Section IV

SUPPORT SYSTEMS

THE BELL SYSTEM TECHNICAL JOURNAL
SAFEGUARD SUPPLEMENT
Printed in U.S.A.

SAFEGUARD Data-Processing System:

Support Software and Support Computers: An Overview

By R. R. CONNERS

(Manuscript received January 3, 1975)

This paper gives a critical overview of the development of a selected set of support software programs. These programs comprise part of the support environment required to make the SAFEGUARD system application software and hardware operational.

I. INTRODUCTION

In the development of the SAFEGUARD system or any other large-scale programmed system, one may distinguish between two types of software—application and support. Application software does the job for which the system is intended. In SAFEGUARD, this means driving the radars, tracking objects, firing missiles, etc. Support software includes all other software required to make both application software and hardware operational.

This paper, and others that appear in this section, cover a limited set of support software programs. Included in this set are

(*i*) CENTRAN—A compiler for a high-level extendible language of the same name.
(*ii*) SNX—A macro assembler.
(*iii*) XPF—A binder for preparing CENTRAN and SNX output for execution.
(*iv*) STACS—A simulator for execution of XPF output.
(*v*) SDRS—A set of programs for reducing, i.e., decoding and formatting, data recorded during CLC execution.

All these programs have several things in common. First, their purpose is to assist the development and debugging of SAFEGUARD application software. Second, for reasons to be discussed in detail later in the paper, these programs operate on computers other than the Central Logic and Control (CLC).*

* The CLC is discussed in Ref. 1. Discussions of support software that operate on the CLC may be found in Ref. 2.

This paper critically examines some key decisions that shaped the programs and the environment in which they operate. In doing this, the groundwork will be laid for the other papers in this section. The paper is divided into two sections: the first covers support software; the second covers the computers on which the support software programs were developed and on which they run.

What is the value of such a critical examination? The frequency with which programming projects fail or repeat the mistakes of their predecessors leads to the conclusion that the knowledge required to manage program development is largely based on experience. Perhaps the communicating of experiences, successful and unsuccessful, may help to transmit some of the knowledge gained on the SAFEGUARD project.

Why is SAFEGUARD a good choice for such an examination? Because its development and that of its prototype span a ten-year period that overlaps the development of the "science" of programming management. Because during those ten years many intensely creative people were involved, and since the magnitude of the project was enormous, their creativity was not constrained. Because the SAFEGUARD effort encompassed a multiplicity of organizational structures within Bell Laboratories and its various subcontractors. Because it seems that, at one time or another, practically everything was tried or seriously studied.

During the ten-year period examined, one can distinguish three successive phases of support software development. Each phase was built on the lessons learned during previous phases, and each phase had its own characteristic spirit. The first phase was part of the NIKE-X antiballistic missile research and development effort in the mid-1960s.[3] The second phase was the development of support software for the Meck test system, a prototype system intended to validate some of the results from the first phase. The third phase was the development of support software for SAFEGUARD, which has applied the results of NIKE-X research on an even wider scale. The topic of this paper is SAFEGUARD support software, but since it has been shaped by a synthesis of NIKE-X and Meck test system experiences, it is with these that the story must begin.

II. SUPPORT SOFTWARE FOR NIKE-X AND THE MECK TEST SYSTEM

In the early days of NIKE-X, support software designers envisioned an environment in which the application programmer "typed" high-level language programs at a terminal under control of a time-sharing system operating on the CLC. Programs would then be processed by a global optimizing compiler and executed, with results routed to the program-

mer at his terminal. In retrospect, these ideas are perfect examples of the optimism that pervaded the computer industry in the mid-1960s. As the industry lost its innocence, these ideas crumbled. The CLC time-sharing system, Program Development Facility (PDF), was ultimately dropped when it became apparent that there was not enough CLC time to allow concurrent development of hardware, support software, and application software. PDF development had lost not only its means but its end as well, for application software development for the Meck test system had already passed the unit testing stage where PDF could have been most helpful.

A global optimizing compiler, NIKE-X Compiler Language (NICOL), was abandoned after completion of three of the four planned stages of its development. A working, but incomplete, compiler (NICOL 3) had been built for the CLC, but there was insufficient CLC time for support software development and compiler maintenance. This, combined with a long time estimated for completion of the optimizing phase given the available manpower and with technical problems encountered in maintaining NICOL 3, indicated that NICOL would not be able to meet the needs of the project. Meanwhile, coding for the Meck test system had been done in assembly language so the compiler was needed only for SAFEGUARD software development.

A message in the NICOL and PDF stories will recur in this paper. Support software goals must be realistic, particularly in the sense that they be attainable at the time they are required. Essential features must be available on schedule. The purpose of support software is, after all, to support the objective of building systems. Building state-of-the-art support software as well is laudable, but only if it contributes to the main objective.

These experiences led to several important decisions that influenced support software development for SAFEGUARD. First, it was decided to move as much support software activity as possible to a computer other than the CLC. This had the desired effect of making the CLC more available for application software testing, but it also resulted in a sizable support computer requirement which is discussed in Section V of this paper.

A second major decision was that application program development would be done primarily in batch rather than in time-sharing mode. This decision was reached over the course of several years and was based on many contributing factors. First, experiences with MULTICS and TSS[4] were disappointing, in the sense that they did not support the expected number of users at the predicted time. Second, other available, or nearly available, time-sharing systems appeared to have serious limitations: CP/CMS was not file-compatible with IBM System

360. TSO, which IBM advocated in place of TSS, was limited in capability and lacking in human engineering. No system provided adequate availability and reliability. Third, interactive support software would have to be written and it appeared easier, and therefore faster, to write support software that would operate in the batch mode. Finally, the exact trade-offs between batch and time-sharing program development were not yet known (and perhaps still are not). It was speculated that time-sharing might improve programmer productivity, but it would do so at the expense of additional computer requirements. It would have been difficult for management, already faced with large development costs, to commit itself to time-sharing without a better understanding of its cost effectiveness.

This shifting support software philosophy affected the development of application software for the Meck test system. From the ashes of PDF and NICOL emerged a set of support software tools that were sufficient to get the job done. Application programs were coded in assembly language. Linking and binding the various application programs was an intricate and awkward process. Computer program source was stored and distributed on cards. The key to effectiveness was an ironclad set of control procedures that made the system work. Despite a lack of sophisticated support software, the Meck test system has consistently met its objectives.

III. BUILDING SUPPORT SOFTWARE FOR THE SAFEGUARD SYSTEM

The decision to deploy the SAFEGUARD system had a direct impact on support software. It would be necessary to improve the programs and procedures used to develop the Meck test system, since SAFEGUARD was a larger and more complex undertaking. However, there was very little time to implement these improvements and still meet tight development schedules.

One of the first decisions made was to "borrow" software. Under the aegis of the ESS project, Bell Laboratories had developed a modular assembler called SWAP[5] that was specifically designed for portability—as long as there was an IBM System 360 available. Borrowing SWAP, converting it to generate CLC machine code, and relabeling it SNX 360 yielded a fast, efficient assembler at minimal cost. However, it was necessary to provide people to maintain and modify SNX 360 as requirements grew. SNX 360 was adopted by both SAFEGUARD and the Meck test system.

What about a programming language for the deployed system? Assembly language had been used exclusively for the Meck test system and was favored by most of the "old pros." Management and support software designers were concerned because experience indi-

cated that a programmer could write and maintain a fixed number of statements per month, no matter what language was used. The use of high-level languages promised greater productivity. Management also knew that system requirements would be fluid in the early stages of development, requiring frequent changes. This supported the argument for the use of a high-level language.

The only high-level language available for the CLC was NICOL, however. At that time there was a working, but incomplete, NICOL compiler of unknown reliability generating inefficient code. It ran slowly and was very difficult to maintain. Estimates of the time required to complete the final phase of NICOL development, an optimizing compiler, were unacceptably long. This was due in part to the inherent difficulty in generating optimal code for a machine whose instruction set had not been designed with an optimizing compiler in mind. In addition to the flaws mentioned above, NICOL did not allow programmers to "get at the machine," i.e., access the CLC hardware registers. This alone was enough to make NICOL unsuitable for a major part of CLC software, the operating system. What was needed was a language that was easy to use and available immediately, could produce optimal code, allowed programmers to access the CLC registers, was efficient, and was sufficiently rich in syntax and semantics to serve the needs of system programmers, tracking and filtering analysts, and radar and missile control programmers.

What evolved from this need was CENTRAN,* a high-level extendible language. The compiler for CENTRAN was coded as a SNX 360 macro package. While this implementation permitted early compiler availability, it did result in long compilation times. Later improvements increased the speed, but not to the point where it was comparable with most compilers.

CENTRAN offered the programmer his choice of language level, from assembly language to something resembling a subset of PL/1, and statements of varying levels could be intermixed. If the programmer felt that the language syntax was inadequate, he could extend it with relative ease. Extendibility allowed for development of the language in stages, so a minimum facility could be made available to users almost immediately. Reference 7 contains a retrospective look at some of the design issues in CENTRAN.

As a postscript to the programming language issue, it is interesting to note that use of CENTRAN was decreed rather than sold. There are several "extreme" reasons why this had to be. On one side, CENTRAN designers felt that the CLC programmers' attachment to assembly

* CENTRAN is described by its inventor under the name ETC. See Ref. 6.

language was excessive* and that the programmers' cries of inefficiency were misdirected. Efficiency concerns, argued the CENTRAN designers, should be directed first to the design of the program and data base, rather than to the programming language. Many programs need not be coded with scrupulous efficiency. Further, with a knowledge of a few basic facts of CLC architecture, the programmer could write CENTRAN statements that would generate code just as efficiently as an experienced assembly language programmer could. The programmers countered with arguments of their own. They claimed, with some justification, that CENTRAN was unreliable. CENTRAN taxed SNX 360 macro capabilities as they had never been taxed before. If a programmer made a syntax error, his compilation would occasionally abort without a diagnostic or produce a SNX 360 diagnostic that was meaningless to him. Programmers also complained about CENTRAN documentation, again with justification. It is impossible to write adequate documentation, construct courses, and reeducate 600 programmers overnight. These things take time to evolve, and while they do, programmers suffer. The battle over CENTRAN raged for some time and became quite bitter. But in the end, CENTRAN was used and programs were written.

Programs were indeed written, several thousand of them, in fact. A better source code storage and control mechanism was needed to replace the card-oriented Meck test system approach. A disc-based filing system was under development, but not near completion. Bell Laboratories accepted the offer of the use of an IBM proprietary system that had been used in the development of IBM programs.

This product is a comprehensive disc-based source-object listing filing system which offers programmers many of the features required in the software development process; for example, an editor for changing source lines, a means of temporarily changing source for testing, and a mechanism to facilitate delivery of completed code. In addition, the system helps to protect the programmer from his own or others' mistakes by allowing limited access to libraries. Initially, users were not enthusiastic about the system, and management pressure had to be applied to ensure its use. Complaints centered around reliability and documentation deficiencies. In retrospect, however, the decision to use the system proved to be a good one, primarily because of the procedural discipline it forced on the programmers.

* Part of this attachment is really an unwillingness to give up comfortable, familiar coding patterns. B. N. Dickman relates an anecdote that illustrates this vividly. When he joined the project in 1967, he found that CLC programmers were hard-coding base registers in their instructions. He implemented a USING pseudo-operation similar to the one in IBM System 360 BAL. But he found it difficult to get programmers to use this most helpful and completely noncontroversial feature.

CENTRAN and SNX 360, like most compilers and assemblers, produce 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 was named the Execution Preparation Facility (XPF).

XPF was possibly the most volatile of the support software programs. As new capabilities were incorporated into the CLC operating systems, XPF had to be changed to reflect these new capabilities in the bound versions of users' programs. This could have been a massive coordination problem, but the XPF designers found a creative solution. First, they implemented XPF in PL/1, which facilitates changes, and made clever use of its preprocessor to automate change as much as possible. Second, they planned a series of releases with incrementally expanding capabilities and coordinated them with the development schedules for the CLC operating systems.

XPF is discussed in Ref. 9. This paper by Van Sciver focuses on the use of PL/1 and its consequences, stressing the additional flexibility which its use affords.

All the support software discussed above is classical in the sense that its operation is well understood by every student of computer science and that most technical problems involved have been studied theoretically and translated into cookbook solutions. Because of its complexity, the SAFEGUARD application requires an additional support facility operating in an area where the theory is not well understood. The basic question is how does one validate a real-time multiprocessing system as complex as SAFEGUARD? How does one really know what has happened inside the CLC? To answer the latter question, the capability of data recording is provided by the CLC operating system. Recording makes it possible to transmit data of the designer's choosing to tape at the rate of three million bits per second during CLC execution. These raw data would fill more than 150 printed pages for each second of CLC execution. Clearly, some automated techniques are required to help the designer through this morass of data. This is the function of reduction programs that group and format the information for orderly and meaningful presentation. These programs are called the SAFEGUARD Data Reduction System (SDRS).

A lack of clearly specified requirements makes designing data reduction programs difficult. The designer of an assembler, a compiler, or a simulator can take much for granted. A large body of knowledge exists about these programs, and techniques for implementing them

* The rationale behind the choice of functions may be found in Ref. 8.

have been studied extensively. The designer of a data reduction system has little theoretical knowledge from which to draw. Although the data reduction problem was documented as far back as SAGE, very few people have extensive experience in debugging large-scale real-time systems, and even fewer people understand the importance that real-time debugging considerations play in data base design. So the data reduction designer gets little help from any one, and must build the most flexible and basic package possible in the hope of meeting user requirements as they are discovered.

SDRS is a generalized information storage and retrieval system, part of which was borrowed from another Bell Laboratories application and adapted for SAFEGUARD.[10]

IV. SOME CONCLUSIONS ABOUT SUPPORT SOFTWARE

What made certain support programs more successful than others? Obviously, the more successful ones met the needs of the users. They were available when they were needed, were flexible enough to react as requirements changed, and were reliable. Various methods were used to achieve these objectives. High-level languages were used to retain flexibility. In fact, flexibility was considered so important that efficiency was sacrificed. Software was borrowed shamelessly, 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 between the facilities to ensure integrity. All these techniques helped to build a successful support software system.

V. THE USE OF SUPPORT COMPUTERS IN SAFEGUARD DEVELOPMENT

As indicated earlier in this paper, a basic decision was made to move as much support software work as possible from the CLC to a commercial computer. This led to major involvement in the use and operation of commercial computers. Items to be discussed include the computers currently being used and how they were selected, their locations, and some of the techniques used to improve cost effectiveness.

In the mid-to-late 1960s, NIKE-X computing was performed exclusively by the Bell Laboratories, Whippany, General-Purpose Computer Center, which operated an IBM 7094 and a GE 635. NIKE-X support software development work had been divided between the GE and the IBM systems. PDF and NICOL development was being done on the GE 635, while the CLC assembler and rudimentary binding

facilities were tied to the IBM 7094. The IBM 7094 was to be phased out when MULTICS became available. However, when it became necessary to choose a computer for installation at Meck Island, the GE 635 PDF-NICOL package was not ready. The only alternative was to code programs from the Meck test system in assembly language. Since there was no CLC assembler for the GE 635, it was necessary to install an IBM System 360/65 (which could emulate the IBM 7094 at lower cost than the original) on Meck Island. Once the commitment had been made for Meck, the IBM 7094 at Whippany could not be released, since it was required for compatibility with Meck. The GE 635 then began to disappear from the mainstream of software development activities. It remains in use full time, principally to produce fault-location dictionaries for SAFEGUARD equipment.[11]

Meanwhile, the Bell Laboratories IBM 7094 was also replaced with an IBM System 360/65 and the CLC assembler and program preparation facilities were run under emulation. Gradually, these programs were converted to run on the IBM System 360/65 in its native mode for increased efficiency. As each conversion occurred, Meck test system dependence on IBM System 360 hardware increased.

Support software for SAFEGUARD further increased the dependence. SNX 360, the IBM proprietary library system, and parts of SDRS were borrowed from other IBM System 360 installations. CENTRAN was built as a macro package on top of SNX 360. XPF was coded in PL/1. Conversion to a non-IBM computer would have severely delayed the operational date of the SAFEGUARD system because virtually the entire support software system would have had to be replaced.

As the project grew, the number (and size) of support computers grew with it. Table I is a summary of the dedicated support computers used for SAFEGUARD and the Meck test system.

The prime purpose of these computers is to provide whatever service is necessary for program development. The challenge in operating them is to do the job in a cost-effective manner. Some methods used

Table I — SAFEGUARD-dedicated support computers (November 1973)

Computer	Location
IBM 370/165	Madison, N.J.
IBM 370/165	Morris Plains, N.J.
IBM 370/155	Concrete, N.D.
IBM 370/155	Nekoma, N.D.
IBM 360/50	Colorado Springs, Colo.
HIS 635	Whippany, N.J.
IBM 360/65	Whippany, N.J.
IBM 360/65/40	Meck Island

to achieve this will be discussed in the following paragraphs. Because the list of such methods is potentially limitless, the discussion focuses on items that might be considered surprising or controversial.

System tuning usually comes to mind first when one contemplates increasing cost effectiveness. Two kinds of system tuning can be distinguished immediately, synthetic and analytic. Synthetic tuning consists of an algorithmic application of such familiar standbys as data set placement optimization and channel balancing. Synthetic tuning is a necessary but not a sufficient condition for a tuned system. The slack is taken up by analytic tuning, which is an attempt to view the computer as a system, to ferret out the bottlenecks through use of the analyst's bag of tricks, to rank the bottlenecks in terms of their system impact, and to propose and implement solutions.

Different types of analysts can be identified by their approach. The most common and perhaps most effective type is the mystic, who appears to find problems by using a logic compounded of experience and intuition. Other types of analysts are the theoreticians, who try to construct classical proofs that one course of action is better than another, and the simulators, who attempt to model the computer with a series of parameters that can be varied to determine an optimum course of action. Neither the theoreticians nor the simulators were well represented on SAFEGUARD. What one did find on SAFEGUARD were the chartists. The chartist believes that it is possible to build one or more computer performance reports that will tell where the bottlenecks are and where tuning is required. One has only to decide which numbers to include in the charts and how to interpret them.

The chartist approach was very successful, but not exactly in the way it was intended. The most significant information derived from the charts was that the biggest system bottlenecks were the users themselves. Reading the reports seemed to show who was a good programmer and who was not, who was getting his job done and who was not, and who was hogging resources at the expense of others. This caused some rethinking about what it really meant to improve throughput. In a global sense, if the programmers do not get their job done, what difference does it make if turnaround time has been reduced or CPU utilization increased? To improve throughput, both the users and the system must be tuned.

Realization of this fact resulted in a campaign to search out those who were using the computer inefficiently. This endeavor was called "bird-dogging."[12]

Another method used to increase the cost effectiveness of SAFEGUARD computer-center operation was the use of plug-compatible components when their use offered equal or superior performance at lower cost.

This had the additional benefit of supporting the government's objective of fostering competition within the computer industry. SAFEGUARD experience gained in replacing IBM 2314 disc storage, IBM 2401 tape drives, and IBM System 370/165 memory does not differ substantially from that reported in the literature. However, certain key points should be mentioned.

First, it is important to have a solid grasp of the economic situation. A commitment to install plug-compatible equipment is also a commitment to expend the funds and the manpower to select replacement equipment and then to make sure that it works. SAFEGUARD has found this to be a nontrivial investment. One must be relatively certain that new announcements or price reductions will not eliminate the expected savings. IBM's announcement of 2319 disc drives cut heavily into the net savings that accrued from SAFEGUARD's switch to plug-compatible 2314s. On the other hand, the conversion to plug-compatible IBM System 370/165 memory has been enormously successful from an economic point of view, primarily because it was decided to purchase the IBM System 370/165s rather than convert to System 370/168s.

Second, it is important that the vendor have a good local service organization with trained, competent people backed by a hierarchy of expertise including the design engineers themselves. Looking back over several years' experience with plug-compatible equipment, the major problems encountered always seem to be due to difficulties within the service organization.

The third and last technique for improving computer-center cost effectiveness is the use of facilities management companies to operate SAFEGUARD computer centers. The IBM System 370/165s at Madison and Morris Plains, N. J., and the IBM System 360/50 at Colorado Springs, Colorado, were all operated this way. There are many advantages to such an arrangement. Perhaps the most significant one is the emphasis that the facilities management companies place on service. In the case of both companies used by SAFEGUARD, the results have been outstanding. This is partially because facilities management companies exist to provide service and partially because of the nature of Cost-Plus-Award-Fee (CPAF) contracts,[13] which make it financially advantageous for the subcontractor to do his best. Another advantage is that Bell Laboratories need not recruit and hire additional computer operations specialists. Finally, there is the issue of cost. Experience on SAFEGUARD has shown that, when all considerations are taken into account, a facilities management company can provide excellent cost-effective service.

The principal disadvantage of subcontracting computer-center operations is that the company selected occasionally takes a parochial

view toward project needs. For example, one company felt that it was extremely important to reduce average turnaround time. What they should have tried to do was to ensure that jobs received turn around in accordance with their importance to the project, regardless of the effect on the average turnaround time. Here again, the CPAF contract is a valuable tool for ensuring that the subcontractor's goals remained aligned with the contractor's goals.

VI. SOME CONCLUSIONS ABOUT SUPPORT COMPUTERS

It is impossible to point out in a few pages all the important lessons from almost ten years of computer-center management experience. In this paper, the emphasis has been put on high-return items—bird-dogging, plug-compatible equipment, and facilities management. Other items, such as hardware and operating system change control and inter-location compatibility, were addressed but have not been discussed. Despite overall success, solutions to many problems were not found; e.g., how does one get users to estimate their computing requirements correctly, or what is an accurate measure of performance improvement? Perhaps our experiences can help others to increase cost effectiveness.

REFERENCES

1. J. W. Olson, "SAFEGUARD Data-Processing System: Architecture of the Central Logic and Control," B.S.T.J., this issue, pp. S41–S61.
2. J. P. Haggerty, "SAFEGUARD Data-Processing System: Central Logic and Control Operating System," B.S.T.J., this issue, pp. S89–S99.
3. N. H. Brown, M. P. Fabisch, and C. J. Rifenberg, "SAFEGUARD Data-Processing System: Introduction and Overview," B.S.T.J., this issue, pp. S9–S25.
4. "No. 1 Electronic Switching System (ESS)," B.S.T.J., *43*, No. 4, Parts 1 and 2 (September 1964).
5. M. E. Barton, "The Macro Assembler, SWAP—A General Purpose Interpretive Processor," Proc. AFIPS FJCC, *37* (1970), pp. 1–8.
6. B. N. Dickman, "ETC—An Extendible Macro-Based Compiler," Proc. AFIPS SJCC, *38* (1971), pp. 529–538.
7. B. N. Dickman, "SAFEGUARD Data-Processing System: CENTRAN: A Case History in Extendible Language Design," B.S.T.J., this issue, pp. S161–S172.
8. J. P. Haggerty, "SAFEGUARD Data-Processing System: Central Logic and Control Operating System," B.S.T.J., this issue, pp. S89–S99.
9. P. A. Van Sciver, "SAFEGUARD Data-Processing System: Systems Programming in PL/1," B.S.T.J., this issue, pp. S173–S180.
10. E. S. Hoover and R. A. Jacoby, "SAFEGUARD Data-Processing System: Data Reduction System," B.S.T.J., this issue, pp. S181–S189.
11. C. J. Rifenberg, "SAFEGUARD Data-Processing System: The Dictionary Approach to Digital Maintenance," B.S.T.J., this issue, pp. S73–S85.
12. J. P. Kuoni, "SAFEGUARD Data-Processing System: A Means to Effective Computer Resource Utilization," B.S.T.J., this issue, pp. S191–S196.
13. W. H. Mac Williams and J. E. Peterson, "SAFEGUARD Data-Processing System: A Cost-Plus-Award-Fee Contract for a Large Software Development Program," B.S.T.J., this issue, pp. S238–S244.