorganization Relations

12.4 Recommendations

12.5 References

List of Tables

Table	Title	Page
1	SDI Software Cost Leverage Factors.....	288
2	Rough Partial Acquisition Imposed Scenario.....	308-309

12.0 PEOPLE & ORGANIZATIONS

12.1 Introduction

12.1.1 Purpose and Scope

This section covers issues concerning the people and organizations to produce SDI software and the technology related to these people and organizations. Success of the SDI effort does not depend on technology alone. Issues of project organization, education and training, professional development, career paths, personnel selection, evaluation, group dynamics, motivation, and productivity will play a significant part and cannot be ignored.

The military has a long history of using psychology successfully to improve individuals, teams, and organizational performance [Watson 78]. The projected size, difficulty, and criticality of the SDI effort require that it continue this history by undertaking R&D and using existing and new results concerning people and organizations in the software engineering process. The projected size of the SDI system alone demands large numbers of people and organizations performing at substantially better than average levels.

As the COCOMO (COnstructive COst MOdel) [Boehm 81] software costing model shows, People and Organizations can have a dramatic effect on productivity and costs. For example, COCOMO has demonstrated that, all other factors being equal, a 90th-percentile team of programmers and analysts will be about four times more productive in delivered source instructions per man-month as a 15th-percentile team [Boehm 81 p. 642].

Figure 1 shows some of the People and Organization-related SDI software cost leverage factors computed using COCOMO for 1989-93, assuming various conditions. With suitable preparation in 1986-88, the SDI could reduce the COCOMO effort multipliers to achieve an estimated productivity improvement of a factor of 2.4, 1.4 of which is attributable to the "people" factors in Figure 1. With further effort during 1989-92, the productivity improvement factor could be improved to roughly 5.2 [Boehm 86, p. 295]. The factors in Figure 1 account for 1.8 of this total. Thus, a large part of this improvement can be achieved by improving the performance of people and organizations.

Improving performance is not the only reason for concern in this area. Other reasons are to identify and select the most effective people and organizations to build the SDI system, to encourage innovation, and to improve the DoD business environment. For example, contracts could require human resource plans, personnel evaluation criteria, or other measures to assure that contractors select and develop effective personnel.

12.1.2 Background

Large variations in performance among individuals have been measured. Such variations present the opportunity for selection and to learn why some individuals are superior at learning and performing software-related skills. This knowledge may be used to improve the skills of others. The personality types and knowledge bases that combine to create the best analysts, programmers, programming managers, etc. are not fully understood. Inexpensive, dependable ways to identify highly productive individuals are needed. More effective techniques for training management and staff in individual effectiveness and group skills, to raise their productivity as individuals and as members of the design or programming team would be useful [DoD 82, pp. 136-137].

Organizational issues of team building, group dynamics, project organization, and interorganizational relations are equally important. The known technology related to group interaction is generally underused in software creation today. The best way to organize at the project level within a given organization to produce software is often not known. Interorganizational relations are important to efforts (like the SDI) that require many diverse organizations to interact effectively.

Significant work exists on these issues in general and relating to software personnel and organizations in particular [e.g. Curtis 85]. However, some gaps need to be filled and methods need to be developed to ensure their effective use in the DoD acquisition context.

12.1.2.1 Relationship to Other Areas

People and Organizations is related to other task areas in this Plan. Software engineering processes, particularly software engineering environments, involve interactions among people and between organizations and the methodology selected. Man-Machine Interface involves people and organizations as does Technology Transition. Software Dependability depends to a great extent on the people producing the software.

12.1.3 Organization of Section

The remainder of this section addresses the SDI requirements, the state of the art, the state of the practice, and the specific projects that relate to people and organizations. The topic of people and organizations is subdivided into individual differences, education and training, selection, evaluation, motivation, teams, projects, organizations, and interorganizational relations. The section concludes with short- and longer-term recommendations.

12.2 SDI Requirements

12.2.1 Functionality

Many people with different skills will be required to implement SDI requirements in software. These skills include software engineering, computer science, and application-oriented knowledge and acquisition and contracting expertise. Contractors, government agencies, and FFRDCs (Federally Funded R&D Centers) all require these people.

These people need to be organized into teams, projects, and organizations to do an unprecedentedly difficult job that requires continued improvement. All the organizations involved must work well together. The Government must ensure and encourage all this to be successfully done using R&D, management, and acquisition mechanisms.

12.2.2 Implications of Architecture

The architecture is projected to be complex, distributed, open, subject to a changing threat, and evolving over a long period of time. This implies the following requirements for People and Organizations.

12.2.2.1 People

The SDI will be a difficult system to develop and, therefore, the best chance for success comes from having people work on it who commit few errors, are highly productive, are knowledgable about the state of the art, and work well in organizations. Identifying, selecting, training, motivating, and managing these individuals are therefore paramount.

	Effort Multipliers		
	1989-1992 Typical	1989-1992 With Preparation	1993 Best Possible
Training and Experience			
Language Experience	1.01	0.98	0.96
Computer Experience	1.01	0.96	0.92
Applications Experience	0.95	0.90	0.85
Modern Programming Practices	0.92	0.87	0.82
Architecture/Management			
Personnel Capability	0.74	0.65	0.60
Product	0.66	0.48	0.37
Gain Factor	1.0	1.4	1.8

Source: [Boehm 86 p. 295]

Table 1. SDI Software Cost Leverage Factors

People working on the SDI will need to have state-of-the-art knowledge. In some cases, they will need to be advancing the state of the art. The SDI project has the opportunity (and the necessity) to actively improve people's knowledge and skills. The situation is analagous to those that produced astronauts, fighter pilots, and special operations troops. This ability to actively make improvements gives the SDI project potential leverage.

The prototyping approach that has been advocated for the SDI development offers the added benefit of allowing incremental improvement in people as well as in the system. As such, it offers a valuable learning experience.

The complex, "never been done before" nature of the SDI system can give organizations and people working on it a sense of eliteness and purpose that will motivate them to be productive.

12.2.2.2 Organizations

The architecture of the SDI requires organizations that are near the state of the art, aimed at fault tolerance and reliability, constantly improving, and pro-active in that they anticipate and react quickly to changes and problems. The use of personnel redundancy, increased formalism, pro-active evolution, systematic error-detection, independent reviewers, and multiple teams can make organizations more fault tolerant. Interorganizational relations are important since so many diverse organizations will need to interact effectively.

Product and architecture issues will also drive organizational structure. A matrix organization serves both project needs and the need to improve functional skills. Other implications for organizational structure include the need for flexibility to adapt to changing requirements, technologies, strengths, and weaknesses. It may even be necessary to reorganize organizations to take advantage of new technical developments.

12.3 Current Status

12.3.1 Individuals

Computer personnel research reveals vast differences in the abilities of individuals [Sackman 86, Schwartz 68, Boehm 75, Myers 78, Frank 79] and in their cognitive and personality styles [Kaiser 85; Adelson 85]. Outside strictly computer personnel much is known about individuals and their performance as is well summarized by Nash and Klitgaard [Nash 85, Klitgaard 85]. These findings have implications for personnel selection, education and training, motivation, management, and communications. To date, however, neither aptitude tests nor academic performance appear to be predictors of on-the-job performance [Glass 80], posing particular difficulties for personnel selection.

12.3.1.2 Individual differences

People differ according to personality, behavior, cognitive processes, experts vs novices, and demographic/situational variables. The types of individuals that will be needed to build the SDI system include analysts, programmers, designers, testers, and managers, as well as many more specialized professionals. Research to date on individual differences has revolved around superdesigners, superprogrammers, MIS personnel, innovators, and expert data modelers. Results can be used in education, automated tool development, knowledge engineering, manpower planning, job design, selection, and evaluation. Curtis has produced a review of the literature on individual differences through 1983 [Curtis 83]. Another review of the literature can be found in [Laughery 85].

Much of the literature acknowledges that the software culture is different from other work cultures [Schneiderman 80; Weinberg 71; Liker 85; Lammers 86]. Some elements of this culture include invisible (or unreadable to the lay person) products, an entrepreneurial style, and a fluid relationship between the worker and the tool. Additional elements include producing the tools they work with and combining attributes of both consumer and producer [Liker 85].

Little is known about why some individuals are superior to others in performance. Questions first asked in the late 1960s that remain unanswered include why some programmers are more than an order of magnitude better than others, what those exceptional programmers do differently from others, and whether or not the answers to these questions can be used to effect a human technology transfer from the exceptional to the needy [Glass 80]. Recent and ongoing research on superprogrammers, superdesigners, expert data modelers and expert vs. novice behavior is aimed at learning why some people are better than others; the goal is to exploit this knowledge through automation and education.

One study on superprogrammers [Molzberger 83] found that such people do not experience programming as a purely rational activity. For them it possesses strong intuitive and aesthetic components. Programs are visualized as three-dimensional structures and programmers often visualize themselves as the processor. Phases of extreme creativity and concentration are typically marked by a reduced need for sleep and food and a changed subjective notion of time. The role of intuition, feelings of absolute certainty, sudden illumination, and feelings for mathematical beauty have similarly been noted in mathematics [Molzberger 83]. This work was done in Germany and by Molzberger's own account has received mixed reviews. While the conclusions reached may be unorthodox, the need to find out what makes excellent programmers remains.

Differences between experts and novices have received considerable attention in the computer field [Shertz 81; Gugerty 86; Silverman 85; Adelson 84; Adelson 85].

In examining differences in problem representation among novice and expert programmers and programming managers, Shertz and Weiser discovered that novices consider the surface features of the problem but represent problems inconsistently; experts consider the deep features and tend to agree with one another on what those features are; and managers use surface structures in a very consistent way to assign problems to people [Shertz 81].

Studies on abstract and concrete concept categorization by programmers also shed light on the problem-solving processes of experts and novices [Adelson 85, p. 428; Adelson 84, p. 483].

Experiments on expert-novice differences in debugging reveal that experts debug more quickly and accurately, largely because they generate high quality hypotheses on the basis of less study of the code than novices. Novices frequently add bugs in the debugging process [Gugerty 86].

Yet another study examined expert and novice programmers in terms of strategies employed to memorize computer programs. Beginning programmers use rote memory, alphabetical orderings, and common language associations while experts' strategies are dominated by functional/meaning of words [Silverman 85].

A survey of 233 citations on how systems engineers use analogy in planning and designing revealed that very little research has directly addressed this question to date. Much of the

cognitive research on planning and design in software programming concerns differences in speed or detail with which experts arrive at the finished code relative to novices. The use of analogies plays an important part in the problem-solving approach of software experts when confronted by new and ill-specified problems. However, software engineering literature in the past 15 years makes little mention of such an approach. Similarly, little cognitive research yet exists to support this argument [Silverman 85].

A study by Kaiser on the relationship of cognitive style to the derivation of information requirements found that experience is more important than cognitive style. She found little correlation between cognitive style and differences in deriving information requirements [Kaiser 85].

There is a dearth of research on kinds of programming tools available and the frequency with which they successfully help programmers create, test and maintain programs [Hanson 85]. It seems that except for expert-novice studies, little research is being undertaken to find out what makes excellent programmers.

Related to studies of superprogrammers are studies of superdesigners. The Microelectronics and Computer Technology Corporation's (MCC) Software Technology Program [Myers 85] is currently undertaking studies of superdesigners to discover their cognitive processes and how to use methods and technologies to improve these processes. The utlimate aim is to better understand the process in order to automate as much of it as possible [Curtis 86a, 86b, 86c, Shen 86]. The software design process has been studied elsewhere [Adelson 84; Adelson 85; Kant 85; Steier 85].

Temple University has undertaken research to determine how users validate the transformation from needs to the requirements specification. After looking at several methods of representation, none were found to be superior, although prototyping appears promising [Nosek 86].

One area where individual differences in design, implementation and usage have been researched is in management information systems (MIS). Individual differences found to be most relevant to MIS success include cognitive style, personality, and demographic/situational variables. However, much remains unknown regarding the specific relationships involved and the relative importance of individual differences when contrasted with contextual factors [Zmud 79].

In contrast, data modeling expertise has not been investigated much. Smelcer's dissertation at the University of Michigan [Smelcer 86] attempts to understand the mental models used by experts for tasks of different difficulty. The hypothesis is that experts use rules for solving complex modeling problems, use analogs for difficult problems, and previously-solved problems for easy problems. Non-experts use none of these models. Smelcer used memorization-and-recall tasks to test the hypothesis by tapping experts' mental models. He contends that a better understanding of models used by expert data modelers can contribute to pedagogy, knowledge engineering, and expert tool development [Smelcer 86]. Such research is of interest to SDI planners, not only from a "people" point of view, but because of its potential contribution to artificial intelligence.

A final area where individual differences have implications for the SDI effort is the effect of behavioral differences on innovation. While the section on Technology Transition treats this subject in more depth, research on the behavioral aspects deserves mention here. Recent research has identified five informal, but critical, behavior functions needed for the effective execution of technology-based innovative projects. They are idea generation, entrepreneuring or championing, project leading, gatekeeping, and sponsoring or coaching.

Two key observations are that (1) some unique individuals have the capability of performing more than one of these functions concurrently and (2) patterns of roles for an individual often change during the course of his career. This research suggests directions for manpower planning, job design, objective setting, and performance measurement and rewards [Roberts 81] and is therefore of potentially intense interest to the SDI software effort.

12.3.1.3 Education and Training

The requirement for large quantities of highly skilled software engineers with state-of-the-art knowledge to build the SDI system mandates an emphasis on education and training. Current and projected shortages for software engineers will only aggravate the problem. Others [Ford 86] within the DoD and elsewhere have recognized the problem and are taking steps to correct it by initiating various education and training programs, including graduate and undergraduate curricula [SIGCSE 85], continuing education, and retraining programs. Concern is growing, however, for the adequacy of university faculty and equipment given future federal budget plans [Gries 86].

12.3.1.3.1 Manpower Shortage

The total amount of software that the DoD needs to build is well beyond the capability of the number of software engineers that can be expected to be available, using current software development methods. According to a recent Department of Commerce study [Commerce 84], the 600,000 to 700,000 programmers and systems analysts in the U.S. have not been able to fill the dramatic increase in demand for their services that resulted from the rapid growth in the use of computers. The 1983 U.S. gap between demand and supply has been estimated in terms of 50,000 to 100,000 software professionals, and if nothing is done, this gap could become 860,000 to 1,000,000 software professionals by 1990 [Martin 83; DoD 82].

From 1979 to 1982, employment in software products (packaged software) and professional services (including custom programming) expanded at a 40 percent average annual rate. Even during the recession years from 1980 to 1982, overall software industry employment grew by 17 percent. The software products sector had the highest growth rate of any sector during the 1981 to 1983 period, increasing threefold from 22,000 to 68,000 employees [Commerce 84].

The National Science Foundation (NSF) projects an 8.9% - 12.3% annual growth rate for defense requirements for computer specialists as opposed to a 5.4% - 6.4% growth rate for non defense requirements for computer specialists [NSF 84]. NSF also predicts that the total shortfall for computer specialists will be in the 15 to 30 percent range (about 115,000 to 140,000) by 1987 [NSF 84]. In the banking field alone, the U.S. Department of Labor projects a 40% increase in programmers by 1990 [Perelman 82].

The U.S. is not alone. According to a National Computing Center survey there is a 7.6% shortage of programmers, 8.2% of analyst/programmers, and 8.2% of system analysts in the United Kingdom (U.K.). Since the survey cannot take into account installations that do not yet exist, these numbers understate the situation [Penney 85]. Elsewhere within the U.K., there is growing concern that the widespread introduction and development of information is being hindered by a growing shortage of manpower with high level information technology skills, despite high unemployment in other fields [Gordon 84].

These shortages are further aggravated by the shortage of college faculty in computer science and software engineering. Although undergraduate student demand for computer

science courses is growing rapidly, the number of new PhD.'s in this area that are taking academic positions each year has not increased since 1975 [NSF 85]. 8.5% of authorized engineering faculty positions are vacant. In its survey of the nation's universities, the American Society for Engineering Education learned that 23% more faculty positions, somewhere between 6,000 and 7,000, are needed to get the quality of engineering and computer science programs back where it was a few years ago [Myers 85a].

Several types of education are needed to meet these shortages including, undergraduate education, graduate education, continuing education, company-based training programs and career paths, and retraining programs. The DoD, through its software initiative comprising the Ada Joint Program Office (AJPO), Software Engineering Institute (SEI), and Software Technology for Adaptable Reliable Systems (STARS) program, has been attacking the problem. Professional societies, such as the Association for Computing Machinery (ACM), develop curricula and sponsor workshops. The private sector contributes by offering company-based programs, private institutes, and cooperative programs with universities. An entire training industry offering seminars in computer science and software engineering is growing at a rapid pace.

12.3.1.3.2 Undergraduate Curricula

The ACM curriculum task force recently completed the elaboration and updating of the first two courses (CS.1 and CS.2) in the ACM Curriculum '78 to reflect the increased knowledge in the computer science field and the shift in teaching emphasis resulting from the growing discipline of software engineering [Koffman 85].

The Carnegie-Mellon Computer Science Department's curriculum design project recently examined the current state of computer science curricula, projected the requirements for undergraduate education in computer science over the next decade, and developed a curriculum suitable for a computer science major [Shaw 85] based around concepts.

A recent article [Gibbs 86a] proposed a more traditionally organized undergraduate curriculum for a B.A.-degree program in computer science. The curriculum is intended as a model not only for high-quality undergraduate colleges and universities, but also for larger universities with strong computer science programs in a liberal arts setting [Gibbs 86a].

In addition, Japan's success in the computer industry suggests that their curricula should not be ignored. Undergraduate engineering programs in Japan have high admission standards, are homogeneous in content, and emphasize problem solving and writing skills [Westney 86]. Communication skills, in particular, are often overlooked in conventional U.S. education programs for computer professionals, despite their importance in implementing integrated information systems [Enger 81].

Under a grant from the National Science Foundation (NSF), Clarkson College of Technology is implementing an Institute for Retraining in Computer Science to attack the related problem of computer science college faculty shortages. It is based on the idea that college faculty in other areas where the personnel situation is not so critical can make use of their background in subjects such as mathematics, their teaching experience and what knowledge they already have of computer science to learn quickly enough material to be qualified to teach a significant portion of the undergraduate curriculum in computer science. [NSF 85]. Similarly, the SEI's Education Division is working to develop software engineering faculty [Gibbs 86] by sponsoring semi-annual faculty development workshops and other activities [SEI 86, p. 48]. The number of software engineering faculty will

grow as more universities such as the Wang Institute begin to offer PhD degrees in Software Engineering [Ford 86a].

12.3.1.3.3 Graduate Curricula

The major effort of the SEI Education Division is design, development, and insertion in universities and industrial education programs of a graduate level software engineering curriculum. Currently, only Seattle University, the Wang Institute, and Texas Christian University (TCU) offer graduate programs in Software Engineering [SEI 86]. If engineering concepts really are to be introduced into computer science, more universities will need to offer a software engineering curriculum. This contrasts somewhat with the computer science master's level program curriculum recommended by ACM in 1981 [ACM 81].

12.3.1.3.4 Continuing Education and Specialized Training Courses

By far, the bulk of non-university computer science education activities sponsored by the U.S. government and industry fall into the category of continuing education and specialized training courses.

Within the DoD, the Ada Program has been particularly vigorous in promoting Ada Education Activities. These include a U.S. Army Ada Training Curriculum Catalog 1984, an Ada Software Engineering Education and Training Plan, and the AFCEA (Armed Forces Communications and Electronics Association) Study of Education and Training for Ada. Key objectives of the AFCEA study are to (1) identify requirements in DoD, NATO and industry for Ada education and training; (2) provide recommendations for effectively meeting Ada education and training requirements; (3) develop methodologies that will be helpful in future updating of Ada education and training requirements; (4) assess the effectiveness of various education and training approaches; (5) increase awareness throughout DoD, NATO and industry of Ada education and training matters; and (6) promote implementation of the recommendations of the study.

The Ada Training Curriculum [SofTech 84] defines a comprehensive set of training course modules or building blocks that can be connected in a variety of ways and coupled with workbooks and supplementary materials to form training programs that satisfy a given set of needs.

An IDA Central Research Project [Bailey 86] developed a capability for evaluating courses in Ada. Indications are that the ability to evaluate Ada courses could be a critical issue for the DoD in the very near future as the need for Ada-literate personnel grows.

The STARS Human Resources area [Oglesby 83] has been concerned with the professional workforce in the DoD software area, including DoD and contractor employees. The STARS Program has recently undergone reorganization and redirection, and they no longer plan to do the following activities.

The goal of the STARS Human Resources area was to increase the size of the DoD software personnel workforce, change its composition in light of the tasks to be performed, and improve its capability. The area was divided into three subareas described below.

The Work Status, Structure, and Requirements subarea was to build a database describing the current situation of the total software workforce, and develop a model to predict needs (including quantities and skill-mix) in light of planned DoD programs.

The Workforce Structure Enhancement subarea was to address the changes needed in the quality and size of the software workforce by influencing career paths, education, and training. This was to be accomplished by influencing, leveraging, and accelerating existing efforts including those in academia, industry, and the DoD. Key DoD institutions for direct promotion of improved education and training in software engineering and management include the military academies, the Defense Systems Management College, the Naval Post Graduate School, the Air Force Institute of Technology, and senior Service schools. STARS was to sponsor a training program for both technical and managerial personnel that would permit tailoring to individual needs and abilities. Also, a traveling team would have been formed to provide "awareness" lectures throughout DoD and industry.

The third subarea, Development of Education Material would have supplemented the SEI Education Division and Ada efforts. It was to create training material for in-house and contractor personnel and investigate the use of computer-aided instruction, including expert systems technology, in the development of the required material and courses [STARS 85].

The SEI Education Division is also concerned with training. It develops and conducts seminars on the evolving state of the art and practice in software engineering for mission critical computer systems [SEI 85; Barbacci 85]. In addition, it is planning to develop materials (textbooks, lecture outlines, transparency masters, student project materials, and educational software) to support its graduate level curriculum. It plans to produce educational versions of software tools which it will collect in a Showcase Software Engineering Education Laboratory. The SEI also plans to be a focal point for software engineering education by becoming a repository of software engineering education knowledge which will be structured in a public database and by conducting an annual conference on software engineering education [SEI 86]. Through these actions and others, the SEI is attempting to increase the number of software engineering educators and improve the productivity of current practitioners [Gibbs 86].

Professional organizations such as the ACM and IEEE continually offer workshops, seminars, and conferences to keep software practitioners abreast of the state of the practice and sometimes the art. While such activities are not in-depth, they do serve an educational function. For many software engineers, they are the only continuing education activities available. Unlike their Japanese counterparts, U.S. companies tend not to offer mid-career training, and an engineer's knowledge can quickly become dated. Japanese firms rely heavily on in-house training programs for mid-career training [Westney 86].

Some companies have begun to implement (some on a mandatory basis), management development programs. These programs consist of two-day workshops spread over a ten month period and they are aimed at creating desired behavioral change. The company commitment is substantial but so is the outcome to the individual and the corporation [Sobkowiak 86].

IBM, through its Software Engineering Institute, offers classes and laboratories addressing critical areas that concern its employees. The Institute has been successful in improving quality and productivity [Carpenter 85].

IBM offers an in-house, non-degree "undergraduate" computer science program called ULCS (University Level Computer Science) for working IBM programmers, managers, and writers. It is aimed at people without a computer science or college background as well as people with degrees in computer science who want to update their skills. The program offers one-week intensive courses team-taught by computer science faculty from over 50 universities. A Curriculum Review Committee sets a curriculum that is typical of college and university computer science programs and is consistent across IBM [Hickey 86].

Companies such as Xerox, Hewlett-Packard, and IBM have arrangements with universities whereby they share revenues and equipment with the universities in exchange for access to graduates and research resources [Myers 85a].

The Rocky Mountain Institute of Software Engineering (RMISE) is a non-profit educational organization supporting the transfer of modern software engineering technology from the research arena to the broader professional community. It sponsors seminars, workshops and tutorials for software engineering professionals; it provides educators with up to date materials supporting software engineering training and education. Through its selection-oriented education program, it provides an exposure to new techniques and concepts sufficient to impart an appreciation of the technology's cost/benefit and allow well-founded decisions about whether or not to adopt it [Riddle 86].

In addition to educating and providing on-going training to computer professionals, the U.S. must provide accelerated retraining programs for individuals from other disciplines to fill the shortage of computer professionals. In many cases, it is not feasible for these people to enroll in undergraduate or graduate programs; continuing education programs are insufficient for their needs. In these cases, retraining programs are necessary.

The National Computing Center (NCC) in the U. K. has offered, since 1967, a two-year systems analysis training program. About 1,000 people go through each year, and there are now more holders of that qualification than there are graduates in computer science. Another NCC initiative is the Threshold Scheme 42 week training program, 24 weeks of which involve gaining practical experience in industry [Penney 85].

12.3.1.3.5 Summary

While there is certainly a crisis in the education and training of sufficient numbers of computer professionals for the future, it has not gone unnoticed. The NSF, DoD, academia, and industry recognize the problem and are attacking it but with expected decreases in the federal budget the problem may not be adequately addressed over the coming years [Gries 86].

12.3.1.4 Selection

Selection is the process of identifying the best people and matching people to jobs. To date, little research has been focussed on the selection process for software professionals. More research seems to have been undertaken in a related area, predicting performance in computer science education programs. Tools that can be used to select professionals include Assessment Centers and simulations like those used for pilot selection.

The shortage of computer science professors has forced many universities to restrict the number of computer science majors [Mazlack 80]. One recent study (Butcher 85) used high school data and ACT (American College Testing Program) test scores to predict college performance in computer science courses that contained only pre-computer science students. Another recent reference [Klitgaard 85], describes how Harvard selects its students, provides a conceptual framework, and reviews the research on academic prediction and prediction of later life success. Much of what is said has implications for selecting elite SDI personnel. Still other studies have found cognitive profiles to be important in determining success in the computer field [Gordon 84; Sheppard 78].

Studies have found the predictive value of the IBM programmer's aptitude test to be low [Testerman 71; Mazlack 80]. Such tests ignore the job environment in which the employee

must work. One study developed a Computer Personnel Job Model that matched the types of information processing styles to the requirements of the job. Further research is needed to create a multi-dimensional model [Testerman 71].

In addition to the traditional personnel selection tools, such as interviews, reference checks, and competency tests [Goldman 81], Assessment Centers can be excellent predictors of job performance [Kraut 76; Howard 74; Klitgaard 85; Worboys 75]. However, they are costly, and therefore usually used only by large, successful companies.

Competency is not the only consideration, however. Personality, the work environment, the ability to work in an organization, and communication skills are other important factors in selection. Interviews are traditionally used to screen for these factors. However, studies have found a number of problems that occur during interviews which decrease their effectiveness as a selection tool [Carlson 71; Schwab 69; Schneider 76].

Based on "behavioral event interviews" and surveys of its systems designer/programmers, General Electric attempted to find what distinguishes an average worker from a superior worker, what makes a productive programmer, and the types of things for which managers should select and train. It concluded that most superior data processors have a mixture of competencies for success [Zientira 81].

A pharmaceutical company, bank, and insurance company are developing an artificial intelligence package for screening prospective job candidates and recommending development actions for incumbents. The package focuses totally on technical proficiency in a specific programming language. The job candidate is queried by the system; the answers are automatically graded, and the individual's development needs are spelled out. While these efforts are rudimentary, have significant EEOC regulations to address, and are not high company priorities (thus being developed with limited budgets), they offer a candidate methodology to qualify software designers [Sobkowiak 86].

Other professions use other techniques to select people. Pilot and pilot trainee se' have received attention for forty-five years and have well developed and suc. selection methods that are continually being refined. One research project in Britain tes.. the feasibility of using a flight simulator to predict success in flying training. It found that it could accurately predict flying training success, but that the cost was relatively high compared with normal selection techniques [Lidderdale 77]. The Federal Aviation Administration (FAA) uses the Airman Certification Systems Development Methodology as a tool to analyze simulator use in FAA airline transport pilot certification [Gilliom 85].

12.3.1.5 Evaluation, Certification, Testing, Measurement

Evaluation, certification, testing, and measurement are closely tied with selection, education, and individual differences. It is another area sparse in activities, despite the great need. DeNelsky and McKee summarize and evaluate the effectiveness of some of the programmer aptitude tests available in 1974 [DeNelsky 74].

The Institute for Certification of Computer Professionals (ICCP) offers four certificates for computer professionals: (1) Certificate in Computer Programming (CCP), (2) Certificate in Data Processing (CDP), (3) Certified Systems Professionals (CSP), and (4) Associate Computer Professional (ACP) [ICCP 86]. Little is known about the success of these certification programs in predicting success on the job.

Wolfe Personnel Testing and Training Systems, Inc. has offered for a number of years several programmer and analyst aptitude and achievement tests used by a number of

organizations including at least one government agency. Some validation studies have been performed.

The STARS Program, through its Measurement Area, is looking at measuring the processes associated with software development, including productivity. An important aspect to evaluating software personnel is having baselines against which comparisons can be made. The STARS Measurement Area plans to provide these baselines.

The SEI Education Division is planning to develop diagnostic and study materials for practitioners of software engineering who wish to study the SEI curriculum. The tests will identify gaps in knowledge and the study materials will help fill the gaps [SEI 86].

Human Resource Tool Kits, which may or may not be automated, but which expedite, improve upon, and control required and useful human resource activities can be helpful in the evaluation process. One example is a software product called People Manager I, designed by People Sciences, Inc. It enables a manager and subordinates to conduct a performance appraisal discussion by interacting through a PC. The system records negotiated goals, allows for the tracking of accomplishments and guides the manager and subordinate in possible development actions [Sobkowiak 86].

12.3.1.6 Motivation

A large portion of the literature is concerned with the motivation of computer professionals [Couger 80; Fitz-enz 78; Lehman 86; LeDuc 80]. Motivation is correlated with productivity and keyed to the work environment.

Research shows that job satisfaction has been generally on the decline in the United States since 1973 [LeDuc 80]. Computer professionals are no exception. A 1977 study found that the chief characteristic of computer professionals was dissatisfaction. Other studies have found that programmers have different internal motivations than other groups. Wages, interesting work, promotion, and growth are important to programmers [LeDuc 80]. Couger and Zawacki, in their 1980 book *Motivating and Managing Computer Personnel*, agree that DP professionals have some unique differences from the general population, but that a good job match is possible and supervisory feedback to employees can be improved. It is also generally agreed that external motivation is often manipulative and not very successful. More successful is internal motivation [LeDuc 80].

Recent research explores the idea of a "pseudo-profit" incentive, or awards, program for development of a more competitive environment at "not-for-profit" R&D laboratories with the ultimate objective of improving productivity [Lehman 86]. Such research may be important to the SDI if it relies heavily on a non-profit R&D environment (e.g., SDI Institute).

12.3.2 Organizations

This section addresses the organization-related topics of teams, projects, organizations, and interorganizational relations.

12.3.2.1 Teams, Group Dynamics, Quality Circles

There is growing interest in the use of the systems development teams for improving the effectiveness and productivity of systems development efforts. As a result of early experiences with team approaches, such as the use of chief programmer teams in the late 1960s, it has become apparent that many factors, other than technical ones, have both

positive and negative impacts on team development. While the understanding of these factors is incomplete, considerable progress is being made [Semprevivo 80].

In the early 1970s, Mills published an article describing a project that not only succeeded but also claimed to have achieved a high productivity rate. He talked about teams, motivation, relationships of people on the team, and the right people for the job [Mills 71]. Since then, the Japanese have demonstrated significant increases in productivity and quality by using teams and quality circles [Davis 81]. Others have studied MIS Project Teams to learn more about the ways successful teams work [White 84; White 84a; Kaiser 82].

A team approach succeeds because it increases the commitment of people to work together effectively. By building teams with key data processing managers, professionals, and users, project cycle times and turnover can be substantially reduced [Stroh 81].

A survey of the literature reveals that MIS Project Teams have been the subject of some research. Several conclusions have been drawn that have implications for other system project teams. One is that different combinations of perceptual styles affect team performance. Project teams exhibiting diversity in perceptual styles were rated as the most successful. Critical long-term decision solutions are more fully developed when all perceptual styles are represented on the decision support team [White 84; White 84a].

Experience with four operating system development projects indicates that team productivity can be substantially increased by the intervention of a skilled communication facilitator. This conclusion is tentative since many factors were not sufficiently controlled. The potential productivity gains, however, suggest further effort to develop reliable communication skills training methods and techniques for evaluating these methods [Unger 77].

Basili and Reiter evaluated different team structures [Basili 79]. They compared individuals programming alone to three-person teams and to teams operating in a disciplined manner. Disciplined teams were found to produce more reliable programs than the other two organizations, whereas undisciplined teams appeared to be no more effective than individual programmers on most measures.

Another study indicates that the situational structure determines the overall effectiveness of the team composition. It offers evidence that heterogeneity in group composition is best for solving complex problems, whereas homogeneity is best for solving structured, less complex problems. It also suggests that one team might not be appropriate for all stages of a project. As the nature of the tasks involved in the project changes, it could be that the optimum team composition changes also [White 84b].

Another factor that distinguishes effective high technology project teams from ineffective ones is the ability to learn in purposeful and cumulative ways. Research in this area suggests that learning may be the most important characteristic of high performance teams that deal with complex, technological challenges. Uncertainty remains about how to transfer learning from one team to another [Wilemon 84].

Because so little work has been done on the subject of team learning in high technology firms, several avenues of future research exist. These include identifying (1) the array of learning types that are most effective at individual stages in the development cycle; (2) the kinds of interventions that can be made to help existing teams be more effective learners and to facilitate learning transfer between teams; (3) the relationship between group size and team learning; (4) communication types and patterns that are most useful to stimulate and transfer learning; (5) informal group activities that are the most powerful conduits or stimuli

for team learning; and (6) how conditions that facilitate individual learning might accelerate or hinder team learning [Wilemon 84].

Research also needs to focus on the information typically exchanged among people in team design. Once captured, these items should be ranked in order of importance. The MCC has recognized this deficiency and is working toward correcting it [Myers 85]. In December 1986, the MCC will sponsor a Conference on Computer-Supported Cooperative Work. The conference will take an interdisciplinary look at computer-supported cooperative work from technological, sociological, organizational, cognitive, and task domain points of view. Teamwork and groups will be heavily emphasized [MCC 86].

Another suggestion for future research is the effect of personality differences, or absence of differences, on the management of project teams and the designs they create [Kaiser 82]. Much further research must be aimed at strategically assembling the most productive teams for any number of organizational situations [White 84b].

12.3.2.2 Projects

The cost, quality, and performance of a system are directly related to the project that produced it. Components that contribute to project success and failure include management, communications, size, structure, attitude, and technology. Considerable research has been done on software projects, primarily for the purpose of creating databases and models to predict productivity and quality on future projects. In compiling this data, people have made several observations about factors affecting project success and failure.

Thomsett developed the following guidelines for successful project management from his observations of systems in many different sizes and styles of organizations:

"a. The first step in achieving workable project management is to correctly draw the boundaries between the various people systems - users, management, and team - involved in the project and around the business system being developed.

b. The second step is to ensure that project management systems act as information systems that enable communication between control and process systems.

c. Third is for organizations to allow teams to evolve so as to allow team members to do what they do best, that is, to let teams evolve from their natural structure. Teams must be given adequate time and resources for learning and growth, and then be left alone to get on with the job of building systems.

d. With the use of technical reviews and structured walkthroughs as reviewing processes, a systems development methodology approach must be implemented for project information systems.

e. Teams must use the functions of work definition, planning, tracking, and reviewing to feed information back to management on project status.

f. Finally, management and teams accepting joint responsibility for the project must recognize the functions they each are best equipped to do and must give each other the respect (authority, legitimacy) and support (resources, information) necessary to perform these functions" [Thomsett 80, p. 85].

The quality and technical competence of managers can have significant impact. Insisting on the technical knowledgeability of all of management had substantial beneficial effects in the IBM Federal Systems Division (even though it caused a number to transfer out) [Mills 86].

Project size also has an effect on management, productivity, and quality. Brooks, in his book *The Mythical Man-Month*, about his experiences developing the IBM OS/360, was one of the first to make observations about systems development projects. One general observation was that large programming projects suffer management problems different in kind from small ones, due to division of labor [Brooks 75].

Others have expanded on this observation. Putnam, in compiling his database of project case studies in order to assess the impact of methodologies on software productivity, has found that the small team approach produces reliable systems. Also, as project size increases, schedule and time required increase exponentially, as do software errors [Putnam 85].

Communication is another project success factor. Schedule compression increases human communications noise, which introduces ambiguities. These ambiguities increase errors and lower productivity [Putnam 85]. Research on the impact of computer-based communications on group performance [Hiltz 78; Murrel 83] has found that groups produce decisions that are superior to average individual solutions. What is needed is research to evaluate the impact of various computer-based communication design features [Murrel 83].

Organizational structure also has implications for project success. An organizational structure that is leveled and dynamic will have leveled, dynamic project management systems. Organizations with such structures have successful project histories [Thomsett 80].

With respect to management, DeMarco has observed that characteristics of the workplace and organization seem to explain a significant part of the wide variation in programmer productivity. Specifically, environmental factors, such as noise, privacy, and interruptability, may be keys to substantial productivity improvement [DeMarco 85].

Examples abound of situations where attitude affects project success [Peters 82]. A prevailing attitude for excellence should be established and/or renewed with the principal participants of a system project at the beginning of the project [Mingione 86]. One study has even revealed a very strong relationship between a programmer's attitude toward his supervisor and programming productivity [Jeffery 85].

Technology's effect on project success, however, should not be ignored. One question is how projects organize and are organized by their technologies. Several relationships are important: (1) that of technology to the production and distribution of relevant information; (2) that of technology to the specialization among participants; and (3) that of conventionally defined standard practices to the use of technology. Human-machine interaction studies at Xerox's Palo Alto Research Center (PARC) have focussed on how technology affects and is affected by the real-time organization of events [Suchman 85].

Humanistics is a phrase that a number of companies are using to label the work of putting human resource skills into a systems organization in order to make the systems more effective. The eight leverage points (that are part of most system development efforts) are business requirements/specifications, screen design and systems flow, documentation/training, acceptance test/pilot, preparation of end-user community, introduction/training, and post-installation assessment. At each point, the human resource

specialist is introducing ways of thinking, approaches and methodologies that significantly increase the usability of the system being developed. This work will eventually lead to a set of guidelines that will place relevant human resource considerations at many points in the standard software development life cycle. It will be a melting of hard and soft issues of technology development leading to improved system usage [Sobkowiak 86].

The SEI recognizes that management of large software development projects needs significant improvement. One principal reason management efforts do not measure up may be that the model on which the principles and systems are based is seriously flawed. Several other aspects needing attention include:

a. Organizational effectiveness would be enhanced by synthesis of information for management and software development;

b. Incentives and risk reduction measures may need to be instituted before more productive work techniques pay off; and

c. Techniques for estimating software costs should be improved.

Furthermore, the lack of organization-level performance measures of the application of new technology makes it frustratingly difficult to assess the effects of technology insertion [SEI 86, p. 31].

Other software project research of interest to the SDI effort include:

a. TRW productivity studies

b. IBM Santa Clara research on rooms and physical facilities

c. Software Engineering Laboratory (SEL) [Basili 77: Basili 85]

d. NRL (Naval Research Laboratory) A-7E Software Cost Reduction Project [Clements 85]

12.3.2.3 Organizations

The SDI effort will need organizations working for it that are flexible, pro-active, innovative, and fault-tolerant. The SDIO will need, therefore, to be able to recognize such organizations and, when necessary, take actions to effect changes in organizations.

Much research has been undertaken to describe organizational structures [Comier 82], processes, and behavior. The research has examined such subtopics as organizational politics [Robey 84], power [Salancik 77; Tushman 80], interrelationships, roles, life cycles, and cultures.

Organizational behavior is a function of the interrelationships between work requirements, individual characteristics and skills, formal organizational arrangements, and the organization's informal organization [Tushman 80]. Effective interaction of entrepreneurial, managerial, and technological roles within a firm is essential to producing innovations [Maidique 80]. A firm's life-cycle is also important since firms act differently as they mature [Adizers 78].

Effective Research, Development and Engineering organizations, needed by the SDI effort, require motivated scientists and engineers and an active and diverse informal organization.

The formal structure and processes must facilitate and nourish idea generation and creative problem solving. Attention must be paid to leadership and supervisory styles [Farris 73] and informal communication networks. Substantial task differences exist within R&D laboratories, and these task differences call for systematically different communication patterns, structures and supervisory styles [Tushman 79].

Recent efforts [Peters 82] argue that organizational culture is the key to organizational excellence. Schein hypothesizes a formal definition of organizational culture but believes that it is necessary to study a large number of organizations using complex interviews, observation, and a joint-inquiry approach to determine the utility of the concept of organizational culture and to relate cultural variables to other variables such as strategy, organizational structure, and ultimately, organizational effectiveness [Schein 84, p.14]. The Japanese and others have shown that quality can be managed within an organization [Crosby 79].

If the SDI effort is to manage both evolutionary and revolutionary change, it will need to understand how best to manage radical technological change. Maidique suggests developing an environment where risk taking by executive champions and product champions will lead to new ventures and products. The literature on entrepreneurship is principally concerned with static behavior. To date, little attention has been paid to the evolution of the entrepreneurial role. Substantial literature on corporate development does exist, but it only makes reference to the entrepreneur and the evolution of his role [Maidique 80].

The SEI plans to conduct a longitudinal study over time to understand and develop strategic managerial and organizational solutions to technology transition barriers. An important precursor to a longitudinal assessment is determining the critical organizational and managerial issues of technology transition such as:

a. Motivation and reward systems including increasing incentives to change software engineering practices of individuals, project teams, and divisions.

b. Authority and decision-making structures and practices.

c. Identifying and using organization-level performance measures for process and outcomes of complex change and advantages of applications of new technology.

d. Goal conflict avoidance or resolution [SEI 86, p. 99].

Organizations that will be developing the SDI system will have a high need to sustain creative talent. They must be able to identify, encourage, and cultivate such talent. Sinetar offers some basic principles that can help business use the ideas and energy of employees with higher probability of success [Sinetar 85]. A related little used organizational practice is human resource accounting [Flamholtz 85].

Strategic planning techniques are being used increasingly by companies to guide new product development [Crawford 80]. Organizations developing the SDI system, as well as the SDIO, need to follow their lead to manage innovation to every extent possible. This includes developing sets of target areas, objectives, and a program [Crawford 80].

Management must build a capability for responsiveness into the overall organizational design and managerial practices to make it genuinely responsive to change. Recognition of this has led to the concept of organization development (OD) [French 78; Starbuck 71;

Patten 81]. The Aeronautical Systems Division (ASD) Management Development Team (MDT) implements the OD approach. It performs management consultation services for ASD managers. It uses survey-feedback, team building, life/career planning, conflict resolution and other organizational activities to help increase the effectiveness of the ASD workforce [Robinson 80].

Organizations also need to be concerned with effective intraorganizational communications. Recent research has focussed on the design of organizational interfaces for groups of users within organizations. Such interfaces include the parts of a computer system that connect human users to one another and to the computer resources [Malone 85]. The extensive use of computers by SDI developers suggests that these interfaces will be important to them.

12.3.2.4 Interorganization Relations

The SDI effort is a large and complex undertaking involving many organizations. Interorganizational relations are therefore quite important. Players include government agencies, contractors, the National Test Bed, FFRDCs (such as the proposed SDI Institute), and others. Each of these players is made up of a number of subcomponents that need to interact. For example, interservice interaction will be important since the Army will assume responsibility for the terminal phase and the Air Force for boost and intermediate phases. In addition, each player may assume more than one role: user, developer, or supporter, for example. The quality of these relations will have an impact on the success of the overall SDI effort. Contractual and other kinds of organizational relationships will be revisited under the area of technology transition, since part of the problem of moving state-of-the-art technologies into practice is effective interorganizational relationships.

The recent Space Shuttle disaster has underscored the importance of interorganizational communications and the potential management problems arising from large complex undertakings involving many organizations. The recent Goldwater/Nunn report to Congress points to structural deficiencies within the DoD. Among other problems, the report cites the destructive effects of inter-Service rivalry and log-rolling [Goldwater 85].

Another thought is that these organizations along with organizations like the SEI and Software Productivity Consortium will be competing for scarce manpower resources. As 12.3.1.3 shows, large shortages in computer personnel are expected in the future. These organizations will have to work together to make the most efficient use of the scarce people resources. Communication and resource sharing will be essential to the success of these organizations. Industrial consortia, such as the MCC, provide a model for pooling resources and loaning personnel. The SDIO may need to set up similar arrangements.

The DoD acquisition and contracting processes need to be simplified and restructured to encourage the timely delivery of innovative products necessary to the SDI effort's success. Both the Eastport Report [Cohen 85, pp. 65-67] and a Senate Staff Report on the SDI system [Waller 86, pp. 58-60] point to these problems. The STARS Program has recently adopted a theme of marketplace stimulation rather than technology development. The desired results are marketplaces in software tools and methods and in end-use MCCR software that will benefit the SDI effort. The key to the new marketplaces will be the technical compatibility provided by interface standards [Kramer 85]. This fits well with the SDI system's requirement for an open architecture.

It has been recommended that the STARS Business Practices area broaden its focus to address an understanding of the DoD business environment as well as tools to support management [Kramer 85]. Recent reorganization and redirection of the STARS Program eliminated the Business Practices area, however. The SEI is also concerned with legal

issues related to the business environment such as software licensing and rights in data [SEI 86, Gibbs 86; Samuelson 86; Samuelson 86a; Samuelson 86b; Samuelson 86c; Samuelson 86d] and is preparing a method for qualifying contractors. Others have noted the inefficiencies of the DoD business environment [Packard 86; RDTWG 84; McNaugher 86] and advocate streamlining the way that the DoD acquires products and services to lower costs and improve quality [Hicks 86; Taft 86; Wade 86].

12.4 Recommendations

How can the SDIO best ensure the high levels of quality and performance required for the people and organizations involved in developing and supporting SDI software? The government has four general mechanisms available

- Government employee- and organization-related mechanisms
- Technology R&D
- Acquisition-related mechanisms
- Inter-personal relationships

The first can help the situation within the government and interorganizational relationships involving the government. The last, personal relationships between government and contractor personnel, is extremely important but cannot bear the burden of "insuring high levels of quality and performance". This leaves technology R&D and acquisition.

For the contractors involved in software development and support -- the largest and most important group -- the essential mechanisms are those related to acquisition. Technology R&D is to insure that concepts or technology exist for the contractors to use or build upon. Because people and organization are areas intimately involved in the contractor(s) internal processes, these same contractor(s) appear to be the best path through which to conduct much of the related R&D. Only R&D that is related to the government's people and organizations, very long range, aimed at establishing independent confidence or validity, or less expensively available elsewhere should be performed elsewhere.

> Recommendation 1: To the maximum extent appropriate, People and Organization R&D should be conducted through the contractor(s) performing the BM/C3 software effort.

The involvement of multiple software contractors complicates the situation particularly if multiple prime contractors are used. The key aspect is the need for the eventual performer(s) to learn new knowledge and skills, adapt the technology, and improve during the early analyses and design, prototyping, and experiments and tests so they may develop into the people and organizations required to do the job.

The contractor(s) involved need to

- Plan for human resource needs
- Select, educate, evaluate, and retain excellent people
- Structure the software-related organization to be near the state of the art, fault-tolerant and reliable, pro-active, and continually improving.
- Match people to places in the structure
- Train people in their duties and technology
- Develop the organizations (OD) including team, project, intra-contractor relationship, and inter-organization relationship building

- Measure, learn, validate, and improve the people and organization including structure, processes, activities, and technology.

To do this the government should make the contractor responsible for the appropriate necessary people and organization R&D and for achievement.

The people-related R&D needed includes a number of aspects. Because the SDI software effort will require large numbers of expert software personnel, SDI contractors must be able to select and educate for excellence. Understanding individual differences and their effects on performance will help in this process as well as in the automation of the programming function. However, selecting qualified computer professionals is a vital aspect of SDI work force creation where most of the research is dated and not specifically targeted at selecting computer professionals. The SDI contractors will also be concerned with evaluating personnel. Based on the STARS Program and others' work on measurement, they should find ways to measure people effectively, collect this measurement data, learn from it, and insert this knowledge into organizations working for the SDIO.

Recommendation 2: The SDI BM/C3 contractor(s) should be responsible for research that identifies characteristics of excellent (and super) programmers and other software-related personnel, and development that results in validated means of selection including empirical studies of certification programs such as the ICCP's to find out how effective such certification is in predicting job success.

Education is also key to the highly qualified and specialized work force required by the SDI effort. SDI planners should monitor Ada, SEI, and other education activities and initiate new educational developments for SDI system requirements that impose education needs widely varying from typical MCCR needs. For example, there may be a requirement to fund courseware development to communicate applications concepts to software designers.

Recommendation 3: The SDIO should fund educational upgrading of SDI software personnel as well as help universities involved. It should take measures to assure that contractors educate and upgrade SDI workers.

Universities near contractor or government sites may need funds to rapidly expand and upgrade graduate software engineering education to meet SDI needs.

Organizational R&D needed by contractor(s) also includes several areas. The requirements for organizations to be near the state-of-the-art, fault tolerant, etc. implies some R&D on the less well understood aspects of these requirements. Project and team structure and placing people in the structure have been shown to affect productivity and quality.

Recommendation 4: The SDI BM/C3 contractor(s) should be responsible for R&D that supports organizational design decisions, organizational and team learning, and computer-mediated work and MMI for software workers as well as for monitoring software, software cost/size/quality data collection and models, and other research to improve software productivity and quality.

The requirements for contractor(s) to not only be responsible for people-and-organization-related R&D but also for achievement in this area demands the government develop new acquisition practices and content that will insure this achievement and encourage supereminent performance. While more work will be needed to develop these new practices and content, Figure 2 sketches a scenario that might underlie such an approach. Even though some items on the lists in Figure 2 may not survive further consideration, clearly a number of items and requirements can be included within the acquisition framework.

The "Early Activities" must be a combination of the best known today with systematic variation so comparison and learning can be speeded. Some of the requirements for personnel selection, such as the Ada knowledge requirement, might be implemented as something that must be met in a fairly short period after assignment to an SDI project rather than before assignment.

The "Early Activities" and those over the next several years are aimed at meeting the goals of "Status after 3-4 Years". A key part of this process is the prototyping of experimental versions and the learning and improvement derived.

Recommendation 5: The SDIO should fund studies of the acquisition process to find mechanisms for assuring that organizations working for it meet the needed people and organization-related requirements.

Much work regarding people and organizations is taking place outside software engineering. Awareness and use of these results are important.

Recommendation 6: To increase awareness within the SDI software development community of the results of research in the People and Organizations area, SDIO should fund an organization to monitor, evaluate, and report on significant developments in this area.

Possibly this responsibility should be assigned to the contractor(s), but possibly it should be done by another organization that is already performing it for other purposes.

Finally, SDIO should not forget its own personnel and those of its close assistants such as FFRDCs and SETA contractors.

Recommendation 7: SDIO and other government personnel involved with SDI software as well as assisting personnel in FFRDCs and SETAs should be of outstanding quality or be upgraded or replaced. Continual upgrading is required for just staying current and should be funded by SDIO.

ORGANIZATIONAL QUALIFICATION REVIEW:

Based on SEI/ESD (Electronic Systems Division) effort but with higher standard

PROPOSAL REQUIRED TO INCLUDE:

Human Resources Plan
Key Personnel
Detailed plans for people and organization structure, activities, and systematic improvement

EVALUATION CRITERIA:

Include appropriate content and weight to people and organization aspects

EARLY ACTIVITIES:

Individual Selection:	Contractor certifies transferred in employees in top fraction of its employees Minimum IQ requirements Minimum math competency requirement Minimum Ada knowledge requirement Use university selectivity and GPA Use existing certification tests Use assessment center with best practices *Also use a number of measurements for selection research purposes* Government blackball capability Expectations Signoff by employee Managers technically competent and managerially competent
Team:	Composition using best practices (with variation for R&D purposes) Team building using best practices (with variation for R&D purposes)
Project:	Fault tolerant structure Best practices and state of the art (with variation for R&D purposes)
Organization:	Independence of Safety, Reliability, QA; Organizational Development Sub-organization; Technology Transition Sub-organization
Interorganization:	Formal High-visibility

Table 2. Rough Partial Acquisition Imposed Scenario

ACTIVITIES OVER 3-4 YEAR PERIOD:

Education	R&D	Eliminate worst people
Training	Measurement	Select better and better personnel
Evaluation	Prototyping	
On-the-job Instruction	Organizational Development	Issue incentive payments
Experiments/Trials	Human Resource Plans (Annual)	

STATUS AFTER 3-4 YEARS:

Validated selection and evaluation methods
Means exist to compare staff with industry and all staff in top 10%
Almost all staff have equivalent of Masters in Software Engineering or better
Key persons clearly in top 10 persons in world in their subarea
Proven high performance team, project, organizational, and interorganizational methods
Staff has application-oriented knowledge
If certification proves significant, then almost all certified
Knowledge of error patterns of persons and methods

Table 2. Rough Partial Acquisition Imposed Scenario (Continued)

12.5 References

[ACM 81] ACM Curriculum Committee on Computer Science. "Recommendations for Master's Level Programs in Computer Science *Communication of the Association for Computing Machinery* 24, 3 (March 1981), pp. 115-123.

[Adelson 84] Adelson, B., "When Novices Surpass Experts: The Difficulty of a Task May Increase with Expertise," *Journal of Experimental Psychology, Learning, Memory, and Cognition* 10, 3 (1984), pp. 483-495.

[Adelson 85] Adelson, B., "Comparing Natural and Abstract Concepts: A Case Study from Computer Science," *Cognitive Science* 9 (1985), pp. 417-430.

[Adelson 84] Adelson, B., and Soloway, E., *A Model of Software Design*, Yale University, YALEU/CSD/RR 342, (October 1984).

[Adelson 85] Adelson, B., and Soloway, E., "The Role of Domain Experience in Software Design," *IEEE Transactions on Software Engineering* SE-11, 11 (November 1985), pp. 1351-1360.

[Adizes 78] Adizes, Ichak, "Organizational Passages - Diagnosing and Treating Lifecycle Problems of Organizations," *Organizational Dynamics*, (Summer 1978).

[Bailey 86] Bailey, John, McDonald, Catherine, and Redwine, Samuel T., Jr., *Ada Course Evaluation Criteria*, IDA Central Research Project 05-81-42, IDA Memorandum Report M-170, Alexandria, VA (May 1986).

[Barbacci 85] Barbacci, Mario, Habermann, A. Nico, and Shaw, Mary, "The Software Engineering Institute: Bridging Practice and Potential," *IEEE Software* (November 1985), pp. 4-21.

[Basili 79] Basili, V.R. and Reiter, R.W., "An Investigation of Human Factors in Software Development," *IEEE Computer* (December 1979), pp. 21-38.

[Basili 85] Basili, Victor R. and Selby, Richard W., Jr., "Calculation and Use of an Environment's Characteristic Software Metric Set," *Proceedings of the Eighth International Conference on Software Engineering*, London (August 28-30, 1985), pp.386-391.

[Basili 77] Basili, Victor R. and Zelkowitz, Marvin V., "The Zelkowitz Software Engineering Laboratory: Objectives," *In Proceedings of the 15th Annual Computer Personnel Research Conference* (August 18-19, 1977), pp. 256-269.

[Boehm 75] Boehm, B., "The High Cost of Software," *Practical Strategies for Developing Large Software Systems*, Addison-Wesley, (1975).

[Boehm 81] Boehm, B., *Software Engineering Economics*, Prentice-Hall, (1981).

[Boehm 86] Boehm, B., "SDI Software Development, SDI Architecture Study Briefing," *TRW-RCA-MDAC-HB*, (April 16-17, 1986).

[Brooks 75] Brooks, Frederick P., Jr., *The Mythical Man-Month: Essays on Software Engineering*, Addison-Wesley Publishing Company, 1975, reprinted 1982.

[Butcher 85] Butcher, D.F. and Muth, W.A., "Predicting Muth Performance in an Introductory Computer Science Course," *Communications of the ACM* 28, 3 (March 1985), pp. 263-268.

[Carlson 71] Carlson, R.E., et al.., "Improvements in the Selection Interview," *Personnel Journal* 50 (1971), pp. 268-275, 317.

[Carpenter 85] Carpenter, M.B. and Hallman, H.K., "Quality and Emphasis at IBM's Software Engineering Institute," *IBM Systems Journal* 24, 2 (1985), pp. 121-133.

[Clements 85] Clements, P., "The Software Cost Reduction Methodology," *National Conference and Workshop on Methodologies and Tools for Real-Time Systems*, Washington, DC, (October 28-November 1, 1985).

[Cohen 85] Cohen, D. et al., Eastport Study Group (December 1985).

[Comier 82] Comier, Antoine Gerard, *A Quantitative Analysis of the Effect of Organizational Structure on Software Engineering Management*, Polytechnic Institute of New York sponsored by the Office of Naval Research, ADA129620, (May 1982).

[Commerce 84] Department of Commerce, Assistant Secretary for Trade Development, Office of Computers and Business Equipment, *A Competitive Assessment of the U.S. Software Industry*, (December 1984).

[Couger 80] Couger, J. Daniel and Zawacki, Robert A., *Motivating and Managing Computer Personnel*, Wiley, 1980.

[Crawford 80] Crawford, C. Merle, "Defining the Charter for Product Innovation," *Sloan Management Review* (Fall 1980), pp. 3-12.

[Crosby 79] Crosby, Philip B., *Quality is Free: The Art of Making Quality Certain*, Signet, 1979.

[Curtis 83] Curtis, Bill, "Fifteen Years of Psychology in Software Engineering: Individual Differences and Cognitive Science," *Proceeding of the 7th International Conference on Software Engineering*, IEEE, 1984, pp. 97-106.

[Curtis 85] Curtis, Bill (ed..), *Tutorial: Human Factors in Software Development*, IEEE Computer Society, 2nd Edition 1985

[Curtis 86a] Telephone conversation with Bill Curtis, MCC, (512) 343-0860, (April 1986).

[Curits 86b] Curtis, Bill, "By the Way, Did Anyone Study Real Programmers?", *Empirical Studies of Programmers*, Saltway and Iyengar (eds.), Ablex, 1986, pp. 256-62.

[Curtis 86c] Curtis, Bill, Presentation at Third International Workshop on the Software Process, Breckenridge, Colorado, 19 Nov 1986.

[Davis 81] Davis, George et al., "Improving Productivity," *Circuits Manufacturing* 21, 2 (February 1981), pp. 45-51.

[DeMarco 85] DeMarco, Tom and Lister, Tim, "Programmer and Performance and the Effects of the Workplace," *Eighth International Conference on Software Engineering* , London, (August 1985), pp. 268-272.

[DeNelsky 74] DeNelsky, G.Y. and McKee, M.G., "Prediction of Computer Programmer Training and Performance using the AABP Test," *Personnel Psychology* 27 (1974), pp. 129-137.

[DoD 82] Department of Defense, *Strategy for a DoD Software Intiative*, Volume II: Appendices, (October 1, 1982).

[Enger 81] Enger, Norman, "Communications Skills Required by Computer Professionals," *Proceedings of the 18th Annual Computer Personnel Research Conference*, (June 3-5, 1981).

[Farris 73] Farris, George F., "The Technical Supervisor: Beyond the Peter Principle," *Technology Review*, 75, 5, (April 1973).

[Fitz-enz 78] Fitz-enz, Jac, "Who is the DP Professional?," *Datamation*, (September 1978), pp. 125-128.

[Flamholtz 85] Flamholtz, Eric G., *Human Resource Accounting*, Second Edition, 1985.

[Ford 86] Ford, Gary, *Educational Needs of the Software Community*, Software Engineering Institute Milestone Report, SEI-86-MR1, (January 31, 1986).

[Ford 86a] Ford, Gary, Senior Computer Scientist, Software Engineering Institute, Pittsburgh, PA, personal communication, (May 1986).

[Frank 79] Frank, "The New Software Economics," *Computerworld*, 1979.

[French 78] French, Wendall L. and Bell, Cecil H., Jr., *Organization Development*, 2d ed., Prentice-Hall, 1978.

[Gibbs 86] Gibbs, Norman E., Director for Education, Software Engineering Institute, Pittsburgh, PA, personal communication, (May 1986).

[Gibbs 86a] Gibbs, Norman E., and Tucker, Allen B., "A Model Curriculum for a Liberal Arts Degree in Computer Science," *Communications of the ACM* (March 1986), pp. 202-210.

[Gilliom 85] Gilliom, D.C. et al., *A Systematic Determination of Skill and Simulator Requirements for Airline Pilot Certification*, Planning Systems International, Inc. under contract to Department of Transportation, PB 85 197275, (March 1985).

[Glass 80] Glass, Robert L. "The Importance of the Individual," *ACM SIGSOFT, Software Engineering Notes* 5, 3 (July 1980), pp. 48-50.

[Goldman 81] Goldman, "The New Competency Test: Matching the Right People to the Right Job," *Psychology Today* (January 1981), pp. 34ff.

[Goldwater 85] Goldwater, Barry et al.., *Defense Organization: The Need for Change: Staff Report to the Committee on Armed Services United States Senate*, (October 16, 1985).

[Gordon 85] Gordon, Alan, "Highly Qualified Manpower for Information Technology," *Computers in Personnel. Today's Decisions - Tomorrow's Opportunities* , (July 9-11, 1985), London, England, pp. 163-171.

[Gordon 84] Gordon, H.W. and Kronz, K., *Cognitive Asymmetry and Occupation: Computer Programmers, Students, and Bank Personnel*, Western Psychiatric Institute and Clinic, University of Pittsburgh, Technical Report 09-84-03, Sponsored by the Office of Naval Research (September 1984).

[Gries 86] Gries, David, R. Miller, R. Ritchie, and P. Young, "Imbalance Between Growth and Funding in Academic Computing Science: Two Trends Colliding," *Communications of the ACM* (September 1986), pp. 870-876.

[Gugerty 86] Gugerty, Leo and Olson, Gary M., "Debugging by Skilled and Novice Programmers," *CHI'86 Proceedings* (April 1986), pp. 171-174.

[Hanson 85] Hanson, Stephen, Joes Rosinski, Rosinski, Richard R., "Programmer Perceptions of Productivity and Programming Tools," *Communications of the ACM* (February 1985), pp.180-189.

[Hickey 86] Hickey, Paul, IBM Austin, (512) 823-2442.

[Hicks 86] Hicks, Donald A., "Streamlining Initiatives: They've Been Around a Long Time," *Program Manager* XV, 2 (March-April 1986), pp. 25-26.

[Hiltz 78] Hiltz, Roxanne and Turoff, Murray, *The Network Nation: Human Communication Through Computer,* Addison-Wesley, 1978.

[Howard 74] Howard, N., "An Assessment of Assessment Centers," *Academy of Management Journal* 17, 1 (1974), pp. 115-134.

[ICCP 86] Institute for Certification of Computer Professionals (ICCP), *1986 ICCP Programs*, brochure, 1986.

[Jeffery 85] Jeffery, D.R. and Lawrence, M.J., "Managing Programming Productivity," *Journal of Systems and Software* 5 (1985), pp. 49-58.

[Kaiser 85] Kaiser, Kate M., "The Relationship of Cognitive Style to the Derivation of Information Requirements," *Computer Personnel* 10, 2 (December 1985), pp. 2-12.

[Kaiser 82] Kaiser, Kate M. and Bostrom, Robert P., "Personality Characteristics of MIS Project Teams: An Empirical Study and Action-Research Design," *MIS Quarterly* (December 1982), pp. 43-60.

[Kant 85] Kant, E., "Understanding and Automating Algorithm Design," *IEEE Transactions on Software Engineering* SE-11, 11 (November 1985), pp. 1361-1374.

[Klitgaard 85] Klitgaard, Robert E., *Choosing Elites*, Basic Books, 1985.

[Koffman 85] Koffman, Elliot B., Stemple, David, and Wardle, Caroline E., Recommended Curriculum for CS2, 1984, A Report of the ACM Curriculum Task Force for CS2,"

Communications of the ACM 28, 8 (August 1985), pp. 815-818.

[Kramer 85] Kramer, John et al.., *An Assessment of the STARS Program September-October 1985, Institute for Defense Analyses*, IDA Memorandum Report M-137, (December 1985).

[Kraut 76] Kraut, Alan, "Management Assessment in International Organizations," *Industrial Relations* (1976), pp. 172-182.

[Lammers 86] Lammers, Susan, *Programmers at Work: Interviews*, Microsoft Press, (1986).

[Laughery 85] Laughery, K.R., Jr., Laughery, K.R., Sr., "Human Factors in Software Engineering, A Review of the Literature," *Journal of Systems Software* 5, 1 (February 1985), pp. 3-14.

[LeDuc 10] LeDuc, A.L., Jr., "Motivation of Programmers," *Data Base* 11, 4 (Summer 1980), pp. 4-12.

[Lehman 86] Lehman, David H., "Improving Employee Productivity through Incentives," *Journal of Systems Management* (March 1986), pp. 14-20.

[Licker 85] Licker, Paul S., "Importing Theory 'Z' to the Software Shop: Cultural Technology Transfer," *Computer Personnel* 10, 1 (January 1985), pp. 8-14.

[Lidderdale 77] Lidderdale, I.G. and Bennett, L.V., "A Cost-Benefit Analysis of a Simulator Test for Pilot Selection in the Royal Air Force," *Symposium on Human Operators and Simulation*, (March 29-31, 1977), Loughborough, Leics, 1977, pp. 25-33.

[McNaugher 86] McNaugher, Thomas L., "Buying Weapons: Bleak Prospects for Real Reform," *The Brookings Review, Summer 1986*, pp.11-16.

[Maidique 80] Maidique, Modesto, "Entrepreneurs, Champions, and Technological Innovation," *Sloan Management Review*, (Winter 1980).

[Malone 85] Malone, Thomas W., "Designing Organizational Interfaces," *CHI'85 Proceedings* (April 1985).

[Martin 83] Martin, E.W., "Strategy for a DoD Software Initiative," *Computer* (March 1983), p. 53.

[Mazlack 80] Mazlack, Lawrence J., "Identifying Potential to Acquire Programming Skill," *Communications of the ACM* 23, 1 (January 1980), pp. 14-17.

[MCC 86] Microelectronics and Computer Technology Corporation, Software Technology Program, "Call for Papers: Conference on Computer-Supported Cooperative Work, December 3-5, 1986, Austin, TX," brochure, for more information call Barbara Smith at (512) 834-3336.

[Mills 71] Mills, Harlen D., "Chief Programmer Teams, Principles and Procedures," IBM Federal Systems Division Report FSC 71-5108, Gaithersburg, MD, (1971).

[Mills 86] Conversation with Harlan Mills at Eastport Group meeting August 1986.

[Mingione 86] Mingione, Al, "Search for Excellence within a Systems Development Project," *Journal of Systems Management* (March 1986), pp. 31-34.

[Molzberger 83] Molzberger, Peter, "Aesthetics and Programming," *CHI'83 Proceedings* (December 1983), pp. 247-250.

[Murrel 83] Murrel, Sharon, "Computer Communication System Design Affects Group Decision Making," *CHI'83 Proceedings*, (December 1983), pp. 63-67.

[Myers 78] Myers, Glenford, "A Controlled Experiment in Program Testing and Code Walkthroughs/Inspections," *Communications of the ACM* (September 1978).

[Myers 85] Myers, Ware, "MCC: Planning the Revolution in Software," *IEEE Software* (November 1985), pp.68-73.

[Myers 85a] Myers, Ware, "The Crisis in Higher Education: Solutions," *IEEE Computer* (July 1985), pp. 69-79.

[Nash 85] Nash, Michael, *Making People Productive*, Jossey-Bass, 1985.

[Nosek 86] Nosek, John, Temple University Department of Computer and Information Sciences, (215) 787-8450, personal communication (May 1986).

[NSF 84] National Science Foundation, *Projected Response of the Science, Engineering, and Technical Labor Market to Defense and Non-Defense Needs: 1982:87*, NSF 84-304, (January 1984), pp. 36-39.

[NSF 85] National Science Foundation, Division of Computer *Research Summary of Awards Fiscal Year 1985*, 1985.

[Oglesby 83] Oglesby, Charles E. and Urban, Josep E., "The Urban Human Resources Task Area," *IEEE Computer* 16, 11 (November 1983), pp. 65-70.

[Packard 86] Packard, David, *An Interim Report to the President by the President's Blue Ribbon Commission on Defense Management*, (February 21, 1986).

[Patten 81] Patten, Thomas H., Jr., *Organizational Development Through Teambuilding*, John Wiley and Sons, 1981.

[Penney 85] Penney, George, "Skills Shortages: Reasons and Remedies", *Information Technology Training* (February 1985), pp. 17-19.

[Perelman 82] Perelman, Ellen S., "Job Challenges, Career Paths Can Lure DP Programmers," *Bank Systems & Equipment* (July 1982), pp. 42-44.

[Peters 82] Peters, Thomas J. and Waterman, Robert H., Jr., *In Search of Excellence: Lessons Learned from America's Best-Run Companies*, Harper and Row, 1982.

[Putnam 85] Putnam, L., "The Impact of Methodologies on Software Productivity: Case Studies," *National Conference and Workshop on Methodologies and Tools for Real-Time Systems*, Washington, DC, (October 28-November 1, 1985).

[RDTWG 84] Rights in Technical Data Working Group, *Report of the Rights in Technical Data Working Group*, Institute for Defense Analyses, IDA Record Document D-52, (January 1984).

[Riddle 86] Riddle, William E. and Williams, Lloyd G., *Technology Selection Education for Software Engineers*, Rocky Mountain Institute of Software Engineering Report No. TR-86-001, (January 1986).

[Roberts 81] Roberts, Edward B. and Fusfeld, Alan R., "Staffing the Innovative Technology-Based Organization," *Sloan Management Review* (Spring 1981), pp. 19-34.

[Robey 84] Robey, Daniel and Markus, M. Lynne, "Rituals in Information System Design," *MIS Quarterly* (March 1984).

[Robinson 80] Robinson, Lt. Col. Daniel G. et al.., "Organizational Development at ASD: A Commitment from the Top," *Proceedings of the IEEE 1980 National Aerospace and Electronics Conference*, NAECON 1980, Dayton, Oh, (May 20-22, 1980), pp. 45-47.

[Sackman 86] Sackman, Erikson, and Grant, "Exploratory Experimental Studies Comparing Online and Offline Programming Performance," *Communications of the ACM* (January 1986).

[Salancik 77] Salancik, Gerald R. and Pfeffer, Jeffrey, "Who Gets Power - and How They Hold on to It: A Strategic - Contingency Model of Power," *Organizational Dynamics*, (Winter 1977).

[Samuelson 86] Samuelson, Pam, *Toward a Reform of the Defense Department Software Acquisition Policy*, Software Engineering Institute Technical Report forthcoming June 1986.

[Samuelson 86a] Samuelson, Pam, *Understanding the Implications of Selling Rights in Software to the Department of Defense: A Journey Through the Regulatory Maze*, Software Engineering Institute Technical Memorandum, (March 1986).

[Samuelson 86b] Samuelson, Pam, *Comments on the Proposed Defense and Federal Acquisition Regulations*, Software Engineering Institute Technical Memorandum, (March 1986).

[Samuelson 86c] Samuelson, Pam, *Adequate Planning for Acquiring Sufficient Documentation about Rights in Software to Permit Organic or Competitive Maintenance*, Software Engineering Institute Technical Memorandum, (March 1986).

[Samuelson 86d] Samuelson, Pamela, Deasy, Devin, and Martin, Anne C., Proposal for a New "Rights in Software" Clause for Software Acquisitions by the Department of Defense, (CMU/SEI-86-TR-2), September 1986.

[Schein 84] Schein, Edgar H., "Coming to a New Awareness Organizational Culture," *Sloan Management Review*, (Winter 1984).

[Schneider 76] Schneider, Benjamin, *Staffing Organizations*, Goodyear, 1976.

[Schwab 69] Schwab, D. and Heneman, H.G., "Relationship Between Interview Structure and Interviewer Reliability in an Employment Situation," *Journal of Applied Psychology 53* (1969), pp. 214-217.

[Schwartz 68] Schwartz, "Analyzing Large-Scale System Development," *Proceedings of the 1968 NATO Conference*, 1968.

[SEI 86] Software Engineering Institute, *SEI Program Plans Draft*, Carnegie-Mellon University, SEI-86-CDRL-104/07, (May 2, 1986).

[Semprevivo 80] Semprevivo, Philip C, *Teams in Information Systems Development*, Yourdon, 1980.

[Shaw 85] Shaw, Mary, ed., *The Carnegie-Mellon Curriculum for Undergraduate Computer Science*, Springer-Verlag, 1985.

[Shen 86] Shen, Vincent, "Empirical Research on the Design Process," *Eleventh Annual Software Engineering Workshop*, NASA/Goddard Space Flight Center, 3 December 1986.

[Sheppard 78] Sheppard, S.B. et al., *Predicting Programmers' Ability to Modify Software*, General Electric, Sponsored by the Office of Naval Research, ADA056079, (May 1978).

[Shertz 81] Shertz, Joan and Weiser, Mark, "A Study of Programming Problem Representation in Novice and Expert Programmers," *Proceedings of the 18th Annual Computer Personnel Research Conference*, (June 3-5, 1981).

[Shneiderman 80] Shneiderman, Ben, *Software Psychology: Human Factors in Computer and Information Systems*, Cambridge, MA, Winthrop Publishers, 1980.

[SIGCSE 85] "Papers of the Sixteenth SIGCSE Technical Symposium on Computer Science Education," *ACM SIGCSE Bulletin* 17, 1 (March 1985).

[Silverman 85] Silverman, Barry G., "Software Cost and Productivity Improvements: An Analogical View," *IEEE Computer* (May 1985), pp. 86-95.

[Sinetar 85] Sinetar, Marsha, "SMR Forum: Entrepreneurs, Chaos, and Creativity - Can Creative People Really Survive Large Company Structure?," *Sloan Management Review* (Winter 1985).

[Smelcer 86] Smelcer, J.B., *Expertise in Data Modeling or What is Inside the Head of an Expert Data Modeler*, Doctoral Dissertation, Computer and Information Systems, The University of Michigan, summary in *CHI'86 Proceedings*, (April 1986), pp. 350.

[Sobkowiak 86] Sobkowiak, Roger T., Software People Concepts, Inc., New Haven, Ct., personal communication, May 1986.

[SofTech 84] SofTech, Inc., U.S. *Army Ada (Trademark) Training Curriculum. Curriculum Catalog 1984*, 1984.

[Soloway 86] Soloway, Elliot, "Studying Software Engineering Documentation From a Cognitive Perspective," Eleventh Annual Software Engineering Workshop, NASA/Goddard Space Flight Center, 3 December 1986.

[Soloway 86] Soloway, E. and Iyengar, S. (eds.), *Empirical Studies of Programmers*, Ablex, 1986

[Starbuck 71] Starbuck, W.H., *Organizational Growth and Development*, Penguin, 1971.

[STARS 85] STARS Joint Program Office, *Software Technology for Adaptable, Reliable Systems (STARS) Program Plan*, Draft, 19 (September 1985).

[Steier 85] Steier, D.M. and Kant, E. "The Roles of Execution and Analysis in Algorithm Design," *IEEE Transactions on Software Engineering* SE-11, 11 (November 1985), pp. 1375-1385.

[Stroh 81] Stroh, Peter, "Team Building and System Development," *Datamation* 27, 2 (February 1981), pp. 89-90.

[Suchman 85] Suchman, Lucy A., "Technology in Use," *CHI '85 Proceedings* (April 1985), pp. 89-91.

[Taft 86] Taft, William H., IV, "Streamlining Has Begun to Payoff," *Program Manager* XV, 2 (March-April 1986), pp. 18-20.

[Testerman 77] Testerman, Ward, "Matching Computer Personnel and the Job Environment," *Proceedings of the 15th Annual Computer Personnel Research Conference* (August 18-19, 1977).

[Thomsett 80] Thomsett, Rob, People and Project Management, Yourdon Press, 1980.

[Tushman 79] Tushman, Michael L., "Managing Communication Networks in R&D Laboratories," *Sloan Management Review* (Winter 1979), pp. 37-49.

[Tushman 80] Tushman, Michael L. and Nadler, David A., "Models of Organizing and Organizational Change," *Organizational Dynamics*, (Autumn 1980).

[Unger 77] Unger, Brian and Walker, Sheldon, "Improving Team Productivity in System Software Development," *Proceedings of the 15th Annual Computer Personnel Research Conference* (August 18-19, 1977), pp. 113.

[Wade 86] Wade, James P., Jr., "DoD Acquisition: What the Future Holds," *Program Manager* XV, 2 (March-April 1986), pp. 27-29.

[Watson 78] Watson, Peter, *War on the Mind: The Military Uses and Abuses of Psychology*, Basic Books, 1978.

[Waller 86] Waller, Douglas, Bruce, James, Cook, Douglas, *SDI: Progress and Challenges*, Staff report submitted to Senator William Proxmire, Senator J. Bennett Johnston, and Senator Lawton Chiles, (March 17, 1986).

[Weinberg 71] Weinberg, G., *The Psychology of Computer Programming* , New York, Van Nostrand Reinhold and Co., 1971.

[Westney 86] Westney, D. Eleanor and Sakakibara, Kiyonori, "Designing the Designers: Computer R&D in the United States and Japan," *Technology Review* (April 1986), pp. 24-69.

[White 84] White, Kathy Brittain, "MIS Project Team: A Theoretical Model Based on Empirical Research," in Beyond *Productivity: Information Systems Development for Organizational Effectiveness* edited by Thomas M.A. Bemeimans, Elsevier Science Publishers, 1984.

[White 84a] White, Kathy Brittain, "MIS Project Teams: An Investigation of Cognitive Style Implications," *MIS Quarterly* (June 1984), pp. 95-101.

[White 84b] White, Kathy Brittain, "An Investigation of Task Team Structure and Its Impact on Productivity," *Proceedings of the National Computer Conference* (1984), pp. 497-503.

[Wilemon 84] Wilemon, David L. and Meyers, Patricia W., "Learning Strategies in High Technology Teams," *Proceedings of the International Congress on Technology and Technology Exchange: Technology and the World Around Us - ICTTE '84 and EMC '84 - Management of Technology and Its Limitations*, (October 8-10, 1984), Pittsburgh, PA, pp. 371-379.

[Worboys 75] Worboys, G.N., "Validation of Externally Developed Assessment Procedures for Identification of Supervisory Potential," *Personnel Psychology* (Spring 1975), pp. 77-91.

[Zientara 81] Zientara, Marguarite, "Looking for Staff with 'Potential'? GE Has Suggestion: Profile Sheet," *Computerworld* 15, 50 (December 14, 1981).

[Zmud 79] Zmud, Robert W., "Individual Differences and MIS Success: A Review of the Empirical Literature," *Management Science* 25, 10 (October 1979), pp. 966-979.

SECTION B13

Technology Transition

Prepared by Samuel T. Redwine, Jr.

Topics covered in Section B13:

13.0 TECHNOLOGY TRANSITION

13.1 Introduction
13.1.1 Purpose and Scope
13.1.2 Background

13.2 Requirements
13.2.1 Information to Make Decision to Build
13.2.2 Transitioning Capabilities to Build SDI "Soon Enough"
13.2.3 Functionality Needed

13.3 Current Status
13.3.1 Process
13.3.1.1 Process Before Use
13.3.1.2 Process of Transition to Use
13.3.2 Product
13.3.3 Summary

13.4 Recommendations

13.5 References

Table of Figures

Figure	Title	Page
1	BM/C3 Software Technology Picture at Time of Full-Scale Engineering Development Decision.....	326
2	Technology Pipeline.....	328
3	Factors Influencing the Likelihood of Adoption or Adaptation.....	335-336
4	Product Characteristics and Components.....	340
5	Organizational Structures and Activities Related to Acceleration.....	342
6	Elements of a Technology Insertion Plan.....	343

13.0 TECHNOLOGY TRANSITION

13.1 Introduction

This section addresses *technology transition* for Strategic Defense Initiative (SDI) Battle Management and Command, Control, and Communication (BM/C3) software technology. Technology transition is the planned overt actions taken to improve the technology base and covers the entire process of transforming research results -- usually through a series of stages -- into usable forms and accomplishing successful use of them. *Technology maturation* differs from technology transition by referring to the unaided natural process. *Technology insertion* is the process of accomplishing the initial use of a technology product on a real project. We will use the term *technology transfer* only in the context of inadvertent, undesirable transfer of technology to a foreign country [McDonald 86].

13.1.1 Purpose and Scope

This section reviews SDI BM/C3 software technology requirements for speeding the use of technology and preventing undesirable technology transfer; for assessing the states of the art and practice in technology transition, especially as related to software and the Department of Defense (DoD); and for providing recommendations. The scope includes transitioning technology from everywhere worthwhile -- not just from SDI or U.S. government technology efforts. All stages of the process are considered, from research to successful use throughout SDI BM/C3 software's development and support.

13.1.2 Background

When Project Hindsight took the first systematic look at the impact of research on the development of weapon systems [Sherwin 67], it found that undirected (basic) science results were infrequently used even after 20 years and that even a piece of highly applied research often takes 5 to 10 years before being used. Recently Redwine and Riddle revisited this question for software technology and likewise found it takes more than 15 or 20 years before a technology is inserted [Redwine 85]. Thus, both a problem and an opportunity exist for SDI -- acceleration of this process is certainly possible.

Other task area assessments sometimes recommend specific steps to speed the use of their technology; this section is concerned with the broad systematic possibilities for improvement across software technology. Some of the issues covered overlap with the areas of People and Organizations and Software Engineering Environments.

Several previous DoD and SDI studies and decisions have been directed at related concerns. DoD has been concerned for a number of years about the need to speed software technology transition [Druffel 83]. This concern led to the establishment in late 1984 of the DoD Software Engineering Institute (SEI) at Carnegie-Mellon University in Pittsburgh. This Federally Funded R&D Center is specfically chartered to aid technology transition of software engineering technology into DoD mission critical systems.

The Eastport Group recommended that the SDI effort not rely on breakthroughs for proving the feasibility of doing the BM/C3 software. This implies a reliance on technology transition for achieving this proof. The SDIO has adopted a prototyping approach to the SDI system that should be a technology transition mechanism, because it permits the trial use of some not yet mature technologies. Likewise the SDIO is establishing a National Testbed one of whose purposes is to aid in technology evaluation.

This section covers SDI requirements for technology transition in Section 13.2 and then briefly reviews the present research state of the art and the state of the practice of technology transition in Section 13.3. Finally, building on prior DoD and SDIO activities, Section 13.4 contains a number of recommendations.

13.2 Requirements

Technology transition related requirements for SDI BM/C3 software technology fall under the two objectives of the technology program:

a. Provide information supporting the full scale engineering development decision in the early 1990s.

b. Provide the capabilities needed to build the SDI system soon enough to prevent BM/C3 software from being the element of the system that delays its deployment, operational status, or upgrades.

The first objective includes technology transition concerns such as evaluation of technologies and the capability to bring the technologies included in the evaluations to adequate maturity for producing the operational SDI system. The second objective relates directly to the speed at which the needed technologies are transitioned to use in the SDI effort.

13.2.1 Information to Make Decision to Build

The full scale engineering deveopment decision involves both the question of the technical feasibility of building the BM/C3 software for an acceptable SDI system and the merit of the most attractive system that could be built with the software technology available. Clearly, if no technically acceptable SDI system is feasible, then the decision outcome should be negative (quit or delay). If the merit of the most attractive feasible alternative is not great enough to convince a majority of Congress (plus the other decision makers involved) then a negative outcome will also occur. (And indeed, this support must still continue over a number of years as the system is built, deployed, operated, and upgraded.)

The decision to enter full scale engineering development may not be made all at one time. However, looking at the decision from the perspective of the time when the decision is being made is useful. Viewed from the perspective of the time of the decision, Figure 1 illustrates four considerations shown by the arrows entering the "BM/C3 Software Feasibility Determination Box".

While the decision on building the SDI sysem will not be based on software technological factors alone, a determination of the feasibility of building the BM/C3 software depends on what constitutes the technically acceptable SDI systems (top of Figure 1). The software functionality and characteristics required by these systems imply a threshold for software requirements. This threshold will determine the state of practice necessary, which yields the first consideration -- technical feasibility criteria for building the software for the SDI system.

The second consideration is that all the necessary software technologies must be mature enough for evaluations to produce convincing evidence of their merit and workability for the SDI system ("Evaluated Software Technology" in Figure 1). This probably means prototype software resulting from the (possibly prototype) technologies must have proved itself in large-scale tests and simulations that are as realistic as possible. The third

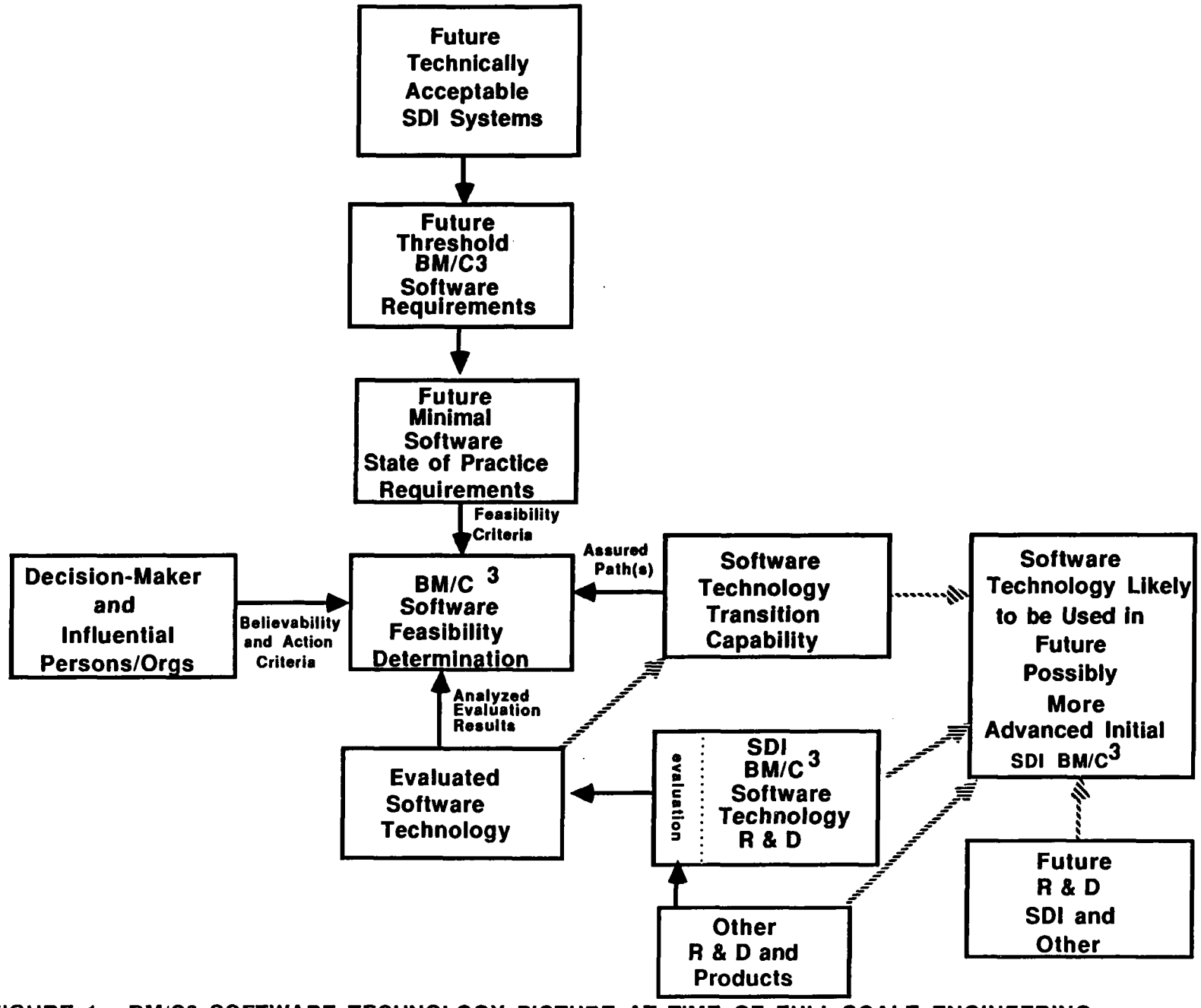


FIGURE 1. BM/C3 SOFTWARE TECHNOLOGY PICTURE AT TIME OF FULL-SCALE ENGINEERING DEVELOPMENT DECISION

consideration is that a clear and certain technology transition path must exist for bringing each technology to production quality, integrating it into the SDI effort, and ensuring its successful use by the time required. Although widespread use of the technologies is not required, the technologies will have to be integrated into multiple organizations within the SDI effort.

Need also exists for the fourth input to the software feasibility determination box in Figure 1 -- establishing the believability and decision criteria for the types of decision makers and influential individuals and organizations that will be involved in this early 1990's decision.

If not done well, efforts to produce these last two could also lead to unacceptable uncertainty and a negative determination. The exact threshold and evidence required will become clearer and more precise as SDI planning and R&D progress. Nevertheless, these considerations already can provide some guidance.

The right side of Figure 1 and the gray arrows indicate that the state of technology used to show feasibility may not be the same as that used to build the system at a later date. Note that the software technology R&D effort should not aim only at meeting the minimal threshold, but at providing the basis for building as good an SDI system as possible. This is a secondary aim when considering feasibility but an important one when considering total system attractiveness.

Thus, technology transition concerns enter into the full scale engineering development decision in several ways.

13.2.2 Transitioning Capabilities to Build SDI "Soon Enough"

Figure 2 shows the abstract process that technologies may follow in their development for use in a DoD program. Technology transition efforts attempt to improve this process by such actions as identifying and communicating requirements, scanning research and development plans and results outside the program's funded ones, evaluating of intermediate results for promise, and providing incentives for early use by contractors.

The SDI technology transition efforts must speed this process for the most promising technologies to allow their successful use soon enough to avoid delaying the SDI system. If warranted, this can include multiple efforts aimed at the same need either to overcome risks or because they are aimed at different time frames.

The SDI system will be evolving and the technology required by upgrades will undoubtly be more advanced with each generation. The SDI system could well either be in a measures/countermeasures spiral or demonstrate such potential to improve (including software) as to discourage this. In either case, continuing improvement in software technology capabilities will be required.

13.2.3 Functionality Needed

While just what mix and emphasis of technology transition functions should be used is a question for the recommendations section and future decisions, a number of types of functions exist.

a. Ascertain and communicate SDI requirements to R&D community in a usable fashion.

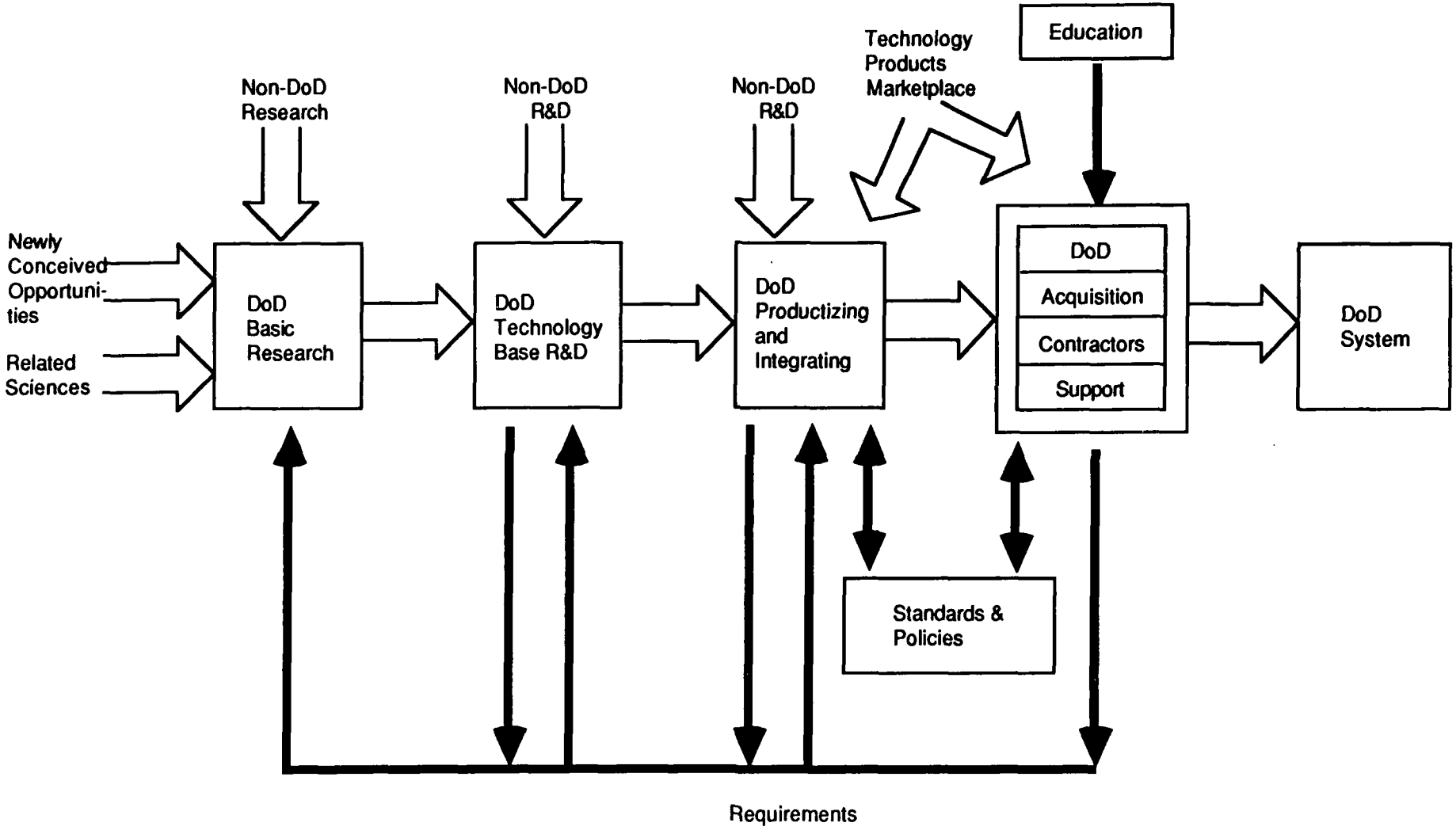


FIGURE 2: Technology Pipeline

b. Scan environment for R&D and product results and plans that might be of merit to SDI.

c. Evaluate concepts, intermediate results, and products for promise.

d. Increase awareness within the SDI effort and its R&D community of items either of possible merit or evaluated as promising.

e. Identify opportunities to accelerate promising technologies.

f. Improve and speed funding decisions, funding distribution, and work initiation.

g. Speed each interorganizational transfers along R&D path.

h. Speed productization (commercialization if done in commercial arena).

i. Ease and speed integration with other technologies and practices used in SDI effort.

j. Achieve product elements and characteristics that make it more likely to be used successfully in SDI effort.

k. Achieve readiness to accept and use in SDI effort personnel and organizations.

l. Speed technology insertion for first successful use in SDI effort.

m. Speed/ensure use of a technology everywhere in SDI effort where appropriate.

n. Reduce cost, technical incompatibilities, and other obstacles to use.

o. Have all needed technologies in use when needed.

p. Provide acquisition contract clauses and methods to aid in achieving these results in SDI contractor community.

q. Avoid undesirable foreign technology transfer.

r. Identify unneeded or less desirable technologies (so as to not invest except as warranted by risk reduction or other rationale).

s. Plan technology efforts and use over multiple prototypes and generations, multiple organizations, and across SDI technology and application areas.

These functions clearly cut across all the other task areas and involve many of the functions of technology R&D management. Of particular importance are the other task areas of Software Engineering Environments and People and Organizations.

13.3 Current Status

This section briefly reviews the current status of the state of the art and the state of practice in technology transition. The review is selective and emphasizes aspects believed to be most relevant to SDI BM/C3 software technology transition. The review first covers concern for the process and then issues involving the technology product. Under both

process and product, coverage is given to issues involved before the use of the technology and then to issues involved in use.

13.3.1 Process

As shown conceptually in Figure 2, the process of a technology advancing from a reseach idea to a technology product used in a DoD effort can include a number of steps. These typically involve a number of organizations in roles of sponsoring, performing R&D, productizing, promoting, educating, selecting for use, using, etc. For many of these organizations the technology must flow in, be worked on, and flow out. In the organizations, the inter-organizational arrangements, and the SDIO funding process many decisions will need to be made. One way of viewing technology transition is that it influences those decisions so that the technology will advance more rapidly and appropriately.

13.3.1.1 Process Before Use

General

Significant relevant literature exists on R&D management [Tornatzky 83, p.113-118]. Much of this deals with managing research personnel, communication paths in laboratories, the place of R&D in an industrial enterprise, and project selection. Unfortunately, the management literature on the last point is often not relevant for the SDI effort because it takes a portfolio viewpoint that does not consider the highly specific mission and design needs of a more focussed project like the SDI.

Significant literature (overlapping somewhat with the management literature [e.g., *Tushman 82*]) also exists on the subject of innovation. One part of this literature is devoted to individual creativity and problem solving. This literature, however, is more relevant to the Software Engineering Environment area that includes concerns for methods and automated support for such activities as designing and the People and Organization area where one concern is for individual performance.

The part of the innovation literature of interest here is concerned with the innovation process in the larger context. In 1983 the National Science Foundation published *The Process of Technological Innovation: Reviewing the Literature* summarizing the "innovation process research" literature [Tornatzky 83]. It covers both before use and use processes.

Among the summary points made are:

a. The innovation process is usually lengthy (spanning years).

b. The process is strongly impacted by organizational and contextual factors.

c. The process usually follows the same sequence of activities involving many organizational units and individuals.

d. The organization, placement, management, and interconnection of the R&D function is particularly important.

e. Government actions have had a checkered history.

Both the management and research literature on communication paths is relevant because it points out the key types of people who must be effectively communicated with in order to improve and speed the research process. One of the key types of communication is across organizational boundaries. These links can be facilitated by a variety of linking mechanisms -- informational and organizational. Information is needed by downstream organizations for awareness and evaluation of relevance and promise. Information is needed upstream for understanding of requirements and acceptability issues.

People are a good way to convey information. Participation of downstream organizations' key communicators and decisionmakers in some meaningful way in upstream activity can be effective in preparing the way for acceptance. Face-to-face interaction is helpful. One particularly effective link is the transfer of technical personnel along with the technology [Tornatzky 83, p.221, Westney 86].

Software Technology

Redwine and Riddle have done the only broad study covering the process of software technology development before use [Redwine 85]. It finds a long process of 15-20 years, points out the characteristics of the technologies that seem to affect this duration, and the significance of (the lack of) good experience or evaluation information in interorganizational transfer.

Ehrlich, in reviewing factors influencing computing technology transfer, emphasizes the importance of communications, interorganizational relations, and the advantages of having research, development, and marketing working together from the beginning rather than taking a serial "pipeline" approach [Ehrlich 85].

Studies of the innovation process in large and small computer firms have been essentially journalistic, and it is hard to draw conclusions specific to software technology from them. The appearance that much initial innovation occurs in small firms, especially in the microcomputer software arena, certainly exists. Some large firms use "entrepreneurial" small internal groups for innovations -- for example, IBM did this to launch its personal computer.

On one hand, the industry has seen a number of examples of successful university spinoffs -- DEC, Wang, Softech along with many others in just the Boston area alone have led to thriving organizations built on computer science. On the other hand, Xerox Palo Alto Research Center, possibly the country's outstanding industrial software research center of the 1970's, has not had its work reflected in successful Xerox software products.

Of special interest are two U.S. industry R&D consortia. One is the Microelectronics and Computer Technology Corporation (MCC) in Austin, Texas, and the other is the Software Productivity Consortium (SPC) in Reston, Virginia. Both are aimed at the "pre-competitive" phase and have a number of major defense contractors as shareholding members. They both are developing mechanisms for aiding the transition of technology to their shareholders including persons explicitly designated to facilitate it and the return to their companies of persons assigned to the consortium for a period -- typically after two years.

The university community has several outstanding research departments. The traditional big three are still strong -- MIT, Stanford, and Carnegie-Mellon. But several others are also outstanding including such schools as the University of Maryland and the University of California Irvine. The defense industry has pockets of innovation in software, but these are usually limited even within the firms themselves.

Several foreign initiatives including software technology are the Alvey and ESPRIT programs in Europe and the ICOT and Sigma efforts in Japan. The European initiatives have forced joint efforts of universities and companies as a way to encourage better transition. Alvey has used industry personnel on temporary assignment to do much of the program management -- hopefully with their increased awareness facilitating transition. ICOT personnel assigned from industry regularly spend significant time back at their firms conveying information about the advances being made at ICOT.

DoD Related

The Department of Defense has a large number of organizations that fund or perform software R&D. (Other U.S. government agencies with significant efforts are NASA, FAA, NBS, and NSF.) Among these DoD organizations and Programs are DARPA, STARS, WIS, Army SDC, RADC, NOSC, ONR, AFOR, NADC, DCA, NCSC and NSA, CIA, DMA, and the service 6.2 and 6.3 programs. FFRDCs involved include Mitre, Aerospace, IDA, NRL, and the SEI.

The Software Engineering Institute was established specifically to aid software technology transition and devotes part of its effort to the process "before use". This includes some scanning for technology, some prototyping, joint projects, and awareness publications.

The same increasing awareness that led to the SEI is reflected in DARPA's sponsoring of a recent study by the Technology Transfer Study Center of George Mason University (now known as The Center for Productive Use of Technology) that included discussion of transfer within DARPA programs, transfer to the military, domestic spin-off, technical knowledge containment, and transfer into DARPA from other sources [Havelock 85]. The study made five recommendations to DARPA for early actions all of which have some relevance to the "before use" phase.

a. Appoint a full time technology transfer facilitator concerned in part with increasing awareness of military needs and parallel R&D efforts.

b. Develop a state-of-the-art on-line retrieval system for tracking all projects.

c. Begin a special process of reporting on critical program transition points and outcomes at each stage.

d. Convene a panel to draft a DARPA policy on access to unclassified technical knowledge.

e. Initiate a systematic periodic search for new technologies.

Elder, also at the center, has done a historical study of Ada [Elder 86].

The coordination among all organizations in DoD doing or funding software R&D, while better than a few years ago, is still relatively minimal. Some exchange of information occurs under the auspices of the Joint Logistic Commanders. The Computer Software and Systems Directorate in OUDRE (R&AT) has oversight responsibility for some of the R&D. For a while the STARS Program provided a forum for exchange of information. More recently, the SDIO has begun regular meetings involving a number of the programs.

Summary

The state of the art in technology transition for the "before use" phase is such that a fair amount is known. The state of the practice in DoD software technology transition is relatively weak, but some promise of improvement exists with the new efforts of the SEI and others.

13.3.1.2 Process of Transition to Use

General

More literature exists on the innovation utilization process than on the "before use" phase. The NSF study mentioned earlier gives one review of the literature [Tornatzky 83]; an even more thorough survey was produced at roughly the same time by Glaser, Abelson, and Garrison [Glaser 83].

Their title for their book is interesting -- *Putting Knowledge to Use: Facilitating the Diffussion of Knowledge and the Implementation of Planned Change.* They review the nature and scope of knowledge utilization (their terminology); factors influencing knowledge utilization and change; stages in problem solving, knowledge utilization, and planned change; linking knowledge with potential users; and guidelines for improved knowledge utilization. The left side of Figure 3, adopted from Glaser, lists many of the factors influencing knowledge utilization. The list illustrates the many factors that are potentially relevant to the SDI effort. While their advice is much richer than this brief restatement might indicate, four practical guidelines are:

a. Consider context, assess readiness, and provide incentives as well as consultation assistance for adoption or adaption.

b. In connection with efforts to use R&D findings to influence policy decisions, focus on altering the patterns of awareness or perception of relevant policy makers, and consider policy makers' interests, values, and constraints.

c. Develop communication networks.

d. Give major attention to possible ways of kindling interest and support of potential users: be clear about cost benefits, other existing evidence of worth, and feasibility of adopting the new technology or procedure.

The NSF review also offers a summary of "lessons learned and unlearned" [Tornatzky 83, p. 217-221]. A few of the points deserving special mention include:

a. Need to focus on all the social units both technology producers and users.

b. User perceptions are important.

c. Organizational characteristics, structure, size, ability to cope with uncertainty, and norms, procedures, and rewards for interorganizational exchanges all have impact.

d. Organizational contexts significantly affect capacities for innovation through constraints on resources and information required for making decisions.

e. Involvement, participation in decision making, perceptions of control, and interpersonal (particularly face-to-face) interaction on the part of those affected by the innovation can be important.

f. Differences between university and industry goals and rewards can hamper transition.

g. The movement of technical personnel is a major vehicle for technology transition.

h. Certain roles played by individuals (e.g., product champion or entrepreneur) appear to be particularly important.

This latter point is expanded on by Maidique who enumerates roles and discusses their performance as firms grow and evolve [Maidigue 80]. His names for the roles are technological entrepreneur, product champion, executive champion, technical definer, sponsor, and business definer. Welsch has established the importance of the facilitative role of information transfer specialists in the successful introduction of decision support systems -- what he calls the missionary and maturation inducing functions. Potential exists for establishing the right climate for such roles and for institutionalizing some of them.

Two recent books on managing change and innovation in organizations in the 1980's are by Kirkpatrick and Humpreys [Kirkpatrick 85, Humpreys 86]. They offer the results of rich experience.

An interesting survey of the field from the knowledge researcher's point of view is offered in a collection edited by Rich [Rich 81]. It points to the many pitfalls a researcher needs to avoid when doing research on the complex topic of knowledge utilization.

Finally, one should not forget that parts of a number of disciplines have relevant aspects. These include marketing, economics, political science, psychology, sociology, management, organizational development, education and training, law, acquisition, standards and regulations, human engineering and computer science. Pulling together all the relevant knowledge to help formulate a technology transition effort, if done from scratch, could be a major effort in itself.

Software Technology

In 1983 the IEEE Computer Society held a workshop entirely devoted to the issue of software engineering technology transition [IEEE 83]. A number of interesting papers were delivered as well as working group reports on Evaluation, Training, Technology Transfer Process and Vehicles, and Behavioral Aspects. A number of conclusions were drawn including the need for a well defined and coherent software engineering process in an organization to form a basis for future transition and improvement, and the importance of behavioral or organizational issues. Of special interest is a unique article by Willis on the internal rate of transition within a major and progressive defense contractor (Hughes) that indicated that after a technology had entered the organization, 4 to 8 years were still required to propagate it throughout the organization.

Also evident from the jobs of the attendees was the fact that a number of organizations have formed sub-organizations part of whose role is to facilitate technology insertion. These departments went by a variety of names including training, toolsmithing, and quality assurance.

	GENERAL	SOFTWARE TECHNOLOGY
Ability to carry out the change	Capability of staff Availability of necessary resources Structuring/planning	Training including prerequisite knowledge High cost Efficiency in upgrading SEE Planned implementation Resource availability Reasonable schedule
Values or institutional/ cultural norms	Compatibility with user's norms Openness/willingness to listen Leadership that encourages efforts to improve operational performance; willingness to entertain challenge Publicness vs. privateness Impact on interpersonal relations	Technically astute managers Management commitment Fit into standard operating procedures Forward thinking organization
Idea or information about the qualities of the innovation	Credibility Observability Ease in understanding and installation Trialability/devisibility/reversibility Susceptibility to successive modifications Availability of technical assistance Adequate personal interaction between innovator and potential users Scientific status Point of origin	Conceptual integrity Tunability Training Too easily modified technology Prior success of source Readily available help Simplicity Visibility of need Ergonomics
Circumstances that prevail at the time	Strong dissatisfaction with the status quo Pressures requiring structural or procedural change Gateway to other innovations Proximity	Latent demand Customer demand/contractual requirement

Figure 3. Factors Influencing the Likelihood of Adoption or Adaptation (page 1 of 2)

	GENERAL	SOFTWARE TECHNOLOGY
Timing or readiness for considering the data	Sensitivity to context factors Early involvement of potential users Linkage with source Synergy	Incremental extensions to current technology
Obligation or felt need to deal with a particular problem	Relevance Shared interest in solving common problems; internal advocacy Prior commitment Energy/investment of time and effort, persistance	Clear recognition of need Personal commitment Salesmanship Severity of need
Resistance or inhibiting factors	Skill in working through resistances Risk and uncertainty Gatekeepers or approval channels	Internal transfer time Psychological hurdles Technology immaturity Human characteristics Size of organization Risk
Yield, or perceived prospect of beneficial payoff from adoption	Perceived relative advantage Incentives/rewards for change Provision of financial support, esteem, status Efficiency of innovation Gateway capacity to other innovations Return on investment	Prior positive experience Contracting disincentives Incentives Technology worthiness
	Source [Glosser 83]	Source [Redwine 85, Willis 83]

Figure 3. Factors Influencing the Likelihood of Adoption or Adaptation (page 2 of 2)

The previously mentioned article by Redwine and Riddle covers the usage history of a number of software innovations [Redwine 85]. The larger study [Redwine 84] from which the article derived contained more detailed histories. In addition, it found on the six DoD weapons projects reviewed that the level of technology used directly reflected the contract and standards requirements.

The right side of Figure 3 lists factors from [Redwine 85], and [Willis 83]. These factors cover much the same issues as the more generally derived list on the left but provide a set of factors specifically found relevant for software technology.

Technical compatibility and ease of integration of new software technology into the "standard operating procedure" is an important concern [e.g., Willis 83]. Interface standards can play a key role in addressing the technical aspects of this issue [Morton 86].

One interesting development particularly strong in the microcomputer software arena is the comparative evaluation of software products. These are becoming increasing sophisticated and useful (e.g., *Software Digest Ratings Newsletter*).

Several articles have appeared in SIGCHI Bulletin and ACM CHI (Computer Human Interaction) conferences reflecting recent thinking in the area of software technology transition [Ehrich 85, Grantham 85, Interest Group 86, Nielsen 86]. Much of this applies known general principles to inserting computing technology, but some is particularly related to software such as the impact of the integration and usability of the software.

As in all disciplines, software technology has an array of publications, conferences, workshops, and education and training that function in part as technology transition vehicles. Likewise the various subdisciplines (roughly corresponding to the task areas in this report) have their own "invisible colleges" providing networks that are often the key means of early communication of results.

DoD Related

The DoD SEI is beginning to develop its education and technology transition divisions and should be making more and more contributions to addressing the issues related to use. Opportunity exists for DoD programs to form close relationships with the SEI. The SEI and the Ada Joint Program Office also have efforts concerned with the evaluation of software engineering environments.

A 1982 independent review of DoD laboratories pointed to a number of concerns that need to be addressed in achieving use of technology in DoD programs. The George Mason study for DARPA also addresses use.

The recent Goldwater/Nunn and Packard Commission reports suggest changes in organization and acquisition practices that might improve technology use. Certainly opportunities exist to use the acquisition process [Lubbes 83].

One possible impediment to technolgy insertion may be the character of the individuals that are selected to be DoD acquisition program managers. Nidiffer reports that the future program managers attending the Defense Systems Management College are distinctly risk averse [Nidiffer n.d.]. Given their key fixture roles, this risk aversion will be an important factor if shared by SDI managers.

Finally, some DoD laboratories have formed close relationships with their user commands. This relationship can be a delicate balance between freedom to pursue the best opportunities and over-constraint to short-term payoff application specific projects. An interesting example is the relationship between the Air Force Systems and Logistic Commands and AFWL.

Summary

Even more is known about achieving the use of technology than about earlier stages. The DoD community is using more of this knowledge as well. However, substantial improvement is possible.

13.3.2 Product

Characteristics and parts of a technology product impact its speed all along the R&D and usage process. Early in the process these include documents generated, empirical evidence published, and perception that the technology addresses a recognized need. Later, components such as training, documentation, marketing and publicity aids, user aids, maintenance aids, operational aids and other support may be critical to success.

Figure 4 lists a number of product factors or components of interest. Of particular importance is the right information tailored for the different types of individuals involved in making usage decisions -- executive, technical manager, user, maintainer, etc.

One important characteristic is how well or easily a new product fits with the others in use. Here, technical compatibility including standards adherence is an important issue.

13.3.3 Summary

Quite a number of the individual issues involving technology transition are fairly well understood. When all the issues are considered together the result is less clear but still useful. The DoD has recently launched the DoD Software Engineering Institute to address these issues specifically for software technology for DoD mission critical systems. A number of opportunities exist either alone or in conjunction with the SEI and others for DoD programs to improve on the usual (not very outstanding) software technology transition performance of DoD programs.

13.4 Recommendations

As indicated by Project Hindsight and more recent studies, fundamental breakthroughs normally take at least 15-20 years before they are used in DoD systems. This means both that the basic research results usable in feasibility demonstrations before the early 1990s full scale engineering development decision and in an SDI system's initial operational capability most likely already exist and that speeding technology transition offers the most promising and surest way to potentially meet SDI needs. Thus, SDIO should give emphasis to managing, improving, and speeding the technology transition of software technology related to BM/C3. The recommendations provided here provide an outline of how to do this and some first steps.

The first needs are to involve the relevant people and organizations, and to think ahead and lay out strategies.

Recommendation 1: Formulate specific technical strategies and plans for transitioning technology for application technologies,

foundation technologies, and software engineering technologies. Cover compatibility, integratibility, standards and conventions, ease of transfer, quality assurance, and interoperability. Plans should indicate paths for transitions of each technology within each task areas and fit closely with contractor technology insertion plan.

These strategies should include establishment of important interfaces among components, identification of generations of technology, concern for the different environments (platform, C2I, and SEE) and their underlying computing bases over time, identification of alternative competing technologies, responsibilities for certification and integration, and the general approaches to speed activities (e.g., evaluation) and pass technology between organizations. Within BM/C3 software technology each technology task area needs to have its own plans for technology transition in congruence with the technology transition strategies and other plans. Plans (in congruence with the strategies) should indicate paths for transitioning each technology within each task area should be reconciled with contractor technology insertion plans. Plans should indicate the feasibility of transitioning the technology successfully including provisions for evaluation and the willingness of the current and next organizations along the path to pursue it (possibly only if it passes evaluation).

SDIO with FFRDC assistance should be the driving force to fulfill this recommendation; however, all the relevant organizations should participate.

The organizational agreements needed to formalize relationships and establish the links and roles desired should be pursued. Informal arrangements in a number of areas should be satisfactory for the near term but eventually many will best be formalized. Certainly contractor roles should be covered in their contracts.

Recommendation 2: Establish relationships with all the relevant organizations; encourage (and in some cases require) speeding up of internal activities, establishing linking mechanisms, and accelerating the transfer of technology toward use in the SDI effort.

Figure 5A lists the types of organizations involved. Of particular importance are the organizations involved in evaluation and integration. The National Test Facility and the integrator(s) of SEE(s) may be particularly crucial in speeding the process. A number of linking mechanisms are listed in Figure 5B. One particular variation that might be considered is inclusion of future users technology on the review groups. The types of activities that need to be accelerated include not just the performance of R&D but all the activity types listed in Figure 5C. Indeed, it may be the other activities that have the most potential for speeding up the process.

The contractor organizations that will use the technology must have such use encouraged -- and to the extent possible ensured -- by features in their contracts (and their subcontracts).

Recommendation 3: Include provisions in the qualifications, statement of work, requirements for proposal content, evaluation criteria, funding and other provisions of acquisitions to encourage and assure the needed technology insertion and successful use. The SDIO should establish what such provisions should be to prepare for future contracting.

- Cost
    - Price
    - Cost of Introduction
    - Cost of Operation, Maintenance, and Use
    - Cost of Conversion to Future Replacement
- Warranty
- Direct Expected Benefits
    - Results
        - Type
        - Quality
        - Quantity
    - Time Reduction
    - Resource Reduction
- Predictability/Risks
- Compatibility/Interoperability
- Ease of Adoption
    - Prerequisites
        - People
        - Facilities
    - Ease of Installation
    - Ease of Startup
- Level of Integration
    - Internally
    - With Rest of SEE and Practices
- Tailored Information Available
- Maintenance Support
- Training
- User Aids
- Maintenance Aids
- Operation Aids
- Technology Insertion Aids
- Other Support (e.g., Hotline)
- Human Engineering
- Customizible/Tailorability
- Reliability and Availability
- Extensibility
- Prior Demonstrations and Use
- Evaluation Results
    - Quantitative and Qualitative
    - Realism and Credibility
- Planned Improvements
- Gateway to Other Innovations
- Conformance to Standards
    (inc. Certification)
- Testing/Verification History
    - Availability of Results
    - Independent V&V
    - Thoroughness
- Source
    - Reputation
    - History
    - Stability

Figure 4. Product Characteristics and Components

Contractors should have a history, organizational structure and practices, and staff with demonstrated willingness and strong educational foundation to adopt/adapt rigorous new software technology and use it successfully. (The government will probably have to help even the best qualified contractors to improve in this area.) The statement of work can call for specific tasks, suborganizations, and deliverables that improve technology insertion. These include requirements for technology insertion planning, technology insertion suborganizations, involvement in linking mechanisms, technical compatibility, and reports on progress. Figure 6 lists elements of a technology insertion plan.

The contractors should be required to cover in their proposals how they will introduce the required amounts and types of new software technology, and this should be an important factor in the evaluation criteria. Adequate funding for technology insertion efforts by the contractors is essential, and incentive payments for outstanding performance in using new technology would be helpful.

The SDI effort must continually scan all possible sources of relevant technology.

Recommendation 4: Continue to scan for promising software technology and products. The results of this scanning plus information on SDI supported efforts should be put in suitable form and communicated throughout the SDI community.

Much of this scanning may already be done by the DoD Software Engineering Institute or others whose results SDI can obtain. SDIO may need to fund some supplemental efforts including translating foreign literature and visits to projects. The information must then be put in suitable form for receivers to be able to decide on its relevance and distributed throughout the SDI community.

The software engineering environment area is particularly important both for incorporating work done elsewhere and for integrating the results of SDI R&D. A number of recommendations are made in Section 8 of Appendix B and elsewhere, but it is important that in the short term SDIO at least not foreclose any significant options.

Recommendation 5: In the near term, ensure that the SDI can piggyback on the NASA Space Station Software Support Environment (SSE) effort and any major DoD SEE efforts. In the longer term, position SDI to benefit from multiple sources of software tools.

A definite strategy in the SEE area should be established quickly, as mentioned in Recommendation 1.

Not only does technology need to flow rapidly, it also needs to be useful, acceptable technology.

Recommendation 6: Ensure that all the parties involved in the BM/C3 software R&D are familiar with the SDIO software needs and with the properties of a software technology that make it more likely to be used.

- Organizational Structure
  - Main Components
  - SDIO
  - "User" Contractors
  - SDIO Agents (e.g., RADC, BMD, ONR)
  - SDI R&D Performers
  - SDII
  - SEI
  - Other FFRDCs
  - SDI National Test Facility and Testbed
  - Government Software Technology Programs
  - Academia
  - Industry Technology Developers
  - SDI SEE Integrator
  - Allies' Software Technology Programs

Figure 5A

- Organizational Structure
  - Linking Mechanisms
    - Formal Arrangements
      - MOUs
      - Planning and Control
  - Technology Insertion Depts.
  - Plans
  - Review Groups
  - People Transfer/Champions
  - Transfer of Software
  - SDI BM/C3 SW Week and other meetings
  - Downstream Involvement
  - Requirements Communication
  - Information and Publicity
  - SDInet

Figure 5B

- Activities
  - Scanning
    - SEI, SDII, IDA, US vs Foreign, Translation
  - Evaluating/Comparing
  - Deciding
  - Acquiring Rights
  - Funding/Acting
  - Movement Between Organizations
  - Performing R&D Steps
  - Integrating
  - Demonstrating
  - Start-up
  - Trial Use
  - Full Use
  - Feedback/Improving
  - Planning

Figure 5C

Figure 5. Organizational Structures and Activities Related to Acceleration

- Goals Over Time
  - For software products
  - For software state of practice
- Current state of practice
  - Measurement
    - Products
    - Process
  - Standards and practices used
  - People and qualifications
- Gaps
  - Existing
  - Expected over time (if nothing done)
- Plans for technology insertion
  - Overall scheme
    - Generation (or waves)
      - Identification
      - Evolution across generations
      - Evolution within generations
    - Organizational responsibilities
      - Internal
      - Inter-organizational
    - People
  - Technology opportunities identification
    - Scanning and assessment process
    - *List of identified technologies*
  - Plans for individual technologies or technology areas
    - Schedule and staffing
    - Integrate
    - Organize
    - Revise standards and procedures
    - Prepare people
    - Install
    - Initial use
    - Measurement
    - Possible tuning, revision, improvement
    - Evaluation/comparison
  - Expected closing of gaps and any remaining gaps

Figure 6. Elements of a Technology Insertion Plan

Figure 4 lists many of the characteristics of software technology products that impact their likelihood of use. The key unanswered SDI software problems should be known to anyone who might be able to contribute. In FY87 DARPA plans to describe SDI problems that are candidates for solution by parallel processing, but the others should also be described by someone.

The SDI requirements, required and desirable properties of the technology, and the approaches chosen for solution all also need to be adequately reflected in the various evaluation procedures used at different stages as a technology advances toward use.

Recommendation 7: Ensure that the technology evaluation processes used at the different stages, while knowledgeable and appropriately independent, properly reflect the requirements and needs of the full scale engineering development decision, the SDI system, and the users of the technology.

In addition, the basis on which a technology will be evaluated needs to be known to its developer. Among the places needing attention is the interface with and use of the NTB. The NTF may evaluate results produced by technologies, but the technologies themselves will also often require evaluation.

A special consideration for technology transition is the preparation for the early 1990s SDI full scale engineering development decision and the impact of the decision point's existence on the conduct of technology transition-oriented activities. In order to plan and prepare for the early 1990's full scale engineering development decision SDIO should establish preliminary versions of (1) requirements for the technically acceptable SDI system(s) to be used to establish BM/C3 software technology feasibility requirements, (2) the believability/decision criteria of the types of decision makers and influential persons and organizations likely to be involved in the decision, (3) the nature of the demonstration(s) needed to convince them, (4) the threshold software technology feasibility criteria, and (5) the nature of the technology evaluations required along with (6) the kinds of technology transition capability needed to provide assurance of meeting the feasibility criteria in the time between the decision and the production of the software.

Evaluations and other technology transition activities should be changed to reflect the results, but it should be remembered that technology advances beyond the minimum required to show feasibility and obtain a go-ahead could make the system more effective and more attractive. Therefore, technology beyond the minimum has a place both in terms of the decision and in terms of future enhancements to an SDI system.

In summary, SDIO needs to seriously plan and manage the technology transition process explicitly involving all the relevant organizations, including the SEI and the contractors who will use the technology, while paying particular attention to technical compatibility, evaluation, the software engineering environment, and the implications of the early 1990s full scale development decision. Doing this should greatly improve the chances that the necessary BM/C3 software technology will be available and used.

13.5 References

[Druffel 83] Druffel, Larry E., Samuel T. Redwine, Jr., and William E. Riddle, "The STARS Program: Overview and Rationale," from *IEEE Computer* (November 1983), pp. 21-29.

[Ehrlich 85] Ehrlich, Kate, "Factors Influencing Technology Transfer," from *SIGCHI* 17, 2 (October 1985), pp. 20-24.

[Elder 86] Elder, Victoria, *Ada: A Case Study of Technology Transfer at DARPA*, Center for the Productive Use of Technology, George Mason University, May 1986.

[Glaser 83] Glaser, Edward M., Harold H. Abelson, and Kathalee N. Garrison, *Putting Knowledge to Use*, Jossey-Bass, 1983.

[Grantham 85] Grantham, Charles E., "Technology Transfer: The Organizational Role," from *SIGCHI* 17, 2 (October 1985), 25-28.

[Havelock 85] Havelock, Ronald G. and David S. Bushnell, *Technology Transfer at DARPA, The Defense Advanced Research Projects Agency: A Diagnostic Analysis*, George Mason/University, (December 1985).

[Hermann 82] Hermann, Robert J., *USDRE Independent Review of DoD Laboratories*, (March 1982).

[Humpreys 86] Humphreys, Watts, *Managing for Innovation: Leading Technical People*, Prentice Hall, 1986.

[IEEE 83] Proceedings from *IEEE Computer Society Workshop on Software Engineering Technology Transfer*, Miami Beach, Florida, (April 25-27, 1983).

[Interest Group 86] "Organizations as Political Social Systems: Constraints on Technology Transfer, Interest Group Discussion at CHI '85", from *SGCHI*, 17,3, (January 1986), pp. 50.

[Kirkpatrick 85] Kirkpatrick, Donald L., *How to Manage Change Effectively*, Jossey-Bass, 1985.

[Lubbes 83] Lubbes, H.O., "Project Management Tasks Area," from *Computer* Special Issue on STARS Program, (November 1983), pp. 56-62.

[Maidique 80] Maidique, Modesto A., "Entrepreneurs, Champions, and Technological Innovation," from *Sloan Management Review*, (Winter 1980), 59-76.

[McDonald 86] McDonald, Cathy, Samuel T. Redwine, *STARS GLOSSARY, A Supplement to the IEEE Standard Glossary of Software Engineering Terminology*, Version 3.0, IDA Paper P-1846, (January 1986), In Progress.

[Morton 86] Morton, Richard and Samuel T. Redwine, Jr., "Information Interface Standards for Software Engineering Environments," from *Proceedings Computer Standards Conference*, 1986, IEEE, pp. 42-48.

[Nidiffer n.d.] Nidiffer, Kenneth E., *How to Effect the Rapid Transition of the Ada(TM) Language by Understanding the Personality of the Program Manager.*

[Nielsen 86] Nielsen, Jakob, Robert L. Mack, Keith H. Bergendorff, and Nancy L. Grischkowsky, "Integrated Software Usage in the Professional Work Environment: Evidence from Questionnaires and Interviews," from *CHI '86 Proceedings,* (April 1986), 162-167.

[Redwine 84] Redwine, S.T., L.G. Becker, A.B. Marmor-Squires, R. J. Martin, S.H. Nash, and W.E. Riddle, *DoD Related Software Technology Requirements, Practices, and Prospects for the Future*, IDA Paper P-1788, (June 1984).

[Redwine 85] Redwine, Samuel T. and William E. Riddle, "Software Technology Maturation," IEEE Computer Society, from *8th International Conference on Software Engineering*, (August 28-30, 1985), London.

[Rich 81] Rich, Robert F. (Editor), *The Knowledge Cycle*, Sage Publications, 1981.

[Rydz 86] Rydz, John S., *Managing Innovations*, Ballinger, 1986.

[Sherwin 67] Sherwin, Chalmers W., and Raymond S. Isenson, Project Hindsight: A Defense Department Study of the Utility of Research, from *Science*, (June 23, 1967), pp. 1571-1577.

[Tornatzky 83] Tornatzky, Louis G., J. D. Eveland, Myles G. Boyland, William A. Hetzner, Elmima C. Johnson, David Roitman, and Janet Schneider, *The Process of Technological Innovation: Reviewing the Literature* , National Science Foundation, (May 1983).

[Tushman 82] Tushman, Michael L. and William L. Moore, *Readings in the Management of Innovation*, Pitman, 1982.

[Welsch 86] Welsch, Gemma M., "The Information Transfer Specialist in Successful Implementation of Decision Support Systems," *Database*, ACM SIGBDP, 18, 1 (Fall 1986).

[Westney 86] Westney, D. Eleanor and Kiyonori Sakakibara, "Designing the Designers, Computer R&D in the United States and Japan," *Technology Review*, (April 1986).

[Willis 83] Willis, R.R., "Technology Transfer Takes 6± 3 Years", *IEEE-CS Workshop on Software Engineering Technology Transfer*, IEEE, 1983.

Appendix C
Plan Charts

This appendix contains a set of planning timelines for the technology task areas that appear in Appendix B and are summarized in Section 4.0 of the main text. The timelines present the R&D recommendations for each area as activities and show when each should begin and end within the 1987-2000 time period. The 1987-1990 time frame is shown in more detail on an accompanying chart. While timelines are inprecise now, they will become more precise over time, as plans become more definite and technology directions clearer. Beginning in 1991, plans become necessarily more vague. As some activities span the entire time period and as resources are not unlimited, decisions on emphasis over time must still be made.

1. Battle Management/C3
2. Network Communications
3. Distributed Operating Systems
4. Data Management Systems
5. Man-Machine Interface
6. Parallel Processing
7. Computer Systems Engineering Technology Foundation Technology
8. Software Engineering Environments
9. Simulation
10. Software Dependability
11. People and Organizations
12. Technology Transition

Plan Chart for Distributed BM/C3

SUBAREAS	1987 - 1990	1991 - 1995	1996 - 2000
ALGORITHM RESEARCH	Research Parallel Algorithms applicable to SDI BM/C3	Contractor/University prototyping for advanced algorithms.	
ALGORITHM TEST CENTER	Definition phase through 1987. Initial Algorithm Library Established 1988. Algorithm Evaluations beginning 1990.	Parallel Algorithm Evaluations integrated into National Test Bed Operations.	Continuing parallel Algorithm trade off studies
BM/C3 STANDARDIZED DECOMPOSITION AND GLOSSARY OF TERMS	Formulate and issue with award of Phase III contract(s).		
	Revise/Update upon conclusion of Phase III contract(s).		
	Revise/Update based upon continuing system definition evaluations		
REQUIREMENTS DEFINITION OF HUMAN INTERVENTION IN BM/C3	Guidance to Phase III contractor(s) *for identification of requirements.*	Requirements reviewed periodically. *Used for prototype and development of* BM/C3 software.	
	Formalize requirements based upon conclusions of Phase III contract(s).		
DECISION SUPPORT SYSTEMS (DSS)	Identify DSS types by architecture	Prototype MBMS coupled with Algorithm Test Center Algorithm Library	
	Prepare specifications for DSS tool suite of model, database, and dialogue management subsystem	Progressive prototyping and integration of MBMS with tools for MBMS and DGMS	
BATTLE-MANAGEMENT-ORIENTED PROTOTYPING AND DEVELOPMENT AIDS	Application-oriented notations, tools, reusable components, etc.	Advanced application-oriented tools	

C-2

Near-term Plan Chart for BM/C3

SUBAREAS	1987-88	1989	1990
ALGORITHM RESEARCH	Identify computationally intensive BM/C3 needs and develop parallel algorithms to meet these needs.		
ALGORITHM TEST CENTER	Define the responsibilities of the test center	Start the algorithm library	Begin testing parallel algorithms on different parallel architechtures to determine performance as a function of architecture
DECISION SUPPORT SYSTEM	Determine the role that humans will play in the BM/C3 architecture	Determine the level of automation that is necessary to support that role and begin to develop prototypes	

C-3

Plan Chart for Network Communications Foundation Technology

SUBAREAS	1987 - 1990	1991 - 1995	1996 - 2000
PROTOTYPE DEVELOPMENT	Incorporate development of real-time reliable, reconfigurable communications network with the development of prototype distributed operating systems		
PROTOCOLS	Develop an open and extensible protocol standard		
	Fund research in the development of automatic protocol generators		
SECURITY	Fund research in the development of security models for network communications system		
NETWORK ALGORITHMS	Fund the development of naming, routing, topology updating, flow control, and clock synchronization algorithms to meet the reliability, security reconfigurability, and real-time performance requirements of the SDI		

C-4

Near Term Plan Chart for Network Communications Foundation Technology

SUBAREAS	1987	1988	1989	1990
PROTOTYPE DEVELOPMENT	Incorporate development of real-time reliable, reconfigurable communications network with the development of prototype distributed operating systems			
PROTOCOLS	Develop an open and extensible protocol standard			
		Fund research in the development of automatic protocol generators		
SECURITY	Fund research in the development of security models for network communications system			
NETWORK ALGORITHMS	Fund the development of naming, routing, topology updating, flow control, and clock synchronization algorithms to meet the reliability, security reconfigurability, and real-time performance requirements of the SDI			

C-5

Plan Chart for Distributing Operating Systems Foundation Technology

SUBAREAS	1987 - 1990	1991 - 1995	1996 - 2000
PROTOTYPE DEVELOPMENT	Fund the development of 4 real-time, secure, and fault-tolerant systems. Base one system on Mach of CMU. Integrate projects on scheduling, security, theory, and reliability.		
ACCESS CONTROL AND AUDITING	Incorporate development of secure kernal with development of prototype distributed operating systems.		
SCHEDULING	Incorporate the development and analysis of real-time schedulers with development of prototype distributed operating systems		
	Monitor research in dynamic scheduling and ensure that current levels of funding continue		
THEORY OF DISTRIBUTED SYSTEMS	Incorporate theoretical projects with each prototype distributed operating system		
	Monitor current work and ensure that current levels of funding continue.		
SYSTEM DESIGN	Assist hardware (in the form of surplus computing power and resources) should be used to reduce software complexity		
DISTRIBUTED TESTBEDS	Place state of the art multicomputer systems at reasearch sites.	Widely distributed prototype operating systems	

C-6

Near Term Plan Chart for Distributing Operating Systems Foundation Technology

SUBAREAS	1987	1988	1989	1990
PROTOTYPE DEVELOPMENT	Fund the development of 4 real-time, secure, and fault-tolerant systems. Base one system on Mach of CMU. Integrate projects on scheduling, security, theory, and reliability.			
ACCESS CONTROL AND AUDITING	Incorporate development of secure kernal with development of prototype distributed operating systems.			
SCHEDULING	Incorporate the development and analysis of real-time schedulers with development of prototype distributed operating systems			
	Monitor research in dynamic scheduling and ensure that current levels of funding continue			
THEORY OF DISTRIBUTED SYSTEMS	Incorporate theoretical projects with each prototype distributed operating system			
	Monitor current work and ensure that current levels of funding continue.			
SYSTEM DESIGN	Assist hardware (in the form of surplus computing power and resources) should be used to reduce software complexity			
DISTRIBUTED TESTBEDS	Place state of the art multicomputer systems at reasearch sites. Widely distributed prototype operating systems.			

Plan Chart for Data Management Systems

SUBAREAS	1987 - 1990	1991 - 1995	1996 - 2000
PLATFORM ENVIRONMENT	Identify[1] and implement prototype algorithms		
	Fund research in exploring parallel algorithms		
	Develop efficient means for replicating data		
	Develop techniques for comparing/ contrasting algorithms		
PLATFORM MANAGEMENT ENVIRONMENT	Develop prototype distributed DBMS's		
	Develop error recovery techniques, data models, etc.		
	Select interfaces, concurrency control and deadlock resolution algorithms[1]		
SOFTWARE ENGINEERING ENVIRONMENT	Fund basic research in data models, active components, amount of data to store issues, etc.		
	Develop distributed prototypes		
	Fund studies to determine security needs[1]		

[1] This activity will task less than 1 year.

Near-Term Chart for Data Management

	87	88	89	90	91
Platform Environment	Identify prototype algorithms required within platform environment.		Begin implementing platform environment algorithms.		
	Determine applicability of parallel algorithms within platform environment.				
	Develop techniques for comparing/contrasting platform environment algorithms.				
	Perform in-depth analyses of current methods of replicating data.	Determine which method of replicating data is most appropriate for platform environment.	Implement techniques for replicating data.		
	Identify and refine requirements for platform management environment distributed DBMS.				
Platform Management Environment	Using refined requirements for platform management environment distributed DBMS, begin implementing error recovery techniques, data models, etc.				
	Select interfaces among DBMS components.				
	Select concurrency control and deadlock resolution algorithms.		Implement deadlock recovery and concurrency control algorithms.		
Software Engineering Environment	Fund basic research in data models, active components, amount of data to store issues, etc.				
	Perform in-depth analyses of current distributed SEE's.				
	Determine SDI's requirements in distributed SEE's, including security.	Begin implementing distributed SEE.			

Plan Chart for Man-Machine Interface Foundation Technology

SUBAREAS	1987 - 1990	1991 - 1995	1996 - 2000
HUMAN FACTORS	Monitor human factors research		Enforce methodology to evaluate an MMI
	Continue NRL Command and Control Human Interface Program	Fund experimentation into decision-making under stress	
		Formulate standards	
INTERACTION TECHNIQUES	Study MMI of SDI applications		
	Monitor state of practice	Adapt and integrate useful, existing interaction techniques	
	Monitor DARPA work in AI interaction techniques	Develop interaction techniques found useful but unavailable	
WORKSTATION MANAGERS	Monitor CG-VDI standard and government development efforts	Fund development of workstation/window manager or borrow technology	
GRAPHICS PACKAGES	Monitor PHIGS standard and its Ada language binding. Integrate into foundation of SDI Software Technology.		
USER INTERFACE MANAGEMENT SYSTEMS	Monitor UIMS R&D	Adapt and integrate useful, suitable, existing UIMS	
DECISION AIDS	Continue USA SDC development of Decision Aids Test Environment		

Near-Term Plan Chart for Man-Machine Interface Foundation Technology

SUBAREA	1987	1988	1989	1990	1991
HUMAN FACTORS	Monitor human factors research				
	Continue NRL Command and Control Human Interface Program				Formulate HF standards
INTERACTION TECHNIQUES	Monitor state of practice		Study MMI of SDI applications		
	Monitor DARPA work in AI interaction techniques			Adapt and integrate useful, existing interaction techniques	Develop interaction techniques found useful but unavailable
WORKSTATION MANAGERS	Monitor CG-VDI standard		Monitor government development efforts		
GRAPHICS PACKAGES	Monitor PHIGS standard and its Ada language binding		Integrate PHIGS and Ada binding into foundation of SDI software technology		
USER INTERFACE MANAGEMENT SYSTEMS	Monitor UIMS R&D				Adapt and integrate useful, suitable, existing UIMS
DECISION AIDS	Continue USA SDC development of Decision Aids Test Environment				

Plan Chart for Parallel Processing Foundation Technology

SUBAREAS	1987 - 1995	1996 - 2000
LANGUAGES	Research techniques for expressing parallelism, both explicitly and implicitly. For explicit expression of parallelism, particular attention should be paid to communication mechanisms and to how much detailed knowledge of the machine's architecture must the programmer have to program the machine effectively.	
COMPILERS	Research the methods for generating efficient code for parallel architectures from high level languages (Ada). Particular attention should be placed on the issue of static scheduling of tasks and how better to predict task behavior.	
OPERATING SYSTEMS	Particular attention should be placed on the issue of dynamic scheduling of tasks in relation to particular architectures.	
ALGORITHMS	Develop a facility for testing and evaluating algorithmic behavior on particular parallel architectures.	

Near Term Plan Chart for Parallel Processing Foundation Technology

SUBAREAS	1987-89	1990-91
LANGUAGES		
COMPILERS	Construct Ada compilers targeted at a broad range of architectures (switched shared memory, networked distributed memory, hierarchical memory)	Investigate the problems and advantages of mapping Ada to the different architectures
OPERATING SYSTEMS		
ALGORITHMS	Identify the algorithms needs that will be of interest to SDI BM/C3 applications. Develop parallel algorithms to meet these needs.	Identify which of the algorithms maps best to which architectures.

Plan Chart for Computer Systems Engineering Technology Foundation Technology

	1987 - 1990	1991-1995	1996-2000
Requirement Specifications	Evaluate work in requirements capture. Fund continuedwork or establish project(s) to build system-level requirements capture prototypes.		
Design Specifications	Evaluate work in specification languages/notations. Fund continued work or establish project(s) to build system-level specification languages/notations.		
Verification	Fund research aimed at developing automatic verification (design meets requirements and specification) capabilities		
System Design	Engineering Environments: Sponsor a tool Fair: Evaluate Existing Engineering Environments. Fund continued work or establish project(s) to build Engineering Environments supporting multiple design alternatives, multiple function to implementation assignments, multiple levels of design representation and propagation of design changes accross representations.		
	Design Tools: Sponsor a Tool Fair: Evaluate existing Design Tools. Fund continued work or establish project(s) to build design tool prototypes for automated decomposition tools, automatic assignment of functions to implementation technology tools, simulators for tradeoff analysis, and tools to transform representations such as microcode compilers and gate array compilers.		
Standardization	Requirement Specifications: Determine Level of Requirements Specification Language Standardization		
	Design Specifications: Determine Level of Design Specification Language Standardization		
	Engineering Environment: Determine Level of Engineering Environment Standardization		

C-14

Plan Chart for Computer Systems Engineering Technology Foundation Technology

	1987 - 1990	1991-1995	1996-2000
Requirement Specifications	Evaluate work in requirements capture. Fund continuedwork or establish project(s) to build system-level requirements capture prototypes.		
Design Specifications	Evaluate work in specification languages/notations. Fund continued work or establish project(s) to build system-level specification languages/notations.		
Verification	Fund research aimed at developing automatic verification (design meets requirements and specification) capabilities		
System Design	Engineering Environments: Sponsor a tool Fair: Evaluate Existing Engineering Environments. Fund continued work or establish project(s) to build Engineering Environments supporting multiple design alternatives, multiple function to implementation assignments, multiple levels of design representation and propagation of design changes accross representations.		
	Design Tools: Sponsor a Tool Fair: Evaluate existing Design Tools. Fund continued work or establish project(s) to build design tool prototypes for automated decomposition tools, automatic assignment of functions to implementation technology tools, simulators for tradeoff analysis, and tools to transform representations such as microcode compilers and gate array compilers.		
Standardization	Requirement Specifications: Determine Level of Requirements Specification Language Standardization		
	Design Specifications: Determine Level of Design Specification Language Standardization		
	Engineering Environment: Determine Level of Engineering Environment Standardization		

C-13

SUBAREAS	1987 - 1990	1991 - 1995	1996 - 2000
GENERAL	Detailed assessment of[1], and coordination with, ongoing SEE efforts	Potential implementation synergy with other government SEE efforts	
	Fund tool interface, architecture studies and definitions		
	Development of prototype SEE(s)		
		Continual refinement of SEE(s)	
METHODOLOGIES	Fund studies, and selection, of methodologies		
	Monitor state of the art		
REQUIREMENTS DEVELOPMENT	Fund R&D in specification languages, simulators, etc.		
	Fund R&D in tools for formal methods		
RAPID PROTOTYPING	Develop methodologies and tools that support rapid prototyping techniques		
	Fund R&D in executable requirements and design languages		
VISUAL PROGRAMMING	Monitor state of the art		
FORMAL VERIFICATION	Develop strategies for formal verification		
	Monitor state of the art		

SUBAREAS	1987 - 1990	1991 - 1995	1996 - 2000
CODE GENERATION/COMPILERS	Develop highly optimized code generators for parallel/distributed architecture		
TESTING	Fund studies in basic R&D (mathematical & empirical foundations)		
	Develop overall testing strategies		
METRICS	Determine relevant types of metrics[1]		
	Consistent measurement and modeling program		
	Monitor state of the art		

[1] This activity will take less than 1 year.

Near-Term Chart for Software Engineering Environment

	87	88	89	90	91
General	Detailed assessment of, and coordination with, ongoing SEE efforts				
	Fund tool interface, architecture studies, and definitions				
	Develop requirements for prototype SEE, using subareas below		Development of prototype SEE(s)		
Methodologies	Fund detailed studies of software methodologies	Select SDI software development methodology (ies)	Monitor state of the art		
Requirements Development	Fund detailed studies to determine state of art, and SDI's requirements, for specification languages		Development of specification languages		
Rapid Prototyping	Develop tools and techniques to support rapid prototyping				
Formal Verification	Fund basic research in formal verification				
Code Generation/ Compilers	Study the potential benefits of highly optimized code generators for parallel distributed architectures				
Testing	Fund studies in basic R & D (mathematical and empirical foundations)				
	Develop overall testing strategies				

Near-Term Chart for Software Engineering Environment

	87	88	89	90	91
Metrics	Determine relevant types of metrics applicable to SDI's needs	Develop consistent measurement and modeling programs			

Plan Chart for the Simulation Task Area

SUBAREA	1987 - 1990	1991 - 1995	1996 - 2000
SDI SIMULATIONS		Perform simulations to analyze performance of SDI components, architectures and strategies	Perform real-time,hardware-in-the-loop, man-in-the-loop, end-to-end, full-threat engagement simulations
SIMULATION PROJECTS	Draft a coordinated, time-phased plan for development Adapt and integrate existing SDI-related models		
SIMULATION PROGRAMMING	Choose Ada as standard simulation language Build small and large-scale simulators and simulation facilities		
SIMULATION SUPPORT	Adapt and integrate useful, existing simulation support tools Develop support tools found useful but unavailable	Standardize an Ada Simulation Support Environment	
MODEL VALIDATION	Study validation of large-scale simulators	Apply validation techniques	
DATA ANALYSIS	Standardize test methodology	Enforce test methodology	

C-20

Near-Term Plan Chart for the Simulation Task Area

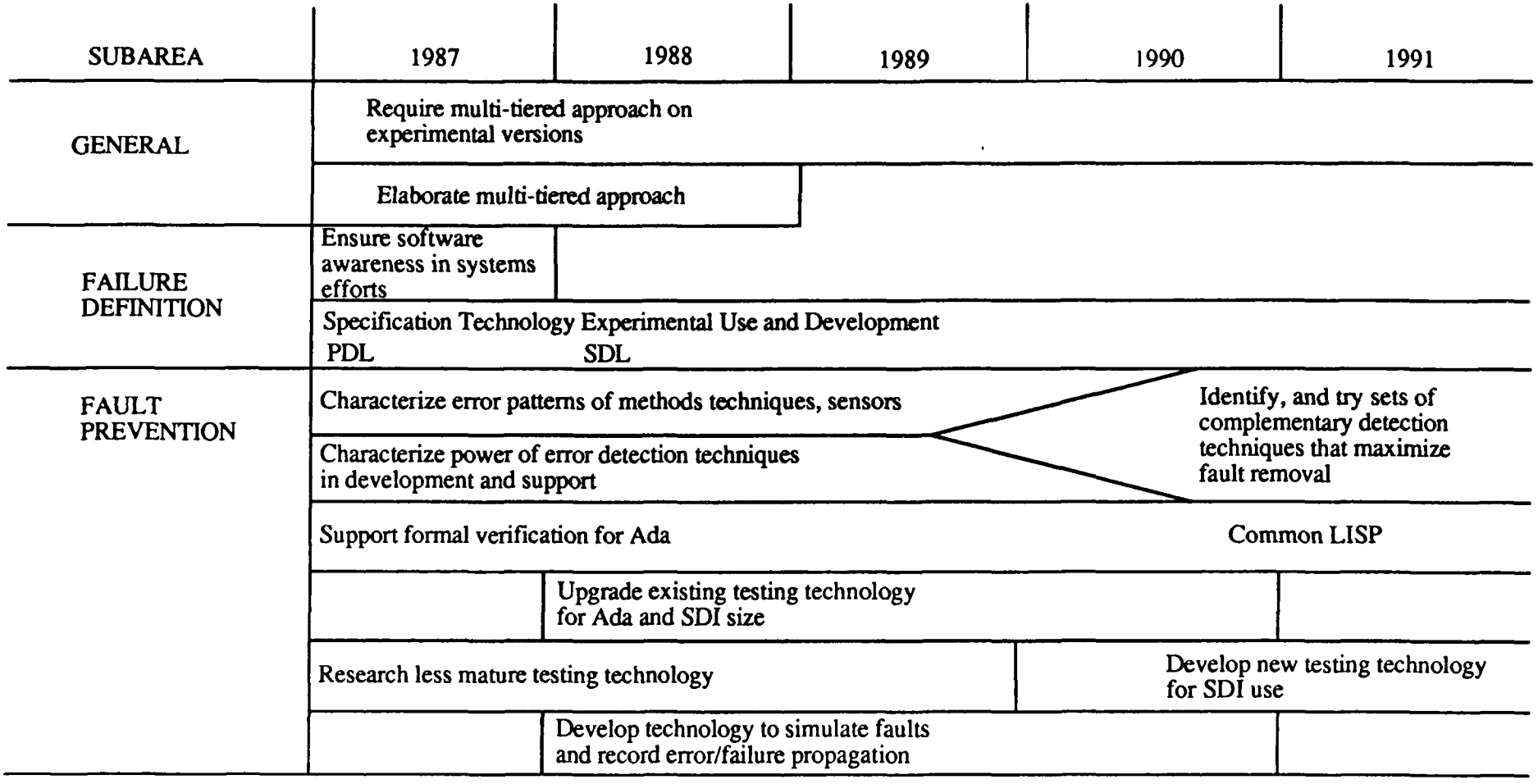
SUBAREA	1987	1988	1989	1990	1991
SIMULATION PROJECTS	Draft a coordinated, time-phased plan for development	Adapt and integrate existing SDI-related models			
SIMULATION PROGRAMMING	Build large-scale simulators (EV88)				
	Choose Ada as standard simulation language	Build simulation and test bed facilities (NTB, ASC) . . .			
SIMULATION SUPPORT		Adapt and integrate useful, existing support tools	Develop support tools found useful but unavailable	Standardize an Ada Simulation Support Environment	
MODEL VALIDATION	Study validation of large-scale simulators.		Apply validation techniques . . .		
DATA ANALYSIS	Standardize test methodology		Enforce test methodology . . .		

Plan Chart for Software Dependability

SUBAREAS	1987 - 1990	1991 - 1995	1996 - 2000
GENERAL	Require and Elaborate Maximal Multi-Tiered Approach		
FAILURE DEFINITION	Specification Technology		
FAULT PREVENTION	Characterize Error Patterns of Development and Support Characterize Power of Error Detection Techniques Formal Verification: Ada	Common Lisp	
	Testing Research & Empirical Studies		
	Upgrade Existing Testing Technology So Can Handle SDI Size Software		
	Simulate Faults and Fault-tolerant Aversion Reporting in Execution		
QUALITY MEASUREMENT AND MODELS	Quality Measurement and Modeling Experiments and Studies		
ACQUISITION/MANAGEMENT PRACTICES	Develop and Try Contractual Requirements for Software Dependability Techniques and Achievements		
FAULT TOLERANCE	Software Level Fault-Tolerance Including Abstract Machine Levels and Fault Tolerance Harness		
	System Level Fault-Tolerance		
	Software Safety and Real-time Failure Mode Analysis		
	Atomic Synchronization in Distributed Systems		
	Error/Failure Recovery Techniques		
	Autonomy and Decentralization		

C-22

Near-Term Plan Chart for Software Dependability Technology



Near-Term Plan Chart for Software Dependability (Cont'd)

SUBAREA	1987	1988	1989	1990	1991
QUALITY MEASUREMENT AND MODELS		Specification, assessment and modeling technology		Tracking and modeling technology	
		Use technology in experimental versions			
ACQUISITION/ MANAGEMENT PRACTICES	Develop contractual practices & language	Try variations on experimental versions		Incorporate in major contracts	
FAULT TOLERANCE	Software level fault tolerance: Prototype fault tolerance harnesses & experiment with mechanisms				Establish stable levels and services
	Study combining system, hardware and software fault tolerance and prototype combinations			Establish scheme for combined fault tolerance	
	Software safety research				
	SDI BM/C3 Failure Mode Analysis				
	Explore decentralization and autonomy models and architectures	Insert into software and systems efforts			
	Explore atomic action and synchronization models and options				
	Explore error/failure recovery techniques				

Plan Chart for People and Organizations

	1987 - 1990	1991-1995	1996-2000
Individual Differences	Fund studies of individual excellence		
	Monitor superdesigner studies		
	Fund studies on effect of behavioral differences on innovation		
Education and Training	Monitor SEI Activities		
		Fund courseware development to communicate applications	
		Fund upgrading of SDI Personnel; fund universities involved	
Selection	Fund research in selection ; monitor ongoing efforts		
Evaluation	Monitor SEI and STARS activities		
	Fund empirical studies of measurement and certification		
Motivation	Monitor research in motivation of computer personnel		
	Fund research in motivation of non-profit R&D organizations		
Teams	Monitor team research at MCC, SEI, and elsewhere		
Projects	Monitor research and data collection on projects at SEI and elsewhere		
	Fund human-machine studies and computer-mediated communciations research; Track work at Xerox PARC, MCC, etc.		
Organizations	Fund studies in organizational excellence		
		Implement OD model	
	Fund research in management of technological change and innovation; Monitor SEI work		
Interorganizational Relations	Monitor SEI efforts to improve the DOD business environment		
	Study contract requirements to assure that contractors select and develop effective personnel		
Organizational	Designate organization to fund, monitor, evaluate, and report on developments in this area		

C-25

Near-Term Plan Chart for Technology Transition (page 2)

<table>
<tr><th>SUBAREAS</th><th>1987</th><th>1988</th><th>1989</th><th>1990</th><th>1991</th></tr>
<tr><td rowspan="3">ORGANIZATIONAL BEFORE USE (Cont'd)</td><td colspan="5">Experiment with methods to speed software technology R&D within and between organizations.</td></tr>
<tr><td>FIXIT report; close links with other parts of SDI; area working groups and workshops</td><td colspan="2">Expand links to cover industrially funded and foreign funded R&D</td><td colspan="2"></td></tr>
<tr><td>Task area technology transition plans</td><td colspan="4">Annual Updates and Progress Reports</td></tr>
<tr><td rowspan="3">ORGANIZATIONAL USE OF TECHNOLOGY</td><td colspan="5">Contractor Technology Insertion Plans</td></tr>
<tr><td>Establish very large software systems group: NASA SS, FAA, WIS, SDI,...</td><td colspan="4">Include developers in technology R&D evaluations and technologists in "Development" R&D Evaluations</td></tr>
<tr><td>Ada PDL experiments;
Software toolsfairs;
SDL experiments</td><td colspan="4">Tools and SEEs Tryouts</td></tr>
</table>

C-26

Near-Term Plan Chart for People and Organizations

<table>
<tr><th>SUBAREA</th><th>1987</th><th>1988</th><th>1989</th><th>1990</th><th>1991</th></tr>
<tr><td>ACQUISITION</td><td>Define contract requirements, language, and practices</td><td colspan="4">Improve contractual practices</td></tr>
<tr><td rowspan="2">R&D AND EDUCATION THROUGH SOFTWARE CONTRACTOR(S)</td><td colspan="5">Establish human resource plan(s), annual updates, progress reports, and incentive payments</td></tr>
<tr><td></td><td>Use initial minimum and best practices; establish R&D scheme with accompanying variations in practice.</td><td colspan="2">R&D, experiments/trials, and measurement on selection, evaluation, motivation, certification, teams, projects, and technological change; educate, train, OJT, evaluate, staff; organize for improvement; use organizational development</td><td>Meet advanced achievement goals; continue improvement and growth</td></tr>
<tr><td>DIRECTLY FUNDED R&D AND EDUCATION</td><td></td><td colspan="4">Fund studies on individual and organizational excellance
Fund studies on behavioral differences impact on innovation
Fund upgrade of SDI-related Government, FFRDC and SETA personnel</td></tr>
<tr><td>MONITORING</td><td>Designate organization to monitor, evaluate, and report on developments in area</td><td colspan="4">Cover superdesigner studies, SEI activities, selection methods, motivation research, team research, man-machine and computer-mediated communications research, project data collection and research, and other people- and organization-related developments</td></tr>
</table>

C-27

Plan Chart for Technology Transition

SUBAREAS	1987 - 1990	1991 - 1995	1996 - 2000
TECHNICAL	Compatibility/Standards/Interfaces		
	Establish Evaluation Capabilities	Refine	
	Opportunity Scanning		
ORGANIZATIONAL BEFORE USE	Augment SEI		
	Act or Experiment and Measure		
	Linking Mechanisms Before Use		
ORGANIZATIONAL USE OF TECHNOLOGY	Technology Transition Plans & Reports		
	Linking Mechanisms to Use		
	Organizations' Technology Insertion Activities, Actions, Research, Experiments and Measurement		
ACQUISITION/CONTRACTS	RFP Requirements	Incentives	
REQUIREMENTS COMMUNICATION	Establish Requirements for FSED Decision	Refine & Communicate Requirements	
SOFTWARE ENGINEERING ENVIRONMENT DEVELOPMENT STRATEGY	Decide if Piggybacking on Particular Efforts	Transition Versions and Enhancements	

Near-Term Plan Chart for Technology Transition

SUBAREA	1987	1988	1989	1990	1991
TECHNICAL	Initial specification, design and programming notation standards; measurement standards; application-oriented standards	Initial SEE interchange standards; BM/C3 software component interfaces; simulation interfaces	SEE internal interfaces; refine, try, expand interfaces and standards	Improved and expanded versions	
	Establish independent software technology R&D evaluation capability	Establish fully integrated technology planning and evaluation capability	Refine		
	Expand and upgrade opportunity scanning	Comprehensive scanning of foreign activities and publications			
ORGANIZATIONAL BEFORE USE	Augment SEI effort on concerns of importance to SDI				

Near-Term Plan Chart for Technology Transition (page 3)

SUBAREAS	1987	1988	1989	1990	1991
ACQUISITION/ CONTRACTS	Define RFP contents and language for inclusion in all contracts and tasks		Provide incentive payments		
REQUIREMENTS COMMUNICATIONS	Describe problems with possible parallel processing solutions; establish requirements for FSED decision; circulate	Describe key problems that could benefit from software technology *innovation; circulate*	Update and circulate		
SOFTWARE ENGINEERING ENVIRONMENT STRATEGY	Decide SEE strategy and efforts to piggyback on	Provide versions for try out; ensure support	Provide enhanced versions for use; insertion aids; support		

C-33