Section I

SYSTEMS ENGINEERING

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

SAFEGUARD Data-Processing System:

The Data-Processing System Performance Requirements in Retrospect

By D. W. MESEKE

(Manuscript received January 3, 1975)

The Data-Processing System Performance Requirements (DPSPRs) specify the required performance to be provided by the SAFEGUARD system software. They were developed primarily by one systems engineering department at Bell Laboratories. Their objective was to specify the required functional performance in sufficient detail to permit software development. The DPSPRs evolved from similar documentation that was developed for systems prior to SAFEGUARD. Their history, development, use, and document control system are described. Suggested improvements are also discussed.

I. INTRODUCTION

The Data-Processing System Performance Requirements (DPSPRs) are a set of documents that specify the required system performance to be provided by the tactical real-time software. A separate set of requirements exists for each site: one for the Missile Direction Center (MDC) site, one for the Perimeter Acquisition Radar (PAR) site, and one for the Ballistic Missile Defense Center (BMDC). The DPSPRs include requirements for such functions as site communications, displays and controls, radar control, interceptor response planning, and missile guidance. Since the SAFEGUARD system must operate continuously in real time with minimum down time, the DPSPRs also include requirements for exercise and fault detection to verify total system performance. The DPSPRs do not include requirements for installation and checkout software, software error control, or process initialization.

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

formalize for the customer—the Army SAFEGUARD System Command (SAFSCOM)—the required system functions, their interactions, and the expected system performance.

The DPSPRs contain detailed requirements for each identified system function. They are not part of the high-level contractual documentation, and they do not contain the detail required to subcontract software development. They are written at an intermediate level along functional lines, but they do not dictate the organization of the software. For example, one section, Target Selection, provides requirements for calculating a set of parameters from quantities specified in another section, Track. When the software was designed, it was found more efficient to have the track software calculate the parameters and pass them to the target selection software. Because the DPSPRs did not specify software organization, it was possible to choose the more efficient software implementation.

This paper provides a retrospective view of the DPSPRs, identifying different aspects of their development that either worked well or should have been done differently. The history of the DPSPRs is given first, followed by a description of how they were developed. A short description of how they were used is given next, followed by a section on document control. The conclusion summarizes recommendations that may be useful for the generation of future data-processing performance requirements.

II. HISTORY

Prior to SAFEGUARD, considerable experience had been gained from the design of the NIKE-ZEUS, NIKE-X, and SENTINEL ABM systems. As part of NIKE-X research and development, a series of documents was developed to specify how various system functions would be performed. They described, for example, how the radars were to gather target trajectory data required to launch and guide a missile to intercept a target. These documents were the forerunner of the DPSPRs.

In January 1967, system studies were initiated to determine the feasibility of deploying a thin area-defense system, later defined as SENTINEL, using components (radars, data processors, missiles, etc.) developed for NIKE-X. The first Data Processing System Performance Requirements documents were written for the SENTINEL system.

In April 1969, Bell Laboratories was redirected by the Department of Defense to develop the SAFEGUARD ABM system. The initial issue of the DPSPRs for SAFEGUARD was completed in July 1969 in accordance with this redirection. The time interval was short because many of the DPSPR concepts and functions that had been developed for SENTINEL

were applicable to SAFEGUARD and because this first issue contained mainly qualitative requirements; i.e., many parameter values were still to be determined. The purpose of this first issue was to disseminate as much information as soon as possible to the software designers, who had already organized to develop the SENTINEL system. This issue was succeeded by a second, more quantitative issue in May 1970, which was placed under internal document control.

On March 31, 1971, the DPSPRs were submitted to the customer for baselining. Baselining the DPSPRs consisted of a detailed document review and preparation of changes, after which both the customer and Bell Laboratories agreed to accept the documents. This process was completed on May 31, 1972. The baselined DPSPRs were then submitted for formal configuration control procedures under which all changes had to be (and must still be) approved by the SAFEGUARD Local Configuration Control Board.

III. HOW THE DPSPRs WERE DEVELOPED

Development of the DPSPRs was the function of the system design department, which initially consisted of about thirty engineers and programmers. Their first step was to write an "operational concept" paper for SAFEGUARD. The concept paper identified the defense objectives, the command and control configuration, and the general operation of the radars and missiles in their defense roles.

Based on the concept paper, the DPSPRs were organized according to the operational functions required at each site. The organization of a typical DPSPR is shown in Table I. The DPSPRs were arranged so that each section addresses a major system function. The ordering of the sections within a DPSPR was primarily based on the sequence in which the functions must be performed. Each section includes three main subsections: objective, operational description, and requirements.

Table I — MDC DPSPR organization

1. General
2. MDC Site Management
3. Radar Management
4. Surveillance
5. Track
6. Target Selection
7. SPARTAN Interceptor Response
8. SPRINT Interceptor Response
9. SPARTAN Guidance
10. SPRINT Guidance
11. Equipment Tests
12. Exercise Subsystem
13. System Constraints
14. Displays and Controls

The operational description includes, in most sections, a general description of how that function is to operate in different system-defense modes. It has been suggested that the DPSPRs should have had one section devoted to a complete operational description of the system rather than appearing throughout the documents. Since the level of detail varies from section to section, this suggested reorganization could probably have provided a more consistent description of the functional operation of the entire system. The concept paper did not provide the detailed descriptive information that was later felt to be lacking on the project.

Original plans called for each DPSPR section to have an inputs/outputs subsection that would define the interfaces among functions. These subsections were never included in the DPSPRs, primarily because there was insufficient time. Since the requirements for each function either specified or implied its inputs/outputs, it was felt that these subsections could be omitted. In retrospect, this probably was a mistake. For instance, an implied output of one function was missed by the software designers in a specific case in which one function was required to stop or inhibit an action previously started by another function. This mission output was not discovered until later during functional integration testing of the designed software. Then, many questions were raised:

(*i*) Is it really necessary to stop the action?
(*ii*) What happens if the action is not stopped?
(*iii*) Can the missing output be implemented without jeopardizing schedules?
(*iv*) How much retesting is required if a modification is made?

Clearly, a perturbation in the software development effort occurred that might have been avoided if the inputs and outputs had been explicitly stated.

The general policy for writing requirements for a function was to state the requirements without descriptions of how the function should be implemented. In many cases, this was difficult to do; it was often easier to say how a function should be done rather than to state a performance requirement for the function. This led to two problems. First, when a requirement specifies how something is to be done, the software designer feels constrained. He may know a better way to implement the requirement or he may want to try other ways. Second, if a performance requirement does not exist for a function and only the technique is specified, then the test designer must generate his own performance requirement, or his tests will check only to see that the proper technique has been implemented. For this reason, it has been

suggested to both system designers and software designers that the DPSPRs should have included only system performance requirements with no mention of implementation or algorithms. This is a philosophically "pure" notion which might or might not work. The method of specifying requirements depends largely on the project organization and the talent of the people involved. For example, before writing a DPSPR function, the systems engineer discussed the particular function with his counterpart, the software designer. In some instances, primarily those in which there was a lack of specific experience, the designer requested that complete design details be supplied. In other cases, the designer wanted only an overall performance requirement because he felt he knew how to produce the design. Requirements were written both ways, but experience suggests that it is probably best to state the performance requirement and then provide a recommended technique to be used at the designer's option. In summary, the system designers tried to reason out the level of detail to be included and took the pragmatic approach of "getting the job done" and trying to satisfy both the software designer and the customer.

IV. HOW THE DPSPRs WERE USED

In software design, the DPSPRs were used in three phases: setting up the software structure, establishing the internal organization of each basic function, and functional testing.

In setting up the software structure, the routines and subroutines needed to perform the functions were based on the requirements in the DPSPRs. In software design, primary emphasis was placed on definition of the inputs required to perform the functions and the outputs required by other functions.

Next, the internal organization of the defined routines and subroutines was established. At this time, emphasis was placed on defining both the particular algorithms required within a function and the interfaces between routines.

As the design of the routines approached completion, the DPSPRs were continually consulted to determine if the designs met requirements. DPSPRs were then used to determine the functional testing required for the completed design.

In system evaluation, the DPSPRs were used primarily as a reference document. The first stage of system evaluation was to verify that the DPSPR specifications would meet system objectives. The evaluation program then determined if the implementation met the DPSPR requirements. The system evaluation effort led to development of new system functions, changes to existing ones to provide better performance, and sometimes modification of the requirements themselves.

The DPSPRs were used by the customer as the documents that specified performance of the system they were buying. The customer coordinated with the design engineers in the formulation of all prebaseline versions of the DPSPRs. After baselining, the customer was deeply involved in the evaluation and discussion of each change proposed for the DPSPRs. In addition, the DPSPRs were used by the customer for his independent evaluation of the system design.

V. DOCUMENT CONTROL

After the first issue of the DPSPRs was published and distributed, an intensive review was held with software designers and system evaluators. This resulted in changes to add new requirements, to expand upon old ones, and to correct errors. No formal accounting of the agreed-upon changes was kept, and some systems and software designers were not made aware of these changes until they received their copies of the next issue. Clearly, there was a need for a better method of keeping track of problems and their solutions and a need for timely revisions.

To solve this, a document control system was established in which all DPSPR-related problems were identified by a Trouble Report (TR) and the solution to each problem was described by a Correction Report (CR). TRs could be written by anyone uncovering a problem, but had to be approved by the writer's immediate supervisor. Once approved, the TR was given a number, recorded in the log book, and sent to the supervisor responsible for the affected DPSPR. After his approval for action, the TRs were assigned to the persons responsible for the particular sections that were related to the problem. Each solution was described in a CR to be approved by the TR originator. So both the TR originator and CR originator had to agree upon the solution. When agreement was reached, the CR had to be approved by the supervisor responsible for the applicable DPSPR.

Since changes to the DPSPR generally implied corresponding changes in the software design, all CRs were reviewed and approved by all affected software design departments, with final approval delegated to higher levels of management as the software delivery date was approached. After final approval, the CR was sent to publications for preparation and distribution of the revision pages for the CR.

Three different methods of achieving this approval were tried before an adequate approval sequence was found. Figure 1 shows a flowchart of each of these methods. First (Method 1), after the DPSPR coordinator approved the CR, a copy was sent to each affected software design supervisor for an assessment of the software impact of the change in terms of cost and schedules. When all assessments were received by

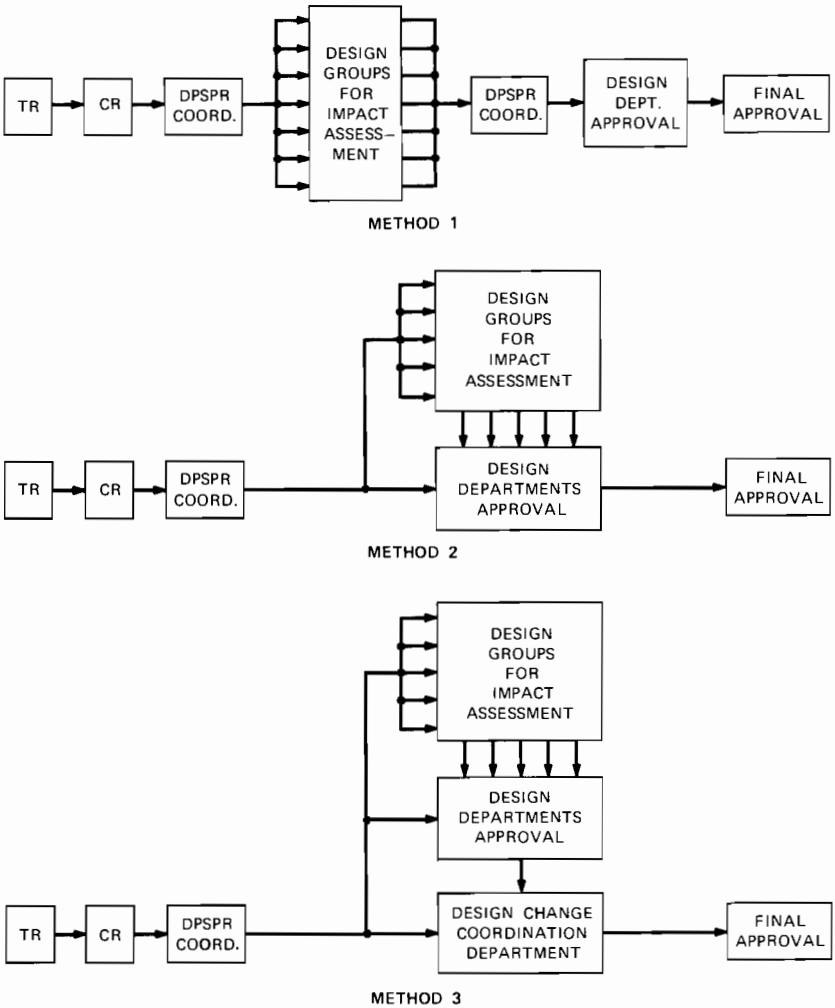

Fig. 1—TR/CR approval sequence.

the DPSPR coordinator, the assessments were attached to the CR and the CR was then routed in turn to the head of the system design department, to the head of the software design department affected by the change, to the heads of all other software design departments, and finally to the director of software design. This procedure resulted in significant delays in the return of impact assessments and in department-head routing. It only worked efficiently when the DPSPR coordinator hand-carried the CR through the approval sequence.

The procedure was then changed (Method 2) to one in which the CR was immediately routed to department heads and, at the same time, information copies were sent to all software design supervisors whose design would be affected by the change. When each department head received the CR for approval, he requested software impact estimates from his supervisors. This procedure was more effective than the previous one; however, CRs tended to become backlogged in the department-head routing process. This resulted primarily because no one representative of the design organization had the responsibility to ensure that each CR received appropriate and timely action.

The final procedure (Method 3) was quite similar to the previous one except that one department head was designated as the change coordinator with the responsibility of ensuring that each CR received the appropriate attention and that all software changes were properly coordinated.

The DPSPRs were submitted to the customer for baselining on March 31, 1971. From that time until the DPSPRs were finally baselined in May 1972, changes were allowed in the DPSPRs by means of the procedure described above. This allowed the DPSPRs to be reasonably current during this period; however, additional effort was required by the customer to review the TR/CR changes as well as the submitted DPSPRs. After baselining, the only change to the TR/CR procedures described above was that approved CRs were incorporated into an Engineering Change Proposal (ECP) which required customer approval before the CRs associated with the ECP were forwarded to publications for generation and distribution of revision pages. There were instances, of course, where software design changes had to be made to make the system work before customer approval could be obtained. The control procedures allowed for this as a "management risk."

The control procedure enabled the project to keep track of all problems and their solutions and to control the changes in system design. However, after the document control procedures had been prepared, a few suggestions were made that might have improved the process.

First, in addition to detailing the specific change to the DPSPR, the CR should have included the rationale and/or study that led to the change. In cases where significant changes were made, they were generally documented in a memorandum; however, little or no rationale accompanied many small changes. Including the rationale would probably have reduced duplication of studies that were conducted by the system designer and the software designer to evaluate changes.

Second, the software design organization always should have been a party to the initial approval of a correction report. This was done

when the TR was originated by software design, but was not done when a TR was initiated by system design or by system evaluation. By coordinating all correction reports through the software designer, there probably would have been fewer unapproved CRs to rework. This would also have made the software designer aware earlier that a change in his design was being proposed.

Third, the TR/CR approval sequence and publication of the change should have been streamlined as much as possible. Even though the designers knew of the change, most other DPSPR users were not aware of it until the revision pages were issued. One change to the approval sequence that might have shortened the approval cycle time would have been to establish a formal calendar date for final review and approval at the highest necessary management level when the CR began its approval sequence. Each CR would be reviewed on that date and rescheduled if a final approval decision could not be reached. This approach would have forced timely attention to each CR in the approval cycle.

VI. CONCLUSIONS

One of the most fundamental needs in a software development project is a clear statement of requirements. The DPSPRs were designed to meet this need and were successful in doing so. They have also provided valuable insight into the design of testing and evaluation procedures. The most notable deficiency in the DPSPRs was a lack of explicit definition of interfaces among the various functions. More concentrated effort in specifying exact definitions of these interfaces would have greatly helped the software designers. The most important lesson learned in setting and maintaining requirements is that changes to the system design must be carefully controlled. It is essential that software designers be made fully aware of the content and implications of each system change.