N. H. Brown,
M. P. Fabisch, and
C. J. Rifenberg

Introduction and Overview

SAFEGUARD Data-Processing System:

Introduction and Overview

By N. H. BROWN, M. P. FABISCH, and C. J. RIFENBERG

(Manuscript received January 3, 1975)

This paper provides the background information necessary for understanding the other papers in this volume, and serves as an introduction to them. It provides a brief history of SAFEGUARD, discusses the hardware and the software involved, and then focuses on the technical and managerial approaches to producing the software.

I. INTRODUCTION

SAFEGUARD is an antiballistic missile (ABM) system primarily designed to respond to attacks by intercontinental ballistic missiles. It is composed of three major subsystems: missiles, radars, and data processing and control. Incoming missiles, after being detected and tracked by the radars, are intercepted and destroyed by defensive missiles. The radars and defensive missiles are controlled by the data-processing system.

Development of the large, real-time data-processing system for the SAFEGUARD Ballistic Missile Defense System was a significant undertaking from any point of view. Developing a system with unique processing and availability requirements led to the involvement of thousands of people and a very substantial commitment of resources. The resulting multiprocessor data-processing system entailed the development of new and sophisticated algorithms, the design of unique testing programs, and the extensive employment of simulations.

These SAFEGUARD papers primarily emphasize the techniques and methods of a software development effort that produced millions of lines of code. Although the classified nature of the project precludes description of a few of the innovations in both software and hardware, most of the important problems encountered involved no security questions and the objective of these papers is to serve the data-processing community by imparting some of the lessons that were learned.

II. OVERVIEW

2.1 Historical context

At Bell Laboratories, research and development on the first antiballistic missile (ABM) system, the NIKE-ZEUS, began in 1957. The data-processing hardware requirements for NIKE-ZEUS were met by the development of special-purpose digital computers, an outgrowth of the use of analog computers in previous air defense systems. NIKE-ZEUS field test sites were established in New Mexico, California, and the Pacific. Applications programs and techniques were developed for using digital computers as controllers for tracking and missile guidance, for trajectory estimation and discrimination, and as planning and resource allocators in battle management. These application programs were installed and tested at the field sites during the late 1950s and early 1960s. In 1962, an historic intercept was achieved when a NIKE-ZEUS missile launched from Kwajalein Atoll in the Pacific successfully intercepted a TITAN ICBM launched from Vandenberg Air Force Base.

With the termination of the NIKE-ZEUS project in 1963, NIKE-X system development began. This system required a highly reliable data-processing system (DPS) that could support a peak throughput of about 10 million instructions per second and a peak I/O transmission for radar control of about 70 thousand 64-bit words per second. To achieve these requirements, a special-purpose digital computer was designed using integrated circuits and core storage techniques. A field test site for the NIKE-X development was established at Meck Island, part of the Kwajalein Atoll. Testing at this site has had significant impact on the development program.

In 1967, the basic design of the NIKE-X machine was incorporated into the SENTINEL ABM system. Throughput requirements were met by a multiprocessor capable of using as many as ten processors.

Originally, the goal of SENTINEL was the protection of cities from a ballistic missile attack. In 1969, new objectives, including the protection of U. S. MINUTEMAN ICBM bases rather than cities, were announced. This redirection was indicated by a new system name, SAFEGUARD. SENTINEL equipment remained unchanged. The field test site for NIKE-X now became the Meck Prototype System. Its objectives were redefined from those of an R&D program to those of supporting SAFEGUARD design. A detailed test program was established for the Meck system, providing indispensable support for SAFEGUARD in hardware, software, and algorithm development, as well as multiprocessor operation and reentry environment characterization.

The entire software development of SAFEGUARD has been directed at the specific needs of a real-time, high-throughput, very reliable computing system. The applications programs, operating system, support software, and data-reduction facilities were all designed to meet these objectives.

2.2 System description

There are three types of sites in the SAFEGUARD system: Perimeter Acquisition Radar (PAR), Missile Direction Center (MDC), and one Ballistic Missile Defense Center (BMDC). Figure 1 provides a functional overview of these sites. Although several PAR and MDC sites were planned, only one of each is being deployed. The PAR site utilizes a single-face, phased-array radar to provide early detection and trajectory data on threatening ICBMs. Functions of this site include long-range surveillance, detection, and target selection of threatening objects, and 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. The MDC complex uses the target trajectory and classification data from the PAR along

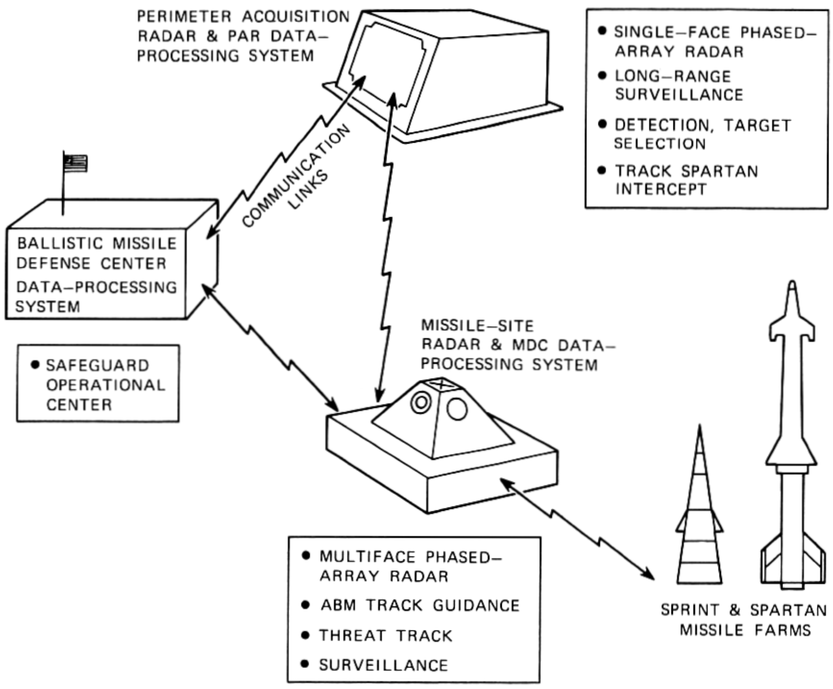
PERIMETER ACQUISITION RADAR & PAR DATA-PROCESSING SYSTEM
• SINGLE-FACE PHASED-ARRAY RADAR
• LONG-RANGE SURVEILLANCE
• DETECTION, TARGET SELECTION
• TRACK SPARTAN INTERCEPT
COMMUNICATION LINKS
BALLISTIC MISSILE DEFENSE CENTER DATA-PROCESSING SYSTEM
• SAFEGUARD OPERATIONAL CENTER
MISSILE-SITE RADAR & MDC DATA-PROCESSING SYSTEM
• MULTIFACE PHASED-ARRAY RADAR
• ABM TRACK GUIDANCE
• THREAT TRACK
• SURVEILLANCE
SPRINT & SPARTAN MISSILE FARMS

Fig. 1—SAFEGUARD system.

with additional data supplied by its multiface phased-array radar. This site provides additional surveillance and target tracking and also performs the functions of track and guidance for the SPRINT and SPARTAN missiles. Both PAR and MDC sites report to the BMDC, a central command center. The BMDC provides a command interface with other military systems and a means of disseminating command directives and controls.

The PAR and MDC radars are controlled by the data-processing systems, collocated with the radars. At the PAR and MDC sites, application programs perform surveillance, tracking, target classification, radar management and testing, intersite communication, and display functions. Additional application programs at the MDC support the battle management and missile guidance functions. The BMDC data-processing system primarily contains display and command/control programs.

Both PAR and MDC radars are controlled by the DPS through the use of digital commands. These commands are used to control beam pointing, frequency selection, receiver gating, thresholding, etc. The SAFEGUARD system design makes use of some constraints on the combinations of radar operations that can be performed and, therefore, on the sequences of pulse transmissions. Appropriate radar commands must be generated by the application programs and sent to the radar at least every few milliseconds. The radar pulse patterns used in SAFEGUARD provide a framework for the time design of the real-time application programs.

2.3 DPS requirements

The data-processing system design was dominated by requirements for high throughput and stringent availability/reliability constraints; i.e., requirements supporting a high probability that the system would be available when required for a mission and highly reliable during the mission.

The fact that the radar is controlled by the DPS contributed significantly to both input/output (I/O) and processing requirements. Appropriate radar commands must be generated by the application programs and output to the radar at least every few milliseconds, yet the DPS must be able to complete processing between two radar events. This contributes to estimates of a peak-load throughput of 10 million instructions per second.

Input/output requirements were further increased by a variety of special-purpose peripherals such as missile controllers and data-transmission controllers for intersite data transmission. The DPS was also required to communicate simultaneously with computing peripherals, especially disc and tape, as well as to provide status information to, and receive commands from, system-control personnel.

The nature of the application imposed requirements for high availability; therefore, a maintenance system was required for fast recovery and quick fault isolation and repair in the event of a hardware malfunction.

Size and complexity increased the problem of verifying the system. This imposed a requirement for a system exerciser that could be used to verify as much of the system as practical.

2.4 Tactical site configuration

This section describes in detail four aspects of a site DPS configuration: hardware, software structure, maintenance and diagnostic subsystem, and exercise subsystem. Except for the absence of an exercise subsystem at BMDC, the DPS structure is similar for MDC, PAR, and BMDC.

2.4.1 DPS hardware

Figure 2 shows the equipment at the MDC site consisting of a central computer and associated peripherals. The central logic and control (CLC) is the multiprocessor computer used to drive each DPS. Under software control, the CLC can be configured into two separate partitions of arbitrary size, each capable of operating as an independent com-

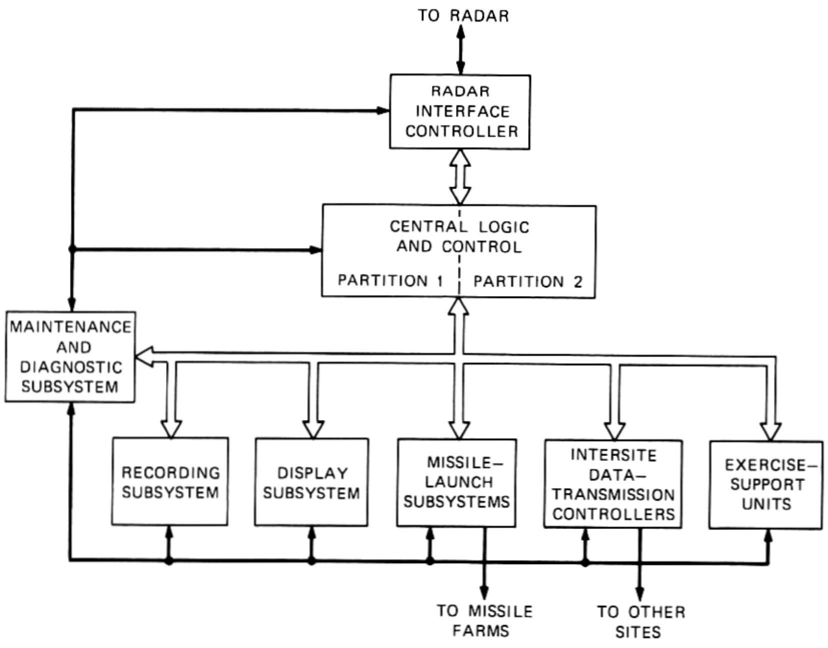
TO RADAR
RADAR INTERFACE CONTROLLER
CENTRAL LOGIC AND CONTROL
PARTITION 1
PARTITION 2
MAINTENANCE AND DIAGNOSTIC SUBSYSTEM
RECORDING SUBSYSTEM
DISPLAY SUBSYSTEM
MISSILE-LAUNCH SUBSYSTEMS
INTERSITE DATA-TRANSMISSION CONTROLLERS
EXERCISE-SUPPORT UNITS
TO MISSILE FARMS
TO OTHER SITES

Fig. 2—SAFEGUARD data-processing system equipment.

puting system. Application software executes on the larger partition. Exercise drivers (described below) for the application software and support activity execute on the smaller partition, which also provides a pool of spare equipment.

The CLC can be configured with up to ten processors. Single-processor throughput of about 1.5 million instructions per second is achieved by a combination of design techniques including instruction execution overlap and use of high-speed arithmetic algorithms. Instruction overlap is achieved by utilization of three asynchronous control units for instruction fetch, operand fetch, and arithmetic execution. 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 ns and a double word size of 64 bits to provide a memory bandwidth in excess of that required for maximum performance of a single processor.

The input/output controller (IOC) controls the transfer of data between the CLC and its peripherals. Since processors do not communicate directly with peripherals, processing and I/O can occur simultaneously. The IOC provides full-duplex operation on 16 channels. Priority circuitry within the controller allows time-multiplexed operation of the channels. The IOC executes commands from IOC programs resident in variable store. Both processors and peripheral devices can initiate IOC program execution.

A timing generator provides a real-time clock and a programmable mechanism for initiating activities at specified times. It can cause the initiation of an IOC program when a specified time of day has been reached. A status unit provides a means of monitoring, in real time, the status of any DPS unit. It also serves as a central point for the distribution of control over the DPS.

CLC peripherals are divided into several subsystems. The Maintenance and Diagnostic Subsystem and the Exercise Subsystem will be described later.

The radar interface controller is the primary interface between the radar and the I/O controller of the CLC. Control and data words are exchanged between these two units. The radar control computer accepts formatted binary words from the CLC and distributes data to the appropriate radar subsystem where a digital-to-analog conversion takes place.

The recording subsystem contains the standard computer peripheral devices: magnetic tape transports, disc memory units, line printers, and card reader.

A man-machine interface is provided through the display subsystem which includes cathode-ray-tube displays with light pens, wall displays, and teletypewriters.

Digital data are transferred between sites by means of the intersite data transmission controllers.

The missile launch subsystems convert CLC commands into control signals for the collocated and remote missile farms and receive missile status conditions, encode them, and send them to the CLC.

2.4.2 DPS software structure

The collection of application software used to drive the DPS is called the application process. The application process 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. Once in execution, a task is not interrupted before completion except for error conditions.

A bit string associated with each task on the priority-ordered task list indicates completion of predecessor condition(s) prior to task execution. The operating system enables execution of the highest-priority task with all predecessor condition bits set. Thus, an important part of process design is development of the priority-ordered task list and the predecessor conditions for each task. The predecessor conditions fall into three main types:

- (*i*) Time—Functionally, the programmable feature of the timing generator is utilized in setting predecessor condition bits.
- (*ii*) I/O completion–Input/output may be initiated by a processor or by a peripheral device. In either case, a task does not "hold" a processor while waiting for I/O completion. Instead, upon I/O completion, a predecessor condition bit is set for a designated task.
- (*iii*) Other task completion—Long-running computations are often subdivided into several shorter ones. Appropriate sequential computational requirements are preserved by designating other task predecessor conditions.

Where possible, the application process is asynchronous, i.e., tasks are only enabled when data are available to be processed.

2.4.3 Maintenance and diagnostic subsystem (M&DSS)

The M&DSS is composed of test equipment and software that supports digital equipment maintenance. The M&DSS verifies the availability and readiness of DPS hardware by conducting nonreal-time, programmed, diagnostic tests on equipment through an independent data bus connected to each digital unit. These special M&D data paths are also used to support other objectives of the M&DSS which include initializing DPS hardware and, in the event of a malfunction, automatically supporting DPS recovery operations. The M&DSS also provides a centralized control point for status monitoring, equipment allocation, and manual interface with DPS software.

The M&DSS has two distinct facilities for running diagnostics. The primary one involves the M&D processor group, which uses a modified CDC Model 1700 computer system to provide fully automatic, high-speed execution of test programs with automatic interpretation of results through use of fault-location dictionaries. The other facility involves the M&D console group, which uses a cathode-ray-tube display console for manual execution of diagnostics and interpretation of results. Each facility is linked to all the digital racks in the DPS and to certain digital racks in the radar areas. These data paths provide the means by which M&DSS software can access each unit as required for DPS initialization, recovery, and diagnostic operations.

2.4.4 System exerciser

A system exerciser was designed for PAR and MDC sites. It provides support for development and integration of the applications processes, evaluation studies that include fidelity validation of various simulators, and site readiness verification of both local and multisite system configurations.

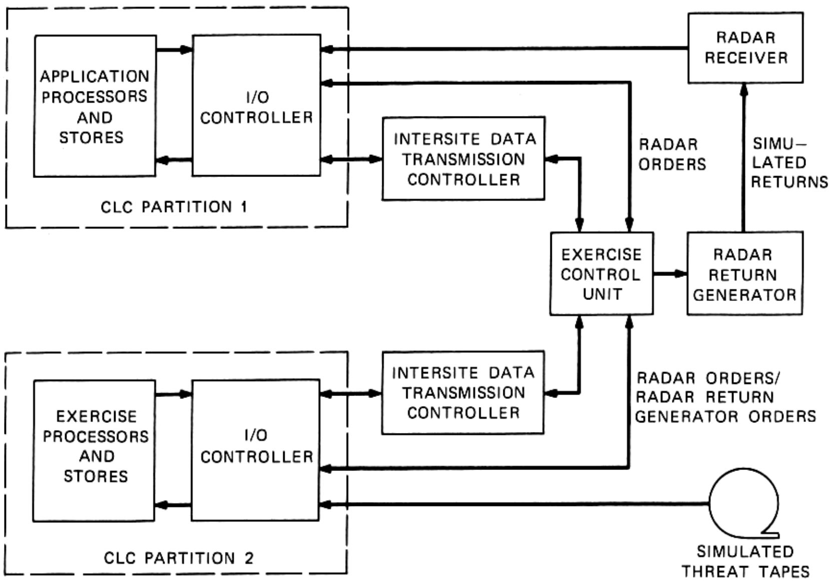
APPLICATION PROCESSORS AND STORES
I/O CONTROLLER
CLC PARTITION 1
INTERSITE DATA TRANSMISSION CONTROLLER
RADAR RECEIVER
RADAR ORDERS
SIMU-LATED RETURNS
EXERCISE CONTROL UNIT
RADAR RETURN GENERATOR
EXERCISE PROCESSORS AND STORES
I/O CONTROLLER
INTERSITE DATA TRANSMISSION CONTROLLER
RADAR ORDERS/ RADAR RETURN GENERATOR ORDERS
CLC PARTITION 2
SIMULATED THREAT TAPES

Fig. 3—Functional representation of the hardware configuration for the PAR system exerciser.

Software was developed to run on the exercise partition of the CLC to generate simulated radar returns and simulated intersite communication. Special hardware was developed to inject the simulated threat data at the receiver of the radar. This allows testing a significant portion of the radar and drives the data processor with realistic data at its actual interface with the radar. Figure 3 provides a functional representation of a PAR exercise configuration.

The principal communication between the two partitions is through the exercise control unit (ECU). The ECU intercepts application program orders to the radar, and intersite messages, and directs them to the exercise partition. The ECU routes simulated radar returns generated by exercise software to the radar-return generator for conversion to analog waveforms and injection into the receiver of the radar.

The exercise software is a real-time process similar in construction to the application process. An off-line facility is used to simulate a threat and generate tapes with a time sequence of the manner in which the threat appears in the radar viewing volume. These tapes are used by the exercise process in generating replies to application-process radar transmissions.

2.5 Software development

2.5.1 Tactical Software Control Site

To develop the large number of programs required for the deployed system and its support, a Tactical Software Control Site (TSCS) was established at Madison, New Jersey. The software development organization, consisting of designers, programmers, test teams, and many others, was located at a few distinct facilities in northern New Jersey, all within a few miles of each other, and a single North Carolina location.

A test bed was required to reproduce accurately the software environment existing at site such that performance of software in its operational environment could be verified; software testing could be accomplished in close proximity to the design organization; and testing could precede site availability to reduce development time. To reproduce the site software environment, the test bed was required to have a representative complement of computing hardware for the PAR and MDC; replicate the interfaces between the computer and peripherals; replicate the peripheral devices to the extent that device performance and characteristics were not completely isolated from the computer; and provide the capability for actually netting the PAR and MDC processes for purpose of system testing. Thus, a test bed was established at TSCS and contained separate PAR and MDC configurations corresponding to the PAR and MDC sites. The configurations provided peripheral hardware needed by software, but did not include all of the analog portions of the radar or missile interfaces. Communication paths between PAR and MDC test-bed configurations were included via the data-transmission controllers. This permitted TSCS netted testing in advance of system testing at the sites.

Experience from previous development projects indicated that all available test-bed time would be required for system testing, operating-system development, and hardware installation and maintenance. Support functions (e.g., software preparation and analysis) were therefore designed for operation on general-purpose computers such as the IBM System/370 and HIS 635. These machines were then also required at TSCS.

2.5.2 *Software development cycle*

The software development cycle for SAFEGUARD was not substantially different from that of other large systems. In practice, individual phases of the development cycle overlapped since the general approach followed was integration of a basic working system with increasingly more complex capabilities. The separate phases of the development cycle consisted first of the *requirements-generation* phase, in which system requirements were determined, established, negotiated, documented, and rigorously controlled. The *design* phase consisted of process design and program design. In process design, the system requirements were translated into a software architecture which defined global data 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 the individual tasks were determined. In the *coding and unit-testing* phases, code was written, compiled, and checked at the unit or task level, using a simulator, drivers, and standard debugging techniques. Next, at the test bed, separate *process-integration* teams combined blocks of new, debugged unit code into processes for increasing functional capabilities. When the tactical software achieved a predefined level of capability, it was sent to site for *site 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. Moreover, it was at site that formal acceptance tests were run. The final phase of system development was *system integration*, in which the PAR, MDC, and BMDC sites were "netted" and the coordinated operation of the entire system was achieved. During all phases of system development, *evaluation* played a strong role. A separate organization was responsible for evaluating system requirements, implementation algorithms, and system-test results. Feedback resulted in frequent changes and refinements in many areas.

Following is an expanded overview of some important features of the SAFEGUARD software-development cycle.

2.5.3 Requirements

The Data Processing System Performance Requirements (DPSPRs) are a set of documents that define the requirements of SAFEGUARD tactical programs for the PAR, MDC, BMDC, and system exerciser processes. Requirements were generated by the system engineering organization in accordance with overall system objectives, which were defined by the Department of Defense. Changes to the requirements were made as a result of detailed software design by the development organization, Meck prototype system-test-program data, system-evaluation efforts, and detailed review with the U.S. Army SAFEGUARD System Command (USASAFSCOM).

The DPSPRs met their original objectives of providing a clear definition of the computing requirements. They have continued to be the up-to-date system definition of SAFEGUARD performance, and have been used to specify all system-testing and acceptance requirements.

2.5.4 Design

Process design was 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 the 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. This activity led to a detailed software specification, including specific mathematical equations or decision tables.

Decisions were made in both process and program design to support early development of a system to which greater capability would be gradually added. Emphasis was placed on modularity in design to ease system growth.

It was found to be essential to initiate the design of the data recording and reduction system early in the development cycle. An attempt was made to define data to be recorded for each computing function, and to design the data base to include consideration of recording and the subsequent analysis to be carried out upon the recorded data.

In many areas simulations 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.

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

Design reviews were held frequently and proved to be an effective means for communicating problems and solutions relating to planning or design issues to other members of the project. These were attended by a review board consisting of both designers and project managers.

2.5.5 Coding and unit testing

All of the software preparation and most unit testing was performed using commercial computers. This was primarily because test-bed time was too valuable to be consumed for compiling and unit testing.

Most SAFEGUARD software was written in CENTRAN, an extensible intermediate-level language resembling a subset of PL/1. CENTRAN generated efficient code. It provided many of the advantages of high-level languages, but could be interspersed with assembly language and system macros when necessary. It was adopted as the project standard.

To facilitate program preparation and unit testing, a linkage editor, a CLC simulator, and a disc library system were also developed for execution on the IBM System/370. The linkage editor bound units of CENTRAN object code for execution on the CLC or CLC simulator. The library system functioned as an editor and disc-file manager, which helped control CENTRAN source and object code. The linkage editor and simulator were developed on the SAFEGUARD project, while the library system was a SAFEGUARD modification of an existing IBM proprietary program.

2.5.6 Process integration

Following unit debugging, collections of units were tested for increasingly greater functional capabilities on the PAR or MDC test beds by independent integration teams. Frequently, large drivers were developed to assist in early functional testing. Subsequently, the system exerciser was used to stress and drive the application process to various conditions and loads.

Detailed analyses of integration tests were possible because the application and exercise processes contain real-time recording functions which were designed as an integral part of the software. Recorded data were reduced and analyzed primarily off line on the IBM System/370 using the SAFEGUARD Date Reduction System, although summary information was available on line.

A hardware/software CLC performance monitor was developed and installed at the TSCS. It was used primarily to validate that the process performance was consistent with its design. Troubles, such as heavily loaded time frames and long-running tasks, were analyzed. When possible, design changes were made to provide a more balanced system.

2.5.7 Site and system integration

When the application and exercise processes achieved predefined capabilities, they were sent to site for further integration. Capabilities already established at TSCS were reverified in the expanded hardware environment. Further testing concurrent with and complementary to test-bed integration was conducted, with primary emphasis on full process testing using the system exerciser. A comprehensive series of acceptance tests was run to demonstrate that system capability was consistent with requirements. Tests ranged from satellite tracking and identification to system exercises which drove the system to design traffic levels.

During system integration, which is the final level of product testing prior to delivery to the customer, it was not possible to exhaustively test all tactical threat environments. An "Endpoint Test" was 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 system.

The stress level was reduced in early testing by selecting subsets of the Endpoint Test environments and by running buildup tests at these lower stress levels before operating the netted system at design traffic levels. The use of a common environment for a number of tests, with traffic buildup by addition to this environment, and buildup of physically internetted sites in stages, led to the "test-chain" approach to testing. This approach, in which all tests in the chain support the Endpoint Test, greatly simplified the problems of integrating a dynamic system.

Commercial computers were installed at site during the site-and-system integration period for data-reduction support. This support was required on location to provide prompt analysis of data recorded during testing. Tight schedules and lack of available CLC time required that this facility be provided by a support computer.

2.5.8 Evaluation

System evaluation was primarily an analytical activity which, because of the complexity of the SAFEGUARD system, relied heavily on simulation. A SAFEGUARD system simulation was designed to provide insight into overall system operation with particular emphasis on battle-planning functions. Initially, the simulated system was made to operate in accordance with performance requirements. Since, quite properly, performance requirements often permit the designer considerable latitude, modeling of the system in this initial phase often entailed considerable invention. The goal was to ensure that objectives would be achieved if the system operated in accordance with performance requirements and that inadequacies in system design would be identified and corrected before resources were wasted attempting to implement a faulty design. Since there was a practical limit to the level of detail in which the various weapon system functions could be modeled, more detailed simulations of the particularly critical functions of surveillance, tracking, target selection, and guidance were added. By employing these simulations in concert, considerable insight was gained into detailed system operation.

As the design of the tactical hardware and software stabilized, these simulations were continually updated to provide a more accurate representation of tactical operation, and a continuous evaluation of the evolving system. Early development of detailed but evolving simulations permitted in-depth analysis of most critical areas of SAFEGUARD operation. A number of significant design modifications can be attributed directly to evaluation activity. A noteworthy example is the restructuring of both the PAR and MDC overload-response software to provide improved performance in a high-traffic environment.

Systematic and detailed analysis of the Meck prototype-system tests, which were designed to stress critical functional capability, provided confidence in the validity of analyses based on simulation. Finally, simulation, in addition to providing a tool for evaluation of overall system performance, permitted the definition of explicit thresholds for use in acceptance tests of the entire netted system.

2.6 Project organization and control

2.6.1 Organization

Organizations were established for each of the major software efforts, PAR, MDC, BMDC, and System Exerciser. A separate systems-engineering organization was responsible for requirements and evaluation. Support-software development organizations were also established for each major support activity such as DPS maintenance software, real-time support software, nonreal-time support software, and computer operations. Each major activity was directed by a project manager.

The software development organization consisted of engineers and programmers primarily from Bell Laboratories, IBM, and Western Electric. While project responsibility rested with Bell Laboratories, IBM was responsible for much of the software development. These development activities were directed by IBM managers who were in turn responsible to Bell Laboratories project managers for completion of the tasks. For the most part, Western Electric engineers and programmers were integrated directly into Bell Laboratories organizations, with the notable exception of test-bed-facilities management, which was turned over to Western Electric early in the development cycle.

2.6.2 Control

Overall scheduling for the project was the responsibility of the system-engineering organization. Project managers were held responsible for coordinating and setting schedules for software under their control, consistent with overall schedules.

Schedules were documented at several levels of detail in a management-information system. Visibility was provided by frequent design/schedule reviews, and by a Principal Event Report. The principal events were selected major milepost achievements in performance, and were scheduled within the total network of activities related to software and system development. A written report as to the performance achieved relative to the defined requirements for a principal event was required within 72 hours of the schedule date. All open items were reported with a schedule for their completion. Upon completion of an open event, written confirmation to management was required.

Further development control and discipline were achieved by the use of additional techniques. A Policies, Procedures, and Standards (PPS) Manual was established and maintained. The manual provided detailed policies and standards to ensure uniformity and control within the project. PPSs were written on change management, documentation, management reporting, programming standards, etc. Software change management standards were established early, and they were extended, modified, and adapted for use on each major activity. Typically, this included documenting troubles on standard Trouble Report forms and keeping track of them and their solutions in a Status Accounting System. Stable software was "frozen," stored, and officially released by a central organization.

Because of the difficulty of employing subcontractors on a large complex software development, very careful attention was given to defining interfaces and a detailed task description, monitoring, and evaluation system was devised. This system was fundamental to the success of the development effort.

Comprehensive documentation standards were also established early. Support software documentation emphasized requirements and user information; tactical software documentation emphasized requirements, design information, test plans, and well-commented listings. In general, documentation and software development were synchronized.

The emphasis on planning was fundamental to the overall management approach. Although no single planning format or technique was prescribed, each project manager was required to plan in detail for the complete design, implementation, and testing of his part of the system.

2.6.3 Resource requirements

Resource estimation and control were generally the responsibility of project managers. Normal budgetary procedures were used, requiring justification to and approval by upper management and the customer on a yearly basis. Manpower needs were estimated by project managers using experience and algorithms from other large projects together with a detailed plan of the work to be performed. Manpower restrictions were resolved by replanning and modifying schedules.

Support-computer needs were estimated by project managers and analyzed by the support-computer project manager, who coordinated the acquisition of support equipment. Application-computer require-

Table I — SAFEGUARD software development—quantities of instructions and statements

Real-Time Software Instructions	
CLC operating system	100,000
MDC applications	300,000
MDC exerciser	50,000
PAR applications	200,000
PAR exerciser	25,000
BMDC applications	60,000
Total	735,000
Support Software Source Statements	
CLC software preparation support	210,000
System simulation	50,000
Exercise support	30,000
Data reduction	150,000
Configuration management	70,000
Logic simulation	70,000
Total	580,000
Installation and Maintenance Software Instructions	
MDC radar installation	50,000
PAR radar installation	110,000
PAR radar test	60,000
Maintenance & diagnostic	300,000
Diagnostic operating facility	120,000
DPS installation & test	190,000
Total	830,000

ments were established and monitored through periodic sizing estimates by the PAR, MDC, and BMDC project managers.

The size and duration of the SAFEGUARD development effort was large indeed. Table I shows the size of the major components of software: real-time software, consisting of MDC and PAR applications and exercise programs, BMDC applications programs, and the CLC operating system, totalled 735,000 instructions; support software, such as compilers and simulators executed on commercial computers, totalled 580,000 statements, some assembly language, and some 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. The total development interval, starting with the generation of SENTINEL requirements and concluding with SAFEGUARD system integration, spanned 90 months.

III. CONCLUSION

Perhaps the most important lesson to be learned from SAFEGUARD is that a large, well-conceived development project, however ambitious, can be completed successfully. During the development, the number of sites was changed, drastically reducing the size of the deployment. This, coupled with test results, as well as changes in objectives, led to modifications in the overall system design. However, it can reasonably be said that the complete development, including the integration of the first installed sites, was performed on schedule and that the system met the prescribed performance specifications. Although cost performance is a little bit harder to define because of the effects of inflation over the period and because of changes in the deployment, it seems clear that costs were controlled reasonably.

To reiterate an observation made earlier, implementation of the SAFEGUARD data-processing system was a significant undertaking, one of the most complex ever attempted. Its production entailed the development of a highly reliable multiprocessor computer system, and the generation of millions of lines of code. The papers that follow describe some of the design of the system as well as the lessons that were learned and the techniques employed.