NAVAL POSTGRADUATE SCHOOL
Monterey, California

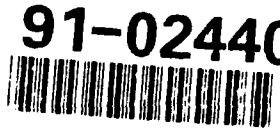
THESIS

A FRAMEWORK FOR UNDERSTANDING THE
STRATEGIC DEFENSE INITIATIVES'
SOFTWARE DEBATES

by

Reginald C. Adams

Thesis Advisor: Donald Lacer



UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

Form Approved
OMB No 0704-0188

1a REPORT SECURITY CLASSIFICATION: Unclassified	1b RESTRICTIVE MARKINGS
2a SECURITY CLASSIFICATION AUTHORITY	3 DISTRIBUTION/AVAILABILITY OF REPORT: APPROVED FOR PUBLIC RELEASE; DISTRIBUTION IS UNLIMITED
2b DECLASSIFICATION/DOWNGRADING SCHEDULE	
4 PERFORMING ORGANIZATION REPORT NUMBER(S)	5 MONITORING ORGANIZATION REPORT NUMBER(S)
6a NAME OF PERFORMING ORGANIZATION: Naval Postgraduate School; 6b OFFICE SYMBOL (If applicable): CODE CC	7a NAME OF MONITORING ORGANIZATION: Naval Postgraduate School
6c ADDRESS (City, State, and ZIP Code): Monterey, CA 93943-5000	7b ADDRESS (City, State, and ZIP Code): Monterey, CA 93943-5000
8a NAME OF FUNDING SPONSORING ORGANIZATION; 8b OFFICE SYMBOL (If applicable)	9 PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER
8c ADDRESS (City, State, and ZIP Code)	10 SOURCE OF FUNDING NUMBERS: PROGRAM ELEMENT NO; PROJECT NO; TASK NO; WORK UNIT ACCESSION NO

11 TITLE (Include Security Classification)
A FRAMEWORK FOR UNDERSTANDING THE STRATEGIC DEFENSE INITIATIVES' SOFTWARE DEBATES

12 PERSONAL AUTHOR(S)
ADAMS, REGINALD C.

13a TYPE OF REPORT	13b TIME COVERED	14 DATE OF REPORT (Year, Month, Day)	15 PAGE COUNT
Master's Thesis	FROM _____ TO _____	JUNE 1990	97

16 SUPPLEMENTARY NOTATION
The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government

17 COSATI CODES			18 SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
FIELD	GROUP	SUB-GROUP	STRATEGIC DEFENSE INITIATIVE (SDI), RELIABILITY, ERROR FREE, BALISTIC MISSILE DEFENSE (BMD) (BM/C3)

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

On March 23, 1983, then-President Ronald Reagan challenged a group of engineers and scientists to make nuclear weapons "impotent and obsolete." This challenge led to the beginning of a new era in space technology and strategic defense, thus creating the Strategic Defense Initiative (SDI), better known as "Star Wars." By 1984, several studies had begun to show that software in conjunction with Battle Management/Command, Control, and Communications techniques would play a major role in determining the effectiveness of the SDI. The results from these studies caused numerous controversial debates on the reliability, dependability, and trustworthiness of the software. This thesis provides a framework for understanding the complexities of the SDI software and points out some of the major issues involved in the software debates. The structure for this thesis is based on presenting the opinions of various computer scientists and engineers, indicating the issues that are controversial and those that have been defined as a necessity for the SDI program. One of the major highlights is the SDI summary chart that provides the reader with a very brief narrative of each individuals' opinion on the software issues discussed in this thesis.

20 DISTRIBUTION/AVAILABILITY OF ABSTRACT	21 ABSTRACT SECURITY CLASSIFICATION
☒ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT ☐ DTIC USERS	unclassified
22a NAME OF RESPONSIBLE INDIVIDUAL: PROF. DONALD LACER	22b TELEPHONE (Include Area Code): (408) 646-2772 / 22c OFFICE SYMBOL: CC/La

DD Form 1473, JUN 86 — Previous editions are obsolete — SECURITY CLASSIFICATION OF THIS PAGE

S/N 0102-LF-014-6603

UNCLASSIFIED

Approved for public release; Distribution is unlimited

A FRAMEWORK FOR UNDERSTANDING THE
STRATEGIC DEFENSE INITIATIVES'
SOFTWARE DEBATES

by

Reginald C. Adams
Captain, U.S. Air Force
B.S., Mississippi Valley State University, 1984

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN SYSTEMS TECHNOLOGY
(COMMAND, CONTROL AND COMMUNICATIONS)

from the

NAVAL POSTGRADUATE SCHOOL

JUNE 1990

Author: Reginald C. Adams
Reginald C. Adams

Approved by: Donald A Lacer
Donald Lacer, Thesis Advisor

Prof. Ludi, Second Reader

Carl Jones, Chairman
Department of Joint, Command,
Control and Communications

ABSTRACT

On March 23, 1983, then-President Ronald Reagan challenged a group of engineers and scientists to make nuclear weapons "impotent and obsolete." This challenge led to the beginning of a new era in space technology and strategic defense, thus creating the Strategic Defense Initiative (SDI), better known as "Star Wars." By 1984, several studies had begun to show that software in conjunction with Battle Management/Command, Control, and Communications techniques would play a major role in determining the effectiveness of the SDI. The results from these studies caused numerous controversial debates on the reliability, dependability, and trustworthiness of the software. This thesis provides a framework for understanding the complexities of the SDI software and points out some of the major issues involved in the software debates. The structure for this thesis is based on presenting the opinions of various computer scientists and engineers, indicating the issues that are controversial and those that have been defined as a necessity for the SDI program. One of the major highlights is the SDI summary chart that provides the reader with a very brief narrative of each individuals' opinion on the software issues discussed in this thesis.

TABLE OF CONTENTS

I. INTRODUCTION 1

II. MAGNITUDE OF THE SDI SOFTWARE PROBLEM 7

A. INTRODUCTION 7

B. NATURE OF THE BALLISTIC MISSILE DEFENSE SYSTEM (BMD) 7

C. SIMULATION: DESIGN AND TESTING 9

D. SOFTWARE DEVELOPMENT PHASES 11

E. ROLES OF SOFTWARE IN BMD 13

III. ANALYSIS OF THE SDI SOFTWARE DEBATES; WHAT THEY DISAGREE ON 17

A. INTRODUCTION 17

B. WHY SDI SOFTWARE MAY BE UNRELIABLE 17

C. DEFINING TRUSTWORTHY: THE PARNAS VIEWPOINT 28

D. VIEWS OF WHY SDI SOFTWARE WILL NOT BE ERROR-FREE 31

IV. ANALYSIS OF THE SDI SOFTWARE DEBATES; WHAT THEY AGREE ON 37

A. INTRODUCTION 37

B. THE NEED FOR A SPECIFIC BATTLE MANAGEMENT/C3 ARCHITECTURE PRIOR TO DEVELOPMENT 37

C. THE NEED FOR A DEDICATED NATIONAL TEST BED . . 43

D. THE STRATEGIC DEFENSE SOFTWARE CENTER 48

V. SUMMARY OF THE SDI SOFTWARE DEBATES 51

A. INTRODUCTION 51

B. EVALUATION OF THE SDI SUMMARY CHART 51

1. Reliability and Trustworthiness 53

2. Availability 53

3. BM/C3 Architecture 54

4. Testability 55

VI. DEVELOPMENTAL CONCERNS AND TRADEOFFS ASSOCIATED WITH SDI SOFTWARE 57

A. INTRODUCTION 57

B. SDIO SOFTWARE CHALLENGES 58

C. THE USE OF SOFTWARE ENGINEERING TOOLS FOR SDI SOFTWARE DEVELOPMENT 62

D. THE SIMULATION AND TESTING TOOLS AVAILABLE . . 69

E. SECURITY FOR SDI 76

VII. CONCLUSIONS 82

LIST OF REFERENCES 86

INITIAL DISTRIBUTION LIST 89

I. INTRODUCTION

Computer technology has advanced very rapidly since the late 1970s. In particular, computer hardware has allowed engineers to develop smaller, faster machines for more advanced computing power. In contrast, computer software has lagged behind in advanced technology. Software development is one of the major keys to the success of a Ballistic Missile Defense (BMD) system. Consequently, the Strategic Defense Initiative Organization (SDIO) has been tasked with finding ways to explore new software engineering techniques. These new developments will enhance U.S. chances of protecting itself from nuclear destruction.

The purpose of this thesis is to analyze the SDI software debates that began in 1985 and continue even today. This thesis will, therefore, provide a framework which will allow individual readers to understand the complexity of the SDI software problem. Many computer scientists have put forth a variety of opinions, but these ideas and opinions have never been compared so that interested personnel from academia, the military, and industry would have an understanding of the SDI software and its complexities.

In the early 1950s nuclear armed ballistic missiles represented an advance in weapon delivery technology. During

that time the United States was vulnerable to missile attack. To counter these missile attacks, many anti- ballistic concepts were conceived; in particular, the SAFEGUARD ballistic missile defense system. It was developed in the mid 1970s to serve as the answer to the Soviet Unions' nuclear threat. Further analysis showed that (1) multiple, independent target warheads from a single booster could saturate defenses, and (2) our ballistic missile inventory proved inadequate against a superior Soviet force. [Ref. 1]

SAFEGUARD was later disbanded and the Soviet threat is still ever present. The Strategic Arms Limitation Treaties (SALT) provided a limited reduction in the arsenals of both the United States and the Soviet Union.

The concept of ballistic missile defense found a new spark in the 1980s under the Reagan administration. In March 1983, President Ronald Reagan tasked scientists to free us from the fear of nuclear weapons by making them "impotent and obsolete." This new outlook on strategic defense became known as the Strategic Defense Initiative (SDI) or "Star Wars." What is SDI? According to President Reagan, SDI is the development of an array of satellites carrying sensors, weapons, and computers to detect ICBMs and intercept them before they can do much damage [Ref. 2:p. 2]. The purpose of this concept was made clear in 1985 when the president stated:

> [SDI's] purpose is to identify ways to exploit recent advances in ballistic missile defense technologies that have potential for strengthening deterrence and therefore

increasing our security and that of our allies. [Ref. 2: p. 2]

The Strategic Defense Initiative Organization (SDIO) is planning a three part phased development for ballistic missile defense. Each phase builds incrementally on the previous, capitalizing on the benefits of the preceding phase. The SDIO and their contractors have studied several options for the first phase. The basic purpose of this phase would be to compel Soviet operational adjustments and compromises, thus reducing the confidence of Soviet planners. The second phase would enhance deterrence by imposing uncertainty on the Soviet strategic attack plans. A follow-on section of this phase would deny the Soviets' the ability to destroy "militarily significant" targets (e.g., Missile Silos and Command and Control Centers). The third phase would be aimed at dominating the threat posed by nuclear ballistic missiles. That phase would be completed either by the Soviets choosing to rid themselves of obsolescent missiles by negotiating arms reduction or by engaging in an offensive-defensive arms race that could prove costly and leave both the Soviets and the U.S. financially handicapped. Figure 1.1 gives an overview of the phase I architecture for SDI. This architecture shows the different surveillance and tracking systems, radars, and interceptors that would be used from boost phase to terminal phase. Table I gives a description of the functions of the system elements in the phase I architecture. The reader should

note that there is no description of the brilliant pebbles function, which replaces the Space Based Interceptor and is a very recent innovative, space kinetic energy concept. [Ref. 3:pp. 2-6]

Many supporters of SDI are very optimistic concerning the technologies used to deter Soviet Aggression. Although many areas lack confidence, scientists feel the 1990s brings promise of the reality of an SDI deployment.

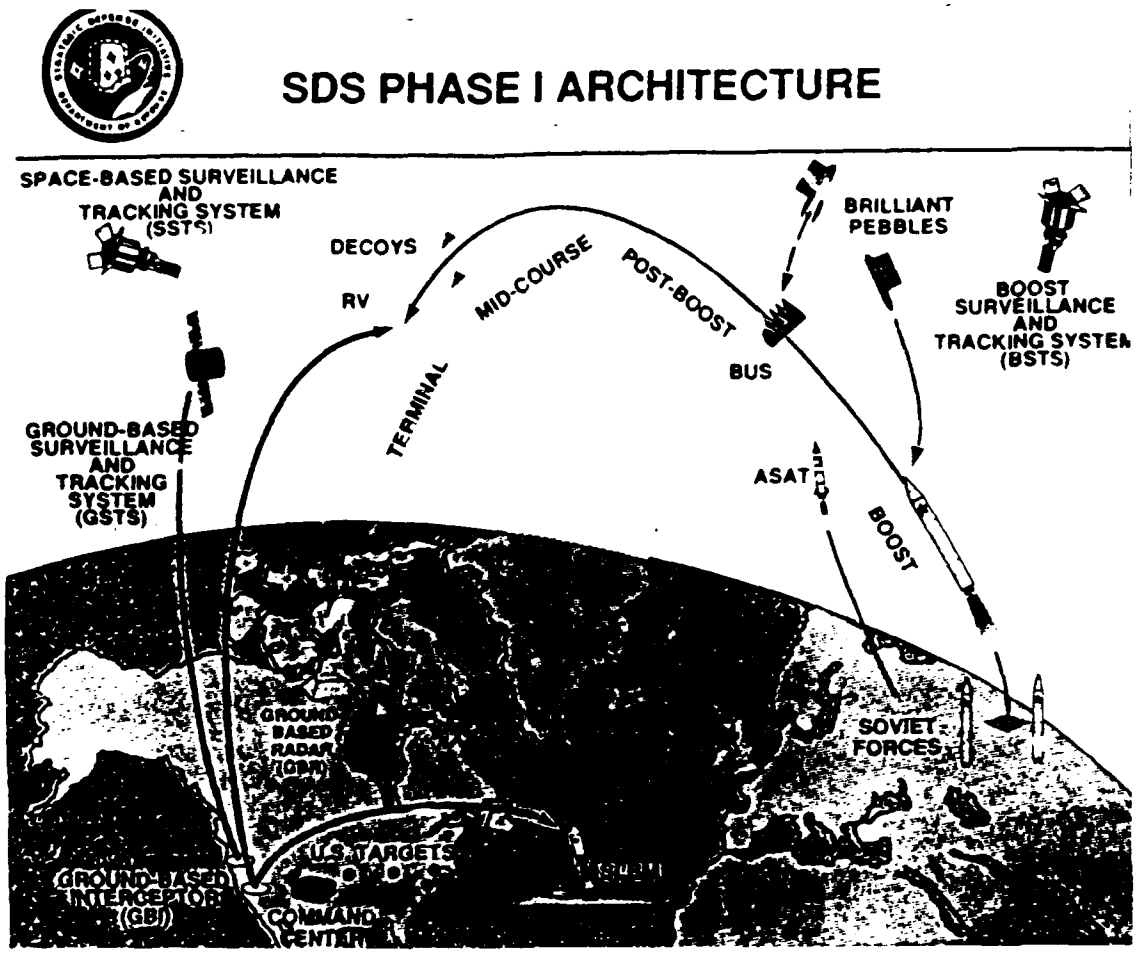

SDS PHASE I ARCHITECTURE
SPACE-BASED SURVEILLANCE AND TRACKING SYSTEM (SSTS)
DECOYS
RV
MID-COURSE
POST-BOOST
BRILLIANT PEBBLES
BOOST SURVEILLANCE AND TRACKING SYSTEM (BSTS)
BUS
TERMINAL
GROUND-BASED SURVEILLANCE AND TRACKING SYSTEM (GSTS)
ASAT
BOOST
GROUND BASED RADAR (GBR)
SOVIET FORCES
U.S. TARGETS
GROUND-BASED INTERCEPTOR (GBI)
COMMAND CENTER

Figure 1.1

TABLE I.

Elements of Initial System

SYSTEM ELEMENTS	PRIMARY FUNCTIONS
Boost Surveillance and Tracking System (BSTS)	• Detection of missile launches • Acquisition and tracking of boosters • Booster kill assessment
Space-Based Surveillance and Tracking System (SSTS)	• Acquisition and tracking of post-boost vehicles and reentry vehicle clusters, ASATs, and satellites • Kill assessment
Ground-Based Surveillance and Tracking System (GSTS)	• Closely spaced object resolution • Tracking of reentry vehicles and penetration aids • Discrimination of reentry vehicles from penetration aids • Kill assessment
Ground-Based Radar (GBR)*	• Acquisition and tracking • Discrimination of reentry vehicles from penetration aids
Ground-Based (Exoatmospheric) Interceptor (GBI)	• Destruction of reentry vehicles in late midcourse
Space-Based Interceptor (SBI)	• Destruction of boosters, post-boost vehicles, and ASATs • Destruction of reentry vehicles in early midcourse
Command Center (CC)	• Human decisionmaking • Communications • Battle plan execution • Guidance for system operation and integration functions (SOIF)

* GBR is currently under consideration for inclusion in Phase I.

II. MAGNITUDE OF THE SDI SOFTWARE PROBLEM

A. INTRODUCTION

The focus of this chapter is on some of the major software issues that directly affect the success of SDI. The main issue that is unique to SDI is its physical size. According to the Office of Technology Assessment (OTA), the software required for the SDI program far exceeds that of any of its predecessors. Other factors discussed in this chapter are the use of software development tools and software engineering practices. Finally, design and testing efforts are discussed in relation to the simulation methods for SDI.

B. NATURE OF THE BALLISTIC MISSILE DEFENSE SYSTEM (BDM)

In the massive conglomeration of satellites, weapons, and sensors required for a strategic defense system lies what many supporters of SDI call the catalyst of the entire system. That catalyst is the software necessary to integrate the sensors and weapons. The SDI software problem is difficult because of the many complexities involved. They include areas such as simulation and testing, computer architectures, reliability, maintainability, compatibility and interoperability, to name a few.

One reason for the confusion over software design and development stemmed from the alternate interpretations of President Reagans' famous "Star Wars" speech. Based on the interpretation of Dr. David Parnas, former member of the Eastport Study Group, the only way to make nuclear weapons impotent and obsolete will be to eliminate the nuclear arsenals of both the United States and the U.S.S.R. The Eastport Study group, chaired by Dr. Danny Cohen, share an alternate interpretation. Dr. Cohen believes that any goals reached toward ultimate ballistic missile defense will be a major accomplishment. Views of Dr. Parnas and Dr. Cohen will be discussed more in Chapters III and IV.

What makes the software such an integral part of the ballistic missile defense system? The main reason is that computer technology hardware and software must be adequate to control individual weapons and sensors and coordinate their operation. Battle Management and Command and Control (C2) are the two most important factors for SDIs' success. The C2 system must receive and act on information pertaining to thousands of missile launches, tens of thousands of warheads, and hundreds of thousands of decoys. To complicate matters, all this coordination must be done within the time it takes Soviet Intercontinental Ballistic Missiles (ICBM) to travel from their launch sites to U.S. targets. This time is estimated to be about 30 minutes. Therefore, any human intervention to correct errors will be minimal. [Ref. 4:p. 46]

Managing an enhanced defense against ballistic missiles presents problems of unprecedented complexity. In 1983, the Defense Technology Study Team (DTST) examined the technologies needed for ballistic defense. The study, better known as the Fletcher Study, concluded that the development of software for a battle management system will be a task that exceeds in complexity and difficulty any that has been accomplished in the protection of civil or military war systems. The estimate for written software instruction at that time was 10 million lines. According to some analyst, that figure has more that quadrupled, exceeding 40 million lines of executable code. [Ref. 5:p. 17]

C. SIMULATION: DESIGN AND TESTING

One of the many questions that analysts in the space program are asking is "How can the system be tested" or "When is the system operating correctly." Without a doubt SDI software developers are asking some of the same questions. Because of its complexity, the BMD system has to be tested from the research phase to the deployment phase. On a broad basis, testing should occur at four levels They include research, component and subsystem testing, system level testing, and deployed hardware and software testing. Research is literally nothing more than achieving an understanding of the process involved. Component and subsystem tests provide designers with a sense of validation or lack of validity for

a given approach to a system or subsystem. System level testing is done to get rid of operational bugs and measure system performance against its specifications and its operational requirements. Finally, deployed hardware and software testing brings a realistic environment into focus. [Ref. 6:p. 1605]

The Strategic Defense System (SDS) will contain many algorithms performing functions for surveillance, tracking, discrimination, weapons assignment, weapons control and guidance, network routing and control, security-access control, system fault tolerance, and fail-safe mechanisms. These algorithms are crucial to the identification and assessment of missile threats and the performance of sensors and weapons. This requires testing to be done on and end-to-end basis using the actual software code instead of an emulated version. The algorithms must be tested to find out how well the chain is functioning (i.e., tracing action from threat to sensor detection and track association to track correlation to track prediction. ...). There is obviously no one solution to the "testing problem, but scientists are measuring system performance through simulation. But, the simulation capability must be credible. Credibility is established when acceptable testing results are provided to those testing the system. SDIO recognized this need and organized the National Test Bed (NTB) in 1988. The author will discuss the NTB in more detail in Chapter IV.

According to the Eastport Study Group, simulation facilities must have the capability of modeling the components of the SDS, and its potential threats to answer questions like "Are the battle management strategies embedded in the BM/C3 software adequate to cope with possible/potential attacks?" The simulation technique utilized to replace simulated hardware or software functions is known as "In-line" testing. With this technique engineering developments are accelerated by allowing the insertion of system components and prototypes into a realistic environment, thus detecting problems much earlier that with traditional methods. [Ref. 2:p. 29]

D. SOFTWARE DEVELOPMENT PHASES

The development of a comprehensive defense system depends very heavily on the software controlling it. If the software is faulty, it is inevitable that the system will fail. BMD software has four crucial areas that are critical factors in attaining the objectives for which the system is designed. These include: planning, design, implementation, testing and debugging. It should also be noted that these phases are not always sequential and some overlap will occur.

The first crucial phase for developing SDI software is planning. Planning involves stating the necessary specifications and requirements to accomplish a mission. These elements must be bounded in some manner to give certain definition to the problem. The precise specification of what

the ballistic missile defense must do is a complicated task. For instance, if the mission is to shoot down Soviet missiles, knowing only Soviet missiles exist, then this problem is relatively simple. Unfortunately SDI will operate in a more complicated environment. For example, how can Soviet missiles be distinguished from non-Soviet missiles or what if a Soviet missile is headed for a target in East Germany? The developer will therefore try to predict every contingency in order to decide how the software should respond. A number of possibilities and circumstances may not be foreseeable, but the developer must somehow properly plan for these contingencies [Ref. 4:p. 48].

The design phase for SDI software development is a large task in its own right. In particular, the developers must implement the specifications and requirements defined in the first stage (planning) through the use of computers. This is where all the algorithms must be incorporated and sequences of action established.

Based on the experiences of other major military systems, integration of individual components generates problems when the entire system must operate near its limit. For instance, during an exercise using the World Wide Military Command and Control System (WWMCCS), a communication network used by military and civilian authorities for message transaction to and from the field, the message transmission rate was satisfactory when the system was in routine operation. On the

contrary, when additional regional centers were connected to the WWMCCS, the system performance was degraded tremendously. [Ref. 4:p. 48]

Simulation and testing is usually used to solve problems on a small basis, but they are limited with respect to large, interconnected systems such as the BMD system. Further discussion of this issue will occur in Chapter VI.

The final phase involves debugging real time software. Debugging can account for almost 70 percent of the total life cycle cost of a software development project. Two salient points that make software maintenance difficult are: 1) Errors don't reoccur easily, and 2) Errors found after a system becomes operational must be eliminated. To add even more complexity to this problem, analysts state that the probability of introducing an error while eliminating a known error ranges from 5 to 50 percent. [Ref. 4:p. 50]

These four phases indicate that software development for a BMD system is critical. The author believes that scientists and engineers have a lot of ground to make up in terms of finding viable solutions to make the Reagan administrations' prophecy a reality.

E. ROLES OF SOFTWARE IN BMD

Ballistic Missile Defenses can't perform without computers, just as computers can't perform without software. Software is nothing more that a set of sequential instructions used to

direct the actions of computers. The software, because it is expected to perform continuously with or without failures, must be reliable and trustworthy.

There are those who support or oppose the easibility of the development of the software based on their technical expertise. One unique requirement for SDI is that the software driving the BMD would have to work reliably the first time it's used in battle. Dr. David Parnas wrote the following on software reliability:

> People familiar with both software engineering and older engineering disciplines observe that the state of the art in software is significantly behind that in other areas of engineering. When most engineering products have been completed, tested, and sold, it is reasonable to expect that product design is correct and that it will work reliably. With software products, it is usual to find that the software has major "bugs" and does not work reliably for some users. These problems may persist for several versions and sometimes worsen as the software is "improved" [Ref. 7:p. 433]

Some computer scientists and engineers believe that software development is advanced enough to make the SDI task tractable. Dr. Frederick P. Brooks, for example, stated:

> I see no reason why we could not build the kind of software system that SDI requires with the software engineering technology that we have today." [Ref. 8: p. 284]

There are several positions on the feasibility of SDI software development; therefore, it is important to understand these positions when analyzing the roles of software in BMD.

According to an Office of Technology Assessment (OTA) report in 1988, software for BMD would be expected to:

1. Be the agent of system evaluation, permitting changes in system operation through reprogramming of existing computers.

2. Perform most complex tasks in the system, such as battle management.

3. Be responsible for recovery from failures, whether they are hardware or software failures.

4. Respond to threats, both anticipated and unanticipated against the system.

These roles or functions of software are just a few among many that software technology must contend with. Futhermore, BMD software would be more complex than any previously built. A conclusion from the Fletcher Study stated:

> 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. [Ref. 9:p. 4]

In this author's opinion, DoD has found that producing C3 and information processing software for weapon systems is not an easy task. There are many similarities between a BMD and today's typical C3 systems, such as weapons guidance, targeting, and real time communication control, but the differences are enormous. They include very little human intervention, much larger battle spaces to manage, operating in a nuclear environment, and many others. BMD software is critical to successfully defending the U.S. and its allies against any nuclear threats. Because of complexity and other software issues, it is very important that requirements be

stated specifically before the development of any software modules.

III. ANALYSIS OF THE SDI DEBATES: WHAT THEY DISAGREE ON

A. INTRODUCTION

This chapter describes the different views of some noted software engineers, professors, analysts, and computer scientists on the reliability issues concerning SDI software. The opinions are those of individuals with first hand knowledge and interest in the SDI program. Among them are: Dr. Danny Cohen, Chairmen of the former Eastport Study Group; Dr. Frederick Brooks, Professor of Computer Science at the University of North Carolina; Dr. David Parnas, Professor of Computing and Information Systems at the University of Victoria, B.C.; Dr. Solomon J. Bushbaum, Executive Vice President for Customer Systems at AT&T Bell Laboratories, and C.A. Zraket, President and CEO of the Mitre Corporation.

The author will try to provide the reader with a framework for understanding some of the complexities associated with designing and developing reliable, trustworthy, error-free code.

B. WHY SDI SOFTWARE MAY BE UNRELIABLE

Defining reliability as it applies to SDI software is difficult because of the extreme demands on the system and the inability to operationally test those demands. The software

will be responsible for ensuring that the BM/C3 systems can bind together a system of worldwide sensors, weapons, platforms, and communication links. Knowing that a task of this magnitude must be completed in a matter of minutes, reliable software is important for a successful strategic defense. [Ref. 2:p. 1]

In 1985, Dr. David Parnas resigned from the panel on computing in support of battle management known as the Eastport Study Group. He felt many of the computer science problems could not be solved prior to the development of an Anti-Ballistic Missile (ABM) system. Dr. Parnas later wrote eight articles that supported his resignation. Among the noted papers were; "Why software is unreliable," "Why the SDI Software System will be untrustworthy," and "The Limits of Software Engineering Methods." [Ref. 7:p. 2]

Dr. Parnas does not apply the term reliability towards a strategic defense system as he does to other programs. Dr. Parnas appeared before the U.S. Senate Subcommittee on Strategic and Theater Nuclear Forces in December 1985. There he stated that reliable software could indeed be built if all the requirements are known and the software can be controllable as well as predictable. These factors can be determined for a system such as the telephone system, but they don't apply well to SDI. First, tools such as mathematical analysis and exhaustive case analysis are useless in some sense because the requirements for the software functions are

controlled by strategies and tactics of the Russians. Secondly, redundancy, which is considered by some authorities to be the key to reliability, is very expensive in space. Redundancy is good for independent component failures, but design failures pose a bigger problem. For instance, if redundancy is implemented in a section of code that already contains a design flaw, we've given ourselves a "double whammy." These tools are great for mathematical analysis that work on continuous functions, but software is a discrete state system that cannot be described by these functions. [Ref. 8:p. 286]

Dr. C.A. Zraket, in his article on "Uncertainties in Building a Strategic Defense" stated that the feasibility of designing and building reliable software for a BMD system depended largely on (1) trustworthiness - known, predictable effectiveness, and freedom from "catastrophic flaw," (2) fault tolerance - ability to continue functioning coherently when part of the system is damaged, and (3) information security - ability to prevent programs from being exploited by using "trusted" programs and coding and authentication techniques. To sum up the problem of reliability, Dr. Zraket stated the following:

> Much more research and a lengthy development effort are needed in dealing with the operational design and implementation of a specific BMD architecture before a conclusive judgement can be made about the reliability of software for this space-based system. [Ref. 6:p. 1605]

Everyone realizes that the SDI system is faced with reliability problems, but how reliable must the software be or as stated by Dr. Frederick Brooks, "How good is good enough?" For instance, if 10,000 nuclear missiles were fired at the United States and our system operated effectively at 99.9 percent, 10 nuclear bombs would still bombard our country. Not good! Figure 3.1 gives the reader some idea of the number of errors created and later found and fixed (Rayleigh Curve). This curve is intended to be a rough representation of a subsystem for SDI. The curve was projected on the basis of a real-time C2 system consisting of 1 million lines of source code. The x-axis represents the time in months while the y-axis represents the number of errors per month. There are several milestones that occur during the development and operation phases. The sequential stages are represented by the vertical lines and are defined as follows:

1. Preliminary Design Review (PDR)
2. Critical Design Review (CDR)
3. First Code Complete
4. System Integration Test
5. User-operational Capability
6. Initial Operational Capability
7. Full Operational Capability (95 percent reliability level)
8. 99 percent reliability level, and
9. 99.9 percent reliability level. [Ref. 10:p. 63]

Expected Error Rate

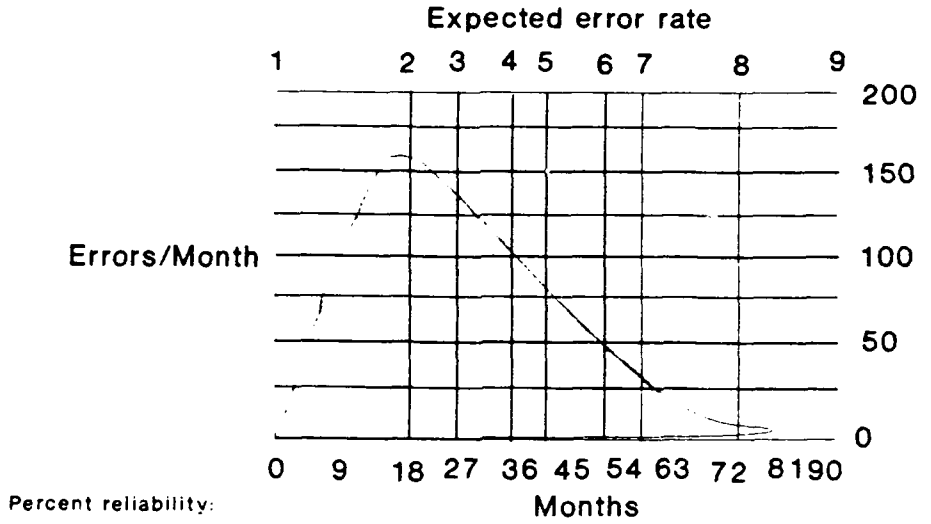


Figure 3.1

Due to the vast size and complexity of SDI, it would take approximately 7.5 years to reach the 99.9 percent reliability level. Figure 3.2 shows the total expected errors. Even at a 99.9 percent reliability level there would still be over 6,000 cumulative errors. (Keep in mind that this is the case for 1 million lines of code.) [Ref. 10:p. 63] At the third annual IEEE Conference on Strategic Software, the number of lines of code to operate SDI was up to 20 millions lines, thus increasing the concern and need for reliable software systems. [Ref. 11]

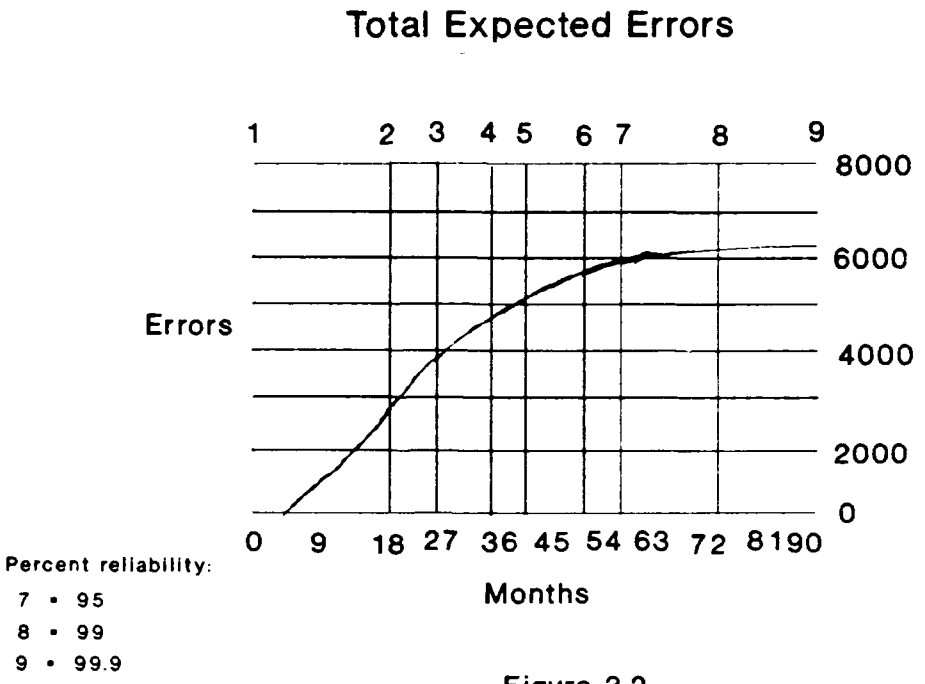


Figure 3.2

SDI software may have a reliability problem, but the real question is how to evaluate the systems' dependability. Other questions are what are the characteristics of a reliable system, and how much emphasis should be placed on each characteristic for evaluating the system? The Office of Technology Assessment (OTA) addressed these issues and labeled the following commonly considered characteristics:

1. Correctness - whether or not the software satisfies its specification
2. trustworthiness - probability that there are no errors in the software that will cause the system to fail catastrophically

3. Fault tolerance - either in prevention (i.e., capability of the software to prevent a failure despite the occurrence of an abnormal or undesired event - or failure of recovery

4. Availability - probability that the system will be available for use

5. Security - resistance of the software to unauthorized use, theft of data, and modifications of programs

6. Error incidence - number of error in the software, normalized to some measure of size

7. Safety - prevention of human life and property under specified operational conditions

These characteristics are critical to system reliability. The process of instilling reliability into SDI software must begin at the software development phase. It is very difficult to "add in" reliability after the software design is complete. As stated earlier, this can lead to exponential cost increase. [Ref. 12:p. 288]

Even though there is a lack of ways of quantifying confidence in software, people trust computerized systems. For instance, no one thinks about the probability of a disastrous error occurring in the software of an automated teller machine. This attitude allows scientists to start the development of systems without having all of the reliability issues resolved. The more confidence that is gained in a system, the less the resistance that will be shown in accepting that system. There are a few entities that characterize dependable software systems. They include 1) extensive use and abuse of the system, 2) predictable

environments, 3) low cost of failures, and 4) stable requirements [Ref. 12:p. 235-236]. A strategic defense system exhibits none of h these entities.

The extensive use of any system, whether it be an automobile or a computer system, may be the most important factor for building confidence in that system. This confidence comes from extensive use and abuse. Simulations cannot test extensive use and in many cases real world complications are expensive and poorly understood. The software for SDI falls within a similar category because it can only be tested end-end during an actual battle. [Ref. 12:p. 45] In essence the first time may be the only time. Dr. Solomon Bushbaum believes that most, if not all, of the essential attributes of the BM/C3 systems have been demonstrated in comparable terrestrial systems, namely, the U.S. Public Telecommunications Network. [Ref. 8:p. 275] Although the telephone system has been extensively "used and abused," it is not a weapon system. Figure 3.3 shows a comparison of the characteristics for SAFEGUARD, SDI, and the phone system with respect to dependability. In a letter to Congress from designers and maintainers at AT&T Bell Labs, they stated:

> "Despite rigorous test, the first time new equipment is incorporated into the telephone network, it rarely performs reliably. [Ref. 12:p. 243]

If the behavior of the software can be predicted in an operational environment confidence can be gained because the environmental factors are known. This is the case with the

Characteristics of Dependable Systems
Applied to
SDI, SAFEGUARD, AND THE TELEPHONE SYSTEM

Characteristic	SDI	SAFEGUARD	Telephone System
Extensively used & abused.....	No	No	Yes
Predictable environment.....	No	No	Yes
Low cost of a failure.....	No	No	Yes
Stable requirements.....	No	Yes	Yes
Well-understood predecessor.....	No	Yes	Yes
Simple design.....	UKN	?	Yes
Disciplined development.....	UKN	Yes	Yes

Source: OTA

Figure 3.3

telephone system where mathematical models can be used to measure the amount of traffic on a switching circuit. On the contrary, in a BMD environment, nuclear background and countermeasures will not be predictable prior to the battle. [Ref. 8:p. 345] Therefore, one's confidence in a system may be lessened by the unknown factors.

All software systems inevitably experience software failures. Users lack confidence in these systems if the risk associated with gaining information is more than that of losing it. In the case of the phone system, the ability to recover from a failure at a low cost increases the users

willingness to use the system more. Besides, no major war will be won or lost because a phone call cannot be recovered or reconnected. On the contrary, software for BMD systems must be reliable. There will be no time to repair errors during a battle. If the error is "catastrophic," many thousands of warheads will reach their targets. According to an OTA report concerning technology for preventing catastrophic failure, the following statement was made:

> OTA found no evidence that the software engineering technology foreseeable in the near future would make large improvements in the dependability of software for BMD systems. In particular there would be no way to ensure that BMD software would not fail catastrophically when first used. [Ref. 12:p. 246]

Finally, defining all the requirements is not feasible, mainly because the threats, the strategies, countermeasures, and technologies are in a constant evolution. In preparation for writing this thesis, most software analysts and engineers when asked "What is the toughest problem facing SDI software development," the majority stated: changing requirements. This is not a new issue. The problem has plagued some of our other major weapon systems, in particular the B-1 bomber program. Many changes were incorporated during the development stage which resulted in major deficiencies. According to a report on the B-1 bomber:

> Defense officials blame many of the program's problems on the decision to begin producing the aircraft at the same time that research and development efforts were underway, forcing engineers to experiment with some systems before they were completely developed. [Ref: 13: p. A1]

Stability in software has a special importance because of the many decisions involved in the design. The earlier a decision is made, the harder it is to correct later in the process. The author believes that the key is to have little change during the development process. This process increases the confidence in the system and keeps cost at a minimum.

The previous discussion dealt with issues that were deemed necessary for reliable software systems. Whether or not technology is able to accomplish this feat is another question. Supporters of the SDI program lean towards the answers from the Eastport Study Group and various other reports that have been submitted to Congress. Dr. Yale J. Lubkin, director of advance technology for a major EW manufacturing company, compared reliability of software systems to that of the human brain. Dr. Lubkin stated that the brain is a discrete state system that works with continuous state components to produce complex sensor/logic systems. He also stated that these systems work well on a scale such as an "inchworm." For example, as the worm rotates its sensor, searching to define the safe boundaries of its domain, it can learn to proceed rapidly on a safe path without reaching a boundary of discontinuity. Therefore, he concludes that the implication that software systems will always be unreliable is not true. It is obvious that these debates have very valid positions on both sides. The winner may not be determined until the battle is over. [Ref. 28:p. 12]

C. DEFINING TRUSTWORTHY: THE PARNAS VIEWPOINT

In the previous section reliability was discussed as a measure of system behavior. As discussed earlier, reliability is traditionally measured as the mean time between failure (MTBF). Trustworthiness and reliability are generally considered by most as almost being the same. It is obvious that reliability has been quantified but not many attempts have been make to quantify trustworthiness. According to the OTA, two possible reasons for the lack of interest in quantifying trustworthiness may be 1) trust is determined qualitatively as much as quantitatively and 2) most systems in critical applications are guarded by human operators. Whether these reason are true or not, the term trustworthy must be defined. [Ref. 12:p. 232]

As noted earlier Dr. David Parnas resigned from the Eastport Study Group because of his inability to perceive that software for the SDI program would be trustworthy. After his resignation, Dr. Parnas wrote several papers about SDI software issues. Of particular interest was the article on "Why the SDI Software System Will Be Untrustworthy." That article described the characteristics of the proposed battle management software system and provided implications of the problem characteristics. Dr. Parnas also stated reasons for not trusting the software before the U.S. Senate Subcommittee in 1985. The rest of this section will focus on the "Parnas Viewpoint" of why software systems are not trustworthy.

Common definitions of trust include the belief that conditions are favorable so that failure does not occur and the belief in one's ability to maintain confidence. This definition seems to fit very well with Dr. Parnas' reasons why the software would not work correctly when really needed. Two basic reasons are 1) we do not know exactly what the software is supposed to do, and 2) we must somehow validate the fact that it operates the way we think it should operate [Ref. 8:p. 290]. These reasons lead Dr. Parnas to believe that the software could not be trusted because it is not known whether the software will be correct or not. Scientists and engineers agree that the software will not be perfect. Everyone involved with the SDI program is willing to accept that fact. The problem is how to stop the system from failing catastrophically. Relating back to Dr. Frederick Brooks' question of "How good is good enough," Dr. Parnas responded in a letter to the editor of IEEE Computer magazine. He stated:

> I think that the answer is clear. To be good enough we have to know, with high confidence, how good it is. To be good enough, we must know that the system will not fail catastrophically. [Ref. 14:p. 6]

In the previous section, the author described some of the characteristics that reliable software systems should have. From this viewpoint requirements of the battle management software and some of the implications of these characteristics will be discussed. The following is a brief explanation of each:

1. Identification, tracking and weapon targeting are required system functions whose ballistic characteristics cannot be known with certainty prior to the battle. The implication of this problem is that fire control software cannot be written without making assumptions about the enemy's weapons and targets. Therefore, if the system is developed without knowledge of these characteristics, there are likely to be fatal errors in the software.

2. Determining the behavior of computers within the network cannot be predicted because of countermeasures by an attacker. This leads to the belief that the component availability and throughput of the system will also be inhibited. There are some systems where the likelihood of failures can be predicted from past history or component failures are unlikely and statistically independent, but this does not hold true for the required battle management software.

3. Real time testing of the software will be impossible. Although operational software for military aircraft undergoes rigorous ground and flight testing, "bugs" can and do show up in battle conditions. With this inability to test the system under real-time field conditions, the confidence and faith in the system is minimal.

4. There will be little possibility of human intervention during a nuclear war. This means that debugging and modifications of the program during a short period are not very likely to occur. Software modifications have been made in the field during previous wars (programming notes on the walls of trucks carrying computers; Vietnam). Some systems have even become reliable through such techniques, but the likelihood of these events occurring in a 30 minute war is extinct.

5. The number of targets detected and identified will determine the computational requirements of each process. The problem here, as in other cases, is that the number of targets or decoys cannot be predicted. Theoretically, this can be done using runtime/pre-runtime scheduling techniques. This scenario would work only if a worse case real-time schedule could be worked out in advance.

6. Each weapon system will include its own weapons and sensors which will require complex software systems to run them. To make matters worse, these systems will be

> dynamic during and after development. With a large number of contractors involved, the components will be subject to independent system modification. It is a known fact that as the size of software projects increase, the level of difficulty involved in terms of integration increases. When interfaces are changed, the problem is worsened. [Ref. 15]

The views shared by Dr. Parnas do not indicate that software cannot function effectively. They only point to the issues that engineers must consider before launching into a massive space project like SDI. In an article in Technology Review entitled "The Software for Star Wars: An Achilles Heel?" Herbert Lin discussed the "can do" spirit of Americans to overcome any so-called impossible feat. He stated that it is fashionable for proponents of SDI to dismiss claims of impossibility by citing lessons from history. The fact that someone said something is impossible does not make it possible. In his closing argument he states:

> No one can know with certainty how the BMD software would work during a large scale attack. How much confidence should the American public have in a BMD system so complex that no one person can understand it? That cannot be certified as error-free? That it is operationally untestable? The lessons learned from the performance of other large software systems suggest that the answer might be "very little" confidence indeed." [Ref. 5:p. 18]

D. VIEWS OF WHY SDI SOFTWARE WILL NOT BE ERROR-FREE

The hazards of error-prone software has become common a part of life on a daily basis. Whether it be a miscalculation on our phone bills, lost airline reservations, or an additional deposit to our checking account, errors are found

that may or may not alter our life styles. In many situations, the magnitude of the error is small and the recovery process is trivial. Unfortunately this is not always the case, especially with the SDI software. In the previous sections of this thesis the requirements for a BMD software system were stated. Errors will inevitably occur, but they must be kept to a minimum. [Ref. 10:p. 62]

Ware Myers discussed the problems of errors in large, complicated software systems. He stated that they are created in the process of formulating requirements, writing specifications, designing software, and writing code. Most analysts try to "spot check" the code by using techniques such as design reviews, module testing, and integration testing. Many errors occur when reprogramming is done to remove the errors found earlier. The fact is, there is no certainty that all errors in large complex software systems can be found and fixed. [Ref. 10:p. 62]

Dr. Solomon Bushbaum tried to silence many critics on the issue of error-free software by rephrasing the question to ask whether or not the total BM/C3 system could be designed to be robust and resilient in an error-prone environment. As alluded to earlier, Dr. Bushbaum is referring to the U.S. Public Telecommunications Network. [Ref. 8:p. 274] The author believes that the phone system is reliable, available, and adaptable because it has been "used and abused" continuously

for years. These are not exactly the characteristics that relate to the BMD software systems.

There are no technological breakthroughs that will make it possible to write error free software. It has not happened in the past and the state-of-the-art tools and techniques for the future are not promising. According to the Eastport Study Group, the focus should be on developing software that will function dependably despite the presence of errors. [Ref. 2:p. 2] This means placing more emphasis on the software research that the SDIO supports and finding ways to reduce errors. About 80 percent of errors found in large real-time systems are said to be caused by faulty requirements and design flaws. Some software professionals think that by using proper development methodologies, intensive testing, formal validation techniques and fault tolerant designs, we can reduce the incidence of errors. The process involves using computer aids and analysis procedures to test smaller components of large systems. This results is trying to narrowly predict and reduce the number of errors in a program. Fault tolerant design techniques employ methods such as multiversion development. In this process, two separate teams of programmers are given the same set of requirements to develop programs. The programs are then compared to each other to reduce common mode errors. [Ref. 10:p. 65]

Although the ultimate goal is to design and develop a software system that operates effectively with some known

errors, some scientists and engineers believe that the process of counting errors is not well understood. [Ref. 10:p. 65] It is quite obvious that, in most cases, quantitative arguments are more convincing than qualitative ones. People are more comfortable with numbers that support an argument. In a letter to the editor of Computer magazine, Dr Parnas stated:

> Error statistics make excellent diversions but they do not matter. A low error rate does not matter. A low error rate does not mean that the system will be effective. All that does matter is whether the software works acceptably when first used by the customers; the sad answer is that, even in cases much simpler that SDI, it does not. What also matters is whether we can find all the "serious" errors before we put the software into use. The sad answer is that we cannot." [Ref. 14:pp. 6-7]

One cannot totally believe that the debate on whether SDI software can be error free hinges only on assumptions about error incidence. The number of errors found per thousand lines of code cannot measure program correctness. The fact remains that all of today's large computer systems contain undiscovered flaws that are revealed only after the systems are put to use. [Ref. 24:p. 94] Even during program development, the "bugs" (errors) began to appear. The simplest type of error, such as typographical slips, sometimes cause the largest consequences. One of the most infamous examples is the failure of the Mariner I Probe to Venus in 1962. In this case, a period was substituted for a comma in the FORTRAN language, consequently, the probe had to be destroyed shortly after launch [Ref. 24:p. 94]. Most fourth generation languages check for notation errors automatically but logic

errors are more difficult to prevent. Even with small programming projects done at the Navel Postgraduate School, logic errors cost valuable computer resource time.

Other errors are caused by a programmer's lack of knowledge of certain types of contingencies. These errors cause major problems. There is no "physical" mistake to be found. The error is caused because the program branches are not available. Noted author, Alan Boring, gave an explanation that in large-scale programs there is usually nobody who understands the entire system completely. Humans play a major role in manufacturing errors. Programming errors are especially insidious because there is no way of indicating there is a problem prior to testing. Another problem is that if the section of code where the error lies is never executed, the error is never found. Unfortunately, one day that "special contingency" may occur and that section of software needs to execute properly, but does not. As stated earlier, redesigning software is expensive. The Department of Defense (DoD) will spend about 10 percent of the budget or $30 billion on software in FY90. With the recent lessening of the Soviet threat, it is not likely that Congress will dispense more money for redesigning systems that do not operate properly. [Ref. 16:p. 26]

Zraket feels that because of the discontinuous and highly discrete nature of software, generating millions of lines of software code may cause small errors with a large operational

impact on the BMD system. The solution: rigorous design and testing. This solution will be discussed more in the next chapter. Zraket also stated that the most important and prevalent uncertainties and flaws occur in the operational design and structure of the software. His approach is similar to that of Myers; build the operational design and software in increments, evaluating each sequence.

IV. ANALYSIS OF THE SDI SOFTWARE DEBATE: WHAT THEY AGREE ON

A. INTRODUCTION

This chapter focuses on a few of the SDI software issues of which both supporters and critics have agreed. This is by no means an exhaustive list of requirements, rather it typifies issues that are less debatable. The reader should be informed that contrary to the style of the previous chapter, the views shared in the debates addressed in this chapter are those of a composite group and not necessarily those of individuals. Anyone who has studied the SDI software issues can testify that there are several other problems that have been heavily debated. The author will attempt to discuss the relative "show stoppers"; What would happen if these issues were not resolved?

The major software issues that have survived the debates are 1) the need for a specific battle management\C3 architecture, 2) the need for a dedicated national test facility, and 3) the need for a SDS software center.

B. THE NEED FOR A SPECIFIC BATTLE MANAGEMENT/C3 ARCHITECTURE PRIOR TO DEVELOPMENT

One can easily categorize the SDI as a vast system consisting only of thousands of sensors, weapons and

platforms. More than likely, it is often overlooked or taken for granted that software, computers, and communications drive the entire system. The Eastport Study Group suggested that too much emphasis was being placed on weapons and sensors and not nearly enough on the software complexities concerned with the design and development. The Eastport study concluded that the choice of system architecture depends on the feasibility of the battle management software that can be simulated, tested, and maintained. Given these facts, one must consider the factors that serve as criteria to evaluate the architectures. Fundamentally, performance, testability, and cost are a few of the criteria considered. For each of these criteria, measures of performances (MOPs) must be defined. For performance, the measures of performance are linked to the survivability or the robustness of the system. These factors go hand-in-hand with reliability and durability, necessities for real world performance. An MOP for testability must be structured from the results of small-scale tests. Reasoning for that approach is based on the mere fact that full-scale tests are impossible. The cost of building, deploying, and maintaining a strategic defense system will be inherently high. [Ref. 2:p. 22]

The SDIO realized that battle management software and its supporting C3 elements must be given the highest technical priority. The task set before them was to find an architecture that implements human control, lowers cost, and reduces the

magnitude of required coordination and communication control. Although these factors were heavily debated, the SDIO considered them a necessity. What type of system architecture would best satisfy these issues - centralized, decentralized, or layered. Viewing the software as a battle manager, there were many questions about the type of structure needed.

A centralized architecture required the software to provide information to every element (sensors and weapons) in the network instantaneously and simultaneously. This process would make fault tolerance and error recovery virtually impossible. Also, human control would have no intercession within the process. Computing resources for a centralized system are focused in one location and may consist of several processors that share common memory devices. These systems are called multi-processors. The processors have a high rate of communication because of high data rates, resulting in a tightly coupled system. Figure 4.1 shows a "centralized" representation of the architecture. Additionally, the centralized architecture would not allow simulated full-scale testing, a major criteria for SDI's acceptance. [Ref. 2:p. 23]

A decentralized architecture does not rely on tight coordination. It is basically organized as a hierarchy similar to the military chain of command (i.e., tasks are delegated at lower levels). As noted in the Eastport Study:

> No system part within such a hierarchy needs to depend on millisecond-by-millisecond detailed instructions from a higher authority. " [Ref. 2:p. 23]

CENTRALIZED BATTLE MANAGEMENT
ARCHITECTURE

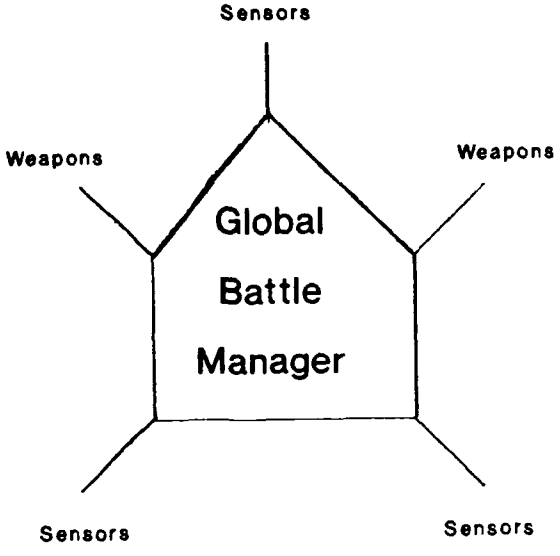


Figure 4.1

In the decentralized system, processors are separated (physically), individual memory spaces are allocated and data communication rates are decreased. These characteristics are also known as important parts of a distributed system. The system follows the loosely coupled concept. Each battle manager reports (when necessary) to the battle manager at the next higher level. For instance, given three levels, the lowest level would be considered the local battle manager performing fighting functions. The next level would be the regional battle managers. They would be responsible for targets passing between battle spaces and resolving

contentions for resources among local battle managers. The top level manager would be the global battle manager. The primary job would be to establish strategies for the regional and local battle managers. They would also provide the man-machine interface for the system. Figure 4.2 depicts a typical decentralized architecture. [Ref. 2:p. 45]

DECENTRALIZED BATTLE MANAGEMENT ARCHITECTURE

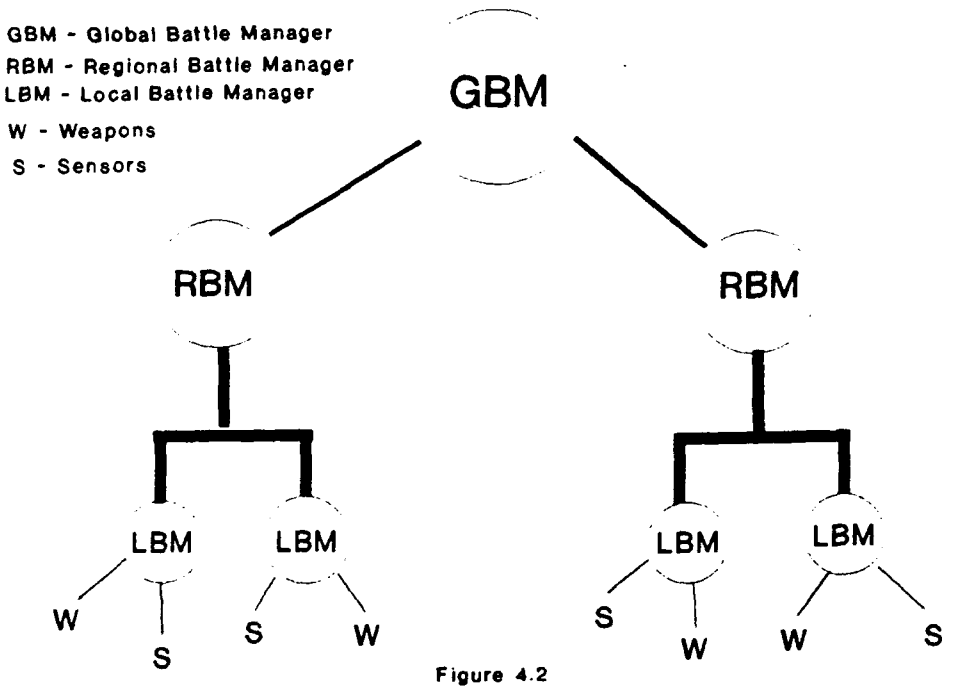


Figure 4.2

The following elements are critical advantages attributed to a decentralized architecture capable of independent action. This list was taken from a report prepared for the 100th Congress by the SDIO in June 1987. The advantages are as follows:

1. Simplicity: A simpler architecture can be produced by eliminating "perfect" coordination. This approach requires less bandwidth, less latency requirements, and reduces scheduling demands that have a profound impact on software.

2. Testability: Elements in a decentralized architecture act independently. In this regard, each element can be tested separately allowing the developer to see how the whole system functions in reference to its individual parts.

3. Evolvability: Since decentralized architectures use relatively simple interfaces, addition of similar elements is far less tedious. Existing computing requirements will not be affected by additional changes. Scalability in deployment must be supported. With every asset that is deployed, the performance of the system is improved incrementally.

4. Robustness: Error and fault tolerance levels in one platform are controlled under that particular battle manager. Thus, when failures occur at one level, other levels are not affected.

5. Diversity: Robustness is reinforced through diversity. Plans for strategic defense must employ the services of vendors as well as implement new technologies as they evolve. The major driver behind a battle management system being diverse is its ability to add competition to the procurement market. The result is lower system costs.

6. Durability/Survivability: A decentralized architecture is more survivable against countermeasures, and fault tolerances are much lower. This results from the unlikelihood of errors to propagate throughout the entire system.

The decentralized architecture has been recommended by the Eastport Study Group for the design and development of battle management software. [Ref. 17:pp. 6-8]

A third type of architecture is known as a layered architecture. This architecture is very similar to the decentralized version (i.e., Global and Local battle

managers). The layered model can have fewer or more layers depending on the particular system architecture or the tactics employed to counter the ballistic missile attack. For example, requirements and strategies may differ from layer to layer by using specific weapons, sensors, and operating in different conditions. [Ref. 9:pg. 12-20]

The basic functions for the decentralized and layer models are the same. Local battle management functions include tracking and classifying targets and allocating proper resources. Global battle management functions provide real time surveillance, establish rules of engagement, delegate resources, provide mutual defense, and perform situation assessment. Figure 4.3 provides a view of the layered model of battle management. [Ref. 9:p. 12-20]

C. THE NEED FOR A DEDICATED NATIONAL TEST BED

Testing and Evaluation (T&E) of any major weapon system development is critical to its success or failure. T&E of software for strategic defense systems is no exception. It is expected to be both difficult and critical. Authorities agree that in order to support design and development efforts, a national test bed had to be developed. In 1986, Danny Cohen, Chairman of the Eastport Study Group, wrote a memo on the National Test Bed (NTB). Dr. Cohen stated that the single central component of the NTB should be simulation. The simulation requirements are numerous and diverse but the two

LAYERED MODEL BATTLE MANAGEMENT ARCHITECTURE

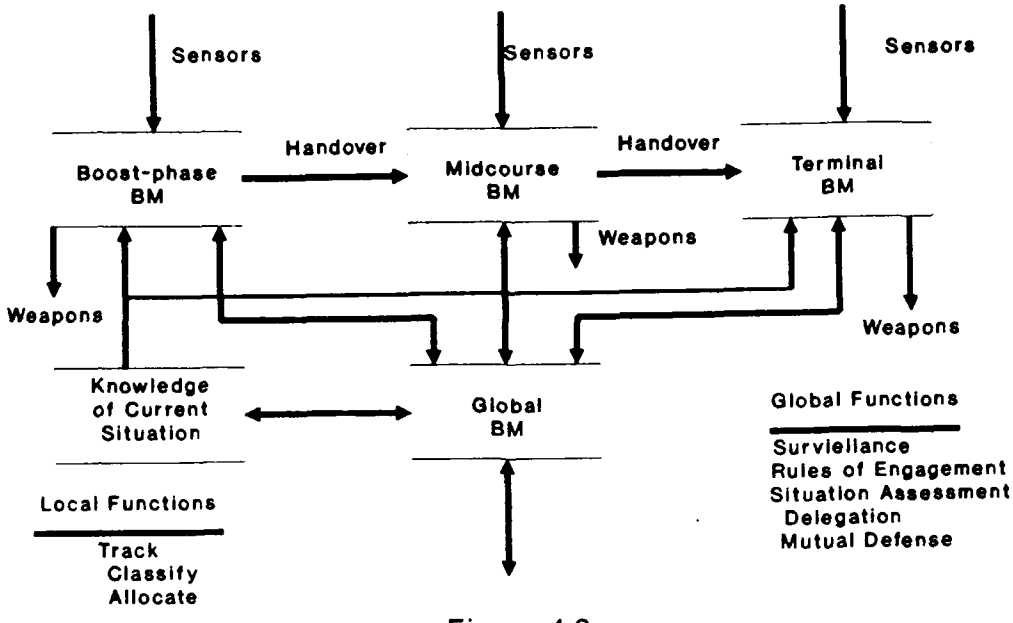


Figure 4.3

main areas for the application of the simulation are design and testing. Simulation's role is paramount because it is impossible to do real time testing on all weapons and sensors. Two basic technical questions that simulation should provide the answer to are: 1) Can BM/C3 software provide an adequate level of reliability and 2) Can the BM/C3 software strategies handle the diversity of attacks. These questions can not be answered without the extensive use of a National Test Facility (NTF).

Concept definition and preliminary design study of the NTB started in March 1986. Martin Marietta and Rockwell

International were selected to complete the preliminary design concept. Martin Marietta subsequently won the contract for development for the NTB. Completion of the National Test Facility (NTF) is scheduled for FY90. [Ref. 3:p. 5.2-15]

Figure 4.4 shows the proposed version of the NTB. The purpose of the NTB is to evaluate systems and new technologies for SDI. Its mission includes demonstrating the feasibility of SDSs through computer simulations, evaluation of the applicability and feasibility of new technologies, and conducting experiments on SDS systems.

SDI National Test Bed

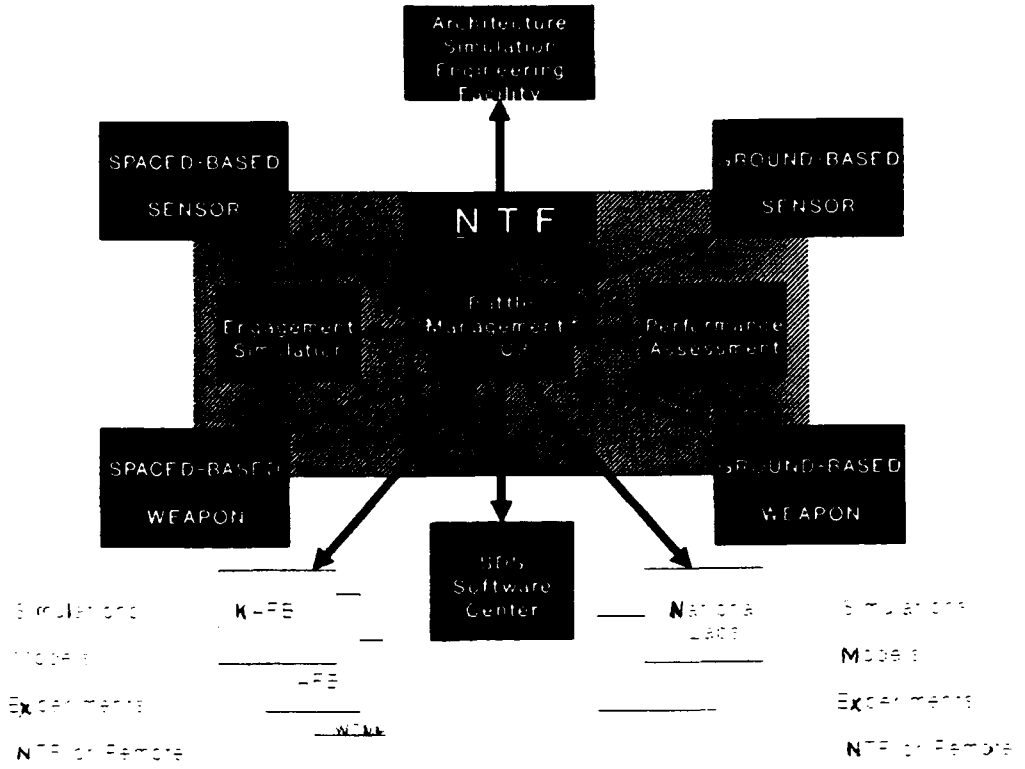


Figure 4.4

This test facility will not only be used for software testing, but for testing communication assets also. [Ref. 2:p. 29] According to the DoD Software Master Plan (Preliminary Draft):

> "The NTB will inter-connect Army, Navy, Air Force, National Laboratories, and Test/Demonstrations facilities into a distributed network. The NTB may be thought of as a network of resources with the National Test Facility (NTF) as its harbor central facility. This composite provides the principal resources dedicated to develop and/or support experimentations and provide analysis support. The NTB is a Natural Resource which draws together contractors, the military, government agencies, academia and others studying SDS issues. [Ref. 18:p. A-13]

The National Test Facility will serve as the hub for the NTB and will be located at the Falcon Air Station, 20 miles east of Colorado Springs, Colorado.

The National Test Facility, NTB, consists of a set of software and hardware tools that support design and development, and execution/analysis of simulation experiments related to SDI and other associated DoD programs. These tools provide support for end-to-end simulation of weapons, sensors, communication systems, and C2 systems. Testing facilities will also be operated at different locations and will be connected to the NTF via communication/computer links. Facilities connected to the NTF will include, Vandenburg Air Force Base (VAFB), Kirtland Air Force Base (KAFB), and White Sands National Laboratory (WSNL). [Ref. 3:p. 5.2-14]

Additionally, in 1988 a Software Center (SC) was developed in order to provide scientists and engineers with efficient development tools. This center will establish software policy as well as invest in development. The major purpose of the center is to provide trusted software tools that reduce risk by allowing each component to capitalize on the investment in the total software environment. The next section will give the reader a fresh understanding of the mission of the Strategic Defense System Software Center. [Ref. 3:p. 5.2-15]

D. THE STRATEGIC DEFENSE SYSTEM SOFTWARE CENTER

Hardware/software integration and testing will be expensive. Current technology may not be reliable and fast enough to handle SDS functions. As stated earlier, the SDS Software Center was proposed as a much needed operation with the National Test Facility. Its primary mission will be to ensure that development, production, integration and validation of trusted SDS software through Full Scale Development (FSD) is done efficiently. The National Aeronautics and Space Adminstration (NASA), Ada Joint Programming Office (AJPO), and other SDIO directorates will coordinate software efforts to avoid duplication and keep cost at a minimum. The Software center will provide help in addressing critical software development areas such as trustworthiness, configuration management, reusability, interoperability, training, technology, and many others. [Ref. 18:p. a14]

Listed below are six basic functions of the software center:

1. Programmatic support for government and contractor developers. This team of experts will assist program managers and government/contractor developers with problems unique to their projects.

2. Produce and integrate high quality, reliable software for SDS systems. This process is critical to the success of the SDS mission.

3. Provide state-of-the-art technology for software engineering environments (SEE) and configuration management system components.

4. Establish and maintain a library of reusable Ada components. Ada will be the language of choice for the NTB-developed software. The NTF also has an Ada software development facility.

5. Ensure consistency of acquisition, design, development, integration, and testing efforts through software education and training. Training will be provided in all phases of software development. The goal will be to produce an Ada competent development community.

6. Assist in the development and tailoring of new and existing standards. Support will be provided to organizations such as IEEE, ANSI, and Ada9x. [Ref. 18:p. A14-15]

SDIO is providing every necessary means to reduce risk in all BM/C3 systems that are software driven. Everyone realizes that exhaustive testing of software tools is not practical, therefore more emphasis must be placed on the development of sound tools for the SDI program. The Software Center will provide that development environment.

If the functions and procedures of the SDS software center are properly executed, research and development for strategic defense software will be able to reach the majority of its goals. [Ref. 18:p. a15] During the 1990 IEEE Conference for Strategic Software Systems held in Huntsville, Alabama, Lt. Col. Chuck Lillie discussed the short and long term goals for the software center. The goals are as follows:

1. Supportability, compatibility and interoperability of SDS mission critical software

2. Evaluate and demonstrate tools, capabilities, and procedures required to implement SDIO policies governing SDS software development

3. Reduction of the time between SDS research, development, production, deployment and operation

4. Efficient production of trusted, high confidence software throughout the SDS life cycle

5. Effective dissemination of Multi-service research results through program technology insertion

6. Compatibility between services and SDS software engineering and Ada policies [Ref. 11].

There is no question concerning the necessity for the SDS software center. In order for SDIO to gain the necessary confidence to deploy reliable, trustworthy systems, one must first develop those tools to help accomplish this vital mission.

V. SUMMARY OF THE SDI SOFTWARE DEBATES

A. INTRODUCTION

In the previous chapters, several views of whether it is feasible to produce reliable and trustworthy software for a BM\C3 battle management system were addressed. Along with determining the reliability of the software, opinions are also expressed on the type of BM/C3 architecture necessary for further development of battle management schemes. Finally, the question of how we test our development was also addressed with several feasible approaches, including the development of the National Test Bed (NTB). From these debates, the following summary chart, Figure 5.1, was developed to highlight the position of each proponent and opponent of the software debates. Figure 5.1 provides the reader with a brief narrative of the positions of each individual discussed in this thesis.

B. EVALUATION OF THE SDI SUMMARY CHART

As stated in the introduction of this thesis, the purpose was to provide the reader with a framework for understanding the software debates. The framework included understanding the software complexities (i.e., reliability, availability, and testability of numerous weapons and sensors simultaneously) and understanding how to apply the corporate knowledge of all

SDI SOFTWARE SUMMARY CHART

	RELIABILITY/TRUST	AVAILABILITY	BM/C3 ARCHITECTURE	TESTABILITY
Dr. David Parnas	SDI software can be reliable and trustworthy only if all the necessary requirements are met: PROBLEM: We don't know what the system is suppose to do....Catastrophic failure	We can't expect any major breakthrough to solve this problem. Software technology lags far behind hardware technology and software engineering tools may not be the answer.	Decentralized architecture supports the SDI project, but it can't be expected to solve all of the battle management problems.	The software can never be tested fully prior to nuclear war. Simulations only provide "near real-time" results. Testing must be done using the "actual" code.
Office Of Technology Assessment	There is no reliable way to demonstrate that BMD software operate properly when first used. The concept of MTBF has historically had limited use for critical software systems.	Availability figures are useful when the conditions under which they were measured are well known. Extrapolation outside these conditions is risky.	Battle management architectures as yet proposed are not specific enough for their claimed advantages and disadvantages to be effectively evaluated.	Confidence in the dependability of a BMD system has to be derived from simulated battles and test during peacetime.
Ware Myers	The number of errors created depends on the type of software, the size of the system, and other factors. There will be errors, but how do we prevent catastrophic failure?	....	Overall SDS should be concerned with the needs of a highly distributed system architecture. Software requirements, algorithms, and designs need to be worked on together.	Test components in actual environment with simulated data. Use a decentralized architecture. Make the software partionable to allow thorough testing.
Dr. Danny Cohen	There will be errors, but for programs that are system critical, resources are allocated to ensure their quality (i.e. errors < 0.1 per 1000 lines)	SDI software can be developed using today's technology without any major breakthroughs.	System architecture should be loose, coupling on various system components.	Make full use of SDI simulated software with the support of the National Test Bed.
Dr. Soloman Bushbaum	Asking whether there'll be errors is the wrong question. The right question '- whether the total BM/C3 system can be robust and resilient in an changing, error prone environment. Answer....Yes	Although SDI is a complex and difficult task, it behoeves the U.S. to do research to determine whether or not we can develop the technology.	The Distributed architecture approach compartmentalizes crucial functions in modules as well as individual systems.	....
Dr. Charles Zraket	To assess the feasibility of creating and maintaining reliable software, one must distinguish between the uncertainties and flaws. (i.e. Trustworthiness of operational design and tolerance.	Much more research and lengthy development efforts are needed to determine the future of the SDI software	Decentralized architecture must be designed to operate in time periods that include system deployments, peacetime and wartime operations.	Levels of software testing should include: 1. Research 2.System/subsystem testing 3. System-level simulation 4. Deployed Hardware and Software testing
The Fletcher Study	The panel recommended that more emphasis be placed on processor developments and fault-tolerant designs that will ensure reliable operation.	Creating an appropriate software development environment will be a difficult task. The environment must be in place and understood before starting to build a BMD BM/C3 system.	Use logical battle management structure that would consist of a single battle manager replicated several times on different platforms	Proposed the development of a realistic simulation environment .
Hebert Lin	Two successful techniques for assessing reliability and trustworthiness are analytical (mathematical proofs) and emperical testing.	The U.S. should consider three questions concerning SDI software development: What is the nature of the BMD system, What are the obstacles, and can the obstacles be circumvented	Global battle management system would assess the extent and nature of an attack in progress and specify the rules of engagement for each tier. (i.e. Boost, post-boost, midcourse, terminal)	Testing real time software often introduces more software errors when a "fix" is implemented
Dr. Frederick Brooks	How good is good enough?	I see no reason why I or any other competent, experienced, better software manager than I... could not undertake to build such a system.	....	....

Figure 5.1

individuals involved with the software aspect of SDI. The summary chart is divided into two sections; those who oppose and those who support the SDI software efforts. The general consensus is that software for SDI is possible, but there are a number of problem areas that are plaguing the system now, and other problem areas that cannot be eliminated. Because the so called impossible has been made possible many times before, no one is willing to say that software for SDI cannot be developed.

1. Reliability and Trustworthiness

Dr. David Parnas, Ware Myers, and the Office of Technology Assessment all support the fact that the software cannot be assured of not failing catastrophically and that a lack of specific requirements for software operations alters the development process. It should be noted that even though Dr. Parnas has had many negative things to say about SDI's success, he did not state that the software could not ever work reliably. He simply stated that the software requirements must be understood. Scientists and engineers may not know all the necessary requirements for an effective battle management system, but the author believes that extensive use of the National Test Facility, and other related agencies may prove to be the focal point in understanding the requirements.

2. Availability

The availability of most systems is determined by the amount of time the system operates between failures. In basic

engineering, this is known as the mean time between failures (MTBF). The big question in this area is whether the software can be developed with present technology with minimal failures. Dr. Danny Cohen took a firm stand on the advancement of software technology. Dr. Cohen feels that no major breakthroughs are necessary and that today's software technology can be used to create a system with high MTBFs. Dr. Parnas stated that the dependence on new technological breakthroughs should not be expected to solve the availability problem, mainly because of the fact that hardware technology is years ahead of the state-of-the-art of software technology. In analyzing the debates, it was interesting to find that Dr. Frederick Brooks once supported the ideas of Dr. Parnas, but later joined sides with Dr. Cohen after testifying at a subcommittee meeting on Strategic and Theater Nuclear Forces.

In essence, the proponents suggested that required software availability was achievable, but that more research and development efforts were needed. Although not in full support of the way availability is assessed, the Office of Technology Assessment (OTA) stated that the conditions for using availability figures must be well known to be useful.

3. BM/C3 Architecture

In Chapter IV, one of the issues that both proponents and opponents of SDI agreed upon was the need for a specific battle management architecture. It should be noted that no one particular architecture was chosen but the structure for the

architecture must be in place prior to development. Most reports submitted to congress for review recommend that a decentralized architecture be used in support of command and control of the battle management system. The decentralized structure includes loose coupling on various system components. In contrast, OTA stated that all BM/C3 architectures as proposed were not specific enough for their claimed advantages and disadvantages. Again, the consensus is that structure must be in place prior to development.

4. Testability

Everyone agreed that the software for SDI will not be perfect. There are numerous ways to simulate the actual operation of the software, but full-scale testing cannot be accomplished prior to nuclear war. Dr. C. A. Zraket stated that testing efforts must proceed from research to deployment using systematic methods at each level. Simulation is the key factor that will determine the effectiveness of the software during the actual battle. The testing issue was given the most recognition by both the critics and supporters. Comments on how the testing efforts should be conducted did not have the negative air that other subject areas had faced. One of the most weighted concerns came from Herbert Lin. His concern was implementing more errors while trying to "fix" old ones. Again, this was not a statement saying that testing efforts would be useless, rather a caution to those correcting errors during the testing phase.

The summary chart provides the reader with an overview of the major issues that both parties were analyzing. The intent of this chart was not to draw conclusions, but rather to highlight some of the pros and cons of software development for a dependable ballistic missile defense system.

VI. DEVELOPMENTAL CONCERNS AND TRADEOFFS ASSOCIATED WITH SDI SOFTWARE

A. INTRODUCTION

The purpose of this chapter is to focus on some of the vital developmental concerns and tradeoffs that may alter or hinder the progress of developing an effective BMD system. In Chapter II, the role of software for a BMD system was discussed. From that discussion important areas that either need more research, or solutions that are available that need to be implemented were identified. Some of these concerns were addressed at the third International IEEE Software Conference for Strategic Systems in Huntsville, Alabama, February 27-28, 1990. Other concerns and tradeoffs have been discussed among the proponents and critics over the past five years.

Among the issues addressed are: 1) The software challenges from the SDIO perspective, 2) the use of software engineering tools for SDI development, 3) the simulation and testing methods available, and 4) the security of SDI software.

This list of issues and concerns is not exclusive. Because of the dynamic nature of the SDI program, it may be that an exhaustive list of concerns can never be attained.

B. SDIO SOFTWARE CHALLENGES

The SDIO views the current problems associated with development of quality military software to be both real and urgent. The plan is to adopt a strategy to promote a change of attitudes, policies, and practices concerning software acquisition. The strategy has to begin early within the acquisition phase in order to control the resources associated with software development for strategic defense. Software for strategic defense, in particular SDI, faces some monumental challenges that must be overcome prior to the development of the last phase. At the third International Conference on Software for Strategic Systems, Lt. Col. Chuck Lillie discussed some of the challenges that SDIO is dealing with. He listed them as:

1. Large Amount of Code
2. Integration of Software Development by Many Contractors
3. Evaluation and Configuration Management
4. Coordination of Distributed Assets
5. Complex Data Fusion, Discrimination, and Tracking Algorithms
6. Parallel and Distributed Architectures for Real-Time Performance
7. Secure, Fault Tolerance Performance
8. Testing and Simulations [Ref. 11]

The list of challenges does not include other important software issues such as software engineering environments, and

unambiguous, testable, and traceable specifications. As if it has not been stated enough, software development for SDI is the most complex task ever faced by DoD. According to a computer resource working group, size estimates for phase one are huge. Figure 6.1 shows a breakdown of the number of lines of source code per system element for the phase I architecture. These figures are based on the 1988 architecture sent to the Defense Acquisition Board (DAB) by General Electric. [Ref. 11]

The size of the software is not the only challenge that SDIO is facing. For example, there are 12 prime contractors involved in the phase I development process. There are even more subcontractors. Therefore, integration of these elements to operate successfully in one environment will be a monumental task. As one author stated they should build software like they build cathedrals; build them, then pray a lot. The more integrated elements that are involved, the stronger the emphasis that should be placed on building good prototypes. These prototypes should assist in the development of requirements, validate size estimates, provide early demonstration of the software's performance, and address the critical elements involved. Dr. Frederick Brooks said it best, "Build a little, test a little, and learn a lot."

The Aerospace Corporation did a study on the "military software problem" during the early 1980s. In that study they explained some of the "causes" of software problems. Two

SDI Phase I Software Estimates

Elements	KSLOC
Boost Surveillance and Tracking System BSTS	1,208
Space Surveillance and Tracking System SSTS	771
Space Based Interceptor SBI	2,830
Ground Based Radar GBR	600
Ground Surveillance and Tracking System GSTS	266
Ground Based Interceptor GBI	309
Command Center CC	14,522
Total	20,506

KSLOC - Thousands of Lines of Code

Figure 6.1

problems in particular that apply directly to SDI's software are management and personnel. The study showed that managers lack some necessary skills to "manage" the software process on an end-to-end basis. For instance, software managers (program managers) had a poor understanding of the technical complexities involved (i.e., hardware selected prior to allocation of resources required, lack of communication and organization, and lack of life cycle perspective). Brigadier General John Vamboise, in his keynote address at the software conference, stated that their biggest problems are managerial not technical. This issue was also a concern to both the Defense Science Board and the Eastport Study Group. General Vamboise used a familiar proverb to identify a peculiar criteria that all program managers must have a vision. For where there is no vision, the people perish. [Ref. 11]

The second part of the software problem involves people. The Aerospace Study pointed out that DoD does not take the systems approach to the availability of human resources. There is a shortage of skilled programmers. Inevitably this leads to using inexperienced personnel. Since the average duty assignment for military personnel is about two years, the turnover is very rapid. This lack of continuity eventually leads to stripping ongoing projects to staff new ones.

In order to fix this personnel problem, more emphasis should be placed on personnel receiving a competent education in software design and development. For example, Ada is the

standard DoD language for software development, but only 15% of the colleges in the U.S. teach Ada as a programming language. Future resources must come through the formal education process.

In addition to defining some of the challenges for SDI, Lt. Col. Lillie also defined their software policy. The purpose of this policy will be to require and promote the use of software engineering approaches for the development and support of all SDS Full Scale Development (FSD) software. Also, the policy should be implemented through the various military services using individual implementation plans. The key to the concept will be consistency factor. Figure 6.2 shows the application and scope of the SDS software policy. The Office of the Secretary of Defense (OSD) has directed that the SDIO will comply with DoD directives 3405.1, 3405.2, and 5000.3 along with MIL-STD-1815A. The directives basically inform the SDIO of the requirements for software development. Since Ada has become the standard programming language for DoD, MIL-STD-1815A is the standard Ada specification.

C. THE USE OF SOFTWARE ENGINEERING TOOLS FOR SDI SOFTWARE DEVELOPMENT

The acquisition process for major weapon system normally consists of four phases which are separated by decision milestones. The phases are the Concept Definition Phase, Demonstration and Validation Phase, Full Scale Development

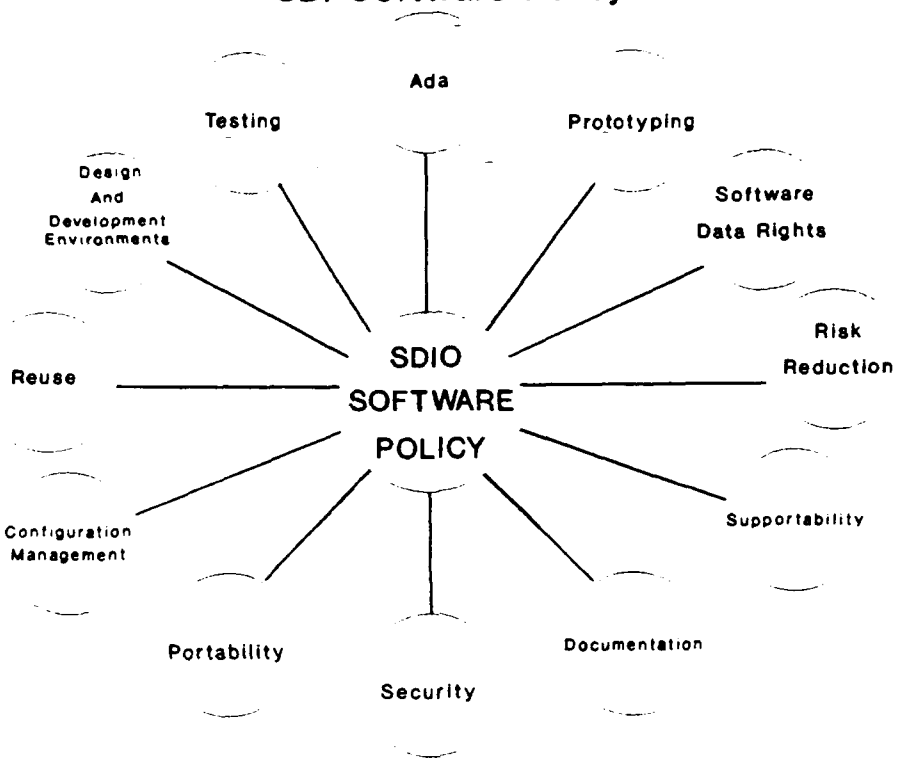


Figure 6.2

Phase, and Production Deployment Phase. This process is initiated when a need is perceived in a particular mission area. Basically, the need may arise from a change in threat, technology advancements, cost reduction opportunities, or a projected obsolescence of existing systems. [Ref. 19:p. 1-13] The software development life cycle follows a similar procedure. First, a feasibility analysis is completed, then the requirements and specifications are completed. Continuing, the software analyst begins to draw up the design from the specifications and actual coding of the software is done.

Finally, to see if reliable, dependable results are available, extensive testing is begun. After this phase, the initial development life cycle is complete. Fig. 6.3 shows similarities between the acquisition phases for a weapon system and the development life cycle concept for software.

This whole process for software development has evolved around a relatively new technology called software engineering. According to an OTA report, the term "software engineering" arrived on the scene in 1968. It was the result of computer scientists focusing on the difficulties in developing complicated software systems. How is software engineering defined? Ask any five software engineers this question and you're bound to get at least three, related, but different answers. Author Richard Fairley, in his book, "Software Engineering Concepts," stated:

> Software engineering is the technological and managerial discipline concerned with systematic production and maintenance of software products that are developed and modified on time and within cost estimates. [Ref. 20:p. 2]

The Institute of Electrical and Electronic Engineers (IEEE) expanded this definition to include maintenance and retirement of the software. Referring to Figure 6.3, one can see that the software life cycle does not stop when the development phase is complete.

Many proponents of SDI tend to think that software engineering tools will enhance the software in terms of reliability, maintainability, and availability. Th e "tools"

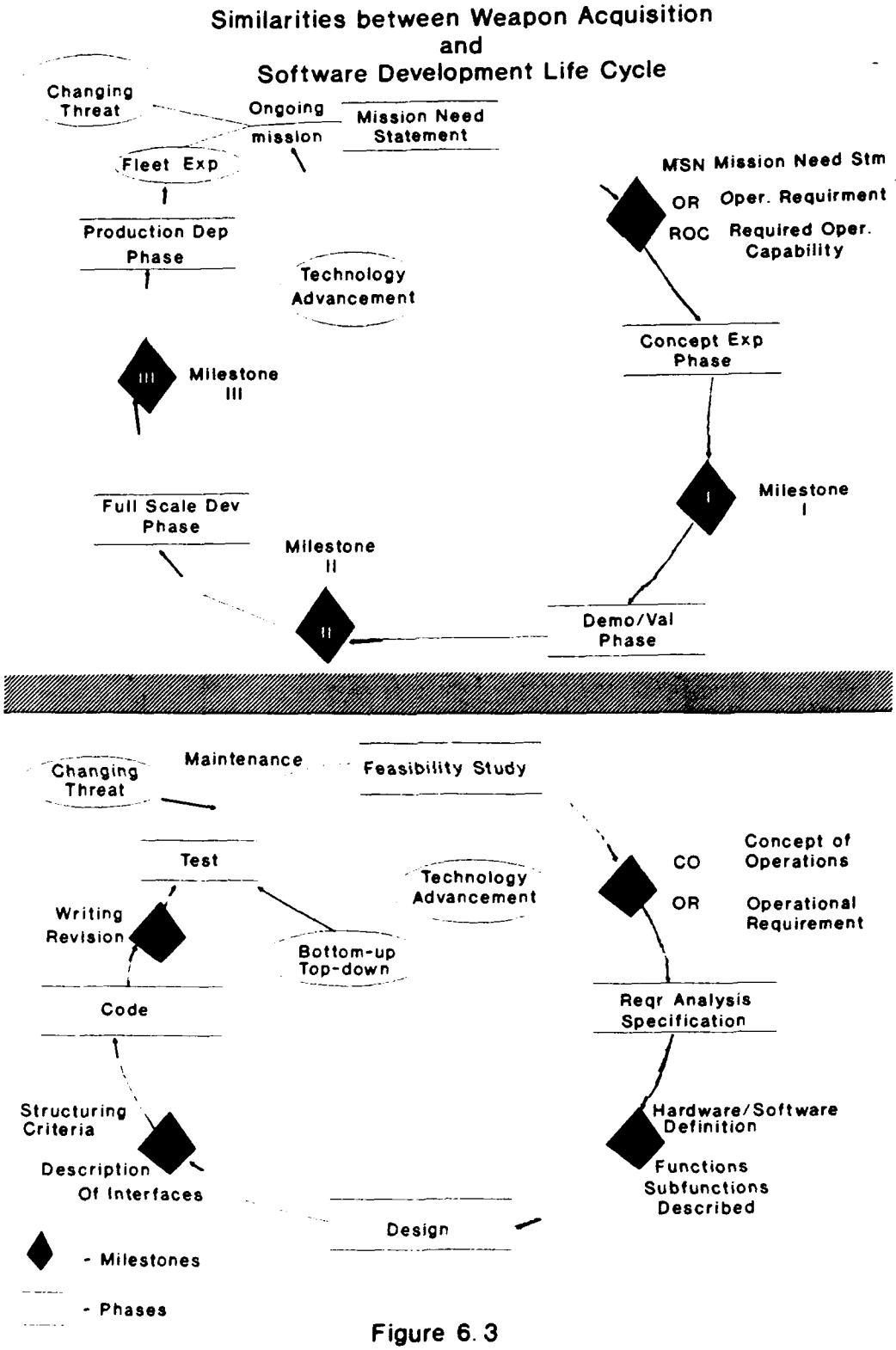

Similarities between Weapon Acquisition
and
Software Development Life Cycle
Changing Threat
Ongoing mission
Mission Need Statement
Fleet Exp
Production Dep Phase
Technology Advancement
III
Milestone III
Full Scale Dev Phase
Milestone II
II
Demo/Val Phase
MSN Mission Need Stm
OR Oper. Requirment
ROC Required Oper. Capability
Concept Exp Phase
I
Milestone I
Changing Threat
Maintenance
Feasibility Study
Test
Writing Revision
Bottom-up Top-down
Code
Structuring Criteria
Description Of Interfaces
Design
Technology Advancement
CO Concept of Operations
OR Operational Requirement
Reqr Analysis Specification
Hardware/Software Definition
Functions
Subfunctions
Described
- Milestones
- Phases

Figure 6.3

are in the form of automated support that may consist of one or more programs. On the other hand, critics believe that software engineering tools for SDI provide no greater support or advantages than any of the previous software tools. [Ref. 21] The following discussion will focus on "additional" paradigms within the software life cycle. Some of these techniques are associated with risk assessment and incremental development while others make an attempt to eliminate or reduce the number of steps within the software life cycle. Specifically, we will discuss the possible uses and limitations of object oriented programming, automatic programming, and artificial intelligence.

Object-Oriented Programming will allow the properties of procedures and data to be combined together. Unlike other data structures, an object data structure contains the properties of reusability and encapsulation. In essence, the variables of the objects are contained within themselves. This process allows data transferring of vast quantities of information on a function to function basis. Although this is just one variation of the term object programming, it is the most relevant to SDI; mainly because formal requirements and design specifications are used. These were just two of the techniques suggested by Dr. Parnas after spending several years on the Navy's Software Cost Reduction (SCR) project. The one downfall associated with this technique was that using the above

stated methods had not lead to reliable code that met the space and time constraints. [Ref. 21]

It was stated earlier that one of the purposes of using software engineering tools was to eliminate or reduce some of the requirements and design phases. One method developed to employ those techniques is automatic programming. With this technique the programmer is literally taken away and becomes a "specification writer". In other words, the programmers would write the specification for the software and the computer would generate the program. Dr. Parnas stated that the automatic programming concept was not more than a "euphemism" for programming with a higher level language, but even the use of improved languages has led to some improvement in reliability. In essence, if one can write "trusted" software specifications, automatic programming techniques are a definite plus. [Ref. 21]

Finally, how does artificial intelligence (AI) help or will it help us to build better BM/C3 software? There has certainly been a lot of progress in the AI field recently. In March 1989, the American Association for Artificial Intelligence (AAAI) held its first annual conference on innovative applications of AI. There were about 30 different creative applications ranging from expert systems for investing to music composition. These applications were using computers to solve problems by applying some form of human intelligence. [Ref. 22:p. 13]

Dr. Parnas stated that there are two very different definitions of AI in use today. He referred to them as A-1 and A-2. By definition, they are as follows:

> AI-1: The use of computers to solve problems that previously could only be solved by applying human intelligence.
> AI-2: The use of specific set of programming techniques known as heuristic or rule-based programming. In this approach human experts are studied to determine what heuristics or rules of thumb they use in solving problems. Usually they are asked for their rules. These rules are then encoded as input to a program that attempts to behave in accordance with them. In other words, the program is designed to solve a problem the way that humans seem to solve it. [Ref. 23]

Even though there has been tremendous work done in AI, most potential applications of AI should be handled on a problem-by-problem basis. Unfortunately, the SDI battle management problem is not an attractive situation at present. According to the OTA, AI-1 and AI-2 should not be applied to SDI battle management problems until a specific set of battle management problems and their solutions are specified. In his conclusion of AI and SDI, Dr. Parnas stated:

> Artificial Intelligence has the same relation to intelligence as artificial flowers have to flowers. From a distance they may appear alike, but when closely examined they are quite different. I don't think that we can learn much about one by studying the other. AI offers no magic technology to solve our problems. Heuristic techniques do not yield systems that one can trust [Ref. 23]

The author believes that software engineering tools have a lot to offer the programming world, but one must be mindful of tools and techniques that appear to be immediate problem solvers. Many software tools may work perfectly on smaller-

scale systems, but may lack the same efficiency and productivity when applied to larger systems.

There are many software engineering tools available. The key is finding the correct tools that apply specifically to the needs of the SDI software efforts. Further improvements in Object-Oriented Programming will continue to strengthen our confidence in software reuse, thus the development of reliable, trusted software to meet space and time requirements can become a reality. Any technique that can improve the reliability of the software must, at least, be considered before it is rejected. As stated earlier, the Strategic Defense Software Center will be the hub for testing and validating trusted software and must be used to the fullest extent. Before Automatic Programming is rejected based on its similarities with other high level languages, the technique for using this tool must be evaluated. Finally, even though Artificial Intelligence has made tremendous advances in the last two to three years, one must first find ways to incorporate AI into SDI. Specifically, the problems that AI must solve must be understood.

D. THE SIMULATION AND TESTING METHODS AVAILABLE

Throughout this thesis our focus has been on some of the concerns for SDI software. Questions such as will the software be reliable, what are the roles of software, and can the software be developed free of catastrophic errors, have

already been addressed. These are questions that cannot, technically, be answered prior to some type of evaluation of the software. Two of the primary means of carrying out this evaluation is through simulation and testing.

There are several tradeoffs and concerns that greatly affect SDI in these areas. Both the Eastport Study and the 1988 Office of Technology Assessment report on simulation and testing, identify numerous areas for additional research and the need to find alternative testing methods. Other recently published literature provides further insight on improving simulation and testing techniques. Although these methods do not apply specifically to SDI, they do provide the latest technology changes and advancements. This section will give the reader a brief overview of what simulation and testing entails in addition to providing possible insights into areas that need further development.

It is the author's opinion that evolving technology has been a major driver behind the need for increased simulation. In the simplest terms, simulation is a system that duplicates the behavior of another. The problem occurs when the simulator cannot reproduce all the behavioral characteristics of the target or the system being duplicated. In the early 1970s simulations were primarily based on single engagement models. In case of a ballistic missile defense, that meant trying to simulate one missile attacking one target. In the late 1980s, the simulation models had to simulate thousands of missiles on

multiple targets. The simulation process included anticipating the threat, developing a realistic environment to operate within, and testing the functions of the model that was developed. These same problems plague the environment for an effective BM/C3 system for SDI. [Ref. 4:p. 51]

The simulation efforts are presently geographically distributed among several facilities (e.g., The Air Force Electronics Systems Division (ESD) at Hanscom Air Force Base, Massachusetts, the Army Strategic Defense Command (ASDC) in Alabama, and the National Test Facility (NTF) in Colorado). One major concern hindering the simulation and testing efforts is a lack of interfacing between facilities. In essence, the simulation efforts should employ the work of several facilities. Specifically, the Eastport report stated:

> "Simulation efforts should cooperate closely with activities at the National Test Bed (NTB). But for several reasons, NTB should not be the only simulation facility. First, simulation is too vital to the strategic defense effort to permit a "monopoly" that would be implied by such a single implementation. Second, comparison of results from different simulation facilities, for purposes of cross-verification and validation is essential. Third, a centralized implementation at NTB would likely lead to classification, very limited access, and narrow focus, which would curtail the simulators' effectiveness and prevent it from producing confidence in the system's functional ability and reliability. [Ref. 2:p. 31]

The more simulators interface with other systems, the more trust one may be able to put in the system. The Eastport Study referred to three classes of levels for simulation operations. Low level simulations would be used for sensor and

weapon development, mid-level simulations for interfacing with other on-line systems, and high-level simulators for studying and evaluating BM/C3 strategies. With this concept working effectively, testing efforts may show considerable improvement.

Simulating the functions of an SDS is a phenomenal task. This is especially true when algorithms for tracking, discrimination, weapons control and guidance are involved. Dr. Zraket stated that the designs of these algorithms must be tested on an end-to-end basis. He also stated that testing should be done using actual code rather than emulating their behavior through the use of simulators. The problem here is unequivocal; no one knows what the performance of the algorithms will be. This leads to the conclusion that credible simulations are needed. For example, simulations are needed to predict the position of a track based on the details of the estimated present position. This need led SDIO to initiate the National Test Bed. Zraket further stated that two capabilities must be developed: 1) a model-evaluation process to establish the credibility of models and simulations and 2) an experiment-design process to foster systematic and informative experiments. The capabilities are greatly needed because most current SDS simulations lack fidelity, high resolution, and the actual code for SDS algorithms and BM/C3. The formulation of a systems-oriented software engineering environment is

needed to integrate the development and evaluation of SDS on an end-to-end basis. [Ref. 25:p. 96]

In order to simulate a target and formulate some type of model, one must first make an assessment of the environment and the equipment involved. When simulations are done without known qualities of the environment and other factors, the accuracy of the simulation lacks quality. This may very well be the major or the most treacherous concern facing simulation experts. How does one model a Soviet decoy or missile if there is no physical access to them? According to an OTA report, simulation experts from the Naval Research Laboratory stated that building simulations without prior access to "real" equipment caused many unexpected surprises. Even if there was access to performance characteristics, the problems would not be solved. Simulators may not be able to reproduce the parameters and different signatures of events such as nuclear explosions. This means that accuracy and realism involved in the simulations would be further diminished. [Ref. 12:p. 206]

The Department of Defense (DoD) is constantly looking for new ways to combat the complex challenges of modelling and simulating the performance of major weapon systems. According to Mr. James H. Atkinson, Test and Evaluation (T&E) specialist, modelling and simulation (M&S) can offer the potential for overcoming the existing testing limitations of weapon systems. He stated that a generic baseline methodology could be used by different developers of weapon systems to

tailor a performance evaluation program to the specific weapon. [Ref. 25:pp. 143-146] This author believes that this is a concept that can be applied to the development of battle management software. The concept involves initiating four basic characteristics that must occur throughout the software development cycle. The characteristics are 1) Degree of Representation - defining the limits of M&S process; 2) Practicality - obtaining useful output from the M&S process; 3) Validation - establishing acceptance and credibility with the M&S process and 4) Configuration Control - tracking of updates/modifications in the M&S process. Using these four characteristics could possibly help establish a good "working" baseline for understanding and modelling software for SDI. Figure 6.4 shows three domains of test and evaluation that can be applied to the software development life cycles. This figure is a mock up version used for the weapons systems development life cycle. The basic functions and relative application are the same. The process does not eliminate the need for thorough testing of the system, but could help to produce better testing results. [Ref. 25:p. 143-146]

One of the issues not discussed in the modelling process was cost. There is no doubt that money could be a driving force for this project. One possible tradeoff is that more money could be placed up front to institute the modelling and simulation process. This would possibly result in reduced costs for validation testing. With emphasis being placed on

T&E Domains, Simulators and Testing

T&E Domain	Relative Application	Functions
1	M&S - 80% Test - 20%	A Establish methodology and formulate approach B Specify requirements and design specifications C Identify limits of simulators (Practicality)
2	M&S - 45% Test - 55%	A Update and improve simulator capability B Validate simulations by using "actual" code for selective testing
3	M&S - 80% Test - 20%	A Predict the performance of system using validated simulators B Compare to limited field tests C Monitor and continue to validate the simulators through out the system's lifecycle

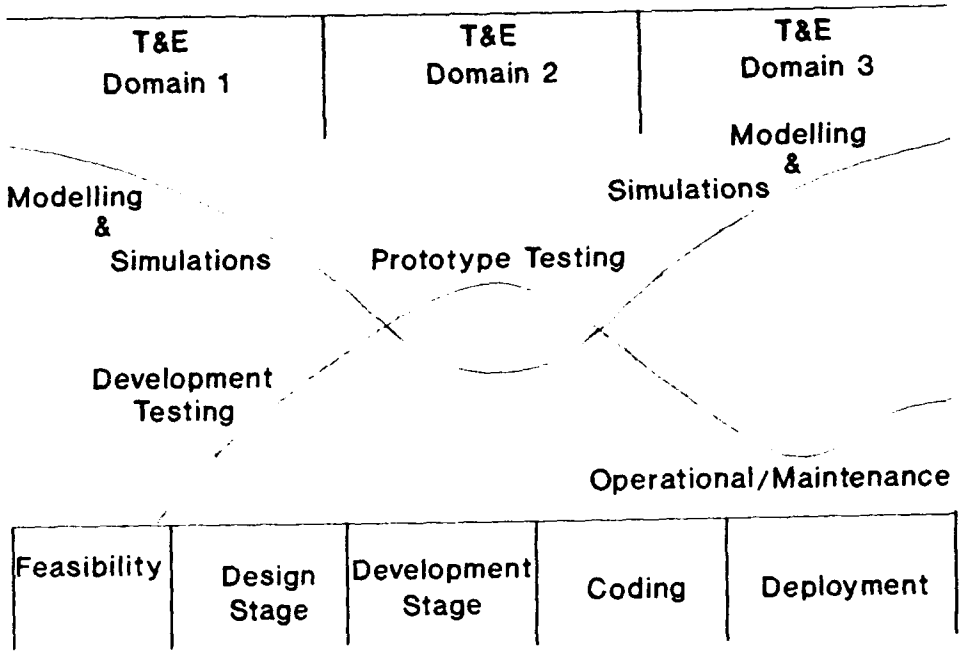


Figure 6.4

accelerating the development process, the models and simulators are needed to increase the effectiveness of the software.

E. SECURITY FOR SDI

Security is an issue that plagues many "sensitive" projects whether it's DoD related or exclusive to the private sector. Over the last 10 years there has been a growing increase in the number of attacks on computers and communication networks. The attackers are not choosy and they don't have "respect of network or computer." [Ref. 26] Because of the critical functions of the BM/C3 networks and computers, greater emphasis should be placed on securing the assets for SDI. The sad part is that the means of security go far beyond locking the door at night. Therefore, in this section the author will focus on some of the failures of security that are applicable to BM/C3 computers.

In 1987, the SDIO submitted an interim assessment of SDI's computing requirements to the 100th Congress. In that report, the SDIO stated that minimizing the complexity in the concept design of SDS was the key to fault tolerance and security with respect to software. This is true especially when the end product results in a slow, expensive system. In other words, security and fault tolerance need to be integrated during the design phase. By now, the results of "adding" requirements to a system are well known. The study also indicated that the

security policies for the SDS must reflect the highly decentralized nature of the system. Decentralization does not solve all of our architecture problems, but in this case it provides us with consistency. [Ref. 17:p. 10]

One of the glaring tradeoffs that needs further research is the performance of the system. A more secure system also means a more complex system. It has been stated in some reports to Congress that there is not a need for another study on computing requirements. The author disagrees for a number of reasons. Mainly, because we have not scratched the surface on items such as security mechanisms for strategic defense systems.

Computer security for SDI should be based on two separate aspects; technological and applicational. By technological, we can address security from logical and physical points of view. Logically, security is the protection of data and access or gateways between programs. Physical security is the action that can be taken to prevent physical harm to the resources. [Ref. 26:p. 433]

Applicational computer security addresses the development of new applications and the maintenance of those applications. Both the technological and applicational approach to computer security can be applied tc the SDI security policy. In the previous section we looked at how to do modelling and simulation with the software development life cycle. Computer security specialists, K.P. Badenhorst and Jan H.P. Eloff,

developed an "ideal methodology" for computer security in view of the typical structured approach to the software development life cycle. They structured the major phases as introduction, implementation, and maintenance. Figure 6.5 shows the layout of each phase within the methodology and description of each phase is listed below:

High Level View of a Methodology
for
Computer Security

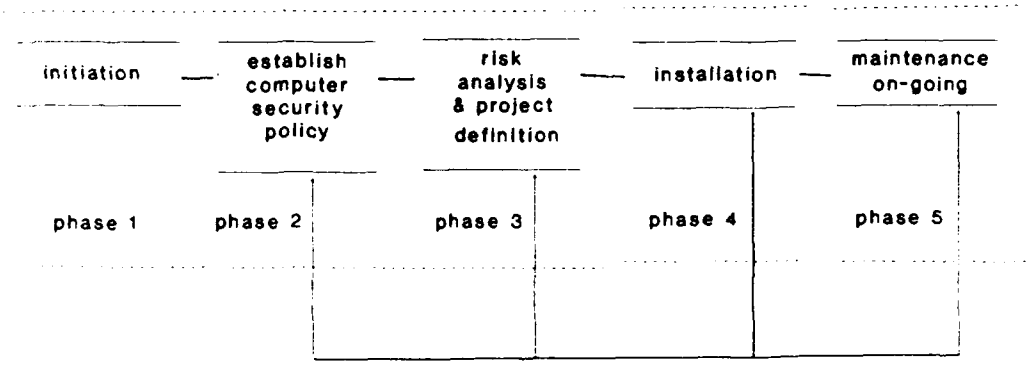


Figure 6.5

1. Phase 1: Initiation - Requires management to establish awareness and support. Also a special group of people should focus on securing the assets by introducing new techniques.

2. Phase 2: Establish Computer Security Policy - Again, top management (Program managers) must establish a

"corporate" policy for security in order to develop good management controls.

3. Phase 3: Risk Analysis and Project Definition - This is where cost factors can be decreased by ensuring that proper security measures have been selected.

4. Phase 4: Installation - Covers technological aspects such as logical access, physical access, encryption, and other methods.
5. Phase 5: Maintenance/Ongoing - Full development of controls for applicational systems. [Ref. 26:pp. 443-435]

These phases or a model similar to this should be considered when employing security tactics.

Security is important because the bulk of SDI's resources rely on communication networks and computers. Whenever a computer system or a communication network is violated there are a variety of losses involved. Those losses include the loss of goods and services, the loss of assets, and the denial of computer services. [Ref. 29] These services are critical because of the time factor that would be involved in a nuclear war. The probability of bringing an entire network of computers back on-line after a failure during a nuclear war is almost impossible. It does not matter whether the method of destruction was accidental or deliberate, the consequences are the same. Much consideration should be given to finding ways of avoiding situations like the "worm" created by computer hacker, Robert T. Morris. A worm is a type of instruction that is placed in computer program causing incorrect results when the program is run. No physical access is required since these "worms" can be placed in the programs through the use of

floppy diskettes, internal/external file transfers, and by having unauthorized access to computer networks. According to an article in Computer World, Morris gained access to computers at the National Aeronautics and Space Administration's Ames Research Center in California, the U.S. Air Force Logistics Command in Ohio, the University of California at Berkeley, and Purdue University in Indiana. These were just a few of the facilities where Morris was able to get in and cause considerable damage. [Ref. 27:p. 1]

Computer security fails often because people have a narrow minded view of security as being only physical in nature. To eliminate some of the failures one must understand the targeting concepts, the motives for attacking computer systems, and how to uncover the penetrator. It's not always the "enemy" that's plaguing our systems. [Ref. 29] Morris and other computer hackers proved that point. In a previous research report on computer security, the author stated that there were three basic approaches to a system's defense. They include: 1) dispersion, 2) duplication, and 3) defense in depth. In the event of accidental or deliberate threats, dispersion is a tactic used to minimize the losses. Since total destruction of a computer system would require multiple attacks, dispersed systems are less susceptible to a total system shutdown. SDI's decentralized architecture would employ the use of the dispersion technique with its BM/C3 operations. Secondly, duplication or redundancy must be maintained in

order to enhance the life of the system. Duplication is the method of providing back-up components for enhancing fault tolerance. Finally, defense in depth would include the use of an alarm system, a response system and a recovery system. The alarm system's function will be to detect a threat in time to either avert it or minimize the damage. The response system must apprehend the villain and neutralize the effect of the error or accident. The recovery system must repair the damage and restore system operation. Obviously these functions alone do not merit full protection against "hackers", but collectively they help form a defensive ring. [Ref. 29]

The Computer Security Act of 1987 and the Computer Fraud and Abuse Act of 1986 are pieces of legislature for punishing individuals who willfully attack government resources. This is not the real problem. Today, there is no quantitative way to measure the amount of security in a system, but DoD has developed a standard for evaluating the security for computer systems. DoD Standard 5200.28 is the Department of Defense Trusted Computer System Evaluation Criteria. The concept used in this standard consists of matching the features of a system against those known to be necessary to provide security.

VII. CONCLUSIONS

Software development for the Strategic Defense Initiative (SDI) is one of the most complex tasks that both government and defense contractors have ever faced. Why is this development effort so hard? Mainly because of the changing requirements associated with the SDI program. Even in the midst of writing this thesis, information that was once a "hot item" is just another passing term in the huge SDI vocabulary. To some extent, this constant change in requirements is good because it shows that technology is rapidly progressing. On the contrary, the analysts and computer scientists that are trying to develop reliable, dependable, and trustworthy software for these systems are facing a "programmer's nightmare." Another problem is that state-of-the-art software technology is still lagging far behind hardware technology. Although there has been vast improvements in software technology, one may still ponder the question of whether or not these improvements can support the operational task of SDI.

The software debates held at Stanford University and the University of California at Los Angeles were not just "another conference" for scientists and engineers to argue over ideas. The debates brought out important issues, some of which are

discussed in this thesis, that were necessary for the successful design and development of the SDI software and battle management/C3 system. It is obvious that all the issues from the debates are not yet resolved, but research and development efforts should continue until acceptable solutions are found.

The software for SDI will be responsible for coordinating a conglomeration of weapons and sensors. It will take over 20 million lines of code to complete this task. Will the software make the Ballistic Missile Defense (BMD) system and the Battle Management/C3 system reliable? This is the question that engineers and computer scientists alike are pondering on a daily basis. The research and development programs must remain in high gear in order to allow an answer to this question with any confidence. It's a known fact that one will not know the answer to this question until war is started. It is the consensus of all involved with SDI that the point of war remains distant. Much consideration must be given to the wise words of Dr. Frederick Brooks, "Build a little, test a little and learn a lot." This is exactly the route that software development for SDI must take. The establishment of the National Test Bed (NTB), National Test Facility (NTF) and the Strategic Defense Software Center in Colorado Springs, Colorado is very timely in support of testing and evaluation efforts for SDI. This certainly should not be the only facility testing SDI components. Other agencies should be

involved, therefore, providing a comparative analysis of the system's operational capabilities under several conditions.

Obviously, there are many other factors that software developers must consider. For example, the choice of Battle management architecture effects the structure and development of the software. The decentralized battle management architecture has been the choice of most agencies that have completed studies for the SDIO. Decentralization does not resolve all of the SDI related issues, but it best suits the needs for the current software development efforts.

Advanced software engineering tools have made very impressive improvements in the areas of "expert systems" and "decision-support" systems. These tools are certain to have an impact on SDI in future development efforts. Special caution must be taken if proponents of SDI plan to depend on these tools for the development of SDI software. Tools that do not allow the software to remain in "discrete" states must be avoided. Dr. David Parnas says that they should avoid using artificial intelligence and automatic programming techniques for program verification. On the contrary, one should not eliminate any possible development tools and techniques until they can be proven "unreliable." Full scale tests are impossible and every one is willing to accept that fact, but improvements in the simulation efforts may shed new light on some of the once unresolvable problems.

Finally, the software is pivotal for strategic defense. It is possible to develop the software, but the following issues must be understood: 1) understand the complete function of the software at all levels and 2) accept the fact that it will not be error-free. The job of the engineers and computer scientists is huge; keeping the system from failing "catastrophically."

LIST OF REFERENCES

1. Lamberth, Benjamin S. Selective Nuclear Operations and Soviet Strategy, The Rand Corporation, September 1975

2. Eastport Study Group, "A Report to the Director," Strategic Defense Initiative Organization, December 1985

3. U.S. Department of Defense, Report to the Congress on the Strategic Defense Initiative, March 13, 1989

4. Lin, Herbert, "The Development of Software for Ballistic-Missile Defense", Scientific American, Volume 253, Number 6, December, 1985

5. Lin, Herbert, "The Software for Starwars An Achilles Heel?" Technology Review, July 1985

6. Zracket, Charles A., "Uncertainties in Building A Strategic Defense," Science, Volume 235, Number 4796, March 1987

7. Parnas, David L., "Software Aspects of Strategic Defense Systems," American Scientist, Volume 73, Number 5, September-October, 1985

8. U.S. Senate, Subcommittee on Strategic and Theater Nuclear Forces (Hearings), Committee of Armed Services, December 1985

9. Fletcher, James C., A Report of the Study on Eliminating the Threat Posed by Nuclear Ballistic Missiles, Volume V, February 1984

10. Myers, Ware, "Can Software for the Strategic Defense Initiative Ever Be Error-free?" Computer, Volume 19, Number 11 November 1986

11. Notes on Third International Software Conference for Strategic Systems, 27 February 1990

12. U.S. Congress, Office of Technology Assessment, "SDI: Technology, Survivability, and Software", OTA-ISC-353, May 1988

13. "New Weapon Suffers From Major Defects," Washington Post, January 7, 1987

14. Parnas, David L., "Parnas:SDI "red herrings" Miss the Boat," Computer, Volume 20, Number 2, February, 1987

15. Parnas, David L., "Why the SDI Software System Will Be Untrustworthy," Software Aspects of Strategic Defense, June 1985

16. Jacky, Johnathan, "The Star Wars Defense Wor.'t Computer," Atlantic, Volume 255, Number 6, June 1985

17. Strategic Defense Initiative Organization, "An Interim Assessment of SDI Computing Requirements (Draft)," June 1987

18. U.S. Department of Defense, "Software Master Plan (Preliminary Draft)," February 9, 1990

19. Navy Program Manager's Guide, 1988

20. Fairley, Richard, Software Engineering Concepts, (New York: McGraw-Hill, 1985)

21. Parnas, David L. "The Limits of Software Engineering Methods," Software Aspects of Strategic Defense, June 1985

22. Moore, Brad, "You Mean It Really Works? or Innovative, Deployed, AI Applications," AI Magazine, Volume 10, Number 3, Fall 1989

23. Parnas, David L., "Artificial Intelligence and the Strategic Defense Initiative," Software Aspects of Strategic Defense, June 1985

24. Myers, Ware, "Software Pivotal to Strategic Defense," Volume 22, Number 1, January 1989

25. Atkinson, James H. Sr., "Modeling and Simulation in the Test and Evaluation Process," Simulation, Volume 54, Number 3, March 1990

26. Badenhorst, K.P., and Eloff, Jan H.P., "Framework of a Methodology for the Life Cycle of Computer Security in an Organization," Computers and Security, Volume 8, 1990

27. Alexander, Michael, "Morris Verdict Stirs Debates," Computer World, Vol.XXIV, Number 5, January 1990

28. Lubkin, Yale, "SDI: Soft? Where?" Defense Science and Electronics, Volume 53, Number 4, 1986

29. Adams, Reginald C., "Computer Insecurity," Research Paper, March 1990.

INITIAL DISTRIBUTION LIST

		No. Copies
1.	Defense Technical Information Center Cameron Station Alexandria, VA 22304-6145	2
2.	Library, Code 0142 Naval Postgraduate School Monterey, CA 93943-5000	2
3.	Director for Command, Control and Communications Systems, Joint Staff Washington, D.C. 20318-6000	1
4.	C3 Academic Group, Code CC Naval Postgraduate School Monterey, CA 93943-5000	1
5.	AFIT/NR Wright-Patterson AFB, OH 45433-6583	1
6.	AFIT/CIRK Wright-Patterson AFB, OH 45433-6583	1
7.	CAPT Reginald Adams 108 Beale St. Belzoni, MS 39038	1
8.	Prof. Luqi Associate Professor Computer Science Dept. Naval Postgraduate School Monterey, C 93943-5000	1
9.	Prof. Donald Lacer, Code CC/La Naval Postgraduate School Monterey, CA93943-5000	1
10.	Dr. Marvin J. Hamilton The Aerospace Corporation Mail Station M8/107 P.O. Box 92957 Los Angeles, CA 90009-2957	1

END

11. Mr. Wesley Mann, Jr. 1
Vice President, Defense ansd Surveillance Operations
The Aerospace Corporation
Mail Station M5/702
P.O. Box 92957
Los Angeles, CA 90009-2957