

SAFEGUARD Data-Processing System:

Process-System Testing and the System Exerciser

By B. P. DONOHUE III and J. F. McDONALD

(Manuscript received January 3, 1975)

This paper considers two problems: how to build the SAFEGUARD software so that it is testable and how to test it as realistically as possible. The first is solved by an iterative process of adding software capabilities, testing them, then adding more. The second problem is solved by driving SAFEGUARD with computer-generated radar echoes.

I. INTRODUCTION

Testing activities play a crucial role in the development of all hardware/software systems. These activities are described in terms of two phases, system integration and system testing. The system integration phase is carried out through tests which determine that all components of the system have been properly connected and are performing their specific function correctly. During the system test phase, the *performance* of the overall system is determined through analysis of the results obtained from some finite set of tests. The tests must reflect, as well as possible, the environment and full range of permissible data and control inputs. Although these phases overlap extensively, much system integration occurs before the system test phase.

It is well known that very difficult problems may be encountered in the system integration and test phases of complex system development programs. The plans and some of the significant techniques used to minimize these difficulties for the SAFEGUARD development are discussed.

Plans for the full SAFEGUARD system tests required large-scale analysis and simulation of the complete system. Since it is not possible to describe all the considerations that went into this planning, discussion is limited to a general description of overall system test planning. However, the relationship between the overall system tests and the

data-processing effort are described specifically. Particular attention is given to the system exerciser because of the important role it plays.

II. SYSTEM INTEGRATION AND TEST PLAN

For several reasons, it is vital to prepare a detailed system integration and test plan. First, the time allocated for conducting the integration and test phases is usually not sufficient to demonstrate system performance under all conditions. This is simply an empirical observation. It could be attributed to the lack of detailed understanding of the objectives at the time the overall system development schedules are being formulated. It is always possible to conceive of an infinite number of tests of any complex system. No matter how carefully planned, the number of necessary tests will still be very large and, therefore, require a significant amount of calendar time to conduct. Since the system integration and test phases are the last activities before making the system available to the user, there is always pressure to make these periods as short as possible. The early existence of a detailed test plan is important because it provides strong support in arguing for reasonable system integration and test intervals and allows optimal use to be made of the allotted time.

Second, the system integration and test phases can overlap and, therefore, interact extensively. The tests that are conducted during the integration phase are designed to verify that system components perform as specified. Results from these tests can serve to increase confidence in overall system performance. The scope of future testing can be significantly influenced by this increased confidence. As a result, the testing activities in these two phases should be well coordinated.

Third, there are always schedule difficulties during the system integration and test phases if planning for test tools, techniques, and procedures does not begin long before the actual test period. Development of the hardware/software products can be influenced by test considerations. The test tools can often be developed more economically, and will better serve needs if identified early. Preparation of a detailed plan is the best way to recognize required lead times and avoid such scheduling difficulties.

Fourth, monitoring of progress is particularly difficult during these phases of the development. It is not uncommon to find that progress has been negative (and unknown) during parts of these intervals. A detailed test plan can serve as a very good measuring guide to monitor this progress.

Some general characteristics of a good system integration and test plan are reasonably clear. It identifies the means to achieve a specific set of objectives in a specific time, it recognizes the availability and

capability of other tests carried out during the development, and it reflects all appropriate constraints on the use of resources. In **SAFEGUARD**, certain features of the plan were more significant than others. Four have been selected for more detailed discussion.

2.1 The incremental approach

Everyone recognizes that a complex system cannot be integrated in one step, so an “incremental approach” must be used; i.e., the complexity of the hardware environment, the software, and the test cases must be built up incrementally.

Several factors were considered in arriving at the specific incremental approach for **SAFEGUARD**. These led to a series of steps of increasing complexity, where each step included a given level of hardware, software, and functional tests. The principal steps were:

- (i) Integrate all the “control” software; i.e., demonstrate the basic operating control necessary to perform initialization and cycling.
- (ii) Integrate those software units that are part of critical timing chains.
- (iii) Integrate additional software, which allows a simple, but consistent, stream of functional processing.
- (iv) Interface this software with hardware; e.g., radars.
- (v) Integrate remaining software to provide complete capability.

These principal tests were supplemented with additional parallel testing of various parts of software. Following is a brief description of how these steps were applied to the Missile Direction Center (MDC) application software.

First, the basic control programs were merged with the operating system, and the ability to load, initialize, and cycle was established. Then software dealing with the radar loop was added; i.e., radar management, search, and track programs. Ability to search and track was then established at low traffic levels, while the radar hardware was simulated with software. After sanity was established in the software, the radar hardware was introduced into the testing loop. In parallel with this activity, application programs supporting intersite communications and command and control were tested in a separate test bed. Similarly, both battle planning and missile guidance software were tested in separate software environments. Ultimately, these programs were merged into a single process, and the complexity of the test cases was systematically increased.

The incremental approach can create difficulties. It is obvious that some mechanism must be provided to represent interfaces of programs

that are not yet a part of the process. Dummy programs, called "stubs," were provided. The requirements for stubs depend on the nature of the programs they represent and the sequence in which programs are added and tested. If this aspect of the incremental approach is not carefully considered during test planning, the stubs may become nearly as complex as the programs themselves, thus defeating the incremental strategy.

The selection of test cases can affect the efficiency of a test plan in a major way. SAFEGUARD has literally hundreds of individual capabilities and operates over a continuum of threat environments. Each test was carefully designed, using a design-of-experiments approach, so that all capabilities covering the full range of operation could be verified with the smallest number of tests. The test design was also approached from an incremental viewpoint, and was found to require an iterative effort.

The sequence used in identifying the test cases for full system testing of the SAFEGUARD MDC is briefly described here.

- (i) The peak traffic level to be verified in full system testing was selected.
- (ii) The types of threats to be countered, and allowable combinations, were delineated.
- (iii) A sequence of tests starting with a single target and building up to peak traffic was identified. The "single target" was common to all test cases, as were other targets added later. Keeping pieces of the threat environment common provided a basis of test result comparisons—peg points along the way.
- (iv) A set of high-traffic test cases was defined and all capabilities tested were identified. This exercise was performed iteratively with the goal of identifying a *minimum* set of high-traffic tests that, as a collection, test all system capabilities and cover all necessary threat mixes.

2.2 Success criteria

The system integration and test phases are intended to demonstrate that the various components and the system operate as intended. Tests are designed to subject the system to various stresses and conditions. The crux of test design is the clear specification of criteria that can be used to measure successful operation. It is obvious that this has to be done, but it is not always recognized that the success criteria will affect a test program in so many ways. For example, the efficiency of the test activities is vastly improved if the success criteria, that is, expected results, are available before the execution of the test. The

criteria can also affect the data recording and reduction efforts. Since the specification of success criteria is a form of testing, it is not uncommon to uncover problems in either requirements or implementation. All these factors recommend that success criteria be identified early in the development sequence.

This effort was both difficult and large. On the SAFEGUARD project, sources of information that provided a basis for establishing success criteria included results of the test program conducted at Meck Island, desk analysis, and simulations. The greatest amount of data came from the simulations of the system. Various portions of SAFEGUARD were simulated in varying degrees of detail. These simulations were in turn calibrated using analytical and field data results. Where possible, the simulations were then used to predict system performance for each test case. The success of a test was measured by comparing data recorded during the test to predicted values. The simulations were large, initiated early, and served as a basis for system evaluation activities.

2.3 Data recording and reduction

One critical step in testing a system is measuring the system's performance. The basic measurement tool in the SAFEGUARD project was the recording and reduction of test data. Because of the complexity of the software processes and the tightness of schedules and on-line computer time, the ability to process recorded data off-line was essential. Recording and data reduction were not treated as two problems, but rather as two aspects of the same problem.¹ A coordinated approach to recording and data reduction was taken to achieve an efficient solution.

In "high-traffic" testing, or in any mode of testing, in fact, recording should be minimized (e.g., so that the off-line data reduction system is not overwhelmed with data). To meet this goal and still preserve the necessary error isolation capabilities, a "hierarchy" of recording selectability was defined.

The basic approach to recording and data reduction for SAFEGUARD was to construct each process so that the ability to select the desired mix of recording per run or per test could be accomplished with ease. Each process has the necessary capability for all possible recording permanently embedded in the on-line code. Data reduction program activities of sorting and formatting are minimized by the real-time association of sort "handles" with the recorded data. The key to the approach lies in a hierarchical structure in which multiple levels of recording are established. In general, three levels (high, intermediate, and low) are sufficient, although additional levels could be used in special instances.

The basic three levels can be described as follows. A process is divided into process functions. The recording necessary to isolate a test failure to a process function or to a peripheral is the highest level of recording. In general, these highest level data should consist of "counts" or statistics usable to determine logic flow, basic time sequencing, etc. The lowest level of recording consists of a detailed record of the processing of an input by a process function on a single logical pass. The intermediate level of recording is designed to aid the tester in selecting the proper low-level recording options.

A quick-look on-line computer capability was embedded in the software to allow off-line data reduction to be bypassed on occasion. This allows critical data to be "recorded" in on-line memory and output on a printer immediately following test completion. The test teams used quick-look and operating-system debugging aids² to support integration. Using quick-look, they determined when and in what portion of the process detailed recording should be performed. In the case of system tests, the system test specification specifies success criteria and prescribes the data to be recorded.

In testing the Meck prototype system, there were several examples of missions in which millions of words of data were recorded. In conducting a test involving missile launches, it is necessary to record all data of any possible interest, for the cost of repeating such tests is extremely high. However, the cost of repeating a test is reasonably economical in the TSCS (Tactical Software Control Site) since no launches are involved. Although tests are not absolutely repeatable, they are essentially repeatable in a functional sense. This means that a hierarchy of recording can be utilized to minimize the data recorded in real time, minimizing the off-line data reduction required. If a test fails, it can be repeated with selective recording performed in the suspect areas of the system. Although this approach forfeits some capability to isolate transient errors, it allows trade-offs to be made in the use of on-line computer time vs off-line data reduction time. With hierarchical recording, better test turnaround and lower overall integration costs were achieved without any serious problem in isolating transient errors.

2.4 Test tools

The need to provide test signals and data to "drive" any system is clear. As the complexity of the system and its operating environment increases, so does the complexity of the driver. It was considered vital to devote considerable resources to the development of a driver, and the effort was started early.

Few ground rules were available to guide its development. As it evolved, both special-purpose hardware and software were required. Because this effort was viewed as one of the more significant ones, the driver, or the SAFEGUARD system exerciser, is discussed in detail in the following section.

III. THE SYSTEM EXERCISER

The primary role of the system exerciser is to support test and integration of SAFEGUARD applications software in the hardware environment in which it was designed to operate. But testing SAFEGUARD against a simulator is difficult for two reasons. First, SAFEGUARD is a complex system involving radars, missiles, and interacting sites; the number of combinations of inputs is immense. Second, in actual operation, some inputs, such as radar noise, are random variables; these inputs should be random during testing as well.

Because of its complexity, it was not feasible to simply assemble the entire system and drive it utilizing the system exerciser. The system was assembled in an incremental sequence. The development of the system exerciser was, likewise, modular in nature. At each building stage, portions of the system exerciser's capability were used to drive that portion of the system included in the test bed. By relating the sequence of capability buildup in testing to the modularity of the system, an efficient development plan was evolved.

During the early stages of SAFEGUARD development, several goals for the system exerciser were established consistent with the primary role. The five most important goals are:

- (i) As much of the system, hardware and software, should be exercised as is cost-effective. The software heavily interacts with the hardware; hence, confidence in the software/hardware combination can only be established through successful demonstration of their interactions.
- (ii) The impact of system exerciser implementation on the application-system implementation should be kept to a minimum.
- (iii) The system exerciser's simulation of the environment should be as realistic as is feasible.
- (iv) The traffic capacity of the system exerciser should exceed the design level of the application system.
- (v) The system exerciser should provide the capability to record the outputs of the application system.

During the development, every effort was made to retain sufficient flexibility to allow the system exerciser to be used in other ways, e.g.,

determining in part the system readiness and verification in an operational time period.

The discussions that follow apply to the MDC and PAR system exercisers. The approach taken for the BMDC exercise was different because of its distinct processing function (control and display) and relatively small size. The MDC system exerciser is the most complex.

3.1 Structure of the MDC system exerciser

Figure 1 shows the normal connections between equipment at an MDC site. During a system exercise, these connections are rearranged under software control as shown in Fig. 2. Data sent by the application data processor to the radar, the missile ground equipment, and other sites are directed instead to the exercise data processor. The system exerciser generates plausible radar returns, missile responses, and messages from other sites, and returns these to the application data processor. The exerciser is separated from the system being tested; it operates in a separate data processor connected to the application data processor through a special digital hardware unit, the Exercise Control Unit (ECU).

Tapes containing target and some environmental data to be used in the simulation are prepared off-line in nonreal time by a program called the SAFEGUARD Threat Action Generator (STAG). The design of STAG and the real-time processes was closely coordinated.

Several decisions were made in the design of the MDC system exerciser. First and foremost, the exerciser software was executed in a data processor distinct from the application data processor. The execution of exerciser programs in no way interferes with the execution of application programs. The alternative of executing the exerciser programs in real time on the application computer had been taken in the pro-

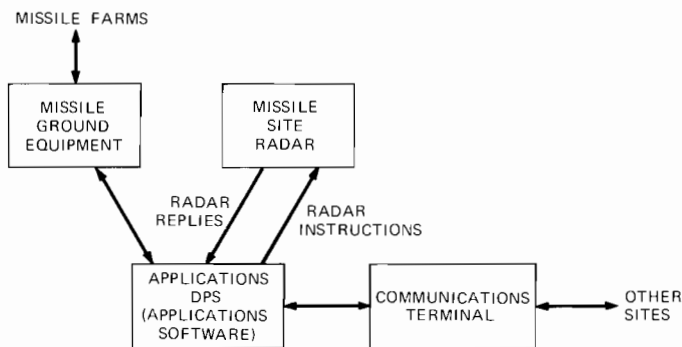


Fig. 1.—SAFEGUARD MDC site equipment configuration.

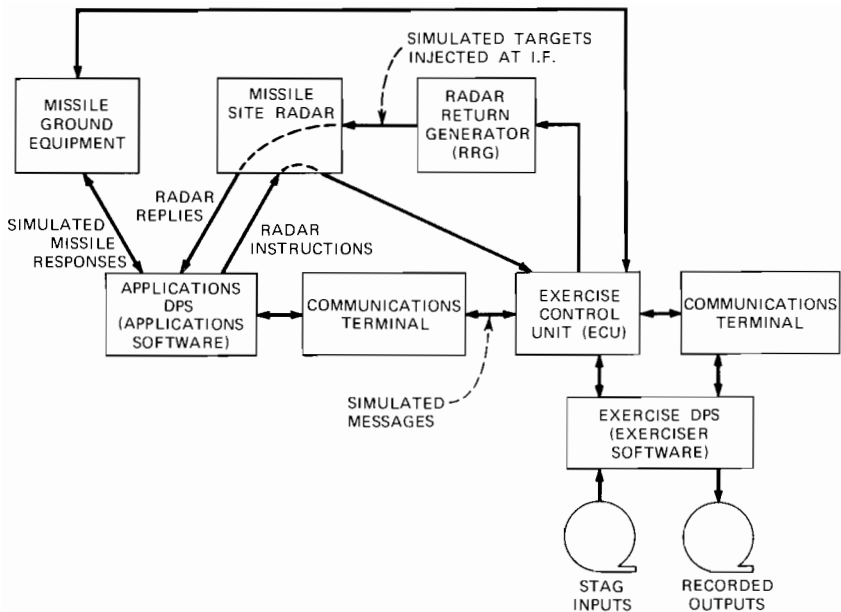


Fig. 2—SAFEGUARD MSR site configured for a local exercise.

tototype system. Separation of the application and exerciser program systems also allows the development of the exerciser to remain as independent of the application system as possible. The potential for the exerciser programs to corrupt the application programs while operating in a combined form was demonstrated on occasion with the Meck test system.

Experience with the separated application and exercise systems has been favorable. No interference or identifiable differences in queuing or timing between the exercise and application modes was found. For instance, exercises were conducted that involved the tracking of "simulated" satellites. The performance of the application process was comparable when similar "live" satellites were tracked.

At one stage of the design, it was recognized that requirements for exerciser data processing throughput could be reduced by about 40 percent if the exerciser's load could be made more uniform. All that this required was to have the application program distribute in time the data which the application data processor sends to the radar (see Fig. 2). Changes were made to accomplish this without affecting the capability of the application system. Other examples include the setting of "flags" by the application program in data that it sends to the radar. When the exercise intercepts the data, it uses the flags to help expedite

processing. This was accomplished without compromising either the applications or exerciser roles.

A second decision made in the design of the exerciser was to utilize as much of the hardware in an exercise as possible. Clearly, the real defensive missiles could not be included, but we note that the exerciser interfaces with the system of missile ground equipment, not just at the software/hardware interface. The full radar could not be included because a real target environment is not available to be viewed and because the cost of injecting simulated signals at the radar face is prohibitive. As shown in Fig. 2, the ECU injects simulated signals into the radar at the RF strip. This has allowed the applications software to be tested with major portions of the radar. This proved to be an effective approach from several points of view. It provided a mechanism to identify numerous problems in the hardware and software at the TSCS (the test bed). These problems included radar instruction sequencing errors, tracking bias errors, miswiring, etc. Corrections were made to both TSCS and site hardware. Software was corrected before it was shipped to the site. As a result of the prior testing at the TSCS, relatively few problems were found with the testing at site. Problems that were found were largely attributed to the detailed characteristics of the hardware not included in the exercise. The number of problems was lower than originally expected.

A third decision in building the system exerciser was to perform as much of the calculation required for simulation as possible before conducting the real-time exercise. Calculations for targets, defensive missile farms, and other sites and of hardware was done off-line, in the STAG facility; and results were placed on tape. The real-time software modified these data as appropriate for the real-time condition. This approach minimized the size and complexity of the real-time exerciser on a nonreal-time, pre-exercise basis. It also allowed programs such as trajectory generators to be used to support exercises for different radars; i.e., both the PAR and the MSR. This reduced the total size of the effort.

Fourth, in designing the exerciser, a number of decisions were made relative to the realism of the various exercise simulations. The approach was usually, but not always, to simulate the effect of a particular phenomenon, rather than the phenomenon itself. For example, in simulating the stream of intersite messages the MDC receives from the PAR, there were several options. The highest degree of realism would be a detailed simulation of the PAR system interacting with the threat environment. A much cheaper option would be to generate a representative sequence of intersite messages per threat. These threat messages

would then be combined with a set of PAR status messages and modified in real time as appropriate. For SAFEGUARD, the latter approach was taken because it was economical, yet sufficient.

3.2 Exercising the exerciser

The system exerciser is a complex system, although considerably smaller than the applications system. As the principal tool in integrating the applications software, it had to be stable and reasonably debugged. There were at least two alternatives to test it. On one hand, the testing of the exerciser could be performed in conjunction with the testing of the applications system. On the other hand, the system exerciser could be tested as a stand-alone system. The latter approach was taken for SAFEGUARD, because it allowed greater control and easier isolation of problems.

Testing the exerciser was conceptually simple. We can view the applications software as outputting radar instructions, missile instructions via the missile ground equipment, and intersite messages. Those three classes of outputs represent the stimuli to which the exerciser responds. To test the exerciser, a simple software package called the Exercise Standard Test Process (ESTP), which resided in the application data processor and output these stimuli, was generated.

In simplest terms, ESTP obtains time-tagged data blocks containing radar instructions, missile instructions, and intersite messages from a driver tape. ESTP outputs each data block at the appropriate time. The key part of all this, of course, is the generation of the driver tape.

The most critical output from the applications software to the real-time exerciser is the stream of radar instructions. The real-time exerciser must determine whether or not any tactically ordered radar operations will cause the simulated radar to view any simulated targets. To test this portion of the exerciser, a stream of radar instructions that cause the exerciser to perform its simulation calculations is required. The target trajectories are known, and the expected response of the applications system is known. With this information, the radar instructions to be generated by the applications system are computed. ESTP assumes a "perfect" tracker but does not simulate the application system tracker. With respect to the missile loops and the intersite loops, similar deterministic test methods were used to exercise the exerciser.

Because of the testing done with ESTP, relatively few problems were experienced with the exerciser when it was interfaced with the applications software. Just as importantly, ESTP provided a vehicle for further isolation and debugging of problems that did occur.

IV. CONCLUSIONS

Some lessons learned from SAFEGUARD system integration and test activities can possibly be applied to other projects. They are summarized as follows:

- (i) Prepare a test plan early; even though it cannot be complete initially, it should address those items that could affect design, or require long lead time.
- (ii) Consider an incremental approach to testing. Several iterations will be required to decide what form the incremental buildup should take. Details will affect the program development schedules.
- (iii) Start the identification of tests early. Don't delay the specification of success criteria. This specification requires lead time and coordination with other activities and can go a long way toward getting design problems resolved early. Make every attempt to minimize the total number of test cases. The expense of doing the necessary analysis, test specification preparation, etc., is large and often underestimated.
- (iv) Make adequate provisions for an exerciser. Consider separating but not isolating the exerciser from the applications system. Try to incorporate as much of the hardware in the exercise configuration as possible. Test the exerciser to create a stable base for system testing.

REFERENCES

1. E. S. Hoover and R. A. Jacoby, "SAFEGUARD Data-Processing System: The Data Reduction System," B.S.T.J., this issue, pp. S181-S189.
2. A. K. Phillips, "SAFEGUARD Data-Processing System: Debugging a Real-Time Multiprocessor System," B.S.T.J., this issue, pp. S133-S145.