Chapter 4.

SAFEGUARD SYSTEM

Chapter 4. SAFEGUARD SYSTEM

SAFEGUARD is an ABM system designed to protect U.S. Minuteman ICBM bases from attacks by enemy ballistic missiles. Development of the SAFEGUARD System began with a redirection of the SENTINEL program in March 1969. Its deployment plan called for a number of sites to be constructed primarily in the western part of the United States. As a result of the Strategic Arms Limitation Treaty (SALT) and related program decisions, actual deployment was subsequently limited to a single complex in North Dakota and a system command center in Colorado.

The initial SAFEGUARD plan called for up to twelve sites deployed in two phases. The first phase, for which authorization was originally granted, provided Minuteman defense at Grand Forks Air Force Base (AFB), North Dakota, and at Malmstrom AFB, Montana, together with a Ballistic Missile Defense Center (BMDC) at Cheyenne Mt., Colorado. The second phase would have added Minuteman defense at Whiteman AFB, Missouri, and at Warren AFB, Wyoming, as well as defense of the National Command Authority (NCA) in Washington, D.C. This phase also retained the option to add additional sites to protect Strategic Air Command (SAC) bases and population centers. In March 1971, approval was granted to proceed with the installation at Whiteman and to plan for the Warren site. Whiteman was designated as the Fire Control Center (FCC) and Malmstrom as the Alternate Fire Control Center. The FCC was an intermediate command center reporting to the BMDC. A year later, however, authorization for the Whiteman site was rescinded and Malmstrom was designated as the FCC.

In accordance with the terms of the Strategic Arms Limitation Treaty of June 1972, and a subsequent Congressional decision not to authorize the permitted deployment in the Washington, D.C. area, the SAFEGUARD System was further reduced to provide Minuteman defense only at Grand Forks AFB. Thus, the current deployment consists of a Perimeter Acquisition Radar (PAR) and a Missile Direction Center (MDC) in North Dakota, both under overall command of the BMDC in Colorado. Included in the MDC are a Missile Site Radar (MSR) and associated SPRINT and SPARTAN missile farms. This deployment is depicted in Figure 4-1 and is aimed at providing a number of defense capabilities.[1-8]

The three types of sites in the SAFEGUARD System are interconnected by communications links. The PAR site uses a single-face, phased-array radar to provide early detection and target trajectory data on threatening ICBMs. Functions of this site include long-range surveillance, detection and selection of threatening objects, and

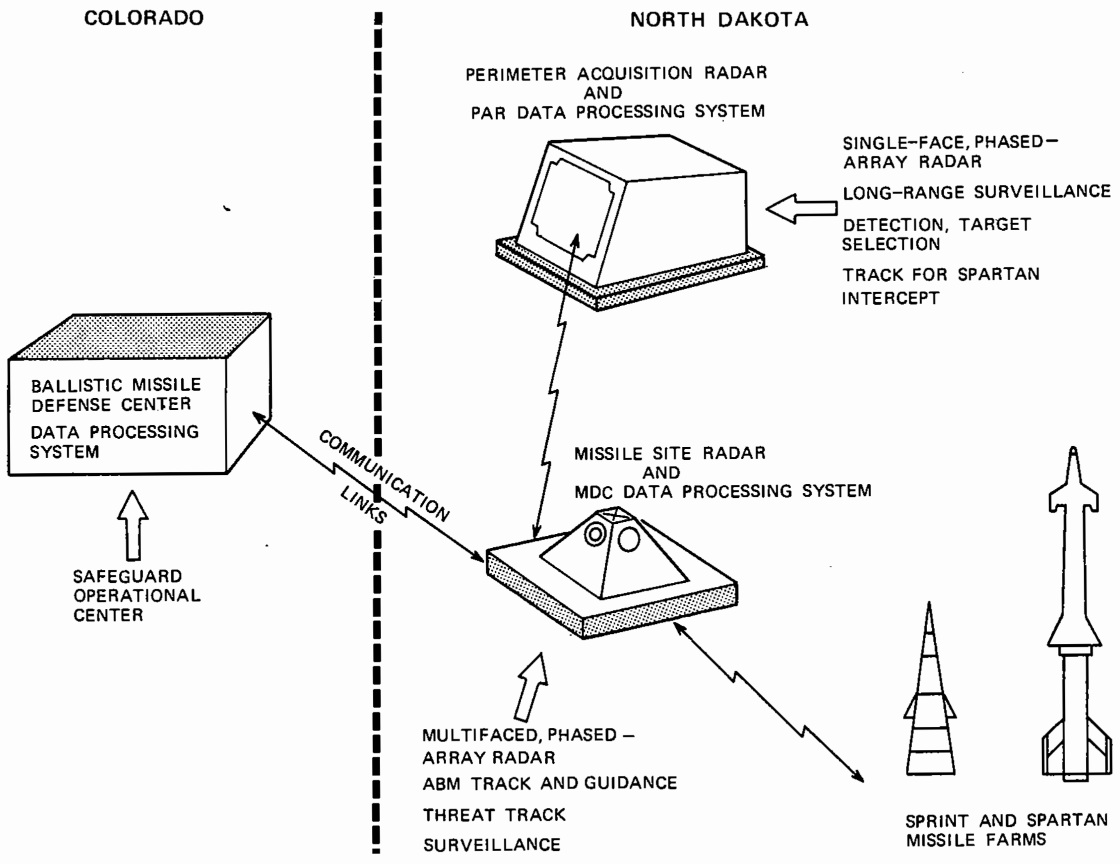

COLORADO
NORTH DAKOTA
PERIMETER ACQUISITION RADAR AND PAR DATA PROCESSING SYSTEM
SINGLE-FACE, PHASED-ARRAY RADAR
LONG-RANGE SURVEILLANCE
DETECTION, TARGET SELECTION
TRACK FOR SPARTAN INTERCEPT
BALLISTIC MISSILE DEFENSE CENTER
DATA PROCESSING SYSTEM
SAFEGUARD OPERATIONAL CENTER
COMMUNICATION LINKS
MISSILE SITE RADAR AND MDC DATA PROCESSING SYSTEM
MULTIFACED, PHASED – ARRAY RADAR
ABM TRACK AND GUIDANCE
THREAT TRACK
SURVEILLANCE
SPRINT AND SPARTAN MISSILE FARMS

Figure 4-1. SAFEGUARD ABM System

ICBM threat tracking for SPARTAN intercept. This last capability significantly increases the long-range SPARTAN field-of-fire. The PAR site does not perform missile guidance, but instead, transmits trajectory and target classification data over the tactical communication links to the MDC. The MDC uses this information together with data from its own multi-faced, phased-array Missile Site Radar. This site provides additional surveillance and target tracking as well as track and guidance for SPRINT and SPARTAN missiles. Both PAR and MDC sites report to the BMDC, which provides a command interface with other military systems and a means of disseminating command directives and controls.

The PAR and MSR are controlled through digital commands issued by collocated Data Processing Systems (DPSs). These commands are used to manage such radar functions as beam pointing, frequency selection, receiver gating, thresholding, etc. In addition, application programs in the PAR and MSR Data Processors (PARDP and MSRDP) manage the major system functions of surveillance, tracking, target classification, radar testing, intersite communication, and command/control/display. At the MDC, other programs support engagement management and missile guidance. The BMDC DPS primarily performs command, control, and display functions.

MAJOR SAFEGUARD ELEMENTS

Perimeter Acquisition Radar

The Perimeter Acquisition Radar performs long-range surveillance, detection, and tracking of ICBMs for SPARTAN missile intercept.[7,9] As an auxiliary function, it can provide track data on selected satellites of interest. To withstand nuclear blasts,[10] the radar equipment is mounted in a reinforced concrete structure which is shielded against nuclear electromagnetic radiation. The front face of the building supports an array of 6888 antenna elements (see Figure 4-2).

At periodic intervals, the PAR searches the volume of space within a large solid angle in front of the array face. It can detect small targets in a typical ballistic missile complex at large distances with a high probability of detection. The radar operating frequency can be changed and the antenna beam-pointing direction can be switched between points in space and between multiple transmissions in the same radar interval.

The PAR transmitter consists of high-power Traveling Wave Tubes (TWTs) whose combined total peak radiated power is in the multimegawatt range. Target returns are amplified by low-noise transistor amplifiers which are an integral part of the phase-steered corporate-fed array. Further details on the PAR are given in Chapter 8.

Missile Site Radar

The Missile Site Radar is a four-face, single-beam phased-array radar operating in S-band. Its function is to acquire and track incoming ballistic missiles and to launch and guide SPRINT and SPARTAN missiles for intercept.[7,11,12] The MSR can also launch and guide SPARTAN missiles for intercept using PAR target data.

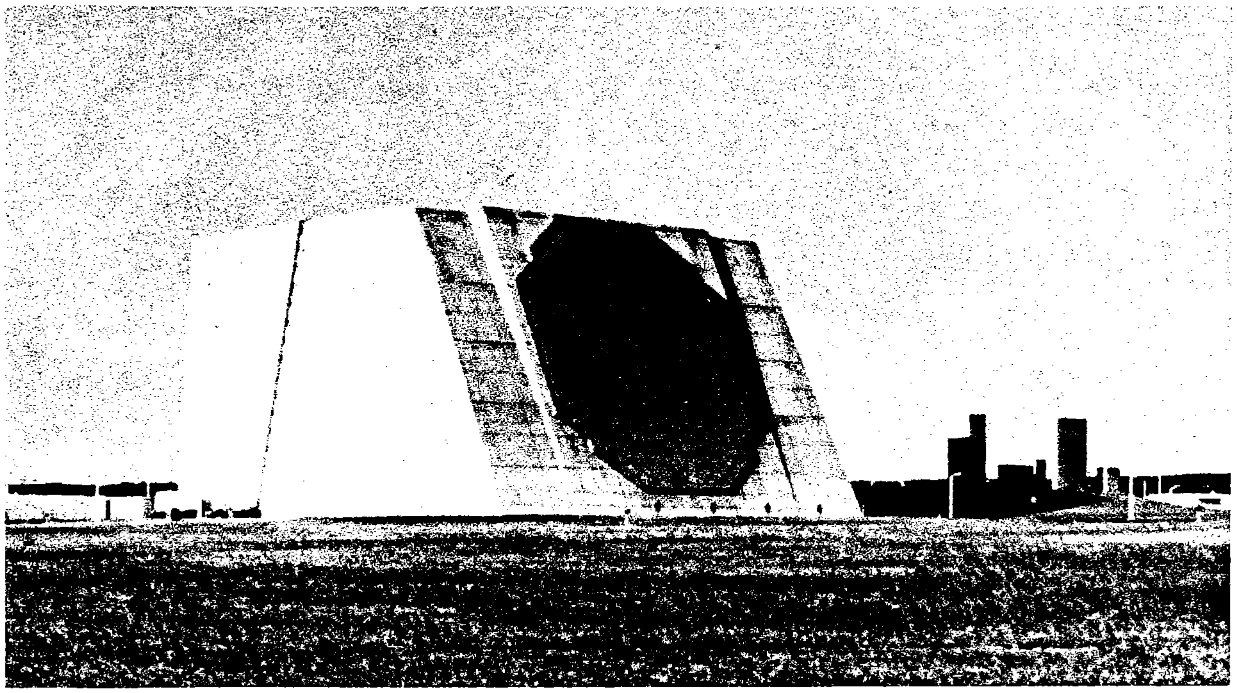


Figure 4-2. PAR Building

To withstand nuclear effects, the radar equipment is housed in a reinforced concrete structure shielded against nuclear electromagnetic radiation.[10] A large portion of the building is below ground with an above-ground turret containing the four phased arrays to provide hemispheric coverage (see Figure 4-3). Each of the four arrays consists of 5001 elements used for both transmission and reception.

The MSR scans the complete hemisphere and can detect targets of small cross section at ranges of several hundred nautical miles.[13,14] The antenna beam pointing direction and operating frequency can be switched between points in space between transmissions. Separate transmit and receive feedhorns are used behind each MSR space-fed array.

The MSR transmitter final amplifier has two high-power klystrons operating in parallel, each with its own power supply. This permits operation at half power, should one unit fail. The transmitter output power is in the megawatt range at high duty cycle. Target returns are amplified by a cooled parametric amplifier before dechirping to achieve the required sensitivity. Further information on the design and performance of the MSR is given in Chapter 7.

SPRINT Subsystem

The SPRINT Subsystem consists of high-performance, ground-radar-controlled, nuclear-armed interceptor missiles and associated support equipment. The missiles are emplaced and ready for launch from underground cells which, together with their support equipment, are deployed in launch areas, or farms, at the MSR collocated site (Figure 4-4) and at farms remote from the MSR.

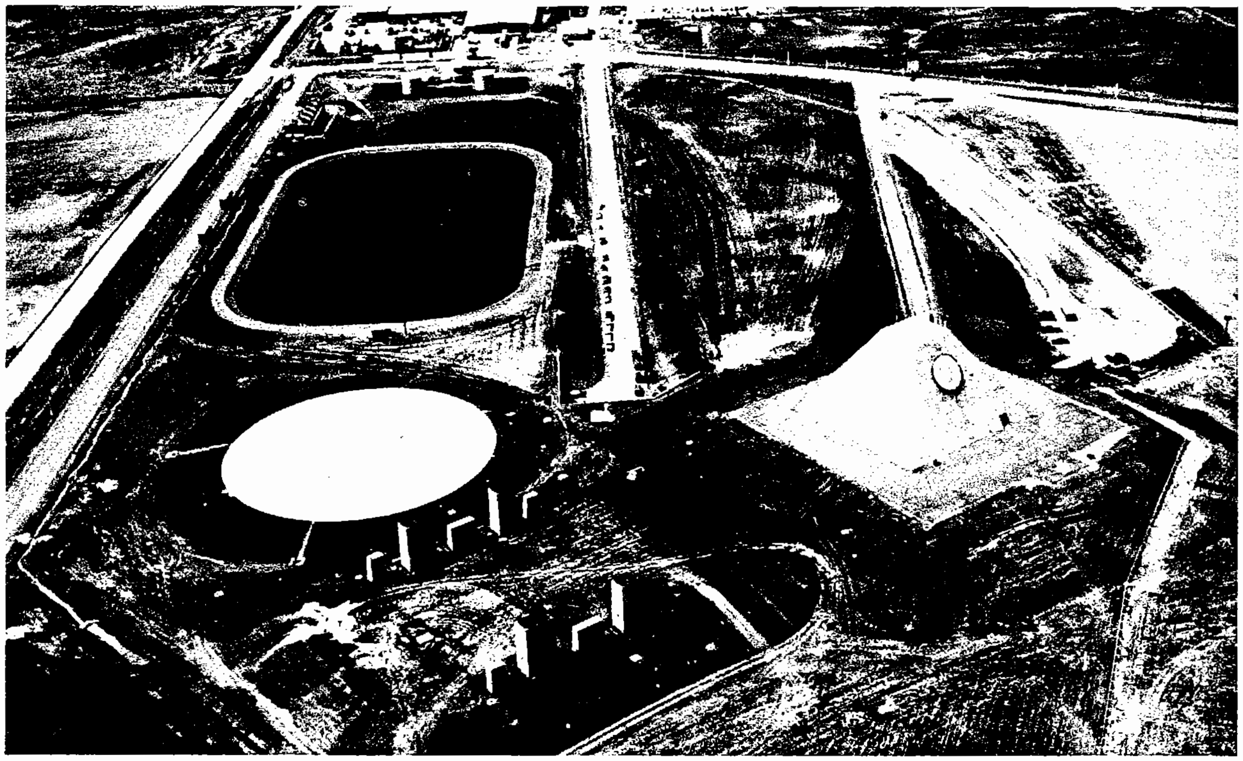


Figure 4-3. MSR Site

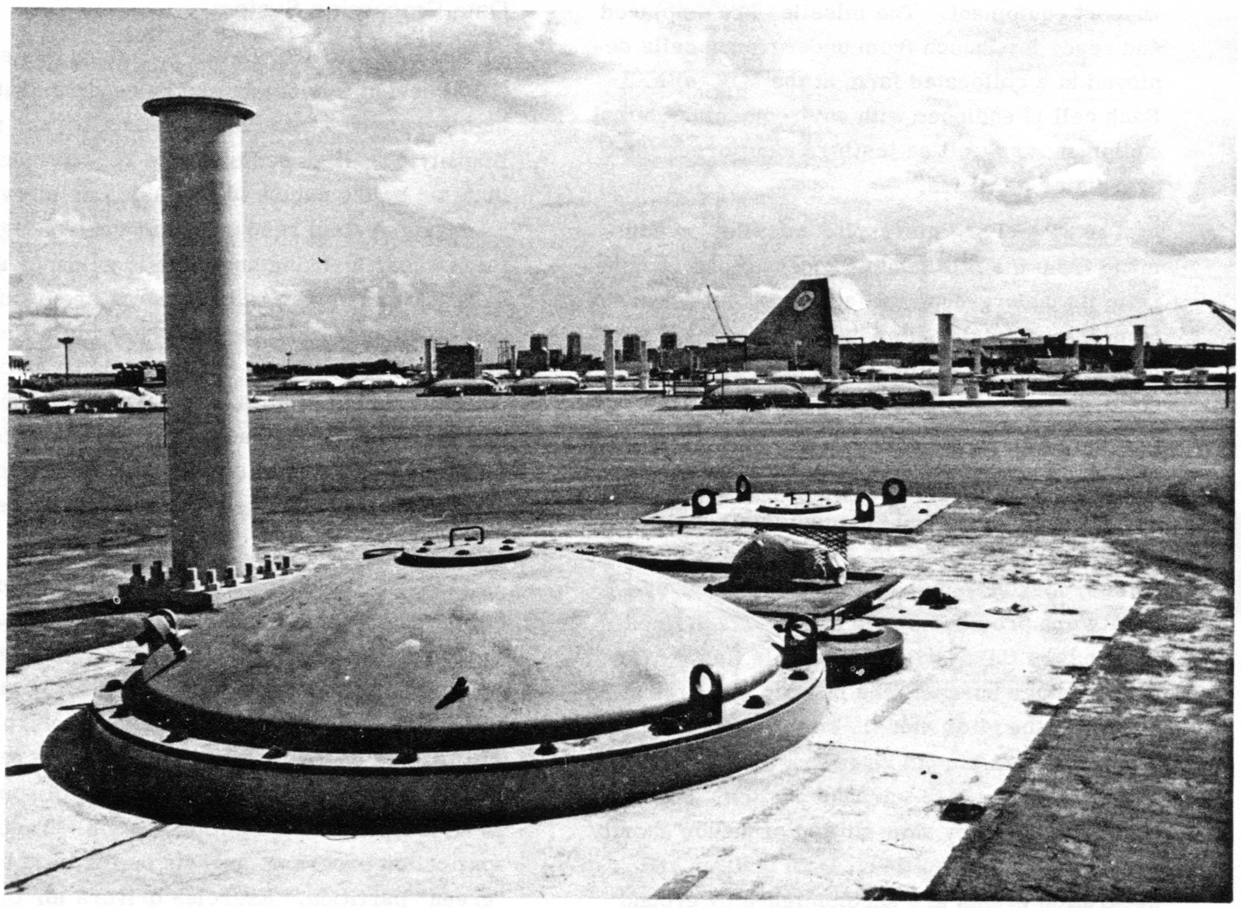


Figure 4-4. SPRINT Cell with MSR in Background

The SPRINT interceptor missile is designed to provide fast reaction-time in reaching endo-atmospheric intercept points.[15] At launch, the missile is ejected from its underground cell and its first stage ignites automatically. A pitchover maneuver is initiated prior to acquisition and track by the MSR. A "missile model" in the computer tells the radar where to look on the planned trajectory so that acquisition can be made as soon as a clear transmission path is available. After first-stage burnout, the second stage is ignited by ground command. The second stage burns for a short interval during which final corrective steering orders are issued and the missile attains the altitude necessary for intercept.

In-flight fratricide related to launch station spacing is a limiting constraint on the rate-of-fire. The launch equipment is capable of accepting launch orders and initiating the launch sequence in a very short interval after receipt of a preparation order. This interval includes the time alloted for tests requested either locally or by radar command and can be reduced somewhat if these tests are not conducted. Refer to Chapter 9 for further details on SPRINT.

SPARTAN Subsystem

The SPARTAN Subsystem consists of high-performance, ground-radar-controlled, nuclear-armed interceptor missiles and associated support equipment. The missiles are emplaced and ready for launch from underground cells deployed in a collocated farm at the MDC site. Each cell is equipped with environmental control equipment, as well as testing, monitoring, and exercising facilities.

The SPARTAN interceptor missile, on command from the MDC Data Processor, is launched from its underground cell at an angle of 5 degrees from the vertical. After launch, the missile follows a radar-controlled flight plan using three solid-propellant-powered stages and two in-flight separations.

After first-stage boost and burnout, the second stage is ignited automatically and booster separation is achieved by second-stage thrust forces. Second-stage control and maneuverability are provided by the second-stage fixed fins and the third-stage deflectable fins. After second-stage burnout, the missile exits the atmosphere in the glide mode. For intercept to occur, the burned-out second stage must be separated from the third-stage warhead section. Separation is achieved by skin-cutting ordnance shortly after third-stage ignition. Both ignition and separation events are accomplished by ground command and occur in the exoatmosphere region. Third-stage exoatmospheric control is accomplished by directing the exhaust from the third-stage motor through nozzles which are an integral part of the third-stage fins. Finally, the nuclear warhead is spin-stabilized during the latter portion of third-stage burn.

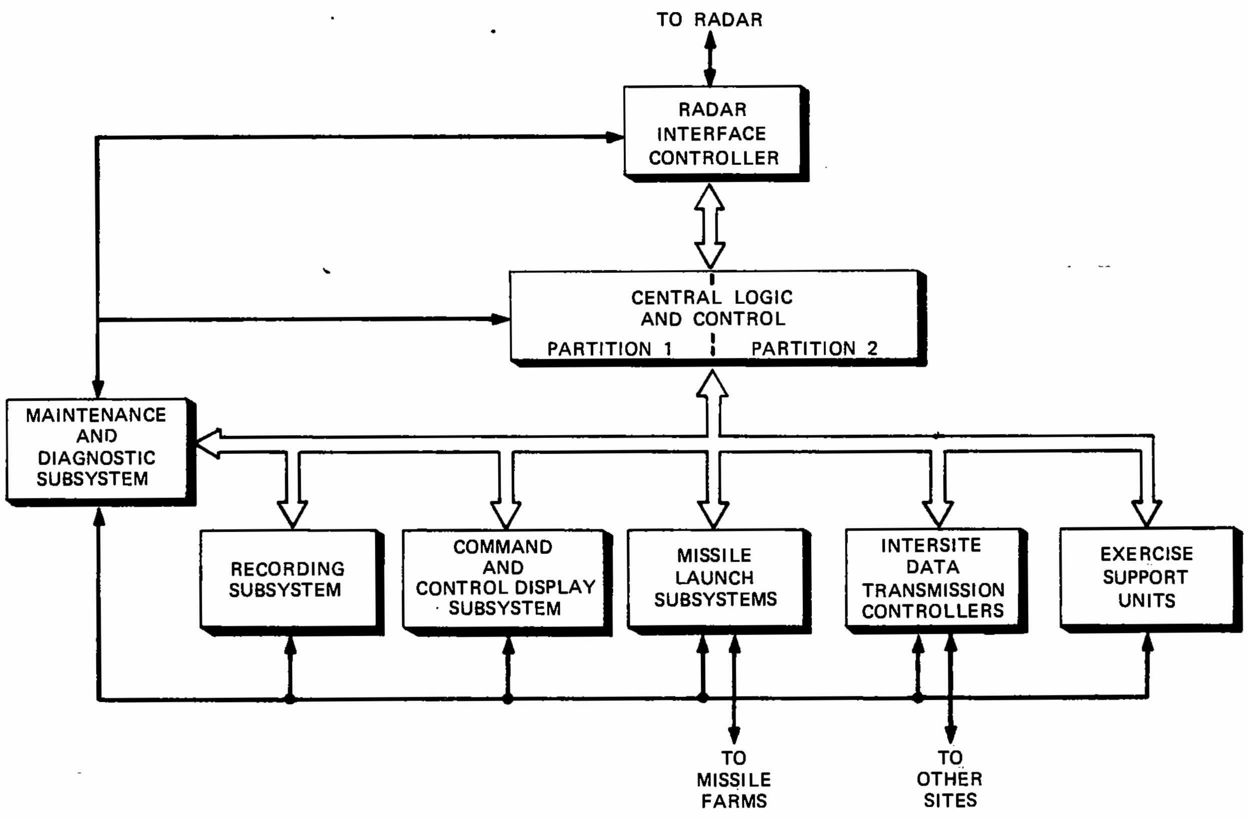
The SPARTAN launch equipment consists of the launch and test equipment and the launch re-radiation antenna. The launch equipment provides for testing launch station hardware and missiles under control of the MDC Data Processor. This testing detects any faulty or unsafe condition to ensure that the subsystem is in a battle-ready state permitting missiles to be launched in rapid-fire sequence. Refer to Chapter 10 for further details on SPARTAN.

Data Processing System

The SAFEGUARD Data Processing System (DPS) design was dominated by requirements for high performance and high availability and reliability.[16-18] High performance requirements are dictated by the nature of the DPS's primary job: managing system resources and controlling a large radar tracking and missile guidance system in real time. After radar returns are processed, new commands must be generated and transmitted to the radar every few milliseconds on a fixed schedule. This results in peak processor throughput of 10 million instructions per second and peak transmission rates to service the radar and other peripheral devices of 200 megabits per second.[19-21]

The DPS equipment at a Missile Direction Center is shown in Figure 4-5. The Central Logic and Control (CLC) is the multiprocessing computer used to drive the radar and the other peripherals shown. Under software control, the CLC can be configured into two separate partitions of arbitrary size, each capable of operating as an independent computing system. Tactical application programs execute on the larger or "green" partition. Exercise drivers for the application programs and independent support-activity programs execute on the smaller or "amber" partition, which also provides a reserve of spare equipment.

The CLC can be configured with up to ten processors.[22] Single-processor throughput of about 1.5 million instructions per second is achieved by a combination of design techniques that include instruction execution overlap and use of high-speed arithmetic algorithms.[19,20] Every processor has access to each of several read-only instruction memories called program stores, and several read-write memories called variable stores. These stores have a memory-cycle time of 500 nanoseconds and a double-word size of 64 bits to provide a memory bandwidth in excess of that required for maximum performance



TO RADAR
RADAR INTERFACE CONTROLLER
CENTRAL LOGIC AND CONTROL
PARTITION 1
PARTITION 2
MAINTENANCE AND DIAGNOSTIC SUBSYSTEM
RECORDING SUBSYSTEM
COMMAND AND CONTROL DISPLAY SUBSYSTEM
MISSILE LAUNCH SUBSYSTEMS
INTERSITE DATA TRANSMISSION CONTROLLERS
EXERCISE SUPPORT UNITS
TO MISSILE FARMS
TO OTHER SITES

Figure 4-5. Data Processing System Equipment

of a single processor. All data transfers between the CLC and its peripherals are controlled by an Input/Output Controller (IOC), thereby allowing processing and I/O tasks to be accomplished simultaneously.[21]

Table 4-1 shows the CLC configurations, both green and amber, in the SAFEGUARD System.

Several major subsystems peripheral to the CLC are shown in Figure 4-5. These include:

- Radar Interface Controller - provides the primary interface between the IOC and the radar for exchange of control and data words. This unit accepts formatted binary words from the CLC and distributes data to the appropriate radar subsystems.
- Missile Launch Subsystems - convert CLC commands into control signals for the collocated and remote missile farms; this equipment also receives missile status information for encoding and transfer to the CLC.
- Intersite Data Transmission Controllers - provide transfers of digital data via data links between sites.[23-26]
- Exercise Support Units - required in system exercises to provide an interface between the simulated threat running in the amber part of a partitioned DPS and the tactical programs running in the green partition.
- Command and Control Display Subsystem - provides the man-machine interface for monitoring and directing tactical operations through Cathode Ray Tube (CRT) displays, consoles, teletypewriters, and wall displays.[27-31]
- Recording Subsystem - contains the standard computer peripheral devices: magnetic tape transports, disk memory units, line printers, and card reader.
- Maintenance and Diagnostic Subsystem (M&DSS) - provides a centralized facility for testing and maintaining digital hardware to ensure the availability and readiness of the DPS.[32-35] The M&DSS uses unique

Table 4-1

CLC Configurations

Equipment	MDC			PAR			BMDC
	Green	Amber	Total	Green	Amber	Total	
Processor Units	7	3	10	3	2	5	1
Program Stores	9	3	12	4	3	7	3
Variable Stores	10	5	15	10	4	14	3
IOCs	1	1	2	1	1	2	1
Timing & Status	1	1	2	1	1	2	1

dedicated hardware to conduct non-real-time, programmed diagnostic tests on the DPS equipment through an independent data bus connected to each digital unit. The M&DSS also supports initialization of the DPS hardware and assists in automatic reconfiguration and recovery of this hardware should it be necessary during the tactical mission.

For more information on the DPS equipment, refer to Chapter 11.

SOFTWARE DEVELOPMENT

Development Cycle

The SAFEGUARD development benefited substantially from prior work on the R&D prototype called the Meck Test Facility. This effort provided an early demonstration of critical ABM concepts and also collected data needed for subsequent design and evaluation. The many lessons learned were applied directly to the SAFEGUARD program.

The software development cycle involved a number of distinct phases. These phases overlapped, since the general approach used required integration of a basic working system with increasingly more complex capabilities.[36] This integration was performed at a separate test facility, the Tactical Software Control Site (TSCS), using special system exerciser arrangements.

The separate phases of the development cycle were as follows:

- Requirements Generation - initial system requirements were determined, established, negotiated, documented, and rigorously controlled.
- Software Design - in process design, system requirements were translated into a software architecture which defined global structures, tasks, task priorities, and task timing requirements for the data processing environment. In program design, the local data base, algorithms, and control structure for individual tasks were determined.
- Coding and Unit Testing - codes were written, compiled, and checked at the unit or task level using a simulator, drivers, and standard debugging techniques.
- Process and Functional Integration - facilities of TSCS were used to combine blocks of new debugged unit code into processes of increasing functional capability. When the tactical software attained a predefined level of capability, it was sent to site for final integration.

Activities at site were similar to those at the TSCS. However, at site the entire complement of peripheral hardware was available for integration with the system. The final phase of system development involved system testing, in which the PAR, MDC, and BMDC sites were "netted" to achieve coordinated operation of the entire system. After this, formal acceptance tests were run at site.

During all phases of system development, evaluation played a strong role. A separate department was responsible for evaluating system requirements, implementation algorithms, and prototype and system test results. Feedback resulted in frequent changes and refinements in many areas.

In the following paragraphs, the software development activities outlined above are discussed more fully.

Meck Test Facility

The Meck Test Facility consisted of a prototype MSR, SPRINT and SPARTAN missile subsystems, and a DPS. During the R&D phase, ABM system requirements were constantly changing; thus it was decided to implement a series of test processes on Meck, each with increasing complexity and more stressing system objectives.

The first process, M-0, was designed to test the radar and missile hardware subsystems. The next process, M-1, tested both SPRINT and SPARTAN intercept capability in a light-threat environment. The final process, M-2, verified SAFEGUARD tactical algorithms for missile guidance, as well as radar surveillance and tracking functions. This inclued remote launches of SPRINT missiles from Illeginni Island. The testing took place from early 1969 through 1974.

A total of 882,000 program instructions were written for the Meck Test Facility, but only 255,000 instructions were included in the three test processes. The remainder were needed for hardware installation and maintenance programs (137,000 instructions) and support software, such as assemblers, process construction, simulation packages, etc. (490,000 instructions).

In addition to the planned testing and verification, the Meck effort made other contributions to SAFEGUARD. Many of the techniques, procedures, and practices developed were later to be used in the SAFEGUARD software development. These were in many diverse areas, such as operating system and process design, software configuration control, and real-time debugging aids, to name a few.

TSCS and System Exerciser

To develop and test SAFEGUARD software and to support system deployment, the Tactical Software Control Site was established at Madison, New Jersey. This test facility contains the PAR and MDC data processing equipment and portions of the analog hardware that interface with the missiles and radars at site. The TSCS also includes office space for major elements of the software development organization, consisting of designers, programmers, and test teams, as well as many other support personnel.

The basic requirement of the TSCS was to reproduce accurately the software environment found at site. In short, its objectives were to:

- Verify performance of tactical software in its operational environment
- Test software in conjunction with the design organization
- Reduce development time through comprehensive testing prior to shipment to site
- Support deployment activities by providing a centralized facility to aid in solving problems encountered at site.

The TSCS was required to have a representative complement of DPS hardware for both the PAR and the MDC. This meant two distinct DPS configurations, each sized in accordance with tactical requirements (see Figure 4-6). Furthermore, each configuration had to replicate the interfaces between computer and peripherals, and the operation of peripherals to the extent necessary for tactical-like responses. In addition, the capability to net the PAR and MDC was required for system testing.

The tactical equipment at TSCS was used almost exclusively to test and integrate the SAFEGUARD operating system, application software, and installation and maintenance software. The support functions needed to develop, control, and analyze tactical software were relegated to general purpose computers.[37] An IBM System 370 was installed at TSCS and off-premises access to a HIS 635 was provided.

A system exerciser was developed to test tactical software in the SAFEGUARD equipment at TSCS. A separate exerciser was provided for

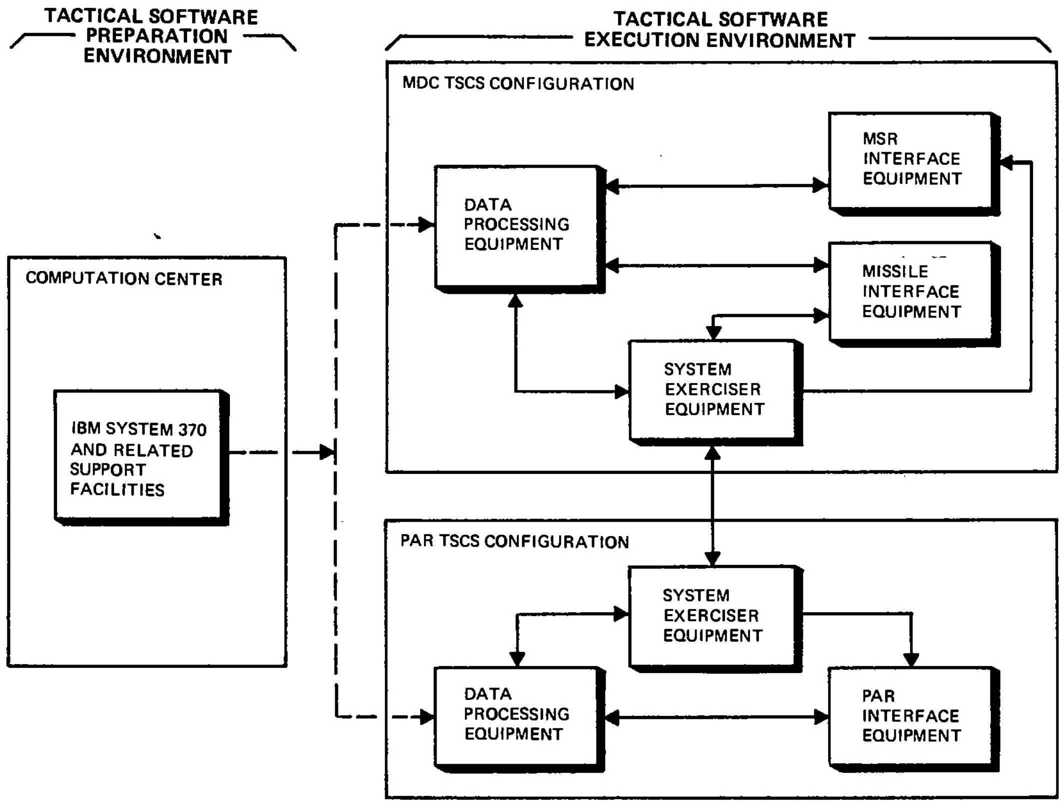
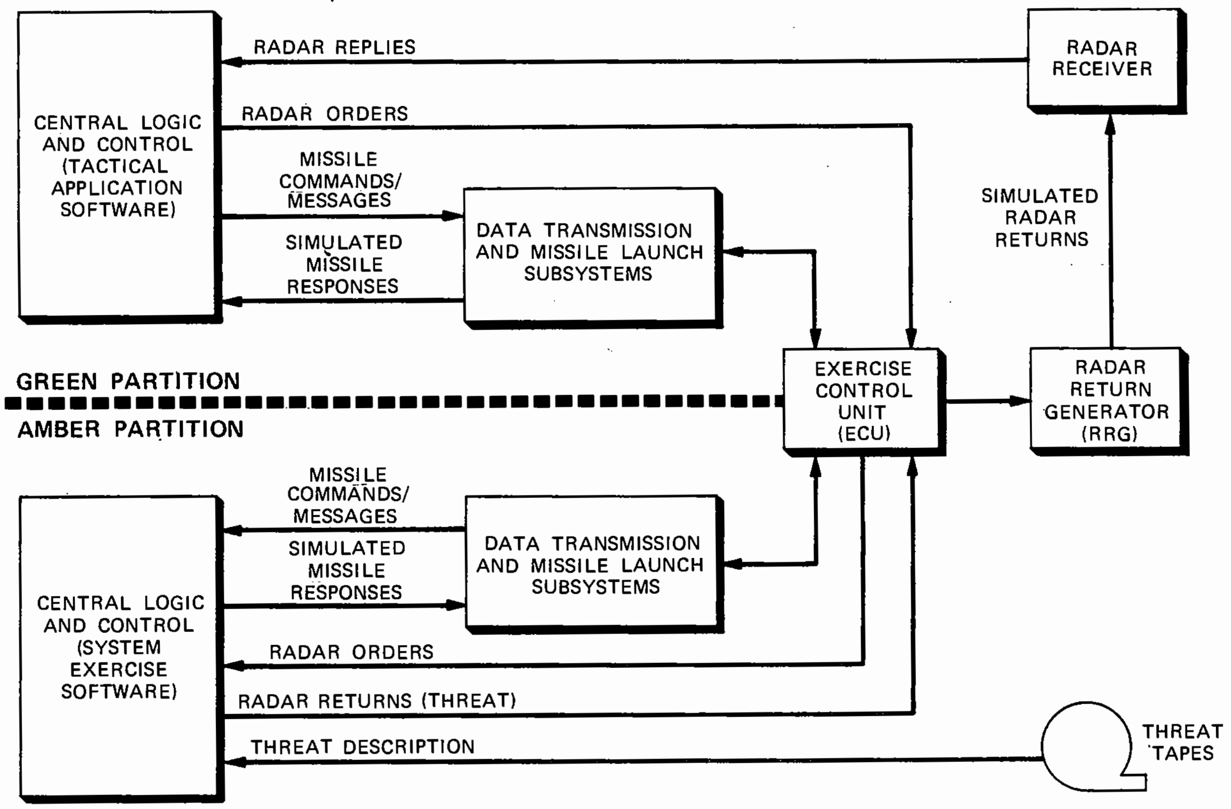

TACTICAL SOFTWARE PREPARATION ENVIRONMENT
TACTICAL SOFTWARE EXECUTION ENVIRONMENT
MDC TSCS CONFIGURATION
MSR INTERFACE EQUIPMENT
DATA PROCESSING EQUIPMENT
MISSILE INTERFACE EQUIPMENT
SYSTEM EXERCISER EQUIPMENT
COMPUTATION CENTER
IBM SYSTEM 370 AND RELATED SUPPORT FACILITIES
PAR TSCS CONFIGURATION
SYSTEM EXERCISER EQUIPMENT
DATA PROCESSING EQUIPMENT
PAR INTERFACE EQUIPMENT

Figure 4-6. Tactical Software Control Site Functions

each configuration (PAR and MDC) for individual testing and for joint testing of the two systems in netted operation.[38, 39] These exercisers are used now in the deployed system for site and system readiness verification.

In designing the system exerciser, care was taken to devise a method that would realistically produce an environment similar to the actual tactical conditions found at site. Two special hardware aspects provided this capability. First and most important, the PAR and MDC configurations were each configured into two separate partitions as shown in Figure 4-7. The larger, or green partition, contained the tactical application software as it would be used in an actual engagement; the smaller, or amber partition, contained the system exerciser software. Second, special exercise support hardware was developed: an Exercise Control Unit (ECU) and a Radar Return Generator (RRG).

During a system exercise, the amber partition generates radar returns representing a preselected threat. These returns are sent via the ECU to the RRG for conversion to analog form and injection into the radar receiver of the green partition. The tactical (or green) partition responds as in a real engagement, issuing radar orders, missile commands, and intersite communications. These are sent via the ECU to the amber partition where they control simulated missile responses. The ECU also appropriately routes these responses and intersite communications from the amber partition to the missile ground equipment and tactical communications links in the green partition.



RADAR REPLIES
RADAR RECEIVER
CENTRAL LOGIC AND CONTROL (TACTICAL APPLICATION SOFTWARE)
RADAR ORDERS
MISSILE COMMANDS/ MESSAGES
SIMULATED MISSILE RESPONSES
DATA TRANSMISSION AND MISSILE LAUNCH SUBSYSTEMS
SIMULATED RADAR RETURNS
GREEN PARTITION
AMBER PARTITION
EXERCISE CONTROL UNIT (ECU)
RADAR RETURN GENERATOR (RRG)
MISSILE COMMANDS/ MESSAGES
SIMULATED MISSILE RESPONSES
DATA TRANSMISSION AND MISSILE LAUNCH SUBSYSTEMS
CENTRAL LOGIC AND CONTROL (SYSTEM EXERCISE SOFTWARE)
RADAR ORDERS
RADAR RETURNS (THREAT)
THREAT DESCRIPTION
THREAT TAPES

Figure 4-7. Functional Configuration for System Exercise

When the system exerciser is used in the netted mode, the tactical communication between the PAR and MDC is not interrupted for any simulated response, but is allowed to flow normally on one circuit of the split link. The other circuit is used for communication between the PAR and MDC exercisers.

The system exerciser has sufficient traffic capacity to test tactical hardware and software at the required design level. Yet, the amber hardware requirements were kept to a minimum by performing as much simulation as possible prior to conducting a real-time exercise. A program called the SAFEGUARD Threat Action Generator (STAG) was developed to generate target, defensive missile, and intersite message information (for local exercises) in a scenario defined by an input deck. STAG is run in non-real-time on the IBM System 370 and records the simulated threat on tape. These tapes then are used during an exercise by the exerciser software in the amber partition to generate the simulated responses.*

Requirements

The Data Processing System Performance Requirements (DPSPRs) are a set of documents that define the requirements of SAFEGUARD tactical software for the PAR, MDC, BMDC, and system exerciser.[38-40] Requirements were generated by the system engineering organization in accordance with overall system objectives.

*For additional information see SAFEGUARD Data Processing System: Process-System Testing and the System Exerciser, B. P. Donohue III and J. F. McDonald, Bell System Technical Journal - Special Supplement (1975).

The primary objective of the DPSPRs was to specify required functional performance in sufficient detail to permit development of software by the designers, yet not limit design freedom. A second objective was to state functionally how the system was to operate in its different defense modes.

Changes in requirements were made through the course of development as a result of feedback from the software development organization, the SAFEGUARD System Tests conducted at Kwajalein and Grand Forks, system evaluation studies, and from detailed review by the Army SAFEGUARD System Command (SAFSCOM). Later on, formal change-control procedures were instituted on these documents to ensure an up-to-date system definition of SAFEGUARD performance and to control the final software products.

Final system testing and acceptance requirements generated by the system engineering organization were based on the DPSPRs.

Software Design

The collection of application software used to drive the DPS is called the application process and is built from basic computing units, called tasks, which are single routines with or without subroutines. The operating system, considered to be part of the process, schedules tasks from a predetermined, priority-ordered task list for execution on the next available processor.[41,42] Once in execution, a task is not interrupted before completion except for error conditions.[43]

Process design is the definition of overall software structure including task assignment and global data base design. The objective of process design is to meet system requirements with a minimum-cost DPS configuration. This activity was complemented by program design which involved developing the algorithms, internal data base, and control structure necessary to implement the function defined for a task. The software design was documented in process design specifications and process workbooks. Between this detailed level of documentation and the DPSPRs, some application process organizations also produced intermediate-level software functional design requirements.

In many areas, various levels of simulation were used to validate the design. In some cases, a few selected equations were implemented on a time-sharing system for a quick exploration of correctness and adequacy. In others, a subset of the real-time computer program, complete with its interface structures, was simulated. In the PAR development, the intermediate-level functional design requirements, together with radar, threat, and environmental considerations, were modeled and simulated to study expected system performance.

The size of individual programs and the time required for their execution were two major parameters that were carefully controlled. Initial sizing and timing estimates were made early in development based on past experience with similar programs. Throughout the course of further development, sizing and timing estimates were tracked on a monthly basis.

It was found essential to initiate design of the data recording and reduction system early in the development cycle. An attempt was made to define the data to be recorded for each computing function and to design the data base with the ultimate use of the recorded data kept in mind.

Coding and Unit Testing

Most SAFEGUARD software was written in CENTRAN, an extensible intermediate-level language resembling a subset of PL/1.[44] CENTRAN is a macro-based language built on SNX, the assembly language for the CLC. As such, CENTRAN provides many of the advantages of a high-level language but can be interspersed with assembly language statements and system macros as necessary. CENTRAN generates efficient code and was adopted as the project standard.

A project-wide attempt was made to include defensive programming techniques during the coding period. Particular attention was given to computations that potentially could cause interrupts which, in turn, would result in task termination. Detection and error recovery codes were then added. Another project-wide technique involved enforcing adequate source-code documentation by requiring well-commented program listings. This proved particularly effective later on in isolating programming errors that were detected during the integration phase.

All programming organizations responsible for a functional area of the application software held design reviews for the total project. These reviews not only helped to educate test and integration personnel, but sometimes uncovered subtle design flaws. Some programming groups used structured programming. These techniques led to an orderly top-down approach in the detailed programming design and coding. In such instances, a substantial saving, both in variable store and in computing time, was often achieved.

Once the coding of individual software units was complete, two stages of testing were undertaken. Units assigned to individual programmers - generally entities of code ranging from 100 to approximately 1000 machine instructions - were individually tested. The purpose of this unit testing was to test the logic paths within a unit. To support this activity, a number of special drivers were developed to allow individual units to be interfaced with the data sets needed in their operation. These drivers sequentially initialized the data sets to produce the environment necessary to execute one logic path after another. Element testing was the second stage of testing. Program units comprising a major functional element of the process were tested together using special software "stubs" in place of the other major functions. These tests were primarily functional in nature. A series of tests was defined that would exercise all major functional requirements given in the system requirements and design specifications.

All software preparation and most of the unit and element testing was performed using commercial computers. This was primarily because time on the TSCS test facility was too limited to support such extensive activity. However, a decided advantage accrued from this approach since a number of testing and debugging tools were already available on the commercial machines. Others were developed specifically for SAFEGUARD in high-level languages such as FORTRAN and PL/1. One such tool was the simulator called STACS (SAFEGUARD Tactical Computer Simulator) which provided the primary unit and element testing vehicle.[45] STACS fully simulates, on the IBM System 370, a single CLC processor and most of the conventional CLC peripheral units. It also simulates many of the operating system capabilities designed for the CLC. Special test drivers were designed to interface with STACS, providing universal testbeds. In addition, a variety of debugging aids were included in STACS to allow exhaustive testing of program units. Such debugging aids range from simple snaps and traces to simulation of error response and the ability to modify or patch a unit.

Configuration Management was an important aspect in development of the software product.[46-49] Methods were developed to identify, control, track, and audit the numerous versions of the programming units to be released for process integration. These methods included rigid control of the patches required to correct the coding problems discovered during integration.[48,49] Central to this activity was a disc-based library system which was an IBM proprietary product. This system offered many of the features necessary to sustain a large development effort. For example, it included an editor for changing source lines, a means of temporarily changing source for testing, and a mechanism to facilitate delivery of debugging code. Perhaps its most significant contribution was the procedural discipline it uniformly forced on all SAFEGUARD programmers.

Other procedures were developed to control all patches made to delivered software. Trouble Reports (TRs) were required to document any problems occurring during testing. Corresponding

Correction Reports (CRs) described the approach used in correcting the trouble via a patch or source change. A large number of support software packages were developed to assist in these configuration management procedures, and included programs ranging from the building of patch files for a process to the identification and reporting of status on TRs and CRs.[48,49]

Process and Functional Integration

Following unit and element testing, collections of functions were combined and tested on the PAR and MDC DPS configurations at TSCS. For example, the basic control programs for the MDC were first merged with the operating system, and the ability to load, initialize, and cycle was established. Then, software for the radar loop was added, i.e., radar management, search, and track programs.

Ability to search and track was first established at low traffic levels, while the radar hardware was simulated by software drivers. After basic operation was established for the software, the radar hardware was introduced into the testing loop. In parallel with this activity, application programs supporting intersite communications and command and control were tested in a separate configuration at TSCS. Similarly, both battle planning and missile guidance software were tested in separate software environments. Ultimately, these programs were merged into a single MDC application process and the complexity of the test cases was systematically increased.

This integration was done separately for the PAR and MDC with the effort planned, as seen above, so that a number of parallel activities could be carried out by independent integration teams. Such an incremental approach allowed for integration of increasingly more complex capabilities into the software, building from a simple nucleus of code and culminating in a full application process for each site.

Early in this integration, drivers were developed to assist in the first steps. These early efforts, referred to as process integration, were devoted mainly to software considerations such as basic cycling, sanity, and interface testing between major software blocks. Later, when the system exerciser was available, it was used exclusively to drive and stress the application processes under various conditions and loads. This testing also covered the radar hardware at TSCS and was planned to include specific functions given in the requirements. This functional integration testing culminated in demonstrating satisfactory performance in both the local and netted modes of the large scenarios planned for eventual system testing at site. Literally thousands of individual test cases were documented, including the success criteria, and then run in support of the overall process and functional integration phases.

A number of support software facilities were crucial in building and testing these large application processes. (See Figure 4-8.) CENTRAN and SNX, like most compilers and assemblers, produced relocatable object code. A program was developed to allocate CLC memory, build the control tables needed by the operating system, perform binding functions, structure the overlays, and perform a host of related services. This program, called the Execution Preparation Facility (XPF), operated on the IBM System 370.[50] Similarly, most testing of software units and elements produced by XPF was performed by STACS on the IBM System 370.

Another software package, called DEBUG, provided software debugging aids for the CLC and proved invaluable early in process integration.[51,52] DEBUG's capabilities included many of the same aids provided on the support computers by such facilities as STACS. Although DEBUG contained some multiprocessor capabilities, its design was oriented more toward a single processor, non-real-time environment. Further extension of debugging aids on the CLC resulted in the development of DARTS (Debugging Aids for Real-Time Systems). This software provided full multiprocessor, real-time capabilities with minimum perturbation on the

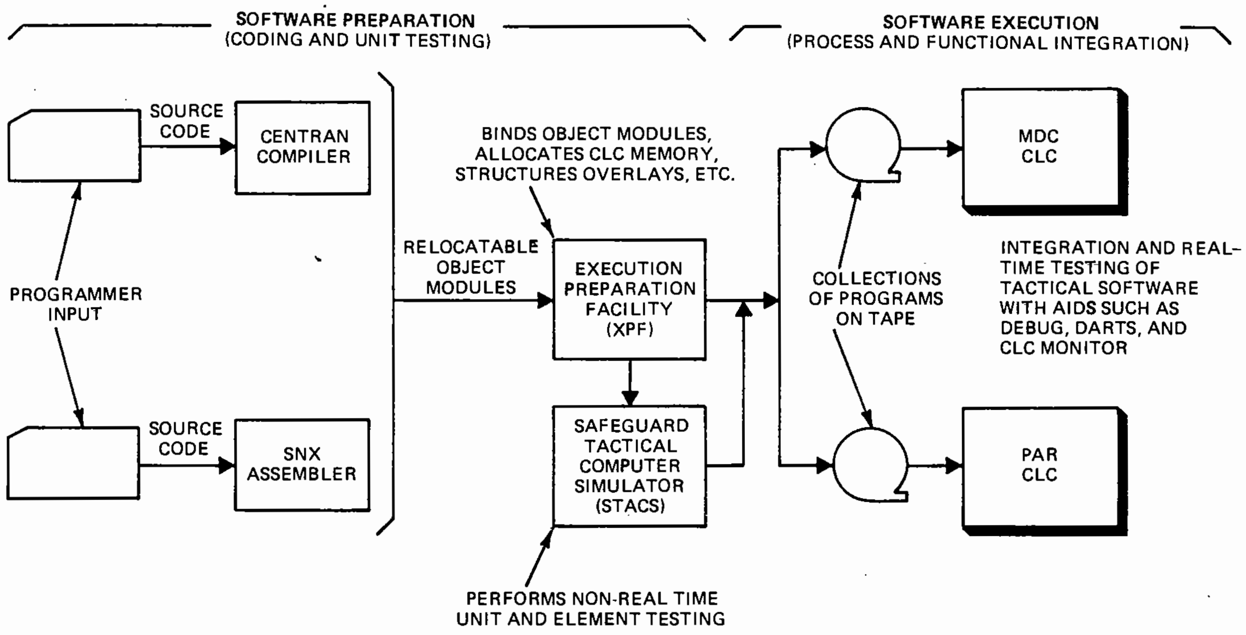

SOFTWARE PREPARATION (CODING AND UNIT TESTING)
SOFTWARE EXECUTION (PROCESS AND FUNCTIONAL INTEGRATION)
SOURCE CODE
CENTRAN COMPILER
PROGRAMMER INPUT
SOURCE CODE
SNX ASSEMBLER
RELOCATABLE OBJECT MODULES
BINDS OBJECT MODULES, ALLOCATES CLC MEMORY, STRUCTURES OVERLAYS, ETC.
EXECUTION PREPARATION FACILITY (XPF)
SAFEGUARD TACTICAL COMPUTER SIMULATOR (STACS)
PERFORMS NON-REAL TIME UNIT AND ELEMENT TESTING
COLLECTIONS OF PROGRAMS ON TAPE
MDC CLC
PAR CLC
INTEGRATION AND REAL-TIME TESTING OF TACTICAL SOFTWARE WITH AIDS SUCH AS DEBUG, DARTS, AND CLC MONITOR

Figure 4-8. Software Preparation and Testing

timing and operation of the application program under test. An array of features was made available, including the ability to simulate manual inputs in real time.

The CLC Monitor was developed to further isolate the effects of running debugging tools on the application process being tested.[53] This monitor is an external hardware monitor which includes its own memory, extensive logic to count and filter data, and two tape units. It was used primarily to validate that process performance was consistent with design. Troubles, such as heavily-loaded time frames and long-running tasks, were analyzed and design changes were made when necessary to provide a more balanced system.

Detailed analysis of functional integration tests was facilitated by designing the real-time recording functions as an integral part of the application and exercise software. Recorded data was reduced and analyzed primarily off-line on the IBM System 370 using the SAFEGUARD Data Reduction System, although summary information was available on-line.[54,55]

Site Integration and System Testing

The TSCS and site integration phases significantly overlapped in time.[56] As tests were completed at TSCS, software updates in the form of patches and data packages relating to performance of the tests were sent to site. These packages included test specifications, exerciser tapes, test results, and reports which itemized any known problems not yet corrected. Site integration then consisted of testing the application software against the full complement of SAFEGUARD hardware using a subset of TSCS tests and special site-oriented tests such as radar-calibration and satellite-tracking tests.

Finally, system testing was carried out at site. This involved a comprehensive series of acceptance tests which demonstrated system capability consistent with requirements and represented the final level of product testing prior to delivery.[57-61] During system testing, it was not possible to exhaustively test all tactical threat environments. Rather, these acceptance tests, or System Technical Verification Tests (STVTs) as they were called, were defined at the design-traffic level for each of the various system operating modes. A series of tests was designed for each mode, at first simulating all communications with other sites, then netting pairs of sites, and finally netting the entire system.

The stress level was reduced in early testing by selecting subsets of the STVT environments and by running buildup tests at these lower stress levels before increasing the traffic load. Once this was accomplished, the "test chain" was continued by physically internetting the sites in stages until the total system was operating at design-traffic levels. This approach, in which all tests in the chain support the STVTs, greatly simplified the problems of integrating a dynamic system.

SAFEGUARD OPERATION AND MAINTENANCE

Operational Overview

The size and duration of the SAFEGUARD development effort were large indeed. Table 4-2 shows the number of machine instructions for the major software components. Real-time software, consisting of MDC and PAR application programs with their exercisers, the BMDC application program, and the CLC operating system, contained a total of 735,000 instructions.[38-42] Support software, such as compilers and simulators, which executed on commercial computers, amounted to 580,000 statements, some in assembly language and some in PL/1 and FORTRAN. Installation and maintenance software for the data processing system and the radars totalled 830,000 instructions. At least several hundred thousand additional instructions were developed for other purposes such as test drivers and specialized simulations. Together this represents nearly 2.5 million instructions written, tested, and debugged over the six years of SAFEGUARD development.

The massive amount of radar and digital equipment installed at site and at TSCS is visibly even more impressive. The radars on the plains of North Dakota are monuments to modern technology housing some of the most complex electronics in the world.[1,2] Within these buildings and at the BMDC site in Colorado and the TSCS in New Jersey, hundreds of digital equipment racks were assembled to provide the computing power needed to control and maintain the deployed system and its test facility.

Table 4-2
Size of Major Software Components

Real-Time Software Instructions	
MDC Application	300,000
MDC Exerciser	50,000
PAR Application	200,000
PAR Exerciser	25,000
BMDC Application	60,000
CLC Operating System	100,000
Total	735,000
Support Software Source Statements	
CENTRAN, SNX, XPF, STACS, etc.	210,000
System Simulation	50,000
Exerciser Support	30,000
Data Reduction	150,000
Configuration Management Programs	70,000
Fault Logic Simulation	70,000
Total	580,000
Installation and Maintenance Software Instructions	
MDC Radar Installation tests	50,000
PAR Radar Installation Tests	110,000
Real-Time PAR Radar Tests	60,000
M&D Operating System	120,000
Maintenance & Diagnostic Tests	300,000
ITP, DUX, RTE, NPD, etc.	190,000
Total	830,000

The remainder of this section briefly describes the operational aspects of the system. The Command and Control Display Subsystem provides the facility for monitoring and controlling tactical operations and maintenance.[27]

Tactical operations are basically concerned with integrating the SAFEGUARD weapon system into the total national defense structure. If an actual threat is detected, the primary tactical function is to provide for the timely release of the system. Maintenance operations are concerned with keeping each element of the SAFEGUARD System at the highest availability level possible.[62-64] This requires monitoring and coordinating maintenance activities for each subsystem consistent with overall tactical demands. Finally, the TSCS test facility takes on a new role during the operational phase of the system — supporting site needs by helping to solve and correct problems as they arise.[65]

Tactical Command and Control

The command and control functions in SAFEGUARD are divided into maintenance and tactical tasks. The maintenance aspects are treated separately in the next section. The tactical functions are designed to be automatic when both feasible and cost effective, with only certain key functions assigned to man. These manual actions complement the automatic system and, except for system release, are not a sequential link in its operation. Hence, man is a controller rather than an operator in the SAFEGUARD System.[27]

The Ballistic Missile Defense Center is the highest echelon of command and control in the SAFEGUARD System. It is collocated with the Commander-in-Chief of the Continental Air Defense Command (CINCONAD) Combat Operations Center (COC) in the Cheyenne Mountain Complex at Colorado Springs, Colorado. CINCONAD exercises operational command of the SAFEGUARD System through the BMDC. The BMDC, in turn, controls and directs the activities of the Perimeter Acquisition Radar and the Missile Direction Center.[40,66]

The COC has special nuclear-employment authority equipment connected directly to the BMDC.[67] In addition, the COC has closed-circuit TV monitors on BMDC displays. These interfaces provide the BMDC with authorization for use of SAFEGUARD nuclear weapons, attack warning and other intelligence data generated external to SAFEGUARD, alert and readiness level coordination with other defense systems, coordination with use of offensive forces, and requests for special intercepts or satellite observations. The BMDC sends SAFEGUARD-generated early warning, attack assessment data, or results of satellite observations to the COC.

Overall manual command and control functions in SAFEGUARD generally relate to implementation of actions associated with this COC-BMDC interface. All such command and control actions at the BMDC, MDC, or PAR require positive feedback to man indicating the application software has recognized the manual input. If a manual action is meaningless or the sequence of actions incorrect, the system indicates this to man. In addition to validity checks on manual inputs, each site performs a validity check on any message it receives from other sites.

A tactical command and control action is classified as either a directive or a control action. A directive is manually initiated by higher authority and requires a corresponding manual action at the PAR or MDC. A control is manually initiated by higher authority and will be automatically effected at the netted PAR or MDC without a local manual action. The Command and Control Display Subsystem includes provision for display of directives (called forced messages) that a tactical officer may implement. The source of each directive accompanies the notification.

System Maintenance

Maintenance is functionally divided into several Maintenance Activity Centers (MACs). The division follows the hardware subsystem division of the site; for example, a DPS MAC, Radar MAC, Missile MAC, etc. The personnel in each MAC are supervised by a Maintenance Director at a maintenance console. From this console, he can request software-controlled subsystem tests that provide current status and fault information. The console also allows him to report his status to the System Maintenance Director. The System Maintenance Director, together with the Equipment Readiness Officer, man a system monitor console in the Equipment Readiness Center.[68,69] In this way, information on all equipment problems is conveyed to the tactical command, with corrective maintenance controlled and scheduled consistent with the tactical needs of the site.

A number of on-line procedures are available to maintenance personnel. The most useful and general method is to conduct System Readiness Verification (SRV) exercises. The SRV uses the system exerciser discussed previously. Scenarios are available to verify specific functional and hardware aspects of the system. The SRV provides a local or netted system test which stresses not only the CLC hardware and its interfaces but also exercises the application software under simulated inputs representative of the design threat. Other on-line tests periodically used for maintenance are the radar Class A and B tests, the missile subsystem Class B tests, Normal Path Diagnostics (NPDs), and Real-Time Exercisers (RTEs). These tools are part of the application software and are used during normal tactical operation.

The Class A tests are cycled automatically by the application software and measure overall performance to ensure that the system is meeting acceptable operating standards. The Class B tests are run on request and are designed for detailed alignment and performance investigations. Whenever the CLC undergoes a reload operation, the NPDs are run to test all processor units, program stores, and variable stores in the green partition, as well as their interfaces, including those to the input/output controllers. Once the application program is cycling, the Data Processing Officer can request the RTEs be run automatically to test each program and variable store every 5 minutes.[31]

A number of off-line maintenance tests also are used. Generally, these tests are designed to check equipment off-line in the amber partition, although a few tests require the full system configuration. The M&D Subsystem provides a fast procedure to isolate hard faults to a chassis or printed circuit board that can be immediately replaced. These tests usually employ fault-location dictionaries that pinpoint the trouble down to a replaceable level.[32-35]

Digital Unit Exercisers (DUXs) also are used to check individual racks.[70] These are special-purpose equipment exercise and checkout programs designed for each unique type of DPS rack. Each DUX program provides the capability to control the functional operation of a rack on a macroscopic level and to dump the contents of individual registers or groups of related registers within the rack. Strictly speaking, these exercises are not tests but have proved valuable in isolating and solving maintenance problems. Some Installation Test Programs (ITPs) developed for site integration have also proved useful in resolving intermittent timing problems and other non-hard faults.[71] The ITPs are being retained at site.

All of these tests are used for periodic Preventive Maintenance (PM) procedures. Scheduled PM activities are carried out according to procedures contained in the Contractor Maintenance Data System (CMDS). As PM checks become due, maintenance personnel follow the steps listed in the CMDS, reporting any faults that occur to a maintenance director. Corrective repair activity is coordinated with the Equipment Readiness Officer to return the system to a fault-free state. At less frequent intervals, periodic validation of system maintenance is performed using closed-loop tracks taken on test objects, such as satellite and calibration spheres external to the system. These techniques help to achieve a cost-effective level of maintenance and provide assurance that the system is properly calibrated and aligned.

All repair operations are conducted off-line. When a faulty unit is identified, it is removed from the system and replaced with a working spare. The faulty unit is repaired in the Technical Maintenance Repair Center (TMRC). Trained technicians at the TMRC use special test sets and documentation to repair the unit and return it to a usable spare stock.

TSCS Operational Support

During the operational phase of SAFEGUARD, the primary role of the TSCS is to assist the sites in resolving any problems that arise. The Configuration Management procedures used during the development phase have been expanded to ensure that any changes in the operational weapon system are handled in an orderly and methodical manner.[46,47]

Configuration Management starts with identification of a baselined product. The baseline consists of the software and its high-level requirements as used at site. These software products include the application software and those software tools needed to control, install, test, and validate the application software at site. The documentation includes Performance Design Specifications (PDSs), Data Processing System Performance Requirements (DPSPRs),[8,38-40] User-Oriented Requirements (UORs), and System Design Requirements for Displays and Controls (SDRDC).[27] This baseline was placed under Configuration Management control and was the version used in final testing.

Changes to this software baseline and SAFEGUARD hardware are rigorously controlled by a series of Contractor/Army Review Boards. Any problem or proposed improvement, either hardware or software, is first documented by a Master Problem Report (MPR), which is an outgrowth of the TR/CR forms used during development. These MPRs are referred to the local design organization at the TSCS. Proposed solutions are reviewed by a Change Review Board for technical accuracy and completeness. Technical approval for change is obtained from a Direction Committee comprised of Contractor and Army management. Approved technical changes that affect the baseline are submitted as Engineering Change Proposals (ECPs) to a local Configuration Control Board. If requirements are involved, the ECP also is sent to a System Board for dollar and resource approval. No changes are permitted at site without the necessary approvals.

Before an approved software change can be shipped to site, it must be tested and verified at the TSCS. This testing, called Technical Verification, consists of a selected set of system-level tests. Once the contractor has demonstrated these same tests at site, the Army conducts User Operational Verification tests using tactical procedures and operators. At this point, the old software baseline is replaced for operational use by the modified product.

The status of all changes to the baseline is tracked from inception of the MPR to final incorporation of the ECP in the baseline. Hardware changes are tracked by the On-Site Configuration Activity Reporting System which ensures that the current site configuration is always available. Similarly, software status and various reports are generated by the Status Accounting System.

LESSONS LEARNED

General

It is impossible to document all of the lessons learned on a project the magnitude of SAFEGUARD. Each participant will take with him a wealth of experience gained from years of hard, dedicated work. And each will have his own view as to what is most important. Part III of this report, Management and Overall Approach, deals with a number of fundamental issues, often involving management decision, that relate directly to the material covered in this chapter. In the remaining paragraphs, focus is placed on some of the more technical issues encountered, some of which may seem obvious but can easily be overlooked in the rush to see a large job completed.

Support Computers

Early in SAFEGUARD an effort was initiated to build a Prototype Development Facility (PDF) that would allow program development in a time-shared environment on the CLC itself, the SAFEGUARD computer. Such an effort seemed admirable since it would have avoided the cost of providing additional computing hardware for unit and element testing. However, it was a mistake. This elegant time-sharing tool competed with demands for CLC time needed to debug the machine itself, to test the SAFEGUARD operating system, and to support early application development, such as installation software. The CLC was being used for testing generic programs and it could not be tied up developing software tools. PDF was abandoned, and very quickly all development tools were concentrated on commercial machines.[37]

Support Software

PDF also erred in trying to be too ambitious. Other efforts followed this same path. A time-sharing system was developed for the GE 635, one of the general-purpose computers available at Bell Laboratories. This system, the GETSS, became operational and supported 20 to 25 users, and in many respects was ahead of its time in providing editing and execution facilities. But it too was misguided. GETSS could not possibly support the hundreds of programmers eventually required, and more importantly, it was developed for the wrong support computer. The project originally was supported by an assembler on the IBM 7094. By decision outside the project, the 7094 was scheduled to be replaced by a GE 635. This led to choosing the GE 635 machine to support the time-sharing development. However, ultimately, SAFEGUARD management adopted the IBM System 370 for project support, leaving GETSS out of the mainstream.

NICOL, the NIKE-X Compiler Language, was an attempt to produce a high-level language compiler, similar to PL/1, for the CLC. It was a very complicated system and went through a number of versions, including one that was to operate on the CLC. The evolution of the project and changing requirements made it a much too complex endeavor. It, like PDF, was abandoned.

There is a message in this experience. Support software goals must be realistic, particularly in the sense that they be attainable at the time they are required. The purpose of support software is, after all, to support the objective of building systems. Building support software using state-of-the-art techniques is laudable, but only if it contributes to the main objective.

Out of these early attempts evolved a system of viable support software for the project. These products were available when needed, flexible enough to react as requirements changed, and reliable. Various methods were used to achieve these objectives. High-level languages were built on a macro assembler to retain flexibility. In fact, flexibility was considered so important that efficiency was sacrificed. Software was borrowed shamelessly and adapted, but with the knowledge that it would have to be maintained. High-risk, state-of-the-art approaches were avoided. Incremental implementations were planned so that programs could be used as quickly as possible. Strict testing and release procedures were adopted to ensure quality. Programs were "frozen" after release and became subject to change-control procedures. Stringent control was placed over the interfaces among the facilities to ensure integrity. All of these techniques helped to build a successful support software system.

System Exerciser

The system exerciser developed for SAFEGUARD was a major innovation.[38,39] This concept can be widely seen in current development on large software systems. The need to provide test signals and data to "drive" any system is clear. As the complexity of the system and its operating environment increases, so does the complexity of the driver. Therefore, it was judged vital to devote considerable resources to the development of a driver on SAFEGUARD, and the effort was started early.

The costs may seem large at the outset of such a development but in the long run they really are not. They more than repay the effort spent. As much of the hardware should be incorporated in the exercise configuration as is cost effective. The impact of the exerciser on the application system should be kept to a minimum. And then, the exerciser should be tested to create a stable base for system testing.

Patching, Debugging Tools, and Testing

The ability to patch programs in a controlled manner was a major contribution to the SAFEGUARD success. A patch to fix a problem could be delivered within days to the integration team. It eliminated the need for time-consuming source code redeliveries and system reverification except at widely spaced intervals. In addition, patching provided a flexible, easy-to-use tool through which new debugging aids and test tools could be created and readily added to the test configuration.[72]

The importance of unit and module testing cannot be overemphasized. A high percentage of the bugs found in process and functional integration could have been eliminated in the unit testing phase. Therefore, it is highly cost effective to provide extensive unit and module test facilities.

Modularity, Redundancy, and Multiprocessing

In the CLC, the use of well-defined interfaces and modular hardware-building-blocks capable of communicating within the framework of a distributed switching system provide the basis for a dynamic computing complex. This structure incorporating multiprocessing is capable of adding new functional units which offer unique economic or performance advantages. The CLC can be configured in a wide range, from a single-processor to a ten-processor system, and into separate computing facilities to provide an independent exerciser.

This architecture can provide high system availability without the need for costly and complete duplication. An additional computing element (processor, store, IOC, etc.) can be added as a single replacement in the event of failure in that type element. This "n+1" approach has reduced the amount of equipment needed for redundancy and for system exercise to a fraction of that required for a complete standby system.

Multiprocessing did not present the expected conflicts and lock-out problems anticipated. Very few such problems were encountered, possibly because of the single-application nature of SAFEGUARD, as opposed to commercial applications involving an unknown job-mix. But probably more important, care was taken to address process design. The decision to use the task structure, in which each task is a small unit of work, was a good one. These tasks were scheduled by a table-driven operating system rather than being dynamically scheduled. Avoiding the use of interrupts for switching between tasks was very deliberate and was the key to maintaining sanity in a large application with many tasks competing for resources.

It was good design practice to use short-running, low-priority, asynchronous tasks wherever possible. This helped alleviate schedule conflicts that would arise if there were a large number of high-priority synchronous tasks. It guaranteed that high-frequency, high-priority tasks would execute as desired, and aided in distributing the workload over a longer time-frame.

Error Control

The following items are a few key points and recommendations based on the SAFEGUARD experience with error control. The recommendations are generally applicable to any large-scale, real-time control system.

Component redundancy and multiprocessor design are inherently fault-tolerant. Similarly, software architecture is fault-tolerant if it has multiple instances of tasks to do specific functions (e.g., n track functions to update data on any of n objects), if the processing work is divided into many small independent tasks, if the software control is decentralized, if there is emphasis to confine errors locally, and if there are system error responses to reinitialize the system to replace faulty hardware components.

System error control guidelines and structure should be defined early.[43] They are required if the total system is to have a consistent approach to error control. Error logging throughout the system should be the first guideline. The next is to design a structure so that error responses can be easily modified. The logging is an invaluable debugging tool. And as operational experience on error occurrence rate is gained, the response can be appropriately modified.

Finally, manual error control or manual override should be provided even for automatic and self-repairing systems. It is invaluable in "bringing up" a faulty system to isolate a complex problem when the automatic software features have failed to circumvent the trouble.

Maintenance Approach

Experience to date has demonstrated the fundamental power and flexibility inherent in the two primary Maintenance and Diagnostic Subsystem (M&DSS) features. These are the extensive data bus interface with the entire DPS and the general-purpose computing capability of the Maintenance Data Processor.[32-35]

In retrospect, the total range of M&DSS capabilities has yet to be explored fully. For example, because of project schedule constraints, the logic-block partitions originally defined have not been changed. But different partitions, chosen with timing faults in mind, might allow these faults to be handled via the fault dictionary method. (In this case, it also would be necessary to increase the speed of the entire M&DSS which now runs two orders of magnitude slower than many of the internal logic events in the DPS.) Other diagnostic tools, such as the Installation Test Programs, could be restructured with fault isolation more in mind, thus yielding isolation to the chassis level.[71]

Development of Computer Languages

CENTRAN was developed to produce code for the CLC. It is an extensible language whose extensions approximate PL/1 in control structure and FORTRAN in data structure.[44] A number of lessons were learned during its development and use.

The designers, implementers, educators, and users should not be disjcinted groups. The designer should be involved as an implementer to keep in touch with reality. He should also be involved as an educator (if a feature is difficult to explain, maybe there is something wrong with it) and as a user (uniformity in extension is best achieved by knowing how the language is being used.) The implementer should act as both educator and program counselor to get feedback on bugs being "programmed around" and to establish priorities for fixing them.

Several things about the implementer-user relationship should have been learned earlier in CENTRAN development. First, the release cycle should be rigidly controlled as soon as possible, no matter how short the cycle. It does not pay to be a "nice guy" and give "fixes" to bugs informally. Next, old versions of the compiler should not be kept around and certainly not maintained. The maintainers are blamed for bugs that no longer exist, and much time is spent rediscovering causes for problems long since solved.

Notices of new releases must go to everyone, not just supervision. Users often underestimate the impact on schedules of changes due to improvements to the compiler, even though the improvements were requested.

Insofar as the designer-educator-user relationship is concerned, the designers should have been more involved in determining the structure and content of the CENTRAN courses. Frequent symposia (e.g., Advanced Topics in CENTRAN Programming) should have been held, with compulsory attendance.

CENTRAN should have been an expression language. This not only would have aided the production of more efficient, clearer, and more concise code, but would have provided a greater degree of uniformity to the language.

More thought should have been given to data types required for data reduction. Maintenance of CENTRAN programs (especially patching) should have been given greater priority in the design of CENTRAN.

Several of the CENTRAN design approaches were advantageous. CENTRAN was implemented by a small group of programmers. This approach avoided communication and other problems typically encountered with a large group of programmers.

The register allocation mechanism, subroutine interface primitives, and extensibility mechanism designs worked well, as exhibited by CENTRAN's short development time. The ability to have partial word variables has been found useful. The ability to program at several levels in one language made the language suitable for systems and applications programming. Finally, and most important, the design of the extended language was sufficient for the implementation of SAFEGUARD software, and SAFEGUARD programs have been successfully implemented in CENTRAN. Several studies of the suitability of CENTRAN for SAFEGUARD have been made outside of Bell Laboratories, and all have concluded affirmatively.

System Programming in PL/1

The Execution Preparation Facility (XPF) performs the linkage editor function on the IBM 370 for CLC software.[50] It was decided to develop XPF in PL/1. Throughout the development, many design and implementation decisions concerning the use of PL/1 were made. Some of these proved to be sound and others had unfortunate results.

The extensive use of the PL/1 preprocessor proved to be an excellent control mechanism. The inclusion of macros, entry point declarations, and global variable declarations via preprocessor procedures greatly facilitated intermodule communication. This standardization guaranteed the integrity of interface.

As originally expected, the liberal use of PL/1 debugging aids was an invaluable development tool. The large number of logic errors detected through on-conditions such as SUBSCRIPTRANGE and STRINGRANGE underlines the value of their use.

The use of assembly language subroutines, although dictated by efficiency and necessity, presents some disadvantages. Since parameter definition is compiler-dependent, assembly language subroutines must be coded to meet the parameter-passing standards and conventions of a specific compiler. In PL/1, these proved even more limiting since assembly language subroutines must be coded for a specific version of the compiler. When such subroutines are utilized, this dependency on a particular version of a compiler should be explicitly documented.

One benefit of using PL/1 not considered in the original decision was the ease with which transfer of responsibility was accomplished. Partial turnover of personnel occurred throughout the project. Transfer of code responsibility to new personnel was accomplished very smoothly with no apparent decrease in productivity. Since PL/1 can be largely self-documenting through the use of meaningful variable names and standard operation symbols, it is easy to read and understand. This ease of understanding was the primary reason for the smooth personnel transitions.

Professional Design Review

For part of the application program development, a technique known as "flowchart review" was undertaken. As soon as the flowchart and data set layouts for an area of the program were completed — and before coding began — a review meeting was held. The programmer was required to give a box-by-box explanation of a detailed flowchart for his program to a small group of his colleagues. In as friendly a manner as possible, this review committee was expected to be critical and to question aggressively every assumption that the programmer made.

After such a review, the programmer coded his unit, tested it, and released it to process integration. Any minor changes due to bugs found during integration had to undergo a second type of review. Each programmer submitting a minor change was required to select a referee from among the senior members of the group. The programmer would discuss with the referee both his proposed change and the procedure to be used in testing the change. The change could not be released until the referee was satisfied with the testing as well as with the code itself. After the referee system was introduced, the problem of bugs in minor changes came to an end.

One lesson learned from these experiments is the extent of the increase in quality and productivity that can be obtained from a disciplined use of professional review. The use of flowchart review:

- Improved and simplified software design.
- Helped detect all major software design errors before code was written.
- Reduced software development time by at least 25 percent.
- Improved quality of delivered software. The referee procedure brought an end to the errors in minor changes turned over to the system integration team. Other forms of professional review have led to similar results.

A second significant lesson can be learned by comparing professional review with some of the other techniques that also have led to improvements in program quality and programmer productivity; e.g., programming teams, modular and top-down design, and structured programming. There is a common denominator to these techniques — the increased structure and discipline that is placed on the process of writing software. Although much remains to be learned about writing software, it is clear that designing and writing software needs to be a much more structured process than it generally is today.

Software Change Control

Two aspects of software change control that were successful in SAFEGUARD were a project-wide library maintenance system to control source and object code and a standard trouble report form. These were not developed over a long period of time, but appeared very early in the project. Because of this stability, software developers grew accustomed to using them. The library maintenance system was available during the first phase (no change control), and the trouble report form was available at the beginning of informal change control. It was recognized that early introduction and acceptance would be beneficial because transition to the later phases would be simplified. Two additional features of the system, change control procedures and software status accounting, proved to be more troublesome to define and implement. Early in the period of informal change control, each process area independently developed its own procedures; thus a certain amount of reexamination and redefinition were required during the transition to formal change control.

Any software change control system is destined to meet with some resistance. Programmers as a rule have very definite ideas about what should be done to their software. This factor, combined with the dynamic nature of software, makes change control a difficult problem, not so much in establishing the mechanisms and procedures, as in dealing with human factors and ensuring adherence to procedures. The first step is to recognize that change control is a necessity that should be addressed early on any large project and, in fact, will be addressed, either early in a systematic manner or later in a less organized but more costly manner. Only the recognition of this need and conscientious addressing of acceptable procedures will guarantee successful change control. These human factor aspects are the more important considerations in successful software change control.

System Constants

The SAFEGUARD application software contains thousands of numerical values or constants to specify site parameters (e.g., radar location), hardware characteristics (e.g., transmitted power), physical modeling (e.g., air density profiles), etc. Accurate constants were needed by designers in many different areas of each application process. Early in process integration, a group was assigned the task to identify, validate, and control the values of all such system constants. Once this was done, no significant problems caused by improper numerical values occurred during the final integration and testing of the SAFEGUARD software.

While this effort was successful, it is clear that the specification and control of constants should have been considered during the initial design of the software processes. Had it been addressed earlier, undoubtedly it would have resulted in some special design features. These thousands of constants could have been controlled in a less burdensome manner either at process construction time, during process initialization, or during execution, by using special constant data sets. The special features could also encompass easier procedures for verifying and changing values.