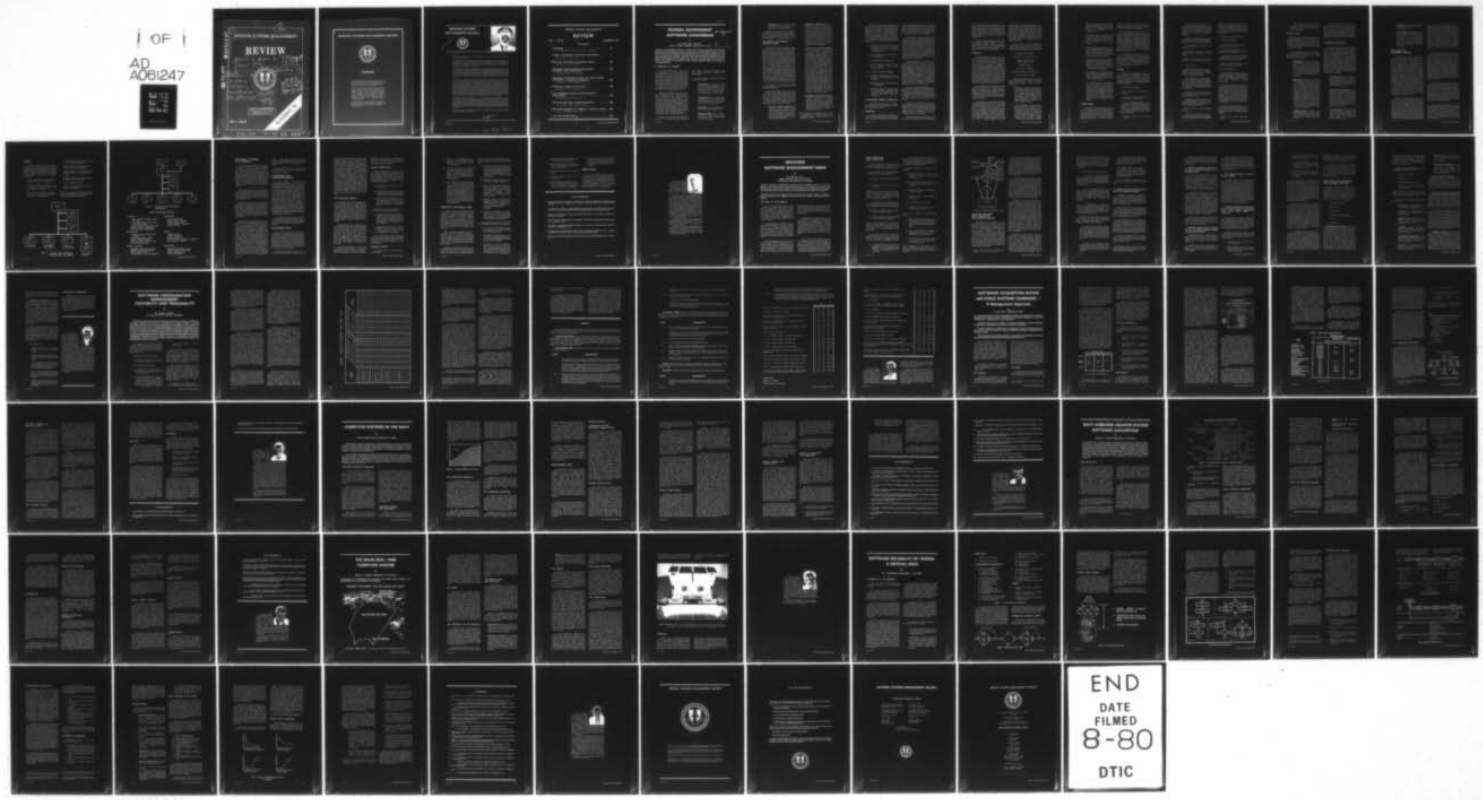
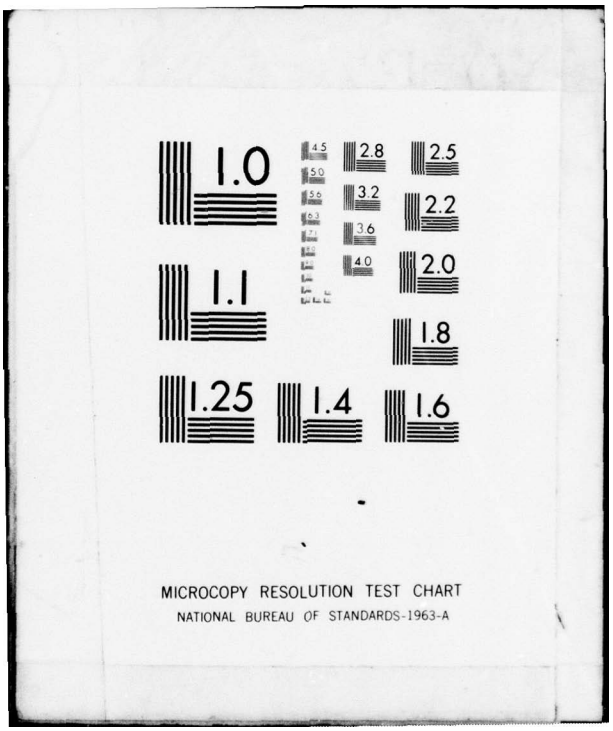
AD-A061 247 DEFENSE SYSTEMS MANAGEMENT COLL FORT BELVOIR VA F/G 9/2
DEFENSE SYSTEMS MANAGEMENT REVIEW, VOLUME 1, NUMBER 6, SUMMER 1--ETC(U)
1978 R G FREEMAN, P OLIVER, R DAVIS

UNCLASSIFIED NL

1 OF 1
AD A061247




1.0
1.1
1.25
1.4
1.6
1.8
2.0
2.2
2.5
2.8
3.2
3.6
4.0
4.5
5.0
5.6
6.3
7.1
MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

AD A061247

DDC FILE COPY

DEFENSE SYSTEMS MANAGEMENT

REVIEW.

Volume 1, Number 6.
Summer 1978.

LEVEL II

Rowland G. Freeman, III
Paul Oliver,
Ruth Davis,
Harvey Tzudiker
John Marciniak

1978

78p.

DDC
RECEIVED
NOV 16 1978
A

DISTRIBUTION STATEMENT A
Approved for public release;
Distribution Unlimited

SUMMER 78

VOL 1, NO 6.

410 036 78 11 15 025 LB

DEFENSE SYSTEMS MANAGEMENT REVIEW

PURPOSE

The purpose of the Defense Systems Management Review is to disseminate information concerning new developments and effective actions taken relative to the management of defense systems programs and defense systems acquisition.

The Review is designed as a vehicle to transmit, between persons in positions of leadership and responsibility in the program management and systems acquisition communities, information on policies, trends, events and current thinking affecting the practice of program management and defense systems acquisition. The publication serves as a means for providing an historical record of significant information associated with defense systems acquisition/management concepts and practices.

The Review supports the assigned mission of the Defense Systems Management College and serves as a medium for continuing the education and professional development of persons in the field.

Review: US ISSN 0363-7727

DEFENSE SYSTEMS MANAGEMENT COLLEGE

AD A035432

Dear Reader:

One of the greatest challenges to the acquisition manager today involves the management and acquisition of computer software systems.

A major weapons software acquisition in conjunction with the hardware requires at least 5 years from initial description of the software system through architecture to programming, coding and debugging, and probably more to get a relatively error-free program in existence. There is a distinct probability that the software system so fielded will not be error-free and that supplemental programs must therefore be developed for error determination and rectification.

Over the past decade, we have seen a burgeoning problem as computer technology has advanced the state-of-the-art in large scale integrated circuits, mini-processors and advanced computer language. Where 6,000 bits of information on a silicone chip ¼-inch square were considered several years ago to be the ultimate in this technology, we are advised that within the next several years a chip may hold 100,000 bits in this same ¼-inch square. This technology requires the use of high order languages which has increased the communications problem between the project/program manager, his contracting officer and the engineering staff.

If you look at the requirements for the software acquisition, there are several areas which must be considered in order to insure a successful software system acquisition. First, the detailed specifications for functional output versus computer hardware are extremely important. The specifications must be tight and deal with the issues of performance, reliability, availability and the cost of computer maintenance and down time as well as detailed documentation of form, fit and function. A systems analyst is required for the systems design of the functional specifications and, to a lesser degree, the hardware for the computer system. Lastly, the documentation supporting the software support system must be of outstanding quality. There is a need for a single architect responsible for organizing, controlling and managing test data as well as the architectural design of the system. Therefore, personnel with advanced education in computer sciences within the program manager organization are essential. We do not know yet how to design an error free system but by rigorous configuration system management, we should be able to avoid an overly costly software system whether it is being developed as an entity in itself or in support of a weapon systems acquisition.

The design and fielding of software systems for a weapons acquisition is a long and complex task. Considering the time scale of developing any software system, about one-third of the time is devoted to system design and functional specification development, 16 percent devoted to coding, 25 percent to individual program testing and 25 percent to total system testing. You can easily see how the opportunity for change will exist during the development and testing process without design control and vigorous configuration management. When you couple this effort with the need to recruit and train highly skilled E-5 and E-6 technicians, a lead time of 8 to 10 years becomes a reality and adds to the acquisition manager challenge.

This issue of the Review is dedicated to addressing this complex problem and will, hopefully, stimulate innovative thought and near term solutions.

R. G. FREEMAN III
Rear Admiral, USN
Commandant

78 11 15 025

DEFENSE SYSTEMS MANAGEMENT

REVIEW

VOL I, NO 6. SUMMER 1978

CONTENTS:

FEDERAL GOVERNMENT SOFTWARE CONVERSION

by

Dr. Paul Oliver, Director
Federal COBOL Compiler Testing Service, USN

ACCESSION for
NTIS
DDC
UNANNOUNCED
JUSTIFICATION
BY
DISTRIBUTION/AVAILABILITY CODES
Dist. AVAIL. and/or SPECIAL
A

Research and development efforts are under way at several universities and research laboratories to determine ways and means of producing portable software, that is, software which is machine and configuration independent over a set of computer installations.[1,2] Until such efforts bear practical fruits, data processing organizations will periodically be faced with the prospect of a software conversion effort. Such an effort is invariably faced with distaste and apprehension. Acceptable means of easing the difficulties of conversion are given by the author in this down-to-earth, practical approach offered for solving an enormous problem.

A PROBLEM—AN ANSWER

A General Accounting Office report[3] published in September 1977, states that the estimated annual Federal Government cost of modifying computer programs is more than $450 million. The modifications are necessary to enable the programs to execute correctly on a computer different from that for which the programs were devised. Comparable industry-wide figures are not available but the assumption is that the overall industry cost of software conversion is enormous. This is a nonproductive cost that does not result in direct improvement in an organization's ability to fulfill its mission.

There are several reasons why system conversion is a disruptive process. First, programmers must be shifted from regular assignments to the conversion task. This is true whether or not an outside contractor assists in the conversion. Proper conversion requires documentation, and old documentation is often found to be inadequate, even in well-managed installations. Aggravating this condition is the fact that the programmers who originally coded the system are frequently no longer with the organization. Finally, conversion often takes place in conjunction with the implementation of a new system, adding to the concomitant disruption.

The only way to ameliorate the difficulty of conversion is to develop a thorough and detailed conversion plan. Such a plan will consist of three basic stages: Preparation, Production, and Implementation.

To set the scope of this article the following definitions are provided.

Conversion: By conversion is meant any change made to a program or system of programs solely for the purpose of enabling such a program or system to execute correctly on a computer different from the one for which it was devised.

Translation refers to a largely automated process of conversion in which the original programs serve as specifications for the new programs to be produced.

Recoding is similar to translation except that the process is largely manual.

Reprogramming refers to a conversion that may entail a system redesign (e.g., batch to on-line) but without significant functional redesign.

Redesign refers to a conversion effort that involves functional redesign and is therefore akin to new development.

Each of the conversion techniques implied by these definitions entails different tools, methodologies, and management guidelines. While this article is directly concerned only with translation, much of the material applies to recoding or reprogramming.

THE FEDERAL CONVERSION SUPPORT CENTER

System conversion is, fundamentally, a problem because there are differences among computer hardware systems, operating systems, and programming languages; and because computer programs and associated files reflect these differences. System conversion is a problem requiring serious attention and the process of conversion is an expensive one. The eventual solution to this problem must entail more and better standards, adherence to those standards, transparency in hardware differences, and improved programming practices. At present, a Federal Conversion Support Center (FCSC) is being planned by the General Services Administration. Such a center could make the problem manageable by introducing consistent procedures for sizing, organizing, and performing system conversions. *A well-planned and organized conversion, based on previous experience and utilizing tools and organization suitable to the task will not be cost-free, but will cost substantially less than a conversion lacking such planning and organization.*

The services to be offered by the Federal Conversion Support Center fall into the areas of analysis and solicitation.

The analysis preceding a conversion includes a definition of the current system, requirements specifications, selection of an approach to conversion, and the development of a conversion plan. Through an assigned project officer, the Federal Conversion Support Center would provide guidelines and procedures, and consulting services in the following phases of preconversion analysis.

Inventory of system components. The inventory of system components will provide documented descriptions of current hardware and software used by the systems to be converted; and, data describing all systems being considered for replacement, this data to comprise program, file, and record descriptions.

Requirements specification. The requirements specification will provide functional requirements identifying features of the current system which the converted system *will not* require, or features the current system does not have that the converted system *will* require; and performance requirements that must be met by the converted systems (to include memory requirements, program processing times, operator intervention procedures, data storage media, maintainability, and ease of modification).

Selection of a conversion approach. Given an inventory of system components, system statistics, and requirements specifications, uniform guidelines and procedures are required to select a conversion approach that will result in a converted system meeting the specified requirements while optimizing costs and time. The approach selection includes a decision as to who will perform the conversion (contractor, government, hardware vendor, combination of these) and how the conversion will be done. The latter may include transformation of current systems, new design and development, partially automated means, mainly manual means, etc. If the approach selected involves contractor participation, the FCSC would provide the selection and contracting support described below. The costs of conversion, implementation, and operation of the current systems for each of the contending approaches must be estimated using consistent formulas, parameters, and procedures. The potential impact of approach and costs on hardware selection must be determined.

Development of the conversion plan. Products of the development plan include inventory development, to include management reports, specifications, programs, data, documentation, and training materials; a work plan describing the conversion phases, tasks, and schedules; an organization structure; and a budget for project plans and teams.

The FCSC project officer would work with a "technical analysis board" consisting of user personnel and additional Federal Conversion Support Center personnel as required.

Solicitation and selection of contractor support may be desired/required in the performance of all or part of a conversion. The Federal Conversion Support Center would assign a project officer to this task. With the assistance of a selection board composed of user as well as Center personnel, the project officer would be responsible for the following tasks leading to the selection of a contractor under a Delegation of Procurement Authority from the General Services Administration for conversion contracting:

- Preparing a schedule for the selection and a budget for any necessary travel, computer time (e.g., for a benchmark development), etc.
- Preparing letter of interest giving a synopsis of requirements.
- Preparing the Solicitation Document.
- Preparing necessary publicity releases (e.g., Commerce Business Daily announcements).
- Contractor briefings as appropriate.
- Validating and evaluating proposals according to predetermined criteria and procedures.
- Participating, with the Contracting Officer, in negotiations.
- Preparing a report for the agency acquiring contractor support. This report would cover the purpose, authority, scope, findings and recommendations of the selection board.

CONVERSION PROJECT OVERVIEW

Some of the technical aspects of a conversion that apply whether or not a contractor is used for the conversion are given here.

Preparation

The first step is the requirements analysis. A review of the planned differences among existing systems and the converted system is particularly important if the language dialect being converted to has new language modules or major changes to input-output modules. It will be necessary to identify the degree to which the compiler being used differs, in its implementation of the language, from the standard specifications for that language.

Tasks, schedules, resource requirements, and end products must be identified. Schedules and resource requirements are particularly difficult to gauge. It is generally a good idea to obtain contractor support in preparing time and resources estimates, since broad experience in conversion is required in making such estimates. The review of existing programs may reveal some programs that will not require conversion. It is doubtful that there will be many of these. Copies of the remaining programs must be collected together with accompanying files and documentation and placed in the hands of the conversion group.

Finally, the specifications of system changes must be defined. Data file changes may be required. File and record sizes, field contents, file organization, access keys, sort keys, access methods, storage media, and labeling conventions are all likely candidates for change. The conversion will present an opportunity for: needed system restructuring; identification of programs to be combined, elimination of intermediate files, and sort/merges which may be deleted if the restructuring involves a shift from tape to direct access storage device (DASD) residence for certain files. Processing logic changes which are necessitated by differing language dialects must be specified with care.

Software tools must be identified and developed. Software must be available to load data, copy programs, convert programs, create extracted versions of test data, perform data and file conversions, compare test results for validity, and measure tests for reliability. Procedures must be developed and controls and quality assurance standards must be specified.

Programs must be collected in a uniform way to ensure that the correct version (release) on every program is being converted. The software indicated above may be used to create test libraries and to control this step. A procedure for maintenance change inclusion must be developed. Thus, a reference base for changes is established.

Adequate test data must be prepared that will exercise an acceptable portion of the converted

programs. Only a test ensures working logic. Even a part of a program that is tested can have bugs. Given adequate input test data, the programs must be run with this input to create output data. Each program should have a set of known inputs that produce known outputs in order to validate each program. Ideally, unit test data can be used for system testing.

Finally, all related materials must be collected. This material includes program and system documentation (flowcharts, narratives, run-books, data layouts), inventories of files and programs, source listings, program assemblies (listings), and a directory of every item's physical location.

Production

As subsystems become ready for actual conversion the translation process begins. This is true even if eventually systems are to be modified. The conventional wisdom is that program translation (i.e., a one-for-one, or close to it, conversion) should precede any modification. Translation is done to avoid intermingling and compounding any translation errors or effects with modification errors or effects. The success of the conversion will be closely related to the adequacy of the controls which are applied during the production stage. Controls must be established for the receipt, handling, and distribution of all materials, for the copying and analysis (to ensure the correctness of the copy process) of program tapes; and for the definition and use of job control language programs.

The translation process itself will be, in part, automated. Many features of programming languages lend themselves to automatic translation through use of commercially available or in-house developed utilities. A large portion of the input-output coding will have to be hand-translated. In some cases the process will be closer to modification than translation.

Thorough unit and system tests should follow the translation phase (and will have to be repeated after the modification phase, if any). It is generally advisable to desk-trace the programs in a gross way, i.e., through job control, housekeeping, and initial input statements. A monitor that intercepts and analyzes abnormal terminations would be a useful tool in testing. The monitor should be capable of displaying the instruction causing the abnormal termination, the data being processed at that time, and of providing snapshots of selected data/program areas. Also useful would be a file-compare utility to determine the validity of the outputs produced by the translated program and a monitor that could recognize units of untested code. Once a translated system has been successfully tested, any required modifications can take place. These modifications may include system restructuring (combination of common subroutines, sort/merge utilities, etc.), changes in logic, and changes in data files.

The entire process must be thoroughly and carefully documented. The precise form of the documentation will depend on the installation's standards, but should include at least the following:

Converted source programs

Flowcharts of the converted systems

Listing of all job control language programs used

Standard file labels

File conversion parameters

Operating instructions and technical notes

Unit and system test reports

A step-by-step summary of how a production team would perform the translation of programs follows. The procedure is used by several contractors.

a. Materials are received by the production team and processed by a Control Section. Each tape is analyzed to ensure readability and is copied to backup tapes. Test data are converted to target machine format. Standard job control code is generated. Task estimates and a schedule for the program translation are created by a resource management system.

b. The source program is converted to the target language by a multistep process through use of appropriate software tools. First the source code is converted to an intermediate language to permit standardized analysis and manipulation. The eventual restructured intermediate language program is then converted to properly formatted target language code. The target

program listings and other documentation are collected and given to the project manager, who assigns the program to an analyst for completion of the documentation.

c. Corrections are made to the target program. The program is compiled until all diagnostics are resolved. Two programmers then desk check every line to verify logical equivalence to the source program.

d. Testing begins with the aid of a cross-reference program (a tool to trap processing exceptions and allow continued testing) and a file compare program to verify output data equivalence. Unexecuted code is identified and desk checked.

e. Unreferenced code is located and identified. Old data and procedure names are replaced by installation-specified new names. The source code is formatted to installation standards to ensure the uniform appearance of all programs.

f. When the program is completely finished, system enhancements are applied. Maintenance changes are identified and implemented. Parallel testing is performed to validate system equivalence.

g. The completed programs are returned to the project manager for a thorough quality control check. The material then is processed by a control section and prepared for shipment to the implementation group. Backup copies of programs are stored on tape, along with microfilmed copies of listings for future reference. The completed programs and documentation are shipped to the organization being serviced.

Implementation

Implementation of the converted system consists of installation and training. Installation (that is, use in a production environment) should not take place until the systems software to be used has reached a satisfactory level of stability (this will be a subjective decision). The conversion manager should anticipate changes (upward) in the resources required to compile and execute the converted programs, and should allow for these changes to avoid serious degradation in throughput.

Unit and integration testing should be repeated on converted programs and test data once these are installed on the target system. After this testing is satisfactorily completed, maintenance changes are applied and tested, problems are corrected, and retesting is performed. The process is repeated until all tests are successful. Note that this inclusion of maintenance changes entails the generation and conversion of production test data to be used in the testing process. Note also that further changes must not be applied to the production programs at this time.

In the meantime, the production data base is converted and tested, and the operating system control language production stream is generated. Finally, production testing using the final version of programs, data, and control language programs takes place (acceptance testing), followed by an appropriate period of parallel testing.

Training

Training is not a separate phase of conversion, but any organization contemplating a conversion should make use of a comprehensive training program as a part of the preparation for conversion. The specific content of such a training program will vary with the organization, but the following topics should be included:

- Basic Conversion Concepts

  Basic concepts and terminology of conversion.

  Comparison of various terminologies describing similar conversion aspects.

  Industry recognized generalized conversion approaches, including translation of existing programs, restructuring of translated application systems, rewriting application systems on the target configurations, implementation of existing software from other systems, emulation, and simulation.

- Conversion Planning and Decision Making

Time/cost factors which effect conversion including programming languages, availability of resources, program size, program complexity, input/output and data conversion, operational considerations, number of programs to convert, and conversion experience.

Decision criteria used to choose a conversion approach or combination of approaches including cost/time/resource considerations.

Available algorithms to estimate conversion time/costs.

Overall project management including project planning, initiation, control, and review.

Design and utilization of resource management systems for conversion projects.

- Conversion Tasks

Detailed tasks include those in feasibility analysis, data and program preparation, production, implementation of the converted software, and postimplementation.

Use of automated tools for all phases of conversion, including test data preparation, test data efficiency verification, translation, text editing, generation of cross reference maps, job control language production, data file comparisons, source formatting, debugging and testing.

Establishment of quality control procedures and preparation of forms for each major task and conversion type.

Establishment of technical standards to be maintained throughout a conversion project and the determination of mechanisms to be used to enforce these standards.

Data file conversion using automated techniques.

Factors that effect data base conversions include the size of the data base, the complexity of the data structures, the time required to reference and retrieve data vs the time required to update or establish the data base, the particular data manipulation languages used by the source and target systems, test data generation, and implementation and testing considerations.

Conversion of job control procedures.

- Conversion Management

Required staffing to manage small and large scale conversion projects.

Cost/schedule performance reporting and management control mechanisms.

Documentation requirements (throughout the conversion) and the recommended documentation forms for each major task.

Specific completion/acceptance criteria for each major task in the conversion.

- Specific Problem Areas

A second to third generation conversion should be demonstrated, including assembler to high level language conversion, following one example throughout all of the conversion steps from preparation through implementation.

A third to third generation conversion should be demonstrated, including one assembler to assembler conversion, one assembler to COBOL 74 conversion, one COBOL 68 to COBOL 74 conversion, and one FORTRAN to FORTRAN conversion. At least one example of each type should be followed through all of the conversion steps from preparation through implementation, and at least one example of each must be included as a workshop exercise.

Conversion problems with vendor extensions. Specific examples should be given for IBM, Control Data Corporation, UNIVAC, Burroughs, Honeywell, and Digital Equipment Company.

Problems with vendor unique data formats.

Special conversion problems involving transaction oriented systems, networks, communications systems, real-time systems, and hybrid configurations.

Software Tools

The description of the conversion phases included several references to software tools. A complete inventory of such tools is expensive, and would be one of the factors that must be taken into consideration when contemplating an in-house conversion. The inventory would not be of great value after the conversion.

A software support inventory can be characterized several ways. The simplest way is according to the three principal conversion stages in which the support inventory is used:

Preparation: A file contents analyzer, a data extractor and modifier, and a data generator can be used for creating test data. Utilities will be required for creating backups of all programs, for maintaining a current version of the software inventory to be converted, and for producing statistics (for example, average program sizes) as required.

Production: The production stage will require software to perform source code to intermediate code translation, to analyze and restructure the intermediate code, to perform intermediate code to target code translation, and to translate test data files. Utilities will be needed to generate the operating system control stream and to apply code corrections to translated code. Additionally, software to produce cross-reference listings, to trap and identify exceptions, to identify unexecuted code, and to perform file comparisons will be needed for testing. A decompiler or depatcher will be needed if the conversion is from an assembler language to a higher level language.

Implementation: Software to validate the results of parallel testing, to identify and implement maintenance changes, and to convert the production data base is required.

Software aids are useful in the management of the conversion project. As a minimum, software tools are needed for resource management (to identify programs and categorize programs by source and content, to estimate resource requirements, and to monitor progress) and for standards enforcement (to format programs to installation standards, replace old names with standard names, etc.).

A DETOUR—EMULATION

Emulation is a process to enable one computer to execute programs written for another computer. Portions of the "emulator" consist of hardware features, namely, microprogrammed circuits to perform the execution of the emulated computer's instruction set. Other portions consist of software features for input/output. Often certain constraints may have to be imposed on the storage and input/output device requirements. Emulation is not really a conversion technique but rather a conversion postponement technique. As such, it is not generally recommended by this writer. There may be, however, instances where emulation may make some sense for some period of time. The following factors should be considered in deciding whether or not emulation is desirable:

Initial considerations. Cost of the emulation packages—hardware and software; the remaining life of the programs to be emulated; the frequency and duration of programs to be run in an emulated mode; costs of file conversion.

Advantages and disadvantages. A new system can be installed prior to reprogramming. Emulation is a useful transition aid and in some cases may obviate the need for conversion. The process allows for greater smoothing of manpower utilization in conversion but may result in inefficient use of a new system, and it can encourage old (undesirable) habits (languages, operating systems).

Cost effectiveness. Need ratio of old processing time to processing time on new computer in emulation mode—then later to new computer in native mode; job mix greatly affects cost effectiveness (input/output problems).

Reprogramming. Cost: projected lifetime of programs; operating time saved by reprogramming.

Cost and limitations of emulators. Costs range from $100 to $1000 per month; translate only about 75 percent of code; input/output is usually a problem; partial translation is confusing; performance is relatively poor (up to 50 percent degradation in central processing unit time, and a 40 percent degradation in memory utilization).

MANAGING THE CONVERSION PROJECT

A conversion project is only slightly different from any other software production project with respect to management. Careful planning is required and, once initiated, the project must be controlled. Finally, there is a completion phase. If there is a significant difference between conversion project management and production project management it is one of emphasis. A conversion project requires (and allows for) more discipline and stricter adherence to procedures. If properly executed, a conversion is very much an assembly-line type of operation. The total effort is broken down into well-defined tasks which are more dependent on experience and strict adherence to procedures than on innovation and ingenuity for successful completion. This is true partly because of the high degree to which the conversion process can be automated. Also note that many of the ground rules for software production do not apply to conversion. Example: Manpower and time are not generally interchangeable in a software production project but, to a degree, are interchangeable in a conversion project.

The first step in the management of a conversion project is to determine the constraints. Constraints may have to be applied to the project organization, the technical approach, the schedule, and the resources available. Further, these constraints are not independent of each other. Limitations on resources for example, will impose limitations on the schedule, the organization, and the products to be produced. After the constraints have been identified the inventory must be identified and catalogued. Where program or file characteristics cannot be precisely identified assumptions regarding these must be made, justified, and documented. Specific tasks must be identified, together with the interdependencies. Task performance schedules and resource estimates for each phase and task are prepared following the identification of tasks. Finally, personnel assignments must be made, lines of communication must be established, and management concurrence for the conversion plan must be obtained. A common mistake is to make the conversion staff a part-time group that participates in conversion activities but whose members continue to report to a parent organization. This is an ingenious way to make a mess of the conversion, particularly since it will be nearly impossible to attribute responsibility for the mess to any one person.

Once the project plan has been approved the project is officially entered in the organization's project management system. The project staff is assigned and relocated as required, members of the staff are briefed on the background and purpose of the project, assignments and schedule, and standards. A crucial step at this time, and one which is often overlooked, is coordination with support organizations. These organizations will include computer operations, system software support, user organizations, quality control, space and facilities, and, if applicable, the contracting staff. The services required by the conversion staff must be documented and must be discussed with each of these groups. Any required training of the staff should be initiated at this point.

The control function of project management begins after project initiation. Project control includes product quality control, progress measurement on a task-by-task basis and the maintenance of project files. It is also important that contingency plans be devised in case it becomes necessary to make changes to the technical approach, the schedule, or the resources budget. The mechanism for communicating the project status to the project team, the users, and management must be established as part of the control function.

Lastly, the project management plan should include procedures for orderly project completion. These will include procedures for acceptance testing, identification of successes and failures for future reference, and for personnel performance review. It is important that all project personnel realize that the conversion project is not a momentary diversion from a regular job but rather, for its duration, *is* the job.

Staffing

The specific makeup and size of the conversion staff will vary with the conversion type and magnitude. The following membership, without specific quantities, is suggested for a large-scale project requiring contractor support. Following each staff category is an indication of the role of that category:

Project Leader (Planning—Project initiation—Project control—Project termination)

Contracting—staff support advising on (Type of contract—Necessary clearances—Terms and conditions—Scheduling)

Operations (Present status and future needs—Performance specifications—Scheduling—Inventory—Computer resources)

Systems Programming (Inventory—Sizing of job—Performance specifications)

Application Systems Developers (Inventory—Sizing—Performance specifications)

Support Programming—software tools for (Inventory—Quality control—Production—Testing)

Material Control (Quality assurance—Backup inventory—Materials transmittal)

Clerical—supports entire team

Analysis and Programming (Production—Testing—Implementation)

Figures 1 and 2 suggest the organizational relation of the staff components. A list of the major tasks to be performed is shown in the checklist.

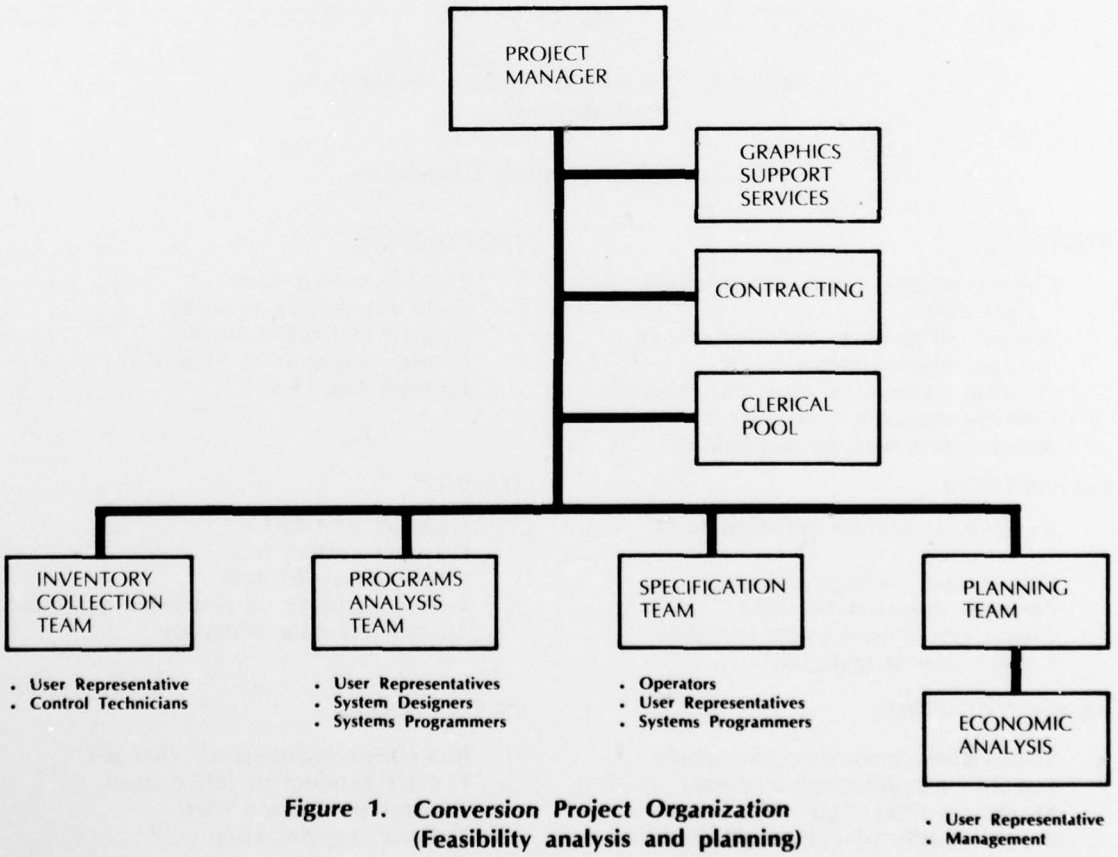


Figure 1. *Conversion Project Organization* (Feasibility analysis and planning)

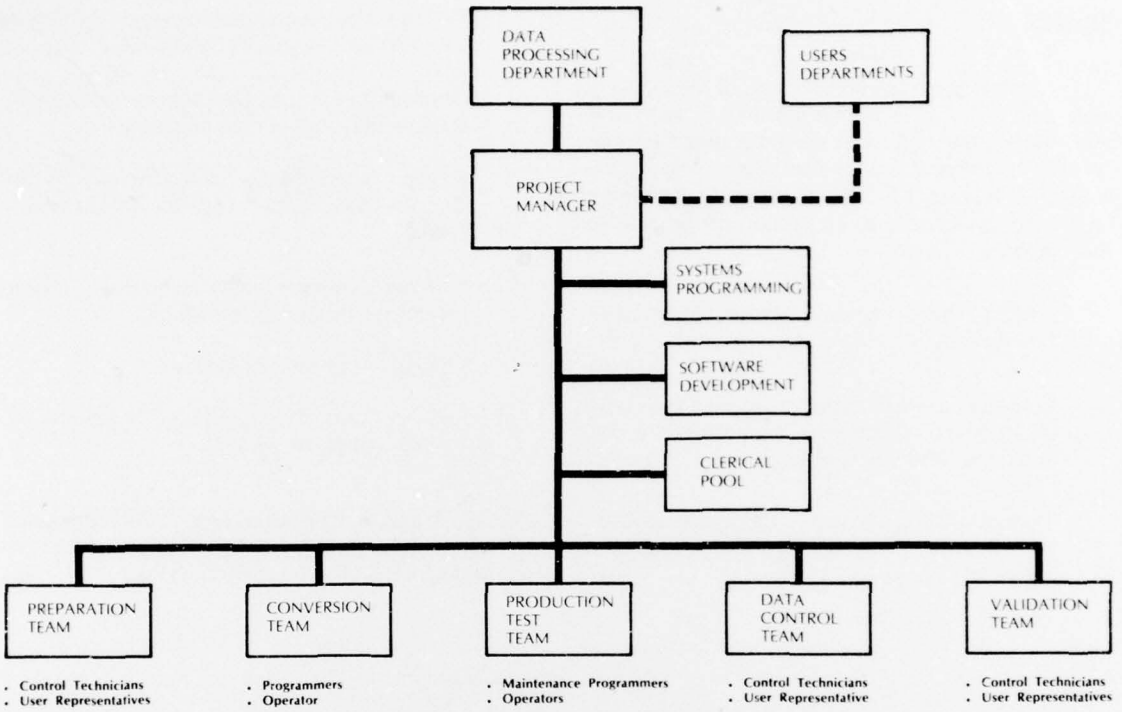


Figure 2. Conversion Project Organization (Post-planning)

System Conversion Checklist

PLANNING

- Collect inventory and define scope of work
- Analyze differences between source and target systems
- Develop conversion plan and schedule
- Estimate resource requirements
- Assign conversion responsibilities

PREPARATION

- Develop or acquire software tools required
- Collect and package programs
- Prepare adequate test data
- Create test output using test data
- Collect related materials

PROJECT CONTROL

- Standardize conversion procedures
- Establish reporting requirements
- Monitor project status
- Assure quality of converted programs

PRODUCTION

- Convert source code
- Code corrections required
- Prepare test job control
- Format programs to standard
- Convert test files

TESTING

- Conduct unit test
- Conduct system test
- Conduct parallel test
- Ensure accuracy of converted programs
- Ensure test data adequacy

INSTALLATION

- Implement maintenance changes
- Prepare production job control
- Convert production files
- Cutover to production

MANAGEMENT PROBLEMS OF CONVERSION

A conversion project results in a large volume of material and serious control problems. Consider for example a conversion inventory of 1000 programs, each using three unique data files. This totals 4000 files that must be controlled, together with 1000 sets of documentation plus 1000 full and 3000 partial listings. Twice these numbers must be generated and stored to provide adequate safeguards. If 20 runs per program are assumed to convert the program there are 20,000 jobs to schedule, run, analyze, and control. Program changes made during the conversion will add to this volume.

Staffing required for the conversion project will not be needed at the culmination of conversion and will be diverted from on-going development and maintenance. Further, there will be periods of peak requirements. These conditions create serious management problems.

Machine time also will encounter periods of peak loads. This, unfortunately, conflicts with growing production owing to the cutover of subsystems. Machine time availability takes on an inverse relation to conversion requirements. In many cases, conversion requirements plus production requirements become more than the total of machine availability. This causes costly delays in the conversion schedule.

Low resources requirements estimates are caused by a lack of understanding of the conversion process. An estimator may take a few programs, convert them, and attempt to extrapolate the requirements for the entire job. This projection is inaccurate because it represents a straight line relationship (which never really exists) between conversion volume and resource requirements. This mistake is the one that has plagued production time estimators for years. As in software production, size has a special effect on conversion time and costs that render a linear relationship invalid. For example, a competent programmer, properly supported by a set of software tools, can convert 500 lines of COBOL source code per day. This suggests that a 10,000 line program could be converted in 20 days. Experience shows that some 36 days would in fact be required, and that this number could go up to 104 days if the conversion is from an assembler language with complex file structures and large file volumes. Another problem in estimating is the tendency to exaggerate the extremes by making simple tasks appear too simple, and difficult ones too difficult.

References 4 and 5 give an overview of the technical problems of conversion, together with some suggested solutions.

CONTRACTING FOR CONVERSION SERVICES

Success in conversion is in large measure dependent upon experience, and few if any data processing organizations possess this experience—conversion is not an ongoing enterprise. The experience required is not of the "we have three professionals who have gone through a couple of conversions" type. Experience, to be useful, must involve an entire team that has performed enough conversions to become proficient in the techniques and tools to be used—a team predictable in its productivity and performance quality. Such experience is most apt to be found in software services or conversion services contractors. Generally, it would be wise for organizations to avail themselves of contractor support for conversion. Contrary to popular opinion, the software producers are *not* the best qualified people to convert the same software. If anything, they are the worst qualified, since they probably cannot resist the temptation to "improve" the system while converting it.

Contracting methods vary greatly. The best, most sound, contracting is probably done by the US Government under its Federal Procurement Regulations. Also useful guidelines can be obtained from Reference 7.

PROCUREMENT STEPS

The first step in a procurement is the same as the first step in the conversion planning, a requirements analysis. Once this is done, the contracting personnel can begin to plan for the type of contract which may be required by the specifications, the time required to prepare a request for proposals (RFP) for potential contractors, the time required for negotiations and evaluation of proposal, and the time required to make an award. The availability of funds also must now be determined.

The technical portions of a request for proposal are prepared by technical personnel on the

project staff. The contracting personnel work on the terms and conditions. This is not to suggest a strict separation of duties. Quite the contrary, the two groups must work cooperatively and harmoniously, since what each does has impact on the work of the other. Some 12 to 24 weeks should be allocated to the accumulation of data to be used in preparation of the Request for Proposal. Four weeks should be allocated to the assembly of information and the final preparation of the Request for Proposal. Four weeks should generally be given to prospective offerors so that they might prepare technical responses, with 2 to 4 additional weeks allowed for preparation of final cost proposals. The evaluation of proposals (technical and cost proposals should be evaluated separately, and the technical team should *not* see the cost proposals) should not take more than from 2 to 4 weeks. Up to 2 weeks should be allowed for higher management review of the award recommendation and final contract award. Thus, the procurement may take from 22 to 38 weeks.

The Procurement Request

The technical staff can ease the burden of the contracting staff by preparing a complete procurement request (PR). This will include a thorough statement of the scope of work (supplies and services, etc.), reporting requirements, property or facilities to be provided to the contractor, and a list of prospective sources of support. Upon receipt of the procurement request the contracting officer can assemble a procurement staff. This staff, for large or complex procurements, may include negotiators, a cost/price analyst, an inspector, legal counsel, and auditors.

The contracting officer can begin to give serious consideration to the type of contract suggested by the work to be done. This may range from fixed price contracts, through cost contracts, to time and materials or labor hours contracts. Each of these forms has several variations. At one extreme, a fixed price contract places all the risk on the contractor, but requires very well-defined specifications; at the other extreme a labor hours contract places all the risk on the buyer, and should only be used when well defined specifications are impossible.

In making a determination of contract type, the contracting officer will have to consider the following factors: complexity of the task, urgency, period of performance, competition, difficulty in establishing performance costs, comparative cost data, business risks, administrative costs, nature of the work to be done, and likely technical and financial capabilities of contractors.

Contract Modifications

Contract modifications are difficult to perform, usually cost the buyer additional money, and often are illegal. Excessive use of options is generally indicative of poor thinking. The Government imposes limitations on the use of options (on the part of its agencies) which are designed to ensure a certain level of discipline.

- Options cannot exceed 50 percent of initial quantities.
- Options may be specified as percentages of contract line items, as increases in specific line items, or as additional line items.
- Options cannot be used when an indefinite quantity contract will suffice or when the option represents a known requirement and funding for this requirement exists.

The government also imposes restrictions on the types of contract modifications allowed.

- Change orders are unilateral changes initiated by the contracting officer and are limited to method of shipment or packing, place of delivery, amount of government furnished property, and changes to drawings and designs.
- Supplemental agreements are bilateral changes which must be within the scope of the original contract, that is, must not be substitutes for new procurement.

There is a third category, "extras," but it is not used frequently. The important point is that changes must be within the scope of the contract. This point is obligatory for government agencies; it is good sense for all buyers.

Evaluation methods

The principal contractor evaluation methods are listed below:

- Cost only—pick low bidder among qualified ones. If the specifications are adequate this is not as undesirable a method as it is reputed to be.

- Cost plus "desirables"—the basic bid price is modified up or down according to the quality or number of "nonmandatory" items a contractor proposes. This is a superficial way of giving "bonus points" that has little to recommend it (in this author's opinion).

- Cost evaluation plus technical evaluation—it is common to attempt to combine "cost points" with "technical points" to determine a winner. This type of evaluation is usually motivated by a desire to substitute subjective judgment for a sound evaluation. One legitimate way of incorporating a technical evaluation is to use the results of a technical review to determine a "technical competitive range," then eliminate all bids falling outside this range and make the award to the lowest of the remaining bids.

ESTIMATING CONVERSION COSTS

Conversion costs for an in-house conversion (performed by a staff of programmers from the organization undergoing the conversion) cannot be accurately estimated. The difficulties of estimating production efforts are well known. Software production is something programmers do all the time. Programmers do not do conversion all the time. Furthermore, the in-house staff does not have the tools and procedures required, and if acquired, the in-house staff has little or no previous usage experience.

One can determine reasonable estimates of conversion costs if a contractor is used for the production stage, which is the most costly of the various conversion stages. The tasks associated with planning and implementation can be itemized, resources required for each of these are estimated, and the result is a cost estimate for these portions of the conversion.

Estimates for the production stage can be derived by reviewing past conversion efforts and prices bid on these efforts. The Federal COBOL Compiler Testing Service has compiled a substantial data base of dollar cost and productivity figures. A few figures can give general indication of potential costs.

- COBOL to COBOL conversion will cost from $.40 to $4.00 a line, with $.65 a line being a good average figure for reasonably "clean" COBOL programs that do not require extensive file restructuring or documentation beyond normal program documentation.

- FORTRAN to FORTRAN conversion costs are similar to COBOL to COBOL costs.

- Data on PL/1 to PL/1 conversion is scarce, but indications are that such a conversion would be from 10 to 20 percent costlier than a COBOL to COBOL conversion. Assembler code—COBOL/FORTRAN will cost from $2+ to $8 per source line.

- Productivity figures range from 300 to 500 lines per man-day in FORTRAN to FORTRAN or COBOL to COBOL down to 20 to 100 lines per man-day in an assembler-COBOL/FORTRAN conversion.

- Extensive documentation (user guides, narratives, etc.) can cost 40 percent of the total per line cost. In-house costs for planning, preparation, and implementation can be 50 percent of the total costs.

Several qualitative observations can be made from an analysis of a sizeable conversion data base:

- Conversion costs can vary greatly with the source machine, but are not very dependent upon the target machine.

- Knowledge of the application is not very important in performing the conversion; but information regarding the application must be available as required.

- Competition, or lack of it, can influence offerors' prices by as much as 100 percent. This author knows of at least one instance where limited competition led to an inflation rate of nearly 300 percent in one contractor's price. A thorough cost

and price analysis on the part of the contracting office could have avoided this.

- Known complexity will not unduly influence costs, but complexity that comes as a surprise can cause havoc.
- None of the above holds for real-time systems.
- Conversion of data base systems is not well understood.
- Conversion estimates should not be attempted by people who do not have access to a sizeable data base of information and who do not do this on a regular basis.
- Cost models consisting of a handful of formulas based on a handful of parameters are worse than useless; they are dangerous because the impression of authority is given.

OBSERVATIONS

Conversion is a disruptive, largely nonproductive process that must nevertheless be faced at some point in the life of a data processing organization. Some ideas have been presented that may be of assistance to those contemplating a conversion. The most important facts are that conversion requires careful planning, and should not be attempted without the services of professionals.

CITED REFERENCES

1. **D. E. Whitten and P. A. D. deMaine, "A Machine and Configuration Independent FORTRAN: Portable FORTRAN (PFORTRAN)," Computer Science Department, Pennsylvania State University, University Park, PA, 1974.**
2. **G. N. Baird and L. A. Johnson, "System for Efficient Program Portability," Proceedings, National Computer Conference, American Federation of Information Processing Societies Inc. Press, Montvale, NJ, 1974.**
3. **General Accounting Office, "General Accounting Office Report to the Congress" FGMSD-77-34, Washington, DC, 14 Sep 77.**
4. **Joel E. Heiss, et al, "Programming for Transferability," technical report, International Computer Systems, Inc., Los Angeles, CA, 1972.**
5. **Paul Oliver, "Transferability of FORTRAN Benchmarks," ADAO39741, Federal Operations Directorate, Dept of Navy, Washington, DC, 1977.**
6. **US General Services Administration, "Public Contracts and Property Management," Code of Federal Regulation, Title 41, Subtitle A, Chapter 1, Federal Procurement Regulation, 1977.**
7. **D. H. Brandon and Sidney Segelstein, *Data Processing Contracts*, Van Nostrand Reinhold Co., New York, 1976.**

Paul Oliver is Director of the Federal Operations Directorate for the Department of the Navy. Included in this Directorate are the Federal Conversion Support Center and the Federal COBOL Compiler Testing Service.

From 1962 to 1965 Dr. Oliver served in the US Air Force with the Air Force Systems Command, Research and Technology Division at Wright-Patterson Air Force Base, Ohio. In this period he worked in the Bionics Branch, the Avionics Laboratory Data Reduction Division, and the Digital Computation Division. After his discharge from the Air Force as a First Lieutenant in 1965, Dr. Oliver remained with the Digital Computation Division as a civilian employee. He also taught in the University of Dayton Mathematics Department. Dr. Oliver joined the UNIVAC Division of Sperry Rand Corporation, Washington, DC, in 1970 as a Staff Scientist. In 1973 he returned to Federal Government service as Director of the Software Development Division, Department of the Navy, ADPE Selection Office.

Dr. Oliver is a member of the Association for Computing Machinery and the Institute of Electrical and Electronic Engineers; he is listed in Who's Who in Computing and American Men of Science. He is on the faculty of the American University.

Paul Oliver received the B.S. degree (high honors) in Mathematics from the University of Maryland in 1962; the M.S. degree in Mathematics from the Ohio State University in 1964; and the Ph.D. degree in Computer and Information Science from the University of North Carolina at Chapel Hill in 1969.

REDUCING SOFTWARE MANAGEMENT RISKS

by

Dr. Ruth M. Davis
Deputy Under Secretary of Defense
for Research and Advanced Technology

Software correctness remains the most elusive goal of computer science. As a result, software is the most unsafe, the least understood, and the most expensive component of total computer system costs. In contrast, costs of computer circuitry have shown a dramatic decrease, especially in the past 15 years, and computer hardware capability has improved.

The author presents some objectives here for better and less risky software. Changes in product control areas are suggested to improve software products and control software expense.

THE RISKS OF TECHNOLOGY

One of the problems with which a civilized society has the greatest trouble is that of dealing with *risk* or danger. The more primitive have the least trouble. Direct confrontation with the enemy in front of your cave, physical destruction of a city or an artifact—such as a statue or a machine—are very tangible satisfying ways of handling a threat or a fear. Some of us still handle our problems thusly: most of us do not.

Science and technology provide us with one of our greatest dilemmas: as a society we admire, respect and envy scientists and science. But the misused products of science and technology confuse us. If we dislike or fear these products, there are very few civilized means at our disposal for dealing with them. Also, mankind is not adept at anticipating technologically-induced risks. Consider a few.

The automobile generated a new era of crime. The criminal from outside the neighborhood, even outside the town, was born along with the quick getaway. The automobile kills nearly 50,000 people a year on highways. Yet, we have individually and collectively assessed the risks posed by automobiles in terms of danger to our lives and property through accident and crime. We drive automobiles daily and require a minimum of licensing for ownership and control. The advantages to us of this means of transportation obviously outweigh the disadvantages.

The telephone line is a superb mechanism for wire-tapping and eavesdropping. The telephone receiver itself is an excellent holder of passive eavesdropping devices in every home—undetected except by physical search. Here again, we have individually and collectively decided to have phones and phone lines—in fact some 161 million telephones exist just in the United States. It is apparent that we want phones more than we fear them.

The scenario gets a bit less determinate as we view other products and unwanted by-products of science and technology. Perhaps, the muddled appearance is because today in 1978, we are closer to the beginning and have not had as much familiarity with products of technology such as nuclear power, computers, and manipulation of the protein molecule to yield seemingly resistant-perfect viruses.

RISKS ASSOCIATED WITH COMPUTERS

If we concentrate just on computers, we see a collection of strong beliefs, genuine interest, free-floating anxieties and highly-articulated concerns.

Not surprisingly, the two principal problems facing managers relying on computers are:

- What risks do they face in using computers, and
- What risks do they face by *not* using computers.

Typically today, most managers are totally unprepared to answer these basic questions. Even the concept of "risk" and its companion concept of "safety" as applied to computer systems is foreign to the computer manager and scientist.

I would suggest that the fundamental questions that managers need to address to evaluate the risks or safety of computer systems are, do I know:

- When my computer system is not performing its intended function? or
- When my computer system is performing a function which was *not* intended

When a computer manager cannot answer these questions in the affirmative then that manager is unable to determine if his organization is taking unacceptable risks in using computers in its operations.

In attempting to manage the risks and determine the safety of computer systems, a definition of safety can be developed (as suggested by William Lowrance in his book *Of Acceptable Risk*) namely:

- A computer system is safe if its risks are judged to be acceptable, where *risk* is the probability of loss or damage due to the occurrence of undesirable events or adverse effects.

Thus, before safety can be judged, many factors about computer systems need to be considered. These factors include the threats, dangers and vulnerabilities, the safeguards, and the alternatives to computers and the attendant costs.

The threats and dangers are well known, for example:

- Natural hazards—fire, water, wind, hail, tornados, earthquakes, radiation, and power outage,
- Errors and omissions of programmers, operators, data input clerks, users and support staff,
- Hardware failures and communications equipment outages,
- Sabotage, strikes, terrorism, disorders, and vandalism,
- Fraud, embezzlement, theft, and
- Interception of information being processed or stored in the system.

The loss due to exposure to possible threats includes:

- Delayed processing of information resulting in increased costs and inability to conduct the required system operations especially those that are near real time (NRT) in nature.
- Loss or unauthorized use of information resulting in harm to whatever operation is dependent upon the computer, and
- Loss of resources or assets such as weapons, personnel, vehicles, targets, signals, information, money, etc.

Figure 1 highlights the essential elements of the safety-risk management process that should be an inherent part of the responsibilities of any computer manager.

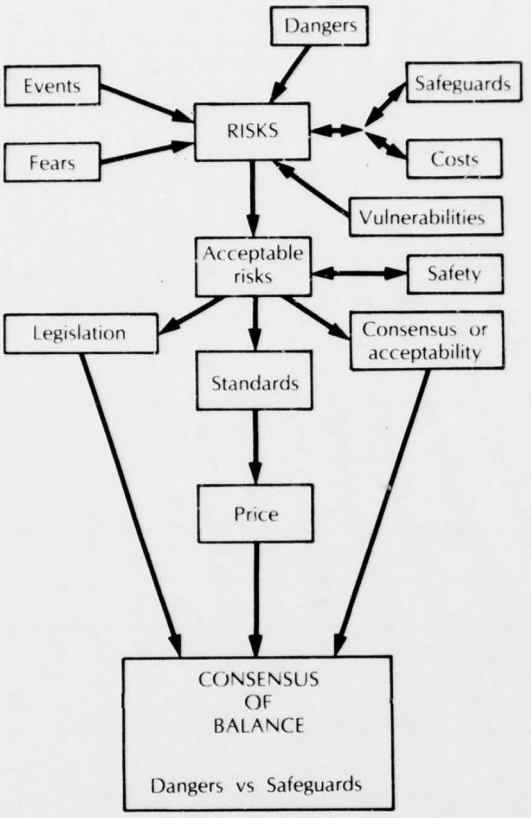


Figure 1. The Safety-Risk Management Process

SOFTWARE THE MOST RISKY COMPONENT OF COMPUTER SYSTEMS

Today software is the component of a computer system to which the most risk is attached. Readers* always evince genuine surprise when told that there is no theoretical (or mathematically rigorous) way to prove programs correct (except for trivial programs containing less than 100 statements). This serious limitation will be difficult to overcome because of the mathematical difficulty of constructing the necessary inductive assertions and the cost of the computer time to generate proofs. The impracticality of exhaustive testing of all program input values can be shown by observing that it would take the "fastest" machine available today more than 30,000 years to try all inputs to a simple multiplication program.

*Ruth M. Davis, "Evolution of Computers and Computing," Science, 195 (4283): 1096–1102 (1977).

The lack of theoretical proof of correctness of computer programs has resulted in the expenditure of considerable intellectual and physical resources in the software field to develop engineering and statistical substitutes. Software engineers and computer scientists have concentrated on quality control techniques for software development, "debugging aids," automatic programming techniques, software validation, and the like, but to little avail. Software correctness remains the most elusive goal of computer science.

As a result, software is the most unsafe, the least understood, and the most expensive component of total computer system costs. Software development costs are now almost 90 percent of total computer system costs. This percentage will probably increase along with the absolute costs of software, since software design, development, and testing is the most highly labor-intensive component of computer system products. The really useful and exciting advances in computing probably will proceed only at the same pace as advances in software engineering. This is distressingly slow.

The software industry, like other spheres of modern endeavor, has entered a period of self-evaluation and redirection. In the past, when software was regarded as an esoteric product, a novelty, or a toy for the technically inclined, a great many excesses were tolerated and forgiven. Software was expensive, unreliable, difficult to understand, debug, and test, and so brittle that even relatively minor perturbations in environment could not be handled satisfactorily. Computers have now become an exceedingly important part of our daily lives. Examples: computers are used by business in accounting, credit, and inventory systems, by health care institutions to diagnose disease and perform blood and tissue typing, by engineers in designing mechanical parts, by the military in deployed weapons systems and by the mass transit industry in handling air traffic control, seat reservations, and routing.

Nonetheless, software quality is uneven at best. Programs thought to be correct (and released for general use) will suddenly produce wrong results, no results, or behave otherwise erratically, because some special condition in the data or in the environment was not accounted for in the logic of the program. Current practice is to design and implement a software system, the paramount considerations usually being production speed, running efficiency, and/or minimal use of storage. The software is then tested for some arbitrary subset of

possible input values and environmental conditions. The system is accepted as correct when it executes the test cases correctly or when time/money runs out, whichever occurs first. It is not surprising that software products are often unreliable and that confidence in new software is generally minimal.

These facts have become increasingly clear:

1. Software (system software and application software) which is reliable, robust, understandable, testable, and maintainable must be built. Society cannot and will not tolerate repeated software failures in systems affecting public health, welfare, and safety.

2. It is worthwhile to sacrifice something in the way of production time, running efficiency, and/or storage required to obtain reliable code. While the details of this tradeoff are, as yet, unknown, it is apparent that software errors are becoming potentially more dangerous and expensive.

3. While software has become increasingly more sophisticated and complex, techniques for producing and ensuring quality in software have not kept pace. Such techniques must now be developed, improved, and integrated into the software production process.

There are four major participants or potential "losers" unless, and until, software reliability and quality is improved so that computer risks can be reduced to acceptable ranges. They are:

The *Customer*, who formulates automated data processing requirements, translates these into system specifications, and uses the specifications as the basis for selection and acceptance strategies and as terms in legally binding contracts. The customer's objective is to obtain the best system for the need at the lowest cost.

The *Producer*, who designs and implements software systems meeting customer specifications, and markets packages of own design. The producer's objective is to maximize profits.

The *User*, who operates and maintains software packages in the field, making modifications in response to malfunctions or changing requirements. The user's objective is to utilize the system effectively.

The fourth player, the unwittingly involved *Public*, has been generally passive in the past. He does not know much about the game right now, but since he keeps losing, he can be expected to learn. The objective is self protection and survival.*

Some of the software realities faced by software customers, producers, users and the public in 1978:

• *Software is the most expensive component in systems procurement.*

Technological developments, especially in the past 15 years, have dramatically reduced the cost of computer circuitry and increased hardware capability. A microprocessor costing about $20 today has the computing power of a large computer that cost $1 million 20 years ago.

The cost of software has shown the opposite trend; it has risen continuously. The estimated cost of software development, testing and maintenance for the Federal government is $4 billion per year (1977). The government in 1977 owned about $25 billion worth of currently used software.

The development costs for new software are enormous. Internal Revenue Service estimated that its proposed Tax Administration System would cost over $500 million to develop. The Air Force estimates that its software costs for command and control systems for the next decade will be several billion dollars. Software costs greatly exceed equipment costs over the lifetime of a computer service.

Overruns of 100 percent in both cost and the time to develop software have not been unusual occurrences. In fact, there have been cases of total failure to develop systems.

---

*Paper presented by Dr. Ruth M. Davis at the International Conference on Reliable Software, Los Angeles, CA, 21 Apr 75. Material prepared by Dr. R. Stillman, Albrecht Neumann and Dr. Dennis Fife, National Bureau of Standards.

The cost of maintaining software is estimated to account for about 75 percent of software costs. Much of this expense is attributable to time spent in fixing up software that was not correctly developed in the first place.

- ***Software or program operation is the most unreliable component of an operating system.***

In 1976 people throughout the world waited expectantly for a program error in the control system for the Viking Lander to be corrected. The Viking was delayed for several days in scooping up a Mars soil sample that was to be examined for signs of life. The computer issued a series of commands to release a protective cover on a 10-foot extendable sampler arm. The sequence was not properly executed, and the arm became jammed.

The Supplemental Security Income Program of the Social Security Administration had a 23.7 percent error rate in the payments made to 4.3 million people covered by the program. In its first 27 months of operation, the program overpaid clients by $622 million. The General Accounting Office, in reviewing this program, levied blame in part on the complexity of the legislation and last minute changes made by Congress. The General Accounting Office also cited the lack of management controls in the Social Security Administration that resulted in software changes being made frequently without the proper documentation.

Software errors in airline reservations and manufacturing process control systems can lead to expensive and frustrating breakdowns of the systems. In systems such as nuclear plant control, software errors can be disastrous.

- ***Software cannot always be proven correct and software quality controls are virtually nonexistent.***

Computer programs can be tested to determine that the program performs as specified for the selected test inputs. However, that the program performs only the intended results cannot be determined.

Mathematical tests to prove correctness of programs are not feasible for large programs. Exhaustive testing over all program input values is often impossible.

Practical validation for the foreseeable future consists of devising systematic tests over a reasonable subset of possible input values. The only validation services for software presently available are provided by the National Bureau of Standards to test COBOL and FORTRAN compilers purchased by the government for conformance to programming language standards.

- ***The software market is a "buyers beware" market.***

The potential software buyer is unguided and vulnerable in an uncertain and complex marketplace. First of all, he has trouble locating the producers of the type of software that he needs. He has few criteria for comparing software performance or for demonstrating the efficiency of software. He cannot compare the features of different products because of the lack of documentation standards. It is very difficult to be positive that the software he buys will be correct and reliable. He cannot be positive that it will perform 100 percent efficiently. He does not know if it will be easily maintainable and transferable. He does not know exactly how much it will cost to maintain the software.

- ***Software development and maintenance is labor intensive and programmers, being highly skilled, are expensive.***

Programming costs are almost entirely labor costs. Programming is a craft that has emerged from business and government, rather than from science and engineering as is the case for hardware technology. Also, the productivity of software development has not significantly improved because of the lack of automated programming techniques.

Despite this rather gloomy picture, I would conjecture and hope that software will flourish and not be replaced by substitution of hardware. The reasons are that:

- Software is a primary means for creativity in computer usage.
- Software changes provide a best means for incremental improvement of existing computer applications.

- Software extends human abilities to explore scientific, mathematical and logical phenomena and behavior.

- The software marketplace, if drastically changed from that of today, may be an attractive one for innovative entrepreneurs and product advances.

- Software as conceived today may provide more product differentiation for the benefit of customers than any currently available substitute.

The fact that software is a primary means for creativity in computer usage is a familiar one. Accomplishments in application areas such as real-time control, funds disbursement and transfer, information systems, inventory control and "robotics" would not have been possible without creative and innovative uses of software. What would be the state of space exploration, national defense and economic well-being without flexible software? One other area of computer usage should not be overlooked—that of using human creativity in concert with software to produce better tools for the development of reliable software.

Changes to software should be incremental rather than continual. Although the invariant types of software substitutes force this discipline upon the user, traditional software still provides the best and least expensive means to accomplish changes. There is no reason why the same discipline cannot be applied to traditional software. Users of software must resist the temptation to make unnecessary and error prone modifications to software. The simplicity of the physical actions involved in modifying software has led to the mistaken belief that modification of software is easy. Requirements do change over the life of a system. However, these changes usually can be accommodated by well-planned incremental changes to software.

Software extends human abilities to explore scientific, mathematical, and logical phenomena and behavior. Problems can be solved through non-analytic means. For example, the LISP language, which is based on the mathematical principles of recursive function theory, has become widely used for research, most prominently in the area of artificial intelligence. The LISP language facilitates the application of mathematical and logical principles to subjects such as game playing, robots, theorem proving, natural language processing and algebraic manipulation. Additionally, software has made possible advances in areas such as automata theory, modeling of stochastic processes, system simulations and simulations of intelligent behavior.

Software allows a broader base for competition in the marketplace—and all parties profit from this. Manufacturers have a large group of customers to satisfy. Software provides the only economical way to meet the entrenched desire "to have it your way" among computer users. Customers benefit from selectivity and lower prices because of similar product competition.

SOME OBJECTIVES FOR BETTER AND LESS RISKY SOFTWARE

Regardless of the form in which software is provided (as now seen or in substitute form), the customer needs some dramatic changes in software products. These changes must be reflected in a dependable software marketplace where proven products can be quickly identified and acquired at lower cost. Changes in the following product control areas are necessary to meet this objective:

Identification of products

Quality control

Cost assessment

Software development management

Validation

Certification

Identification of Products. The dispersion of the software industry and its resulting marketing weakness make it difficult for a customer to locate needed products. There is further difficulty in comparing the features of products, because the necessary documentation is unavailable or is inconsistent among competitive software products. Then, too, the customer gets confused because specifications for software frequently have little in common with the printed material about the product. In fact, the only material available on some products is the guide intended for training purposes! Worse yet, in some cases a supplier may consider design specifications proprietary, and

available for review only under a trial lease agreement. The result is that users are forced to "make do" with haphazard comparisons. Under these circumstances, the lack of customer confidence in selection of a product and mistrust of purchased software is not surprising.

Quality Control of Products. The quality of a computer program lies in:

Reliability and correctness,

Suitability of functions,

Efficiency and performance, and

Myriad design factors for maintainability, transferability, etc.

Unlike beauty, quality is not in the eye of each beholder—it must be uniformly understood and subject to definition and measurement.

Specific features that affect the quality of software products:

Correctness—the property of performing as intended for all acceptable inputs.

Clarity—a measure of the effort required to understand a program (and its documentation).

Robustness—a measure of the extent to which a program will remain well-behaved despite violations of basic assumptions.

Portability—a measure of the effort required to install a program in a new environment (example, another machine, operating system, etc.).

Modifiability/Maintainability—a measure of the effort required to alter a program in response to a change in specifications or requirements.

Performance—a multidimensional description of the program's demands for accountable resources.

Life-Cycle Cost—the costs of initial design and implementation, testing, operations, maintenance, modification, and documentation.

Human Factors Engineering—a measure of the ease, convenience, error-protectedness, and "naturalness" with which a human being can use the program. In other terms, palatability of the user/program interface.

Cost Assessment of Products. Cost is complementary to quality in the customer's selection. He needs data for evaluation of productivity, maintainability and recurring overhead costs. However, today, the customer must perform the original analysis, usually without benefit of any published cost or productivity factors accompanying the product.

Software Development Management. Every computer user should appreciate the basic principles of managing software development, to communicate effectively with programmers on development progress, whether the programmers are in-house or outside contractors. The customer's attempts to develop understanding are thwarted, however, by the wide disparity of views among practitioners and so-called "management experts."

Most reasonable practitioners believe software development is controllable. The essential elements of successful management are recognized: use of proven software engineering techniques, well chosen programming tools, workflow organization, substantive reports and customer reviews, and realistic cost and schedule allowances. Good reporting of software development must be done to give customers confidence that the job is being done well. These actions help overcome the "black art" image of software production.

Validation of Products. Validation is the process of showing that a program is consistent with the specification—that the program performs as intended without error. There are two basic approaches to validation: testing under appropriate conditions and proving correctness. The first of these, testing, will detect program errors for correction and demonstrate that the program performs as specified under the selected test inputs. The other approach, proof of correctness, would establish that a program has no errors and performs only the

defined actions or results. The difficulty of validation in general is that neither exhaustive tests or correctness proofs are feasible for most programs of interest.

Certification of Products. Many software customers, particularly in organized user groups, believe that the quality of programs can be assured through "certification." Software certification means that an independent agency evaluates and tests a program, and makes an authoritative declaration on its usability. Certification is, of course, an empty statement without a definition of the tests. These tests must validate the program, rather than probe its operability at random. The generation of such tests and the testing activity, including fixing the program, are demanding technical tasks that require almost 50 percent of any program developed today.

Changes in software product areas can be achieved—and, indeed, probably can be achieved only through setting some goals or targets and focusing the forces of research, standards, economic assessments, and customer "push" on the goals along with established schedules for achievement. Customer "push" must be based on better customer understanding of technical issues—not on emotional issues.

Some needed goals:

Reduce, by 90 percent, errors in software products delivered to Government customers.

Subject all software development for Government procurement to specified quality controls.

Have validation services in place for all major programming languages, for Data Base Management Systems and commonly used application packages.

Initiate a Government-wide research program to automate software production and provide a standard library of program generators.

Establish applications design and standards for conversion that assure direct transportability of software to any computer system.

CONCLUDING COMMENTS

Despite my pessimistic comments, I think that software is here to stay—for the foreseeable future, at least. Software gives the capability of using computers creatively, flexibly and economically. Computer software is a product that can be designed to benefit users innovatively and uniquely. However, software's continued viability depends upon the user being able to make sensible choices and to control software expenses.

Dr. Ruth M. Davis is Deputy Undersecretary of Defense for Research and Advanced Technology, Research and Engineering. Dr. Davis was Director of the Institute for Computer Sciences and Technology of the Commerce Department's National Bureau of Standards prior to assuming her present position. Dr. Davis has received many awards during her public service career, including the National Civil Service League Award in 1976, the Rockefeller Public Service Award for Professional Accomplishment and Leadership in 1973 and the Department of Commerce Gold Medal in 1972. She has been elected to membership in the National Academy of Engineering and the National Academy of Public Administration. She serves on the Electric Power Research Institute Advisory Council, on the Board of Directors of the American Association for the Advancement of Science and on other scientific and technical advisory groups and councils.

Dr. Davis received her B.A. degree from American University in 1950. She received her M.A. degree in 1952 and a Ph.D. degree in 1955, both from the University of Maryland.

SOFTWARE CONFIGURATION MANAGEMENT TESTABILITY AND TRACEABILITY

by

Mr. Harvey Tzudiker
US Army Computer Systems Command

Although the computer program development process has matured significantly, the lessons learned have been slow to permeate the software industry. The large investment required for a hardware development and production facility makes the implementation of time-proven organizational patterns, methods, and procedures mandatory. On the other hand, a software development activity can be established with virtually no investment in facilities and with shoestring operating budgets which leave no room for vital project controls, among which are technical reviews and audits, control of changes, and acceptance testing. Most of these controls fall within the area of configuration management or are primary concerns of configuration management organizations. Some of the software lessons learned, but not generally applied, are discussed here within that context.

The standard life cycle scheme for computer program configuration management is described in widely known and applied government directives and industry implementations. That scheme establishes requirements for *what* is to be done but, appropriately, not *how*.

The objectives of this article are to present:

- Lessons derived from the application of configuration management.
- Ideas on technical procedures.
- An approach for tracing individual requirements from system specifications to delivery of the system.
- An approach for improving the quality of system specifications by performing "Testability" analyses starting at the requirements review. In particular, the article discusses "testability" and "traceability" of computer programs or "software" requirements throughout the developmental process.

Briefly, "testability" is a determination of whether and how a requirement can be tested, thereby assuring that a requirement can be tested and *will* be tested. The identification of testable requirements establishes the point of departure for their "traceability," for example accounting for testable requirements through the developmental process to final test and acceptance.

One of the fundamental problems associated with software development is the inadequacy of the requirements statement. Inadequately stated requirements lead to false starts, excessive rework, extended periods of fixing and testing, overall schedule slippage, cost overruns, project turbulence and patchwork design, all of which result in fielding a system which is difficult and costly to maintain and modify.

If that problem exists, purification of the requirements specification must be done, sooner or

later, within the system developmental cycle. It can be accomplished in the beginning provided the developer has sufficient strength of purpose and credibility to insist on an acceptable specification before proceeding into system design, or it can be accomplished during the later phases of the developmental cycle with the problems, cost overruns, and delays described above. It is the classic case of not having enough time to do the job right in the first place, but having the time to redo it.

The software world, particularly that portion associated with large scale software acquisition, recognizes the problem. Current approaches to solutions include the development of special languages which "automate" the statement of requirements, the use of simulation, the development of software "breadboard" models or "throwaway" models, and incremental software development. However, getting customer personnel to specify accurate and complete requirements that are thoroughly understood and accepted by developer personnel is a fundamental communications problem. This problem is sure to persist.

A solution to the problem lies in not moving past the system requirements review until an acceptable specification is in hand. Most requirements reviews address the completeness and comprehensibility of the system specification. The quality of this review is based on the diligence, expertness, and objectivity of the reviewers. However, there is a danger that, having worked through the requirements portion of the specification, the reviewers will be content with a gross and cursory treatment of the quality assurance requirement.* But it is exactly at this point that the greatest potential payback in additional effort exists. At this point, a detailed analysis of the specification for testability should take place to finalize the specification and to set the stage for execution of one of the key quality assurance and configuration management processes of the project. For the purpose of this discussion, this process is called Testability Analysis.

Testability Analysis is a process of expansion or explosion of requirements statements down to the lowest possible level which expresses a single requirement, a determination of whether that requirement, as stated, can be tested; the type of test, that is, demonstration, analysis, simulation; and the level of test, for example, program, system, etc. The important product of this kind of analysis is the identification of those individual requirements which cannot be understood and agreed to by both customer and developer, those which are not complete or which, perhaps most important of all, cannot be tested and hence cannot be true statements of requirements. The opinion of this author is that the requirements review should take the form of a Testability Analysis. If the statement of a requirement will not support an objective, quantitative test, it should either be eliminated from the specification or identified as a requirement to which the developer need not respond.

Testability Analysis must identify and treat, in the proper perspective, three kinds of requirements statements: functional requirements (what actions must be accomplished), performance requirements (timing constraints, throughput, etc.), and design requirements (modularity, language, etc.). Testing of functional requirements is usually straightforward. Testing of performance requirements can be straightforward provided requirements are carefully and realistically stated. Too often performance requirements are stated in terms that are too general or too subjective. The customer must be very careful to isolate the key indicators of system performance, and the developer must be equally careful to make sure he understands the contract to which he will be held. The surest way to accomplish the objectives of both the customer and the developer is to agree on the specific tests to be used to demonstrate satisfaction of performance requirements. Both parties must also consider the technical feasibility of the tests to be accomplished in relation to schedule, personnel and facility requirements; there may not be enough time or money to perform them.

The treatment of some performance and design requirements can be bothersome. Design requirements (or constraints) are sometimes more properly stated as performance requirements; for example, reserving system capacity by restricting main memory utilization, and vice versa. For this reason, design and performance requirements should be very carefully validated, restated where appropriate or identified as a guidance rather than

*Military Standard, Mil Std 490, Specification Practices, calls for a Section 4 on Quality Assurance provisions in the specification. The purpose of this section is to identify how conformance to requirements is to be established. Among the Quality Assurance provisions, Mil Std 490 suggests a tabular presentation of requirements paragraph numbers together with the examinations and tests to be performed for each. A sample of a software oriented table is at Exhibit 1. Note that a single level and method of test is assigned to an entire paragraph containing multiple requirements. Under these conditions, demonstration could occur only at a system level test.

Exhibit 1, Verification Cross-Reference Index (Sheet 5 of 7)

SECTION 3 REQUIREMENT PARAGRAPH	VERIFICATION METHODS				LEVEL OF VERIFICATION			CATEGORY OF VERIFICATION			SECTION 4 VERIFICATION PARAGRAPH(S)
	NOT APPLICABLE	EXAMINATION/ANALYSIS	DEMONSTRATION/TEST	SUBPROGRAM	PROGRAM	INTEGRATED SUBSYSTEM	INTEGRATED SYSTEM	DEVELOPMENT	ACCEPTANCE	PRODUCTION	
3.2.3.2.2 E/V		X				X		X			4.2.3.1
			X			X		X			4.2.3.2
							XQ		X		4.3.3
3.2.3.2.3 Data Ret		X				X		X			4.2.3.1
			X			X		X			4.2.3.2
							X		X		4.3.3
3.2.3.2.4 File Maint		X				X		X			4.2.3.1
			X			X		X			4.2.3.2
							X		X		4.3.3
3.2.3.2.5 Dissem		X				X		X			4.2.3.1
Cont			X			X		X			4.2.3.2
							X		X		4.3.3
3.2.3.2.6 Data Access		X				X		X			4.2.3.1
			X			X		X			4.2.3.2
							X		X		4.3.3
3.2.3.3.1 FRENSIT4		X			X			X			4.2.2.1
			X		X			X			4.2.2.2
		X				X		X			4.2.3.1
			X			X		X			4.2.3.2
		X					X		X		4.3.3
			X				X		X		4.3.3
3.2.3.3.2 FRENSIT5		X			X			X			4.2.2.1
			X		X			X			4.2.2.2
		X				X		X			4.2.3.1
			X			X		X			4.2.3.2
		X					X		X		4.3.3
			X				X		X		4.3.3

formal requirements, leaving only truly essential requirements for the developer to fulfill. Normally, the developer should be allowed the widest possible design latitudes in fulfilling functional and performance requirements. Once again, these determinations are all too often made by default at system acceptance time. One of the key indicators of an effective technical manager is that he forces these decisions back to the beginning of the developmental process.*

After Testability Analysis has been performed, the quality assurance portion of the specification (Section 4 or its equivalent) should be annotated to indicate which statements of requirements do not require test. Test execution and testability determinations may have to be deferred, in which case it is imperative that such decisions be documented and communicated to all parties.

If accomplished as described above, Testability Analysis would have provided its own return on the basis of having purified or clarified the requirements specifications. However, there are still further benefits to be obtained. For example, after the requirements explosion described above has been accomplished a unique number or identifier is assigned to each stand-alone requirement along with an identification of level and method of test. This establishes a basis for tracing each unique requirement through the design baseline (with each of its associated design reviews) and into the product baseline. Additionally, Testability Analysis provides the basis for development of master and subordinate test plans with additional insights into technical approaches and constraints, the need for special facilities and identification of critical tests. A cross-referencing of testable requirements against the design baseline provides the necessary assurance that all requirements have been addressed or otherwise accommodated and constitutes a formal point of departure for each of the subsequent design reviews to be accomplished during the detailed design phase. Design reviews should reassess the validity of the testability determinations made during the earlier requirements review.

If the nature of the approved design precludes test of a specific requirement with the component of the system in which it exists, the test requirements must then migrate elsewhere in the test scheme and the cross-reference must be updated accordingly.

Each design review must also address and approve the test plan associated with that portion of the design baseline. At this level, test plans should include the criteria for accomplishing a successful test and the technical procedures for its execution to include statements of expected results. Development of test data and final detailed procedures are accomplished at a later time but should be subject to prior approval.

As previously discussed, Testability Analysis should have identified the level at which each requirement is to be tested. If the cross-reference of requirements to design baseline and then to test plans is maintained throughout the developmental process and the mapping of requirements into the design baseline is proper, then it should be possible to test (demonstrate, etc.) requirements at the earliest appropriate time. Only those requirements for test at the subsystem and system levels, that cannot be accommodated elsewhere, should be deferred. This concept meshes neatly with the recently emerged top-down development concept which, among other things, provides for early incremental demonstration of requirements. This avoids a typical situation in which every requirement must be tested at the system level. Accountability for testable requirements can be accomplished on a continuing basis as earlier, lower level tests are executed and testable requirements are satisfied.

An important aspect of a properly developed and maintained traceability scheme established by Testability Analysis is the facilitation of regression testing analysis. Where a requirement has been modified, the determination of which tests must be repeated, as well as the nature and extent of those tests, can be more quickly and accurately established. This approach is a more acceptable alternative to brute force repetitions of system level tests.

Up to this point, the discussion has addressed the treatment of a customer's explicit requirements. However, a project may require the development of support software (file maintenance, data management, system recovery, developmental tools, etc.), which is not specified in the customer's requirements. It is essential to recognize that testability and traceability of such software should be established irrespective of its specified or implied

*Exhibit 2 (appearing at the end of this article) is an excerpt from the initial product of a testability analysis effort. The exhibit includes explanations of the keys used in the analysis and representative pages produced in the analysis of a software requirements specification.

origin, if the software is a part of or affects the deliverable system.

The techniques and products described in this paper can be applied across a broad spectrum of software development activity. It is important to note that they complement and reinforce new software development methods and procedures. Finally, the discussion has emphasized the importance of front loading the project with these kinds of configuration management and quality assurance efforts. The manager must be prepared to initiate an education program to insure the understanding and support of key project personnel.

In conclusion, one major caution is necessary in applying these techniques. Project schedules and available manpower and skill levels must be carefully weighed in conjunction with the technical characteristics of the system being developed to arrive at a cost effective level of testability and traceability implementation. It is also possible that testability analysis should proceed only as far as the requirements review if system scope and complexity so warrant. On the other hand, if accountability for requirements must be established as a prerequisite to system acceptance, the project manager is strongly urged to apply these concepts with clear assignments of responsibility and periodic review to insure quantifiable and objective results.

EXHIBIT 2

The Verification Requirement Tables (VRT's) contain an expanded version of original system specifications. For each individual requirement on the left side of the sheet, data is provided in four columns on the right side of the sheet. Explanations follow.

a. **Expanded Specification.** Section 3, Requirements, Mil Std 490, of the original specifications has been expanded to isolate individual requirements. Each paragraph of the expanded specification contains a single requirement statement and is identified by a unique paragraph number which can be used to assist in locating the requirement in the original specification. The paragraph numbers used in the expanded specification are the original specification paragraph numbers, further indexed as required, to arrive at a unique reference number for each requirement.

b. **Responsibility (RESP).** The purpose of this column is to identify oranizational responsibility for verification of requirements.

CODE	DESCRIPTION
H	This requirement is strictly hardware and does not require any system operational software for verification purposes. The requirement should be verified by the hardware developer.
S	Requires operational system software (programs, data base, etc.) for verification purposes. The requirement is a candidate for verification by the software developer. The primary purpose of the verification is to demonstrate software and/or hardware/software integration. The inherent hardware dependencies have been separately specified elsewhere as hardware (H) requirements and will be verified by the hardware developer.
S/H	Requires operational system software for verification purposes. The requirement is a candidate for verification by the software developer. The primary purpose of the verification is to demonstrate software and/or hardware/software integration. The inherent hardware dependency must be established. The system integrator must verify that the hardware dependencies have been separately specified as a hardware (H) requirement to assure verification by the hardware developer.

S/O	Requirement specifies manuals, documentation, or training for which software developer is responsible.
N/A	Verification responsibility has not been assigned in the Verification Cross Reference Index (VCRI) by the system integrator.
–	Denotes title or introductory clause paragraph.
*	Allocation of verification responsibility in the VCRI by the system integrator conflicts with the software developer's position. (System integrator's position is given within the brackets on the left side of the Responsibility Column.)

c. **Verification (VERIF).** The purpose of this column is to identify requirements that will be formally verified by the software developer. Table entries used in the VERIF column are defined below. Requirements designated "Y" also have entries in the LEVEL and METHOD columns. The remaining requirements do not have additional entries in the VRT.

CODE	DESCRIPTION
Y	The software developer will verify this requirement directly, or by developing (and verifying) references to specific supporting requirements.
N1	Requirement is a reference to documentation other than system specifications (field manuals, regulations, etc.). Specific requirements, as applicable to the system, must be identified and incorporated into the system specifications.
N2	Too "general" for verification by the software developer.
N3	Specifies capabilities that have been omitted from the system.
N4	Operational/procedural oriented requirement that does not have impact on the system's software. Will not be verified by the software developer.
N5	Definition (of terms, test criteria, etc.) that does not have direct impact on the system's software. Test criteria, applicable definitions, etc., will be incorporated as test criteria in test plans/procedures.
N6	Specifies manuals, training, SOP's, etc., to be developed/performed by other than the hardware or software developers.

d. **LEVEL.** Identifies requirements that will be verified during Preliminary Qualification Tests (PQT's), and/or Formal Qualification Tests (FQT's). Multiple entries are used, as appropriate, in this column.
P = PQT F = FQT

e. **METHOD.** *The general method used to verify the requirement.* Table entries are defined below.

CODE	DESCRIPTION
E	Examination. A non-functional verification such as visual inspection of documentation physical characteristics of the system, and/or of the documentation associated with the system.

A — Analysis. A non-functional verification such as reduction or translation of data, analysis of test data, review of analytical data or analysis or performance of a detailed analysis.

T/D — Test Demonstration. The requirement will be verified via a functional verification such as actual operation wherein the element of verification is instrumented, measured or displayed directly (test) or where the element of verification is logically obvious as being a necessary constraint to some other result, but not itself displayed (demonstration).

	RESP	VERIF LEVEL		METHOD
3.2.2.2.4.2 Data Retrieval shall return the following to FRENSIT:	–			
*3.2.2.2.4.2.a The number of records,	S	Y	P	T/D
*3.2.2.2.4.2.b The content of all records found in the file that satisfy the retrieval criteria, or	S	Y	P	T/D
*3.2.2.2.4.2.c A flag indicating that the number of records retrieved exceeded the limit.	S	Y	P	T/D
*3.2.2.2.4.2.1 This data shall be passed to FRENSIT via main memory or RAM.**	S	Y	P	T/D
*3.2.2.2.5 SRI*** Processing.	–			
*3.2.2.2.5.1 FRENSIT shall call upon SRI Processing to:	–			
*3.2.2.2.5.1.a Make additions to the SRI tables,	S	Y	P	T/D
*3.2.2.2.5.1.b Make changes to the SRI tables,	S	Y	P	T/D
*3.2.2.2.5.1.c Make deletions to the SRI tables.	S	Y	P	T/D
*3.2.2.2.5.2 SRI Processing shall return to FRENSIT, flags indicating that:	–			
*3.2.2.2.5.2.a The requested action was successfully completed,	S	Y	P	T/D
*3.2.2.2.5.2.b An SRI to be added was already in the tables,	S	Y	P	T/D
*3.2.2.2.5.2.c An SRI to be changed could not be found,	S	Y	P	T/D
*3.2.2.2.5.2.d An SRI to be deleted could not be found.	S	Y	P	T/D
*3.2.2.2.5.2.e The originator of the change or delete message was not the same as the originator of the referenced SRI.	S	Y	P	T/D

***Friendly Situation**

****Random Access Memory**

*****Standing Request for Information**

*3.2.2.2.5.2.1 These flags are passed to FRENSIT via main memory.	S	Y	P	T/D
*3.2.2.3 Processing. There shall be one control program to govern the processing of FRENSIT queries and SRIs.	S	Y	P	E
*3.2.2.3.1 FRENSIT3.	–			
*3.2.2.3.1.a The purpose of FRENSIT3 shall be to govern the processing required for all FRENSIT queries and SRIs.	S	Y	P	E
*3.2.2.3.1.b Approach.	–			
*3.2.2.3.1.b.1 The sequence of events is depicted in Figure 4.	S	N2		
*3.2.2.3.1.1 FRENSIT3 shall call Edit and Validation.	S	Y	P	T/D
*3.2.2.3.1.2 FRENSIT3 shall provide edit and validation with the input message.	S	Y	P	T/D
*3.2.2.3.1.3 Edit and validation shall be responsible for validating the input message.	S	Y	P	T/D
*3.2.2.3.1.3.1 Upon completion of this function:	–			
*3.2.2.3.1.3.1.a Control shall be returned.	S	T	P	T/D
*3.2.2.3.1.3.1.b All error flags, if any, passed to FRENSIT3.	S	Y	P	T/D
*3.2.2.3.1.3.2 If any error flags are returned to FRENSIT3, then the Error Process shall be called to generate an error message to the originator.	S	Y	P	T/D
*3.2.2.3.1.3.3 If there were no errors, FRENSIT3 shall determine if the input message is a query or an SRI.	S	Y	P	E

Mr. Harvey Tzudiker is Chief, Test and Configuration Management Division, Technical Evaluation and Support Directorate, US Army Computer Systems Command, Fort Belvoir, VA. The Command's Quality Assurance Program is one of the major responsibilities of the Directorate. The Test and configuration management function have been performed largely within the quality assurance context for tactical and business systems developed and maintained both in-house and under contract.

Mr. Tzudiker joined the US Army Computer Systems Command in 1969. During his tenure, he was responsible for the publication of a military specification for software quality assurance program requirements, (MIL-S-52779 AD). The majority of his experience in private industry was in the application of computers for military use within the tactical environment, and the automation of engineering and production support functions. Mr. Tzudiker received his BS and BA from Boston University (1950).

SOFTWARE ACQUISITION WITHIN AIR FORCE SYSTEMS COMMAND — A Management Approach

by
Lt Col John J. Marciniak, USAF

For reader convenience selected definitions extracted from Department of Defense Directive (DOD) 5000.29 entitled, "Management of Computer Resources in Major Defense Systems," dated April 26, 1976, are reprinted.*

Computer Resources. **The totality of computer equipment, computer program, computer data, associated documentation, personnel, and supplies.**

Computer Software. **A combination of associated computer programs and computer data required to enable the computer equipment to perform computational or control functions.**

Software Engineering. **Science of design, development, implementation, test, evaluation, and maintenance of computer software over its life cycle.**

Software development is a recognized problem area in the acquisition of Department of Defense (DOD) weapon systems. The Air Force Systems Command (the responsible agent in the Air Force for the development and acquisition of weapons systems) has focused attention on bringing this problem under control. A management program has been developed and currently is being implemented. The program is intended to surmount and blend the software development activities into the systems engineering process within the next 5 years.

The purpose of this article is to present the management program that Air Force Systems Command has developed and is in the process of implementing.

The Problem

Although the use of software, or computer programs, has been employed in Air Force systems since 1960, recent advances in computer technology have extended its use through every aspect of the Air Force. Software applications in the Air Force vary from word processing for administrative functions to management information systems; and from command, control, communications (C^3) to spaceborne vehicles. Within these expanding applications, software often is the key element in development schedules. It was only natural that as major problems in development occurred, the Department of Defense and the Air Force would expend great resources and devote special management attention to gaining control over this area. This is better stated as the ability to manage the development and acquisition of software by the responsible system Program Manager and his organization.

The Studies

To the managers of the Command, at both the corporate and program level, the software problem is seen as excessive costs, schedule slippages, and

*Other definitions used in this article also appear in DOD Directive, No. 5000.29.

reduced performance compared to initial requirements. These are the symptoms. The actual causes have been the subject of numerous studies. While too numerous to list, from an Air Force Systems Command standpoint the key studies affecting our management program are the Computer Resources Management Study[1] and the DOD Weapon Systems Software Acquisition and Management Studies.[2] These studies formed the basis for the Air Force Systems Command program targeted against software acquisition management problems.

The first step in the formulation of the Air Force Systems Command program was a relative assessment of the critical problems discussed in the studies mentioned. The task was to formulate a strategy to implement specific recommendations within available resources to assure the earliest possible payoff. To accomplish this, each study recommendation was assessed as to where the potential payoff would be in the acquisition cycle with respect to three different management levels (Figure 1). The first level, Management, represents corporate Air Force management extending to the Program Manager. The second level, Engineering Management, is the program management organization which is responsible for managing the contractual effort, support engineering resources, and to an extent, program management in the developing organization (that is, the contractor). The third level, Engineering, represents those activities contributing directly to computer software development. Each X in Figure 1 represents a specific study recommendation, and takes into account that a recommendation may be applicable to different levels of management and life cycle phases.

ACQUISITION LIFE CYCLE PHASES

	CONCEPTUAL/VALIDATION	FULL SCALE ENG DEVELOPMENT	PRODUCTION/DEPLOYMENT
MANAGEMENT	XXXXXXXXX XXXXXXX	XXX	X
ENGINEERING MANAGEMENT	XXXXXXXXX XXXXXXXXX XXXXXXX	XXXXXXXXX XXXXXXXXX XXXX	XXXXX
ENGINEERING	XX	XXXXXXXXX	

FIGURE 1. APPLICABILITY OF STUDY RECOMMENDATIONS

As indicated by the relative number of X's, corrective actions should be concentrated at the Management and Engineering Management levels in the Conceptual and Validation Phases, and at the Engineering Management level in the Full Scale Development Phase. This is consistent with the premise that the greatest leverage for successful software development is in the planning accomplished at the front end of the system.

Another key consideration in the formulation of the Air Force Systems Command program came directly from a Rand Study that concluded, "the difficulty experienced by the Air Force in managing computer resources stems principally from the failure to follow an adequately structured and properly managed development process."[1] While this conclusion is seemingly an encompassing generalization, it has been recognized by several Air Force Systems Command program offices as failure to adhere to a disciplined development process.

THE AFSC APPROACH

The Air Force Systems Command approach to solution of the software problem is a logical extension of the referenced conclusions. Five steps are outlined:

- first, the definition and description of a software acquisition management discipline;
- second, the employment of an organizational mechanism to implement the discipline;
- third, the use of education and training to communicate the discipline to personnel who manage computer resources;
- fourth, the development of procedures, tools and techniques necessary to support the discipline;
- and finally, the monitoring of organizational and discipline implementation to measure effectiveness.

The Discipline

An espoused premise is that success would be greater if software were treated like hardware—that is if the same rigor were applied to the software development process as is applied to hardware

development. This premise has been extended recently in recognition of the fact that software is a unique entity. Nevertheless, software is an integral part of the system. The rigorous discipline that must be applied in its development is from the viewpoint of total systems engineering. In order to create a discipline, the tools, techniques, and procedures need to be described in a systems context, and promulgated as a baseline. To have a discipline, one must be able to communicate it; this connotes a written description. The discipline itself, unfortunately, is not adequately described. Software often has been regarded as more art than science, the inference being that it is not susceptible to discipline or uniformity of approach. Although general approaches are described and followed, the great artists each had an individual "technique." In the software world the "art" of the programmer may be in this person's choice of individual logic. However, the techniques of analysis, design, and structure do lend to uniformity and discipline. Techniques such as structured programming are now being defined and put to use.

The structure for the definition of the software acquisition management discipline, in the Air Force, is shown in Figure 2 as a hierarchy of Regulations, Standards, and Guidelines. Department of Defense Directive (DODD) 5000.1, "Acquisition of Major Defense Systems," sets overall policy within the DOD for the management of major defense systems. Within the Air Force it is promulgated by Air Force Regulation (AFR) 800–2, "Program Management." Department of Defense Directive 5000.29, "Management of Computer Resources in Major Defense Systems," along with the companion instruction DODI 5000.31, defines Department of Defense management policy for computer resource management in systems. Air Force Regulation 800–14, "Management of Computer Resources in Systems," is the promulgation of DODD 5000.29. The regulation describes life cycle management policies for the acquisition and support of embedded computer resources in systems. The key concepts of this regulation are the Computer Resource Integrated Support Plan (CRISP) and the Operational/Support Configuration Management Plan (O/S CMP). The Computer Resource Integrated Support Plan describes the support concept and delineates required life cycle support resources. The Operational/Support Configuration Management Plan details procedures to be followed for the control of the computer resources configuration during the deployment phase. Air Force Systems Command Pamphlet 800–3, "A Guide for Program Management," describes the general considerations in managing the acquisition of a system and the principal functional processes used. The pamphlet is the guide for Air Force Systems Command Program Managers and is a "bible" of acquisition management.

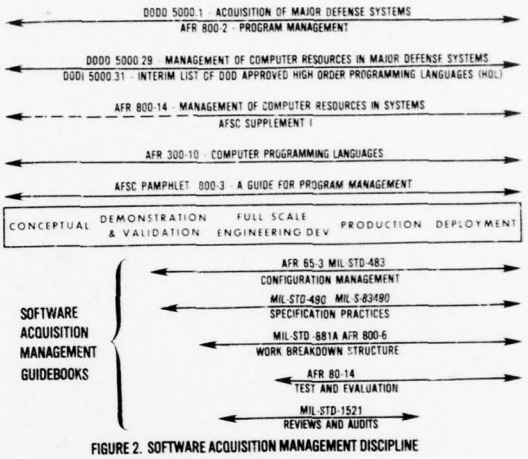


FIGURE 2. SOFTWARE ACQUISITION MANAGEMENT DISCIPLINE

The functional disciplines are described in various military standards and regulations, for example MIL-STD-483, "Configuration Management Practices and Procedures." Each of the documents shown below the Acquisition Phases in Figure 2, describes a specific functional discipline. However, in these documents the treatment of computer resources is weak. Often only a few paragraphs are dedicated to this area. The description of the regulatory structure shown was derived from the top of the hierarchy—genealogically with respect to computer resources has been from the bottom up. The functional disciplines came first, followed by AFR 800–14 in 1974 and by DODD 5000.29 in 1976. Thus the discipline to date has been derived on a vertically structured functional basis. The addition of software engineering procedures appears as an afterthought.

In the past few years the discipline of software engineering has emerged, taking form and substance as an entity and formal discipline. Barry Boehm's article entitled, "Software Engineering," is an excellent treatise of this subject.[4] In the Air Force by filling the voids of the regulatory structure shown in Figure 2, integration of software engineering with management has begun. The key effort is the *Software Acquisition Management Guidebooks*. Each

guidebook takes a specific functional topic, for example, *Configuration Management, Contracting for Software Acquisition,* etc. Practices and procedures are described from a computer resource viewpoint. This effort started at the Electronic Systems Division of Air Force Systems Command. Initial guidebooks were developed by the Mitre Corporation; then in 1975 the program received new emphasis and was accelerated. Currently three sets of guidebooks are underway; *C^3, Airborne (Avionics/Space and Missile)* and *Ground Based (Crew Trainer/Simulator, and Automatic Test Equipment).* Three sets were required to insure adequate coverage of different application perspectives and to capture different viewpoints of preparation. Each set is being prepared by a different contractor. After completion, the guidebooks will be analyzed to see if these texts can and should be published as one integrated series. A list of guidebook topics with completion dates and Defense Documentation Center (DDC) numbers is given in Table I.

A separate chapter for Air Force Systems Command Pamphlet 800–3 is being prepared. The objective is to provide guidance on computer resources, in particular organizational options for Program Managers. The purpose of this chapter is to emphasize the need for software engineering expertise in the embryonic systems engineering team when the system Project Office is formed. The team then can provide proper computer resources planning on the front end of the system acquisition where it is critically important to the success of software development.

Other regulatory measures have been taken, or are underway. The Air Force Systems Command has published a Command supplement to AFR 800–14. Command regulations to implement language control are in the process of development in response to DODI 5000.31 and AFR 300–10, both entitled "Computer Programming Languages."

Language control is a subject of particular importance. Standardization on a single language, or a set of languages, connotes cost savings. By making Air Force intent clear to industry, investment decisions on compiler implementation are

TABLE 1. SOFTWARE GUIDEBOOKS

TOPICS	FUNCTIONAL AREAS		
	C^3	AIRBORNE/SPACE	GROUND SUPPORT
	(COMPLETION DATES OR DDC ACCESSION NO)		
MONITORING STATUS	AD-A016488		NOV 78
REGS· SPECS· & STNDS·	AD-A016401	MAR 78	
CONTRACTING	AD-A020444		SEP 78
DOCUMENTATION	AD-A027051	SEP 78	APR 78
SOW/RFP PREPARATION	AD-A035924	MAY 78	
LIFE CYCLE EVENTS	AD-A037115		
FACILITIES	AD-A038234		
CONFIGURATION MANAGEMENT	DEC 77		NOV 78
QUALITY ASSURANCE	DEC 77	MAR 78	SEP 78
MAINTENANCE	JAN 77		
VERIFICATION	FEB 77	SEP 78	
VALIDATION & CERTIFICATION	FEB 77	SEP 78	
REVIEWS & AUDITS	FEB 77	MAR 78	
REQUIREMENTS SPECIFICATION	APR 77	MAY 78	APR 78
COST ESTIMATION	MAY 77		JUN 78
OVERVIEW TO GUIDEBOOKS	JUN 77		

AS OF: 31 DEC 77

simplified. Within the Program Manager's environment, language control simplifies the language selection process by providing (via a language control facility) expertise for compiler development, plus a host of tools for testing standard compilers and the code generated from them—in short, discipline. In the JOVIAL family of languages the Air Force has standardized on J3 and J73, specified by MIL-STD-1588 (USAF) and MIL-STD-1589 (USAF), respectively. In the near term J3 will be phased out in favor of J73 until a common higher order Department of Defense language is fully developed for operational implementation in the period from 1980 to 1990.

The remaining task is to provide an overall perspective or architecture to the software acquisition management discipline in the Air Force Systems Command. Recognizing that the discipline has been structured from a bottom up and vertical orientation, the Command initiated the evaluation process described. A team of experts was assembled and has evaluated the many different ways software is developed in the Air Force. The functional discipline as set forth in MIL-STD-483, MIL-STD-1521, and others, will be studied to assess adequateness, consistency, and accuracy. This effort should produce the shape of the future discipline. Problems contained in the current structure will be eliminated and eventually an adequate baseline description of the computer software development process will be provided.

The Organizational Mechanism

As a result of one recommendation made by authors of the Rand Study to "Establish centers of expertise in computer technology within selected organizations in the Air Force," the Command conducted its own study. The result was an expansion of the Office of Assistant for Processor and Software Planning to a full Directorate of Computer Resource Development Policy and Planning.[1] This office is the computer resource focal point for Air Force Systems Command policy, technology, and management actions. Further, this office has the responsibility for the Command thrust of solving software acquisition management problems. Specific functions are shown in Table 2. An important part of this organizational change was the institution of field focal points at each product division and at key laboratories (See Figure 3). This computer resource infrastructure throughout the Command is an effective mechanism to:

- coordinate management and technology programs/actions,
- monitor policy implementation,
- work critical problems such as higher order language standardization, and
- gain visibility into problem areas within the programs.

Essentially, the infrastructure provides implementation, across the Command, of the software acquisition management discipline.

TABLE 2.

DIRECTORATE OF COMPUTER RESOURCE DEVELOPMENT POLICY AND PLANNING

- COMPUTER RESOURCES REQUIREMENTS REVIEW (CHAIRS THE COMMAND COMPUTER RESOURCE COMMITTEE)
- COMPUTER RESOURCE LONG RANGE DEVELOPMENT PLANNING
- COMPUTER RESOURCE POLICY
  - AFR 300-SERIES (GENERAL PURPOSE AUTOMATIC DATA PROCESSING EQUIPMENT)
  - AFR 800-14 (EMBEDDED COMPUTERS)
- PLANNING APPLICATIONS OF COMPUTER RESOURCE TECHNOLOGY
- ADVOCATE PROCEDURES/STANDARDS/DATA REQUIREMENTS
- PERSONNEL REQUIREMENTS IDENTIFICATION
  - SPECIAL SKILLS
  - EDUCATION AND TRAINING
- COMPUTER RESOURCE FOCAL POINT

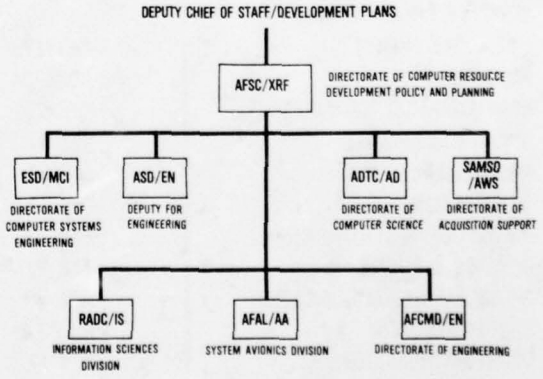


FIGURE 3. AFSC COMPUTER RESOURCE FOCAL POINT ORGANIZATION

Education, Training, and Personnel Initiatives

Communication of software engineering discipline is more than an Air Force Systems Command role and the Air Force is playing its part. The Air Force Institute of Technology has instituted a graduate software engineering curriculum, and work continues toward developing a series of short courses. The Command is attempting to initiate short courses for procurement personnel, and the Education with Industry (EWI) program is being expanded from two to ten spaces per year. In this latter program, Air Force officers spend 10 months working in industry learning software engineering from the inside out. After the Education with Industry tour, the students will go to key Air Force assignments.

With the help of the Military Personnel Center, the records of computer resource personnel now incorporate Special Experience Identifiers. This is in recognition that there are differences, as in the hardware world, in personnel specialities in computer resources. Example: a radar engineer as opposed to a communications specialist. The idea is not to stratify Air Force personnel, but to capture peculiar expertise for future application. For example, computer resource expertise in real-time systems, that may have been attained in a ground-control environment for a space vehicle, can be applied to a real-time avionics system.

Finally, a Computer Resource Newsletter has been initiated to bring critical information to all computer resources personnel in the Command. Three issues have been published and have contained articles about the software acquisition management guidebooks, microprocessors, data rights for computer programs, new policy on language control, and many other subjects. So far the newsletter has filled a void in communications with success. Current distribution of this publication is approximately 800. Requests for additional distribution continue to be received from organizations throughout the Department of Defense.

The Technology Program

A software acquisition management discipline cannot be implemented without supporting tools and techniques. In the past 4 years the computer resource technology program within the Air Force has been transformed from a focus on computer hardware efforts to the support of management needs in software acquisition. The Office of the Under Secretary of Defense Research and Engineering has taken a leadership role here and is developing a 5-year technology plan that integrates the efforts of all Services in a coordinated and concerted effort. The plan is under the cognizance of the Management Steering Committee for Embedded Computer Resources, chartered by DODD 5000.29. The plan provides an overview of activity, insures that duplication of effort is avoided, and makes certain that the program is balanced and given priority across described management areas. By coordinating the technology efforts of the Services, the minimum resources available for these efforts can be consolidated and greater leverage can be achieved.

While only a small amount of money is spent in this technology area by each Service—approximately $3 to 5 million per year in the Air Force—the payoff is big. Compared to the approximately $1 billion per year that is spent on software in the Air Force, the leverage attained is about 200 to 1. The real plus is the critical role that the technology program plays in support of the management discipline.

The Software Acquisition Management Guidebooks are a prime example of a product of this program. Another key effort is development of the Computer Aided Design and System Analysis Tool (CADSAT) (formerly the Computer Aided Requirements Analysis (CARA)). The objective of CADSAT is to provide a tool for computer analysis of system requirements. This area has been determined to be a key problem in software development and is listed as one of the primary causes for software development failures.

A major effort in Air Force Systems Command is the institution of language control. The Air Force has had a JOVIAL standard for years (Air Force Manual (AFM) 100–24, now MIL-STD-1588). Adherence to the standard has been ineffective owing to lack of control. The formation of a Language Control Facility will require support from the technology program to attain operating capability. The Facility will be established at the Rome Air Development Center and then, when operational, will be transitioned to a permanent location and organization. A JOVIAL Compiler Implementation Tool (JOCIT) for J73 is currently under development by Rome Air Development Center. When complete, this tool will provide a uniform basis for developing future compilers. Attendant support tools such as the JOVIAL Compiler Validation System (JCVS) and

a JOVIAL Automated Verification System (JAVS) also will be developed. These tools will provide, to the System Project Office, the means to test J73 compilers developed under contract, and the ability to test code generated by the compiler. This capability will enhance the support of future programs that are developed by Air Force Systems Command. Other efforts in requirements analysis, cost estimating, and testing of software are underway.

The Future

While advances have been made in the development of software, many problems remain. Concerted effort will be required to eliminate these problems so that computer software development can become a normal part of the systems engineering process. The special emphasis it now receives then will decline to a sustenance level necessary to continue evolvement of the discipline. The planning for this has already begun. Department of Defense Directive 5000.29 will expire 6 years after issue date.

An important part of any program is its measures of effectiveness, not only to judge success, but also to assess program adequacy and to identify deficiencies. It is extremely important, because of the limited resources available, that efforts in the software development discipline be directed to the critical payoff areas. In the Air Force Systems Command metrics based upon individual program tracking are being developed. The idea is to compare a program with similar past program acquisitions. Key factors will be tracked, such as divergence of proposed software development cost and size vs the actual cost and size, and the ratio of testing and validation to analysis, design and coding. The first two factors are obvious—the last stems from the fact that the ratio of testing to the total software development cost has traditionally been 40 percent. By tracking the trend of this ratio, perhaps development success may be assessed, and software reliability predicted.

The time table for the program is 5 years. A period of intensive study of the problem is complete. Implementation of the program has begun. Indications of success are appearing although direct results will not be apparent for about 2 years.

SUMMARY

The Air Force Systems Command program is a logical approach that has been given priority to solve the weapon systems software acquisition problem and bring it under management control. The program has been developed in concert with higher level policy direction and is based on many study recommendations. The approach is:

- Define and describe a software acquisition management discipline
- Employ an organizational mechanism to implement the discipline
- Use education and training to communicate the discipline
- Develop supporting procedures, tools, and techniques through use of the computer resource technology program
- Measure effectiveness

If the Air Force is to make progress in this critical problem area, it will need the continued support of top management to insure that resources are available to carry forth a well-balanced program. The program is started and benefits will accrue. Real assessment will not be attainable for a period of from 2 to 5 years. In that time the continuing commitment of Air Force leadership to the program must be maintained to maximize the effect that the program will have. Hopefully, in 5 years this special attention area will become of a second nature and normalized as an integral part of the systems acquisition process.

CITED REFERENCES

1. **Rand Corporation, "The Computer Resources Management Study," R-1855/PR, Apr 76.**
2. **Mitre Corporation, "DOD Weapon Systems Software Acquisition and Management Study," MTR-6908, Vol I and Vol II, Jun 75.**

3. The Johns Hopkins University, Applied Physics Laboratory, "DOD Weapons Systems Software Management Study," SR 75–3, Jun 75.

4. B. W. Boehm, "Software Engineering," IEEE Transactions on Computers, C-25 (12): 1226–41(1976).

Lieutenant Colonel John J. Marciniak is the Director of Computer Resource Development Policy and Planning, Deputy Chief of Staff/Development Plans, Air Force Systems Command.

Lieutenant Colonel Marciniak's previous assignments include a tour with the Air Force Directorate of Data Automation, Chief of Data Systems, Air Force Satellite Control Facility, and as a Research and Development staff officer at the Electronic Systems Division. His primary experience is in command, control and communications (C^3) and space and missile systems. He participated in the Command Automation Master Plan, a primary automation study for the Military Airlift Command (MAC), and was one of the original project officers for Airborne Data Automation (a pilot project to automate one of the Strategic Air Command Airborne Command Post aircraft). As Chief of Data Systems for the Air Force Satellite Control Facility, he was responsible for development of the ground support hardware and software for major defense satellite programs. He has had extensive experience in the development of software for defense systems.

Lieutenant Colonel Marciniak received his BEE degree from the City University of New York (1957) and his MEE from the University of Oklahoma (1969). He is a graduate of the Air Command and Staff College, Maxwell AFB, AL (1973).

COMPUTER SYSTEMS IN THE NAVY

by

Richard Edwin Fryer, Department of Navy

Traditional use of computers is viewed by many with growing distrust.[1,2,3] Software costs are being measured and often found to be higher than even the brave dared predict. Quality metrics for software are being developed[4] and much existing software has registered poorly against these goals. Hundreds of articles, reports, and studies are pointing the way to improved methods for software definition, management, testing and production.

In this article the author explores the purpose and nature of software currently in use in the Navy and compares today's systems with earlier tactical Naval systems. Examination is made of several shifting technology areas that directly affect the way new Navy systems are being developed. Examples of recent system developments at the Naval Weapons Center that incorporate these directions or technology shifts are reviewed. Against this background, the impact of near term technology advances are discussed.

SOFTWARE SIMPLIFIES HARDWARE

The Navy, for many years, has applied computers in traditional tactical (system control, weapon delivery) and management (personnel, logistics, supply) areas. Now with the advent of low cost computers and significant applications-oriented software packages the way that problems are viewed and the way solutions conceived has altered.

Analog computers, electronic and mechanical, have been used in Naval tactical, ground support, and system development applications. Analog computers are based on straightforward physical laws. The cost of an analog solution to a complex problem is significantly greater than for a simple problem. Analog computers fail to meet the requirements of modern problems in at least two major ways. The lack of accurate and inexpensive storage elements for analog data prevents the time sharing of computation modules; resulting in large systems. A second problem is that an effective analog has not been developed for manipulating symbolic information.

Digital computer systems, at first, were used primarily in applications such as command and control. Problems in command and control require great logic and computational complexity for solution. The first digital computers were complex devices. However, it is the capability of a general purpose computer to be programmed and not the hardware complexity that makes complex problem solution feasible. That is, the inherent complexity of a problem is expressed in a sequence of instructions rather than in discrete circuitry. Programmable digital computers have resulted in reduced hardware size when compared with analog systems. Elements of the computer—busses, arithmetic and logic units, and storage—are reused many times during the execution of the software instructions. Software variability allows for the solution of a more complex problem without significant increase in the complexity of the hardware. Ordinarily, there are no hardware imposed costs related to minor alterations in the software or in the definition of the problem being solved.

SOFTWARE BENEFITS ARE INCREASING

For the last several years, articles assessing the status of software in industry and articles on software development and management have used a

graph similar to Figure 1 to emphasize the importance of software.[1567] The real project funds used for software also show an upward trend. The unfortunate aspect of this figure is that it seems to carry a negative connotation for software to the average reader and, apparently often to the author using the figure as well.[5] Not shown by this figure are several other aspects of systems that are increasing: system complexity, the number of software based systems in use, and software value per unit cost. The implication of Figure 1 is cause for hope as much as it is a case for alarm.

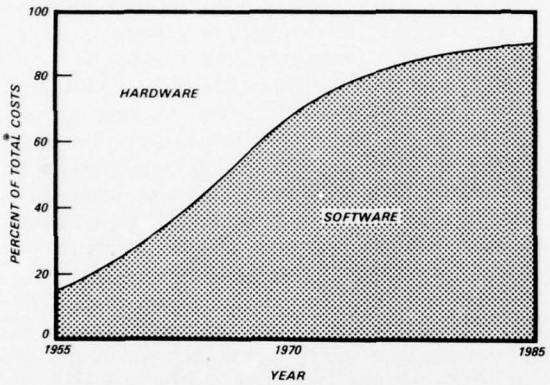


Figure 1. Hardware/Software Cost Trends

THE PACKAGED PROGRAM

A 'packaged program' denotes a standard application package.[8] Packaged programs have been available since early in 1950. In recent years, developments in this field have accelerated owing both to the maturity of the market and the reduction in hardware costs (expansion of customer base). At this time the effect of software and hardware standards is a strongly positive force in the usability of packaged software. At the Naval Weapons Center, packaged software has played a significant role in support of data base manipulations, graph generation, project management, report retrieval, mathematical analysis, and computer aided drafting. A human engineered data base system is an excellent example of the maturity of this field and the benefits that can accrue from the use of such systems.

The economic factor gives strong impetus for the utilization of packaged software. When computer operations are ruled by hardware costs, programmers are usually encouraged to get the last drop of efficiency from each application. This requires application programs customized to a degree not required by the application itself, but rather, demanded by the economic environment. As hardware costs become less significant this barrier drops and the actual economic value to the user of packaged software can be exploited. As the extent of customizing is reduced, the programming manager can concentrate on acceptance testing instead of program support and maintenance.

The major problem facing the potential customer of a packaged software system is the determination of requirements. All too often, requirements are developed in retrospect after software acquisition and/or implementation.[9] The customer must understand his requirements sufficiently well to evaluate the utility of a packaged program for his needs and further to select from competing approaches. The recommended solution to this problem is similar to that for the problem of producing software or acquiring hardware: *develop the requirements analysis and consider design alternatives first.*

Packaged systems (sometimes called 'turnkey' systems) include both the packaged software and the required hardware to operate the software. The Versatile Training System (VTS) is of this type, and is in operation and under further development at the Naval Weapons Center.

Packaged software is available commercially from computer vendors, software contract companies, educational and other special interest groups, user libraries, trade associations, and US Government agencies. A Navy network currently under development also will support a software repository.

THE EMBEDDED COMPUTER

A complete information processing system will include both computing hardware and software (perhaps packaged). Traditionally, embedded computer systems have been parts of an overall weapon system such as an aircraft or missile.[10] The Memory Loader/Verifier, described in the examples, uses an embedded computer system to significantly improve on the utility, human factors, and self test features of specific ground support equipment.

Embedded computer systems may be implemented with "Read Only Memories." In this form, software is sometimes renamed 'firmware'. The

absence of 'software' in small (usually microprocessor based) systems has been advertised by many as a solution to the software *problem*. This is a false path[11] to reducing software costs. The same development procedure and software engineering talent is required for the definition of the Read Only Memory (ROM) contents. This approach has been advocated as a means of managing the configuration of software and appears to be based on the assumption that if software may be altered only with great difficulty, then project management can regain control.

Reliability is demanded of embedded computer systems as these systems normally are used by a disinterested or an unsympathetic customer and not a protective developer. The key to the development of adequate reliability in embedded computer systems appears to be proper management of software development[10]—essentially the same recommendation that is made for acquisition of packaged software.

PROGRAMMED LOGIC

At the other extreme from large central site computers that have a major program package working for the user is a class of systems that lack altogether a general purpose programmable computer. It is still possible, however, to replace discrete or 'random' logic with software. The approach[12] allows for the use of top down design and other engineering techniques to make efficient use of complex circuitry (Programmable logic arrays, multiplexers, read only memories, and bit slice processors). Since hardware costs do not dominate, multiple machines may be used. The use of a number of machines (also called 'state machines') reduces the need for exceptional case handling (the source of interrupts in computer architectures) and further reduces the complexity of the state machine software.

Hardware designs using these techniques can exhibit improved reliability, modularity and flexibility, and are testable and maintainable. The designs are easily documented to be traceable from requirements to implementation, and for self-descriptiveness. These aspects are indicators of software quality.[4] The Software Validation and Control System (SOVAC) is a design example of these techniques.

SYSTEM EXAMPLES

Configuration and Data Management Support System

The Configuration and Data Management Support System (CADMSS) is an automated engineering support system for configuration and data management. The system was developed at the Pacific Missile Test Center and encompasses contract monitoring, baseline accounting, and change monitoring for Naval systems. At the Naval Weapons Center, CADMSS is built on SYSTEM 2000, a large packaged software data base system that is executed on the Univac 1110 host computer. The customers of the Configuration and Data Management Support System are neither computer operators nor programmers. For these reasons, terminal interaction is designed to include many messages and prompts. The system concept is one of integrating and providing access to several related data bases—contract status, baseline accounting, change accounting, configuration management, and data management. A data base of 60,000 drawings and related documentation occupying 250 million bytes of storage is currently in operation. Access to this data base will be provided to authorized users nationwide. The use of standard ASCII COBOL and SYSTEM 2000 languages provide a degree of host independence to reduce future conversion costs as hardware is replaced.[13] The Configuration and Data Management System will provide a means for configuration management of several production tactical systems.

Versatile Training System

The Versatile Training System (VTS) was first deployed in 1972 with the A-7E Fleet Readiness Squadron (VA 122) and two operational A-7E squadrons. The system supported the training of enlisted Naval aviation personnel for aircraft maintenance. Computer assisted training at the squadrons has reduced the loss of personnel assets accruing from schedule peculiarities and other training deficiencies that were created by the large number of students processed through many varied training programs (VA-122 schedules in excess of 1000 students each year). Computer assisted training also has significantly reduced clerical and instructor workload. The result is improved individual counseling. The present scheduling of instructors, training media, classrooms, students and simulators has greatly improved the effectiveness of the Fleet Readiness Aircraft Maintenance Personnel

(FRAMP) in VA-122. The data base access and text processing aspects of the Versatile Training System have incorporated form letters, reports, and many other manual tasks.

The Versatile Training System has been duplicated to support Naval Air Stations at Cecil Field and Jacksonville, Florida; Oceana, Virginia; Whidbey Island, Washington; and, Miramar and North Island, California. Marine Corps applications, Naval Air Station, Naval Aviation Maintenance Training Detachments, Operational Squadrons, the Chief of Naval Reserves, and the TRIDENT training facility all are users of the Versatile Training System. As such this training system qualifies as a packaged system with complete software and hardware ready to perform upon installation. A site manager maintains the capability to customize aspects of the system that are site specific.

The Versatile Training System consists of a generalized data base and associated packaged software systems capable of operating student terminals, report generation, resource configuration, resource scheduling, optical mark grading, data base development, data base inquiry, and simulator operation. All software is written in BASIC (a computer language) and runs under the Resource Sharing Timesharing System (RSTS) on a Digital Equipment Company PDP-11 family computer system. The computer system, the peripheral equipment required and the computer model depend on site specific details. The combination of inexpensive minicomputers used in Versatile Training System management has been a critical factor in the wide success enjoyed by the system.[14] The Versatile Training System is modifying many aspects of Navy personnel training.

Memory Loader/Verifier

The Memory Loader/Verifier is an intelligent computer loader/tester for tactical systems. This item of ground support equipment is for operational squadron routine use and for operational and intermediate (O&I) level maintenance of avionics systems. Originally designed for the A-7E aircraft, the system is finding application on many other Navy aircraft including the A-6E, the A-4M, and the EA6B. The basic functional requirement for the loader/verifier is to save, reload, and check the load in the memory of an avionics processor. A thorough awareness of the various logistics, training, and maintenance problems faced by squadrons led the developers to many enhancements in the basic functional capabilities with the assistance of an Embedded Computer System.

The Memory Loader/Verifier uses a microcomputer (8080A) and a sealed cassette unit adapted for military use as primary components. An alphanumeric display has sufficient capability to issue prompts to the operator, reducing the time required for squadron personnel training in its use. Supplemental instructions are included on a panel in the cover of the unit. The basic operations (load, store, and verify) are accessible through function buttons on the unit. A numeric keyboard is also included for annotation functions, for example, selecting a specific program from the tape to be loaded.

System operation begins with a power up Read Only Memory bootstrap that loads the embedded computer system volatile memory with self-test software and initiates test execution. The successful completion of this operation (including checksum testing of the cassette tape transfer) causes the embedded computer system to be reloaded with operational software. This action initiates operator dialog. The operator is given the opportunity to test the interconnection cable by plugging the cable into a compatible socket on the Memory Loader/Verifier panel. As previous diagnostics have tested to the cable drivers, a failure at this point gives the operator confidence that the cable is defective. Diagnostics are in English and use the alpha display. Maintainability of the Memory Loader/Verifier is improved and the incidence of erroneous trouble reports are greatly reduced as compared to maintainability problems and trouble reports experienced when non-'intelligent' ground support equipment is used. After connecting the Memory Loader/Verifier to the avionics computer connector, the operator may elect to store and save the current version of the tactical program (or the test program if it was installed) in the cassette. This retains any information stored within the program that was executing in the avionics computer. Parameters that are of interest are error counters, hardware biases, aircraft system dependent constants such as boresight values, platform constants, etc. The operator then may elect to load and exercise the avionics computer self-test. The operator may examine error counters, reload the avionics computer with a new master copy of the tactical program (constants for that specific aircraft can be automatically inserted by the Memory Loader/Verifier), or replace the previously saved program to continue accumulation of failure data.

At each cassette transfer, a checksum is performed by the microcomputer 8080. The embedded computer system records failures on the tape. When re-reads continue to fail the checksum, that section of the tape is marked unreliable on a tape log. Backup copies of critical tape files are used to extend the lifetime of a cassette load. The embedded computer system will report self-diagnosed problems and error counters to a maintenance person having a valid access key. The environmentally sealed cassette unit is removable from the front panel of the Memory Loader/Verifier for maintenance or for secure stowage when required.

Many unexplored potential applications of a basic Memory Loader/Verifier system exist. The combination of removable mass storage, an embedded computer system, and signal conditioning electronics in a small enclosure opens avenues for maintenance in many areas of Naval systems. These factors have not gone unnoticed.

Software Validation and Control System

The Software Validation and Control System (SOVAC) is a device for enhancing the development and validation of modifications to a tactical software system. The system is a combination of minicomputer, a microcontroller, and several state machines. The combination balances capabilities and system costs. The system design began with a list of missing features or requirements not fully met by previously developed computer monitor systems. The process led to a prototype users manual that included technical tradeoff rationale, syntax diagrams for complex operations, and sample problems. Careful consideration was given to the man-machine aspects. Upon review and approval of the draft manual, hardware and software requirements were defined. Hardware design proceeded after assignment of functions to the microcontroller and to state machines. System design details were documented through use of an Applicon computer aided drawing system (another packaged system).

The user interfaces with the system through a high level symbolic computer language. The language is similar to BASIC in style, but is oriented to software test functions. Commands include TRACE, TIME EVENTS, DISPLAY, and ON CONDITION *spec* ACTION *spec*. The high level language is interpreted through use of a small Digital Equipment Company Computer, the PDP-11. The interpreter is written in PASCAL. Hardware interface to the microcontroller is provided through an assembly language interrupt and an input/output port capability that can account for slight variations in Software Validation and Control System hardware interfaces with various avionics computers. A FORTRAN connection to collected data gives the user access to a graphics terminal for bar, statistical, and graphical charts.

The Software Validation and Control System concept and implementation, although developed for the A-7E program, was carried out with consideration given to various tactical systems. Implementation to support the A-6E, A-4M, and the F-18 aircraft are anticipated in FY-79 and FY-80.[15]

TRENDS IN SOFTWARE AND HARDWARE

The trend toward decreasing costs for hardware will certainly continue in both military and commercial systems. A significant improvement in capability for low cost computing elements is becoming visible: the megabyte micro is available, and 16 bit supermicros will soon compete with the superminis of the 1970 decade.[16] High level languages will be used routinely with this capable hardware (COBOL, PASCAL, and FORTRAN languages are readily available now). Mass storage is now inexpensive—a 300 Megabyte storage module interfaced with a microcomputer costs approximately $25,000. An order of magnitude increase in the capacity, for little change in cost, will occur in the near term.

The hardware trends will have several effects on systems similar to those in the examples above. Embedded computer systems will find utility in virtually every piece of electronic equipment if only to prompt the operator in proper usage and to diagnose failures.[17] Operation of systems using both computers (or equipment) and humans will be reconsidered for proper man-machine balance. Three key aspects of such a revised system methodology are:

- Give equal status to people and machines when considering requirements;
- Design well structured interfaces to ease the replacement of functions with technology improvements, and

- Place highest regard for the integrity of the system that multiprocesses people and machines, especially in terms of means for specifying and validating the system.[18]

Packaged software may find wide success as part of packaged systems. The added cost of a complete hardware package to accommodate the packaged software will often be less than the value it adds for the customer. This factor will reduce the major requirements for host machine independence of the software and the market will be further improved. (Customized hardware with packaged software will be less expensive than use of software that is host machine independent.) Packaged systems of great variety[19] are expected to become available wherever a clerical or critical rote function is performed, and where large scale computations can lead to even minor improvements in 'profit'. The development of software itself provides an unusual opportunity[2021] for a packaged system. Such packaged systems may be no more successful at satisfying all of a customer's specific requirements than current packaged systems. However the continuing reduction in system costs will reduce the desirability of customizing local systems. This expected effect is analagous to the present state of scientific calculators: only in extreme cases will a customized calculator be developed by an end user.

Software costs in actual dollars will most likely grow with the addition of many new systems in the near future. The effectiveness of Navy systems will surely enjoy more than comparable growth.

CITED REFERENCES

1. B. W. Boehm, "Software and its Impact: A Quantitative Approach," *Datamation,* 19(5):48–59 (1973).
2. H. R. J. Grosch, "The Challenge of Doing the Right Thing," in *Effective vs Efficient Computing,* Prentice Hall, Englewood Cliffs, NJ, 1973, pp 1–11.
3. B. C. DeRoze, "An Introspective Analysis of DOD Weapon System Software Management," *Defense Management Journal,* 11(4): 2–7 (1975).
4. J. A. McCall, et al., "Factors in Software Quality," General Electric Company interim technical report, Oct 76.
5. A. J. Driscoll, "Software Visibility and the Program Manager," *Defense Systems Management Review,* I(2): 12–27 (1977).
6. R. McCarthy, "Applying the Technique of Configuration Management to Software," *Defense Management Journal,* 11(4): 23–28 (1975).
7. R. W. Wolverton, "The Cost of Developing Large Scale Software," in *Practical Strategies for Developing Large Software Systems,* Addison-Wesley, Reading, MA, 1975, pp 73–100.
8. R. V. Head, "The Packaged Program," *Effective Program Development: The Choices, Data Processing Digest,* Los Angeles, CA 1969, pp 11–25.
9. The John Hopkins University, "DOD Weapon System Software Management Study," Silver Spring, MD May 75.
10. J. H. Manley, "Embedded Computer System Software Reliability," *Defense Management Journal,* 11(4): 13–18 (1975).
11. W. R. Archibald and E. S. Hinton, "Critical Needs in Avionic System Software," *Proceedings 1974 NAECON,* pp 475–481.

12. W. I. Fletcher, *An Engineering Approach to Digital Design,* Prentice Hall, Englewood Cliffs, NJ, 1977. (DRAFT)

13. Dept of Navy, "CADMSS Automated Data System Plan," Pacific Missile Test Center, Point Mugu, CA, Oct 75.

14. _____, "The Versatile Training System Compendium Naval Weapons Center," Memorandum Reg. 31408–119–77, May 77.

15. L. Lemon and R. Fryer, "Software Validation Using a Microcontroller/Microcomputer System," paper, Eastern Conference, IEEE, Washington, DC, Sep 76.

16. D. A. Hodges, "Microelectronic Memories," *Scientific American,* 237(3): 130–145(1977).

17. Dept of Navy, Naval Air System Command, "Avionics Systems and Technology for the 1990 Time Period," AIR 533 study group report, Washington, DC, Mar 77.

18. H. D. Mills, "On the Development of Systems of People and Machines," unpublished manuscript, Aug 75.

19. E. R. McLean, "Assessing Returns from the Data Processing Investment," in *Effective vs Efficient Computing,* Prentice Hall, Englewood Cliffs, NJ, 1973, pp 12–25.

20. P. Freeman, "Automating Software Design," *Computer,* 7(4): 33–38(1974).

21. H. Bratman and T. Court, "The Software Factory," *Computer,* 8(5): 28–37(1975).

Mr. Richard E. Fryer, a physicist, is Head, Computer Engineering Branch, Avionics Division, Naval Weapons Center, China Lake, CA. The Computer Engineering Branch is engaged primarily in the development and operation of simulation systems that are used for tactical avionics systems. Mr. Fryer has developed computer monitoring techniques for test and evaluation of software based tactical machines.

In the area of man-machine problem solving Mr. Fryer's interests are focused on aspects of computer graphics and on embedded computer/controller systems. He has published several articles about computer monitoring and graphics and was a member of the Association for Computing Machinery Committee to plan a graphics standard.

Mr. Fryer received his BS from Eastern Illinois University in 1962 and his MS from Michigan State University in 1964.

NAVY AIRBORNE WEAPON SYSTEM SOFTWARE ACQUISITION

by

Dennis W. Farrell, Department of the Navy

In the past 10 years, weapons systems acquired by the Department of Defense (DOD) have become increasingly dependent upon digital computers and computer programs. Unfortunately, experience has shown that the computer programs often were not delivered on time. Software became the major contributor to system cost and failed to reliably meet user requirements and expectations. This article provides an overview of steps taken at three levels—DOD, the Naval Material Command and the Naval Air Systems Command—to solve these problems. The requirements that have resulted from these steps and application of the requirements to the F-18 program are discussed.*

DOD INITIATIVES

As the dimensions of the so-called "software problem" became apparent, the Department of Defense initiated several measures to define, examine and alleviate the problem. Notable was the establishment, in 1974, of the DOD Management Steering Committee for Embedded Computer Resources with representatives from the Assistant Secretary of Defense level.** The committee was charged with identifying and carrying out a comprehensive solution to the problems of weapon system computer and software acquisition, management, and use. To assist in providing a firm basis for software problem identification, the Applied Physics Laboratory (APL) of Johns Hopkins University and the MITRE Corporation were asked to conduct separate, but coordinated, studies. The study efforts were directed to a review of the results of previous studies, a review of software design and management practices in selected weapon system acquisitions, and to consultation with both service and industry organizations. Figure 1*** shows those problems most often identified in previous studies. Problem relationships to one another and to the principal phases of the system life-cycle are illustrated.

Based on the results of the Applied Physics Laboratory and MITRE studies and on other findings (including those of the Joint Logistics Commanders), Department of Defense Directive (DODD) 5000.29, "Management of Computer Resources in Major Defense Systems," was issued. The directive established "policy for the management and control of computer resources during the development, acquisition, deployment and support of major Defense systems." The directive addresses four major areas in computer resources and software acquisition: management and planning, requirements analysis and validation, supportability, and language standardization. Emphasis is placed on applying lessons learned in hardware system acquisition to

*See References 1 and 2 for additional information concerning requirements and applications. Reference 1 is about pertinent DOD and Air Force directives and the implementation of same. Reference 2 addresses software acquisition activities within the Naval Air Systems Command.

**The Management Steering Committee for Embedded Computer Resources is now chaired by the Office of the Under Secretary of Defense (Research and Engineering) and includes representatives from the various Assistant Secretaries of Defense, as well as from the Military Departments and Defense Agencies.

***Taken from Reference 3.

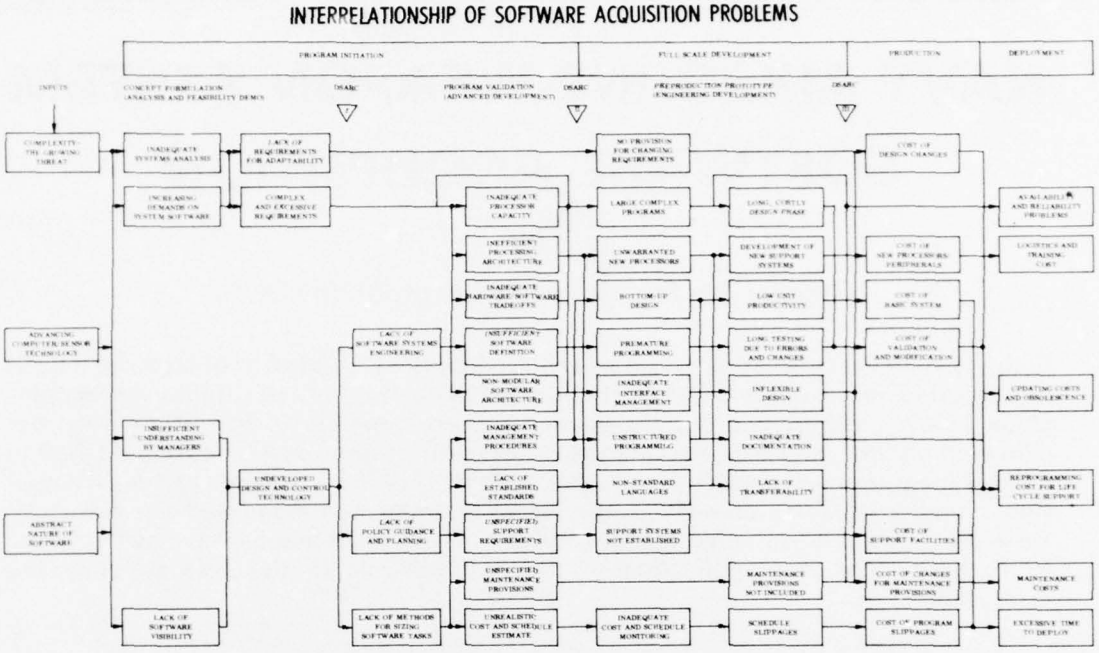


Figure 1. Interrelationship of Software Acquisition Problems

the computer and software field. These lessons emphasize:

- the importance of early planning for the entire life-cycle,
- careful determination and documentation of requirements and the design, and
- the necessity for configuration management.

Based on software experience are requirements for acquisition of support software such as compilers, simulators, and test aids, and requirements for the development and use of standard high order languages (CMS-2, JOVIAL, or a future Department of Defense standard language).

In past acquisitions, computer software often was treated as data, rather than as a critical subsystem of the total weapon system. This approach did not provide for visibility into the software development process. Lack of in-process review and testing delayed discovery of software deficiencies until acceptance test, or in some cases until operational use. By identifying computer software as a system element of major importance throughout the system life cycle, and by treating it as property rather than data, DOD intends to insure that future software acquisitions will provide capabilities and quality commensurate with cost. The intent is to attain a product that will be supportable throughout the life of the system. As a part of this insurance, DODD 5000.29 includes a specific requirement that the requirements, management, initial development, and life cycle planning for weapon system computers and software be treated at the Defense System Acquisition Review Council (DSARC) II review.

NAVAL MATERIAL COMMAND

The Department of the Navy developed a management approach to software problems based on careful documentation, configuration management, and organized in-service support for both ship and airborne weapon system software. The types, contents, and formats of documentation are established as Navy-wide requirements by Secretary of the Navy Instruction (SECNAVINST) 3560.1, "Tactical Digital Systems Documentation Requirements." This instruction defines the interrelationships of the chain of user requirements, technical requirements, design, description, and user documents that are required. Of significance is the fact

that the instruction was issued by the highest levels of the Navy organization, supporting the importance of adequate, timely computer software documentation. The Naval Material Command (NAVMAT) has further promulgated instructions specifying policies and procedures in each of the mentioned areas. Further implementation was left to the individual Systems Commands. The NAVMAT directed[4] that configuration management policy and procedures that previously applied only to hardware be applied to software. This direction specifically included establishment of software change control boards at appropriate levels to insure that all software interfaces are considered in proposed change actions.

To provide for continuing support of software after acceptance by the Navy, NAVMAT also required[5] that the activity to provide the support be identified at least 1 year before the planned support date. This minimum time requirement is to allow the support activity to build staff and facilities as required, and provide technical assistance to the development activity. Technical involvement of this type allows for application of experience gained from previous programs and helps to assure that the final software product is supportable.

The Naval Material Command established the Tactical Digital Systems Office (TADSO) that reports directly to the Vice Chief of Naval Material. This Systems Office is responsible for developing computer and software policy and guidance within the Naval Material Command and addresses both acquisition requirements and standardization considerations on a continuing basis.

NAVAL AIR SYSTEMS COMMAND

Early in 1972 the Naval Air Systems Command (NAVAIR) conducted an internal study of software management policies and procedures. The review resulted in assignment of the NAVAIR Avionics Division as responsible organization for technical adequacy, standardization, and supportability of NAVAIR weapon system computer and software acquisitions. The Avionics Division received responsibility for implementing the growing Department of Defense and Naval Material Command guidance discussed above. With regard to weapon system software, this responsibility specifically includes that of developing instructions addressing:

- **software documentation standards,**
- **software life cycle management planning,**
- **software maintenance/support activity planning, and**
- **programming language selection.**

The tasks assigned to the Avionics Division were complicated by rapid changes in software technology that affect software acquisition management methodology. Example: Although the techniques of structured programming were presumed to provide high quality software, the approaches to, and definitions of, structured programming were so diverse as to be contractually unenforceable. Further, while software quality was an established intuitive (and in some cases, analytic) concept, lack of suitable quality measures made it difficult to specify, in a contract, software quality requirements.

The Avionics Division established an advisory panel of experienced software developers and managers. The panel members were selected from Navy activities involved in development of airborne weapon system software. The purpose of this Naval Air Software Management Advisory Committee (NASMAC) was to insure that the instructions developed by the Avionics Division reflected the Navy's actual experience in the development and acquisition management of airborne weapon system software. With the assistance of the committee, the Avionics Division developed instructions for life cycle management planning[6] and software change review boards[7], and is developing a software management manual. Software life cycle management plans meeting the requirements of these instructions have been declared acceptable for the DSARC II review required by DODD 5000.29. Based on Navy acquisition experience, the planning instruction and the software management manual emphasize in-process reviews, audits, test and evaluation, and early application of configuration management. Design reviews, conducted after documentation of performance requirements and detail design, are treated as particularly[8] critical milestones.

Developers should not proceed from requirements formulation to design, or from design to actual coding, until the Navy is satisfied that the resulting system will meet user performance, quality, and supportability requirements. In-process test, and evaluation against the documented performance and design specification, is used to provide continuing visibility into the progress and success of

the development process. Configuration management is applied to each of a series of performance and design documents as the documents are approved at design reviews. This procedure assures that the documents and the developing computer program are consistent and traceable. Configuration management helps to assure that the program sections, or programs, being tested are not modified without Navy permission.

The Naval Air Systems Command was concerned also with methods and procedures for providing in-service support for weapon system software. The objective was to insure knowledge gained from that support is used in subsequent systems. To provide in-service support, Software Support Activities were established for major weapons systems, generally at Navy laboratories or centers.

Naval Air Systems Command recognizes the value of involving Software Support Activities early in the system acquisition cycle. The Avionics Division Software Management Manual, in preparation, stresses Software Support Activity involvement before preparation of the Engineering Development/Full-Scale Development contract. This action provides at least two major advantages. First, a team with extensive experience in the technical aspects of software development and support is available to furnish valuable assistance in preparing the Request for Proposal. Team input to the Statement of Work, the Contract Data Requirements List, and the Instructions to Offerors can assist the Naval Air Systems Command in establishing a viable contract. Second, continuing participation of the team in design reviews and audits helps to assure that the Navy will acquire a system that meets performance, quality, and supportability requirements. In addition to motivation provided by the technical challenge, the Software Support Activity is motivated by the knowledge that it will be living with and supporting the software product for years to come!

Based on internal impetus and the requirements of higher authority, NAVAIR has come to its current position on weapon system software development and acquisition. This position rests on two basic principles:

- disciplined management, and
- the utilization of System Support Activities.

Disciplined management provides for acquisition of software as a critical subsystem which is integral to the overall system, rather than an adjunct to it. The System Support Activity provides continuing, in-depth technical support to the Project Manager, following with trained staff and equipment to support the software throughout the remainder of the system life cycle. This current position should not be considered static. At least three forces will drive toward change. The forces include:

- utilization of current and future developments in computer and software technology,
- the increasing perception that the System Support Activity has both expertise and equipment to provide certain types of support to the total avionics system, and,
- increased emphasis on managing interfaces among the weapon system and its associated trainers and test equipment.

Countering these forces are existing organizational guidelines and missions, and increasingly stringent budget and staffing limitations.

F-18 SOFTWARE ACQUISITION

The F-18 aircraft is a single-place, twin engine, carrier based weapon system. Primary missions of this aircraft are fighter escort and interdiction. (An A-18 aircraft, an attack version of the F-18 with many common hardware and software elements, is being developed.) Principal weapon delivery system elements include an airborne radar, an inertial navigation set, head-up display, multipurpose displays, two programmable digital mission computers, and an integrated stores management system. Primary requirements of the F-18 are to provide accurate air-to-air and air-to-ground weapon delivery under one man operation. To meet these task requirements, numerous and varied computational tasks must be performed. There will be Operational Flight Programs in six airborne computers:

- Mission computers (two)
- Radar
- Inertial navigation
- Stores management
- Air data.

The mission computer Operational Flight Programs are being developed by the prime contractor, McDonnell Aircraft Company. The other Operational Flight Programs are being developed by the individual subsystem subcontractors.

The full-scale development contract seeks system performance. Design definition and implementation, within these performance contraints, is left to the contractor as the design levels become detailed. This approach, although common with such contracts, is in conflict with the increasing emphasis on detailed Navy visibility into software product processes and development. Thus the F-18 development requirements represent a dynamic compromise among contracting practices, funding, and the developing software acquisition practices. Five aspects of F-18 software development, in terms of the effect of these acquisition practices, are given here.

Standardization

The Assistant Secretary of the Navy for Research and Development required that the Naval Air Systems Command standard airborne computer, the AN/AYK-14, and its high order language, CMS-2M, be used as the F-18 mission computer. This computer, itself under development, was felt to be a risk by the contractor, as was use of an unfamiliar high order language. The risks associated with the AN/AYK-14 were those of schedule and capability to meet performance requirements. "Brass-board" versions of the computer were delivered, and will be used to develop further information on risk areas. The language issue primarily involved the capability of high order languages to yield computer programs efficient in memory and execution time usage. Additional concerns centered around the amount of computer support time required to compile the CMS-2M source language into actual AN/AYK-14 programs, and the assignment of responsibility for the CMS-2M compiler. Because of these concerns, the contractor was allowed to use lower (assembly) level programming techniques for time-critical portions of the computer program. Additional memory capacity was allowed.

Subsystem computers, provided as internal elements of the subcontracted subsystems, were not included in the processor and language standardization requirements.

The pressure for processor and language standardization has succeeded in applying a standard, Government furnished computer and high order programming language for two of the six major applications in the F-18 airplane.

Data and Documentation

Documentation of programs for the six major F-18 computer applications is to meet the requirements of Weapons Specification, WS-8506, a Naval Ordnance Systems Command specification entitled, "Requirements for Digital Computer Program Documentation." The specified requirements include deliverable performance and design specifications, detailed subprogram and data base descriptions, and computer program packages. Also, test plans and procedures for the mission computer programs are required. Data deliveries are scheduled sequentially through the development process, providing the Navy visibility into that process. The delivery of all computer programs is specified in the Contract Data Requirements List and is consistent with Armed Services Procurement Regulations.

Although WS-8506 has been superseded by SECNAVINST 3560.1, "Tactical Digital Systems Documentation Standards," documents that address detailed performance and design requirements have not been significantly changed. The F-18 software design documentation is essentially consistent with current documentation requirements.

Software Configuration Management

The Weapons Specification WS-8506 documents provide configuration identification for each mission computer program. Changes to the configuration during development are handled as either permanent or temporary changes.

Changes intended to be permanent are established by means of a Computer Program Change Request (CPCR), and require approval by all affected areas. The contractor's software documentation control group is responsible for logging and tracking all Computer Program Change Requests, assuring that accurate configuration status accounting is available. When a change is approved, the baseline program tape is updated. A complete new tape is prepared, configuration identification documents are revised, and any required modifications

to associated simulations are made. This process assures that the development team has a consistent set of programs, documentation, and simulations.

For temporary or developmental changes, for a particular test facility, a Computer Program Deviation Request (CPDR) is used. The deviation is approved for a given baseline and is for use only in affected facilities. A CPDR can never be used to alter a deliverable flight tape; to effect such alteration requires conversion of the CPDR to a CPCR, with the latter subject to full application of CPCR controls.

"Lessons learned" in configuration management of developmental hardware, when applied to software development, provide a level of control over deliverable computer programs and supporting documentation. This control should help to assure that the deliverable programs perform as documented and tested, and that the programs are supportable by the Navy.

Software Quality Assurance

The F-18 mission computer software quality assurance has three aspects: modeling, testing, and controlled reviews. The use of models is central to the development process. The FORTRAN language models run on the contractor's IBM-370 facility. These software models are used to validate the systems analysis and design approach and show that the Program Performance Specification and the Program Design Specification are adequate. These same FORTRAN models are used with a cockpit simulator to evaluate control and display interfaces with the pilot. By use of these models, alternative mechanizations can be explored, and a data base is generated to support later testing of the actual mission computer programs.

Both in-process and Navy acceptance testing will be conducted. In-process testing will be conducted in accordance with the contractor's internal procedures. Acceptance testing will include software verification testing against the Computer Program Performance Specification in accordance with a Navy-approved Test Plan and Test Procedures. The software will undergo further formal Navy testing after integration as a part of the total weapon system.

Quality assurance procedures cover the development process from performance specification to the final product. This approach should contribute to the quality of the software product and provide increased Navy confidence in that quality.

Formal and informal reviews of the software development process will include reviews of test results, and quality assurance procedures and results.

Support Software

In accordance with the requirements of DODD 5000.29, the F-18 program has provided for Navy acquisition of support software for the six major computers in the system. All support software is required to run on the Government-furnished IBM-370 at the contractor's facility. Each subcontractor responsible for developing subsystem software is responsible for ensuring that the associated support software will run on this machine. Since it is planned that the IBM-370 will be returned to the Government toward the end of system development, all necessary support software will be captured. As a hedge against changes in this plan, it is required that the support software be written in FORTRAN, to the extent possible, using capabilities available within a 16-bit word length.

According to plan, the support software and the IBM-370 will be transitioned to the supporting activity at the Naval Weapons Center. As an alternative, the support software alone can be transitioned. In either event, the approach provides for avoiding costs associated with redevelopment of support software.

OBSERVATIONS

The F-18 program, appearing in the midst of a significant transition in software management styles, has implemented many new requirements. In the absence of detailed official guidance, some of the implementation details are experimental. As the system is developed, a continuing record of "lessons learned" will provide precedents and guidance for future programs.

CITED REFERENCES

1. Lt. Col. Alan J. Driscoll, USAF, "Software Visibility and the Program Manager," *Defense Systems Management Review*, I (2): 12–27 (1977).

2. V. E. Skullman, "Impact of DODD 5000.29, Management of Computer Resources in Major Defense Systems, On the Naval Air Systems Command (NAVAIR)," Defense Systems Management College Study Project Report PMC 76–2, Dec 76.

3. "DOD Weapons Systems Software Management Study," Applied Physics Laboratory, Johns Hopkins University, technical report, AD-AO22,160, Jun 75.

4. Dept of Navy, Naval Material Command, NAVMATINST 4130.2A, "Configuration Management of Computer Software Associated with Tactical Digital Systems and Other Technical Computer Systems Developed by or for the Naval Material Command," 19 Jul 76.

5. _____, Naval Material Command, NAVMATINST 5200.27A, "Transfer of Navy Tactical Digital System Software Responsibility; Procedures for," 18 Apr 73.

6. _____, Naval Air Systems Command, NAVAIRINST 5230.5, "Responsibility and Requirements for Preparation of Software Life Cycle Management Plans," 21 Jul 76.

7. _____, Naval Air Systems Command, NAVAIRINST 5230.6, "Establishment of Tactical Software Change Review Boards," 1 Jun 77.

Mr. Dennis W. Farrell is Head, Operational Computer Systems Office, Naval Weapons Center (NWC), China Lake, CA. Mr. Farrell is responsible for providing software acquisition and development policy and procedural guidance for Naval Weapons Center programs. Entering the field of weapon system computers through the application of early solid state digital differential analyzers, he has held several performing and supervisory positions in computer hardware and software development.

Mr. Farrell is a member of the Naval Air Software Management Advisory Committee. Mr. Farrell's primary professional interests are software technology and acquisition management, as practiced in a total systems environment.

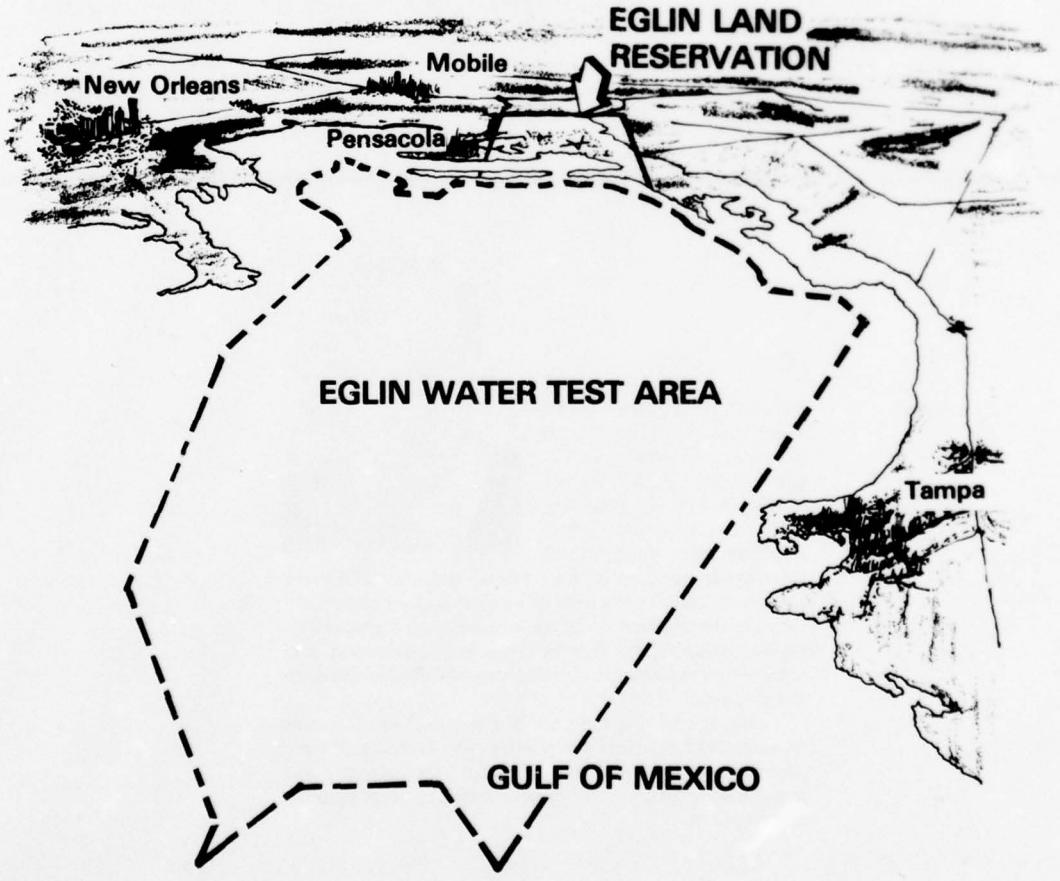
THE EGLIN REAL-TIME COMPUTER SYSTEM

by

George C. Suydan, Department of Air Force

Development of a Computer-Driven Command and Control Facility Supports the Management of Air Force Weapons Test Programs

ARMAMENT DEVELOPMENT AND TEST CENTER TEST AREAS



A PICTORIAL PRESENTATION of the location and relative size of the Eglin test areas is shown.

As its primary mission, the Armament Development and Test Center, a research and development component of the Air Force Systems Command, supports the development, procurement, and testing of Air Force tactical weapons sytems. Center headquarters is located at Eglin Air Force Base near Fort Walton Beach on the Florida Gulf Coast. The Eglin test complex encompasses extensive land and water areas. The Eglin land reservation spreads over 464,000 acres and extends nearly 42 miles along the Gulf Coast. The Eglin water test area reaches nearly 300 miles offshore and includes a great portion of the eastern half of the Gulf of Mexico. Continued productiveness of Center weapons testing is closely related to assurance that testing activities do not present hazards to private activities in the surrounding areas and on the high seas.

THE NEED

In the past, test safety could be accommodated by isolating tests to the vast expanses of land and water in the test complex or by placing restrictive limits on test conditions. The more modern weapons systems, however, are characterized by increased energy, aerodynamic performance, and destructive power. In the late 1960s it became apparent that population pressures and trends in modern weapon design soon would result in safety restrictions so stringent that test productiveness would be limited and thus degrade the effectiveness of the Center's weapons testing mission. Conclusions resulting from a study of this problem indicated that an instrumented test control capability could be developed—a real-time computer system that would assure test safety and vastly expand the potential for cost effective use of Center test resources.

THE APPROACH TO THE PROBLEM

The basic concept of an instrumented test control capability requires extensive real-time monitoring of all facets of a test operation. This permits immediate detection of hazardous or unfavorable conditions and initiation of prompt corrective action. Feedback systems are provided so that corrective action instructions can be communicated directly to test participants; or, remotely controlled transmitters can be used to alter or terminate a flight or a malfunctioning test item.

Estimated costs for procurement of such a system from commercial sources greatly exceeded funds available, and 3 years was a typical schedule quote. The need was for a capability within half that time, and only limited funds could be programmed for associated expenditures. The greatest costs were associated with the engineering of the computer complex and the generation of systems and applications software. The Center possesses analytical and engineering capability within its own personnel resources. Hence, the decision was made to develop the required capability in-house with fund expenditures mainly limited to procurement of replacements for obsolete display and interface equipment.

THE PROJECT PLAN AND THE TEAM

The development activity was organized under a dedicated project team concept. Project direction, engineering, programming, and production tasks were accomplished by Center engineers and computer scientists. Operational requirements and basic design concepts were provided by the ultimate users of the system, especially range safety and test operations agencies. System integration activities involved instrumentation operation and maintenance groups as well as the developers and users of the system. Talented and productive people were required to meet schedule and cost limitations. All team members were carefully selected with special attention given to individual skills and proved performance records.

Adherence to straightforward guidelines was stressed to assure effective utilization of the project team. Three of the most important factors in the project plan were requirements, operations concept, and systems design.

Requirements—Emphasis was given to immediate needs so that existing problems were assured of a near-term solution. Specific and realistic goals were established. In short, requirement goals were reasonable.

Operations Concept—Test operations were centralized at a single control site. The resulting system minimized impact on existing operations activities and test management. An orderly integration of the system through simultaneous development of systems tests, documentation, training plans, and procedures was assured. In short, the plan was workable!

Design—Designs were to assure satisfaction of the requirement above all. Future growth and expansion were not precluded. Risky or questionable techniques or subsystems requiring research and development in themselves were avoided. In short, the plan was kept simple.

THE SYSTEM

An extensive instrumentation complex and a data handling network were already in existence. However, these systems were oriented toward supporting remote special purpose control sites, necessitating relocation of control functions to a newly designed central facility. Some data handling equipment and new cathode ray tube display devices were provided. An extensive communications network was utilized to tie the central facility to remote test ranges and instrumentation sites.

Most of the weapons test scenarios have common requirements in that vast quantities of data must be collected and presented to controllers in forms suitable for decision making. This function required considerable computational powers beyond the capability of any single Center computer. However, three digital computers were available that, with suitable division of function, could easily handle the workload. In the resulting system, an IBM 360/65 computer provides master real-time controller functions, a Control Data Corporation CDC 6600 computer performs mathematical solutions of various test algorithms in a multiprogramming environment, and a Digital Equipment Company PDP-15 computer provides processing of telemetry data. Real-time multiprocessing with these computers required the generation of an extensive package of systems routines compatible with vendor supplied operating systems. The systems software so generated provides the basic data base management, intercomputer communications, and display driver functions. The systems software is maintained independently from specific test requirements.

In order to support tests, applications programs peculiar to test support requirements were developed. These applications programs function under control of the system software. The resulting application software library consists of various packages covering testing functions in support of range safety, test control, and range operations. Individual applications programs are kept small to permit simultaneous support of multiple tests. Dedication of one or more of the computers to a single test or application is avoided with this design. As objectives change from test to test, only the applications program need be modified. Configuration control problems of the more complex system software are minimized, and reliability is kept at a high level.

THE SAFETY PROGRAM

Use of the system in support of hazardous tests required the development of an application program for range safety. To control and direct this effort, an additional management entity separate from the project team was formed. This Configuration Control Board is composed of members from the software development agency and the using organization. The Board is charged with responsibility for managing the life-cycle application software package, to include production, debugging, operational integration, and maintenance. In the case of the range safety function, since the security of lives and property may depend upon the software used for test support, configuration control for this application package is of great importance.

THE APPLICATION

The software resulting from this development enables test support to be conducted with required levels of safety and control. Pretest, real-time, and post-test applications are supported with the following features available.

Prior to launch or release of a test item, various instrumentation and system checks are conducted to assure test readiness. During the test, the computer complex processes data from range instrumentation and presents it, in the control center, to range safety officers and test controllers in forms suitable for decisionmaking and control purposes. Various presentations are provided on cathode ray tube display devices (at operator request) during the course of the mission. The information displayed may consist of maps, graphs, or parameters that have been derived from appropriate simulation models. The maps depict Eglin test area outlines, coastlines, boundaries, sites, facilities, and targets with appropriate grids, annotation, and labeling. The maps are used to plot space positions of airborne items, targets, and aircraft in real time. Weapon hazard footprints and impact areas are overlayed on the map backgrounds for safety adjudications. Also presented are various test parameters resulting from conversion of radar and telemetry data or the solution of an algorithm or simulation model involving the input data. Presentation of

these parameters may be in alphanumeric digital form or graphic presentations. Post-test evaluations frequently require playbacks of recorded data to re-create the test for purposes of quick-look report generation, data reduction, or investigation of test anomalies.

SAFETY AND TEST *control is exercised through display and control mechanisms provided by this* console station. Displays, communications, and command control functions are available.

SUMMARY

The total system was completed on schedule and proved successful from date of first test application. Test operations have been supported with this system since 1975. Safety, flexibility, and test productivity have been improving continuously as experience in system use grows. Also the system has provided a baseline for a continuing phased development of an improved advanced centralized control facility reflecting future test requirements. The experiences and lessons learned in the development of the initial system have generated a corporate pool of knowledge that provides further assurance that future developmental actions can be addressed with confidence.

George C. Suydan is Chief, Analysis Branch, Range Support Division, Armament Development and Test Center, Eglin AFB, Florida. In prior assignments he served as a systems analyst at the Pacific Missile and Test Center, Point Mugu, California, and systems project engineer for the Space and Missiles Test Center, Vandenberg AFB, California. Mr. Suydan's experience includes design and implementation of real-time range safety systems and test control facilities in support of various weapons and missile test programs.

Mr. Suydan holds a BS degree in Chemical Engineering (1961) and an MS degree in Chemical Engineering (1963) both from the Univ of Nebraska.

SOFTWARE RELIABILITY BY DESIGN: A CRITICAL NEED

by

W. J. Willoughby, Department of the Navy

STATEMENT OF THE PROBLEM

Modern Navy weapon systems are becoming increasingly dependent on embedded digital computers.

The F-14 fighter aircraft contains three. Without these, its primary weapon system can be extremely degraded; its mutually dependent fire control, navigation, and flight control systems may be largely ineffective and complex aerial maneuvers and manual flight control become hazardous.

Two of the Navy's newest ships, both CGN36-class cruisers, lacked full combat capability for several years following launch and commissioning because the multiple computers which control their entire weapons array would not operate properly due to software problems. Stories of faulty software in Navy weapon systems computers continue unabated and are increasing with system complexity.

Over the last four years, the Navy has made giant strides in revamping its whole approach to reliable hardware. Abandoning ineffective numerical requirements and demonstration testing, the Naval Material Command has adapted, from NASA experience, a set of acquisition fundamentals (Figure 1) whose conscientious application to hardware acquisition programs has resulted in reliable hardware. But those weapon systems whose operation depends upon digital computers, whether tactical (embedded software or firmware) or dedicated general purpose (user-programmable), continue to be plagued with software failures. In terms of the systems' abilities to function, such failures are as real as hardware failures. Why can't a similar set of acquisition fundamentals be identified and applied to the software?

The Naval Material Command is confident that such fundamentals *can* be developed, resulting in a transition of reliable software design, testing and maintenance from an art, which describes the present state of software reliability achievement to a real science. This belief is reinforced by the multitude of independent efforts underway in this direction and the limited success each has attained even though software failures are continuing.

What remains to be done is to integrate the successful design and test approaches into a unified body of fundamental software acquisition program requirements, which can then be written into contracts and reviewed for compliance at periodic program milestones. This is a major objective of current efforts by the Naval Material Command to reduce drastically the incidence of software-related failures in otherwise reliable weapon systems.

The purpose of this article, therefore, is to encourage the software community:

1. to re-orient its thinking toward software design disciplines and techniques which are known to avoid or minimize the likelihood of problems; and

2. de-emphasize the traditional approach which utilizes standard programming methodology followed by exhaustive testing to discover errors and conflicts. An oft-illustrated example showing the futility of this traditional approach bears repeating.

OBJECTIVES

- IMPROVE FLEET READINESS
- MINIMIZE LIFE CYCLE COST

ACQUISITION FUNDAMENTALS

- CONTRACT FOR RELIABILITY
  - Requirements Not Goals
  - Incentives For Reliable Design
  - Reliability in Source Selection
  - Life Cycle Cost Consideration
- DESIGN TO MINIMIZE FAILURE
  - Mission/Environmental Profiles
  - Design Alternatives Studies
  - Numerical Allocation
  - Conservative Derating Criteria
  - Stress Analysis
  - Sneak Circuit Analysis
  - Worst Case Tolerance Analysis
  - Failure Modes & Effects Analysis
  - Parts & Materials Selection/Screening
  - Design Reviews
- INTEGRATE TESTING TO VERIFY DESIGN
  - Missile Profile Development Test (TAAF)
  - Design Limit Qualification Test
  - Mission Profile Demonstration Test
  - Failure-Free Random Vibration Acceptance (Electronics)
  - Failure-Free All Equipment Screening
- PREVENT FAILURE RECURRENCE
  - Failure Reporting
  - Failure Analysis
  - Corrective Action
- SUSTAIN RELIABILITY IN PRODUCTION
  - Quality Assurance
  - Process Controls
  - Acceptance Testing & Inspection
- SUSTAIN RELIABILITY IN SERVICE USE
  - Initial Fleet Tracking
  - Contractor Corrective Action Responsibility

IMPACT

- REDUCE MAINTENANCE/SUPPORT BURDEN
- INCREASE CERTAINTY OF RELIABLE MATERIAL ACQUISITION
- STRENGTHEN NAVY/CONTRACTOR TECHNICAL TEAM

Figure 1. Navy Acquisition Fundamentals

Consider the extremely simple computer program represented by the flow chart in Figure 2. (Navy operational programs may be orders of magnitude more complex.) Largely due to the loops (which are very common in current computer programs) there are some 10^{20} different sequences (paths) which could be followed during program execution. If a single path could be checked every nanosecond (10^{-9} second), and if this debugging had started at the beginning of the Christian era (1 A.D.), the job would be about half done at the present time. Clearly, programs can never be fully tested. Software reliability by design, not by debugging, is therefore a critical need in the acquisition of reliable software-based Navy weapon systems.

WHERE THE ACTION IS NOW

Practitioners in software design, and its integration into military hardware, have become acutely aware of and sensitive to some of the basic

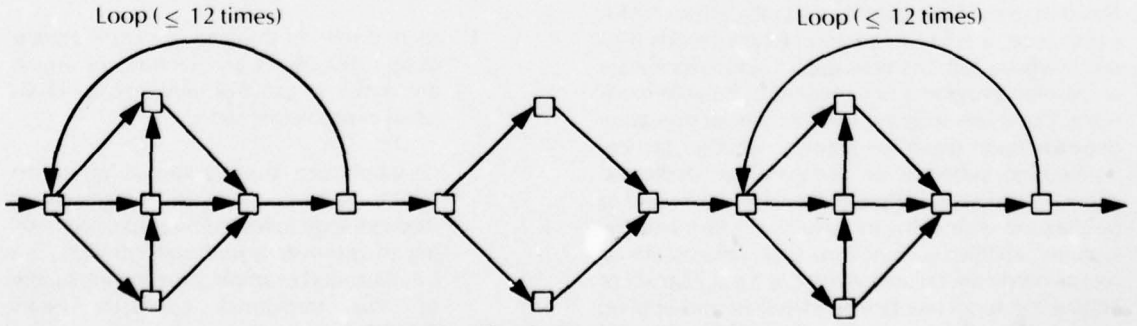


Figure 2. Simple Program Flow Chart

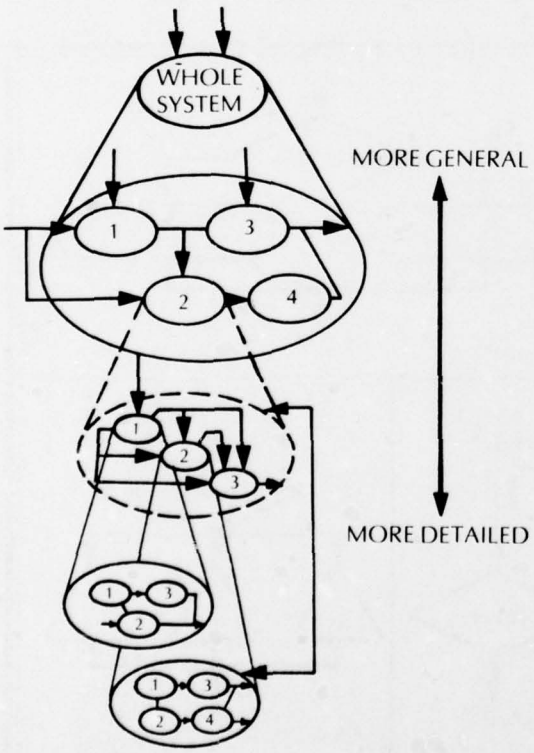
sources of software failures. Out of this awareness is gradually emerging a better understanding as well as the rudiments of a more disciplined, scientific approach to the problem. Software reliability experts are few and far between, and even the term "software reliability" is a recent concept; nevertheless progress is being made towards acquisition fundamentals for the production of reliable software. The following paragraphs discuss some of the currently recognized problem areas and describe some of the actions underway to resolve them.

Modular Design Methods

It has become axiomatic to software developers that the two most common sources of software system problems are (1) requirements analysis errors, and (2) design/coding errors during development. Recognizing that significant cost-savings result from early detection and correction of such software problems, system developers have turned to the so-called "modern programming practices" in order to gain control over the stages of requirements definition, program design and coding. Some, but by no means a complete list, of these practices that are continually being evaluated and refined are listed below—

Requirements Definition. Defining system operational and technical performance requirements is really a series of investigations to discover the required output of each level cumulatively in the system. As with later design, coding and testing steps, it is most logical to start with the required total system output and work down toward smaller, more detailed components that are included therein. When defining system requirements, a trade-off analysis is an important step because it can disclose significant life-cycle cost savings through proper assignment of hardware/software roles. Among the most essential aspects of this analysis are *hardware/software interface specifications*, that should be important topics at preliminary system design reviews.

Top-Down Design. Using this technique, the software is initially designed from the top down, as illustrated conceptually in Figure 3. The designer



- **WORK FROM GENERAL ITEMS TO SPECIFICS**
- **COMPLETE EACH LEVEL BEFORE GOING ON TO LOWER LEVEL**
- **ASSURE TRACEABILITY**

Figure 3. Top-Down Design Approach

starts with the highest level control function and works down to the lowest processing elements. The design of each level is completed as extensively as possible before the next lower, supporting level is started. Simulated inputs or outputs are used where related modules are not yet completed. Various flow charting techniques may be used to map control and data flows. Regardless of the kind of top-down technique employed, it is essential that complete traceability be assured from one level to the next.

Structured Programming. This technique is used to implement the top-down design by carrying the modular approach to lower, more detailed levels. The modules are separately coded and documented using a limited number of program structures in the necessary combinations. Figure 4 illustrates the five basic structures, each of which has only one data entry and one exit. One intent is to make the structure of the design match the structure of the program so that changes to parts of the specification result only in changes to small parts of the program. Use of proven *standardized application modules* further minimizes the opportunity for the software failures.

Chief Programmer Teams. This programming organization provides a subject management tool which complements the structured programming approach. For a small system a single team may be involved. For large or complex systems, the programmers may be organized into a hierarchy of teams, each responsible for a particular portion of the program. Each person's function is specifically defined and the group is built around three primary individuals:

- Chief Programmer—provides wide design/coding experience and team management.
- Back-up Programmer—assists the Chief Programmer and provides peer-level design review.
- Librarian—maintains the program development library and monitors the program status.

Additional support members may be included as necessary. The essence of the Chief Programmer

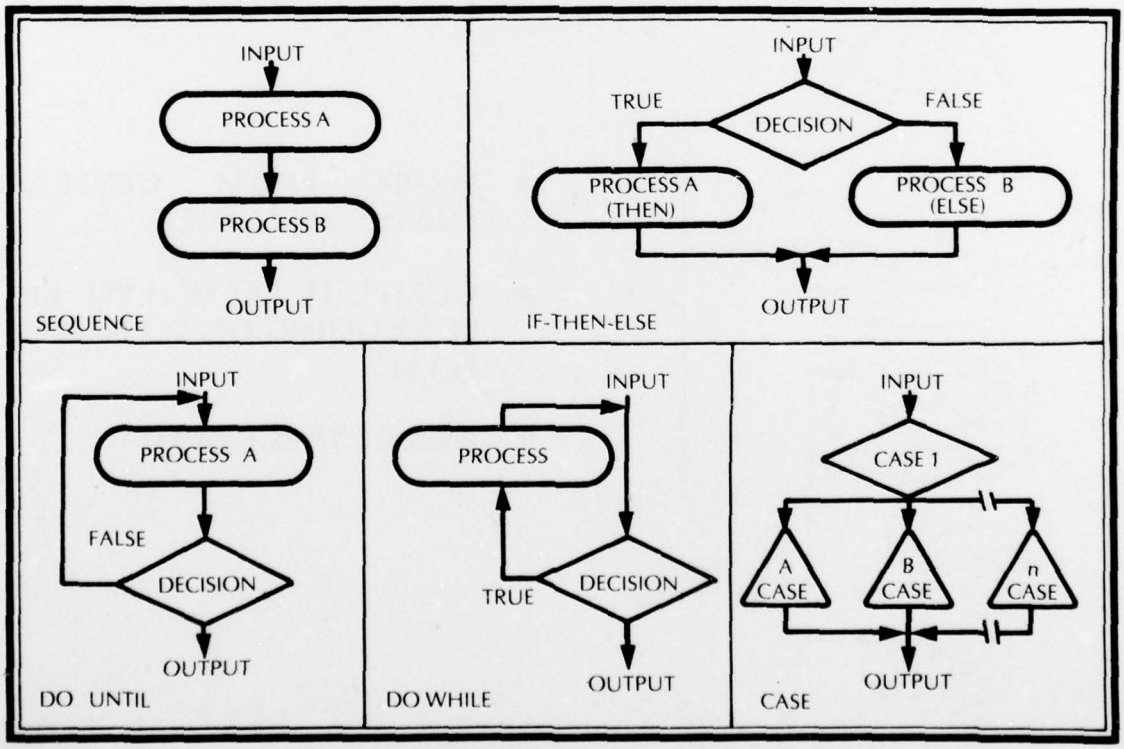


Figure 4. Basic Program Structures

Team concept is to provide an efficient organization for implementing the structured programming approach and other *programming standards.*

Structured Walk-throughs. These reviews are set up to periodically allow the programmer for a particular module the chance to explain his design and coding approach to the rest of the team. This critique provides a measure of fresh thought to the programmer and lends a degree of cohesiveness to the overall team effort. *Early documentation* at the modular level is an important adjunct to this close intercommunication.

Higher-Order Languages. For Navy software systems, compatibility and standardization require that certain approved higher-order program languages be used. These languages are:

- For Tactical Systems*: CMS-2
- For Non-tactical Systems**: COBOL, FORTRAN IV, Basic, APL, PL-1

Software Design Tools. The number and variety of available automated tools (programs) for software design is growing continually. Where these tools are available to developers, and will realize life-cycle cost savings, they should be used. Commercially available tools include some capable of analysis techniques derived from accepted hardware design disciplines, such as: software sneak circuit (path) analysis and worst-case analysis.

Error Tolerance and Error Recovery. There is growing awareness that no complex software system can ever be designed totally error-free. Hence, designers must consider the ramifications of designing the system to be so sensitive to data or logic errors that numerous system faults and shutdowns may occur. An analysis must be made of what types and levels of errors are tolerable under full and reduced capability conditions. The software must be accordingly designed and coded for this tolerance and/or rapid recognition and recovery from errors.

*"Standard Shipboard Tactical Digital Processors and Program Language," TADSTAND 1, Naval Material Command, Code 09Y, Washington, DC, 29 May 73.

**"Standard Higher Level Digital Computer Programming Languages; Policy for," DOD Directive 7905.1, Department of Defense, Washington, DC (Draft).

Verification and Validation

Verification and validation (V&V) includes the systematic evaluation of a program by an agency independent of the program developer. Program development management employing V&V produces an orderly development process. A major advantage is that many errors are found early in the development cycle which otherwise would be discovered only through the use of elaborate simulation or field exercise. Verification and validation aids program development in a number of ways that are traceable to its employment as a structured activity proceeding in parallel with program development, and establishing formal review, communication, and program modification criteria. Establishing V&V as a *formal* activity with strict accountability and reporting procedures has the important results of causing the computer program development contractor or subcontractor to adhere more rigidly to programming and documentation constraints and standards, and of requiring program documentation and code deliveries on a time-phased basis. Thus, a well-planned V&V effort not only has the effect of improving end-item quality and reliability, but also serves as an important program management tool, making the program development cycle highly visible

The application of automated and manual validation processes is scheduled by considering the type of program being validated, the resources available (test beds, automated tools, etc.), the personnel available, and the schedules. The term "V&V" includes all the processes which are applied by an independent organization leading to the certification of a program. Certification that the program can fulfill its mission is accomplished by:

- assuring that each level of documentation has been translated completely and correctly to the next level,
- that the final object code executes correctly,
- that the input medium to be loaded operationally is identical to the code which underwent test, and
- that unused or unnecessary coding which could subvert the intent of the program is not present.

More specifically, V&V refers to the processes demonstrating that a set of requirements or specifications have been correctly translated into the next lower set of documentation, including the actual program code. The verification and validation processes are differentiated and defined more explicitly in Figures 5 and 6, respectively.

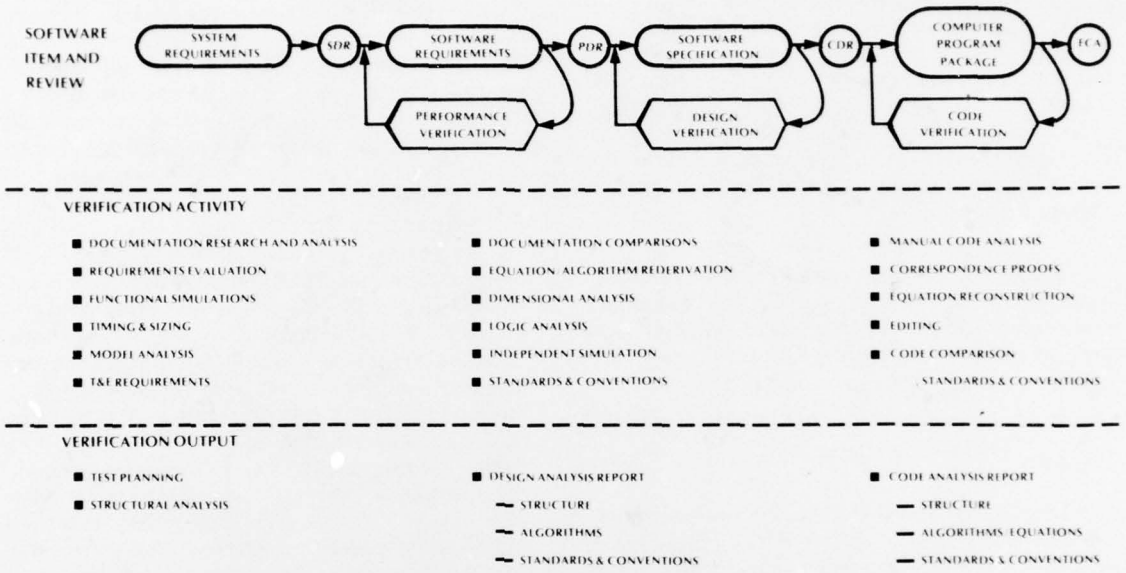


Figure 5. Verification Process

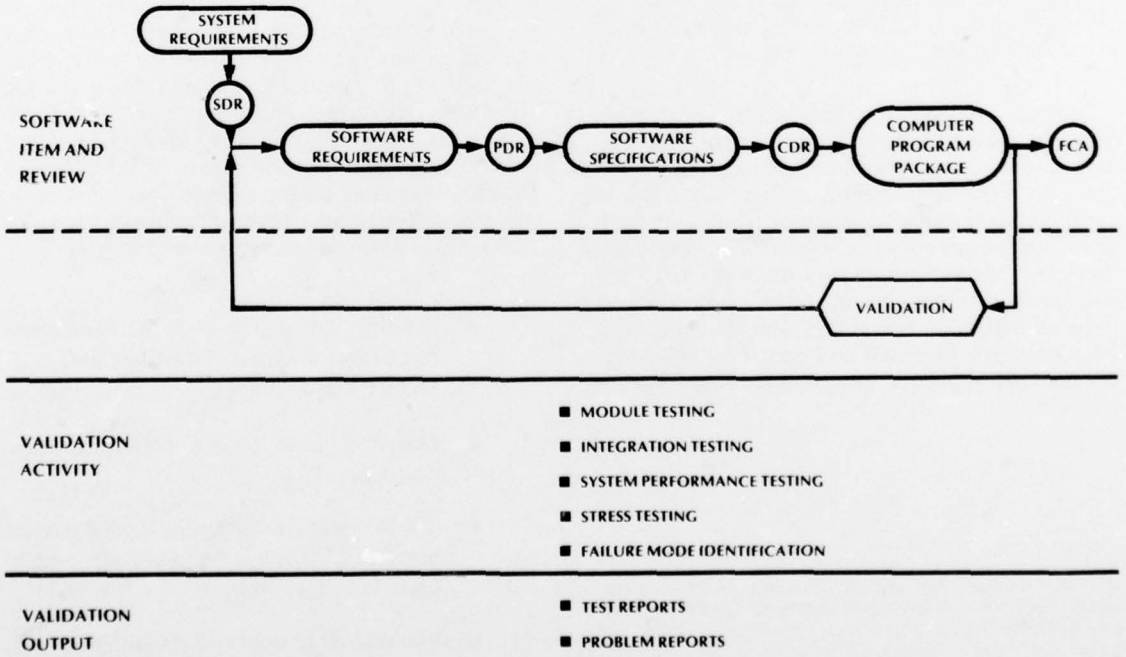


Figure 6. Software Validation Process

Documentation Requirements

Complete and correct software documentation is an essential element of a successful software system development effort. The software documentation is comprised of various categories of descriptive documents which act as building blocks for the design, verification, and validation of the computer program. Therefore, it is essential that its preparation be initiated at the earliest possible point *during* the software/hardware design phase, as a vehicle for team intercommunication and for module interface compatibility. In addition, the documentation is the only tangible product of software design which may effectively serve as a vehicle for project monitoring and configuration management. Once the design has been completed and validated, the documentation becomes a valuable by-product of the development effort and can be used regularly by the Software Support Activity maintaining the system after its deployment.

All software design documentation for tactical Navy applications is governed by Secretary of the Navy Instruction (SECNAVINST) 3560.1.* The instruction contains format and content requirements for all the required documents and specifications. It also contains sample data item descriptions to be included in such software development contracts. All Navy software under development for non-tactical systems is documented according to Department of Defense Standard 7935.15 of September 13, 1977.** This manual provides format and content requirements for all software development documents and, in addition, offers guidelines for acquisition managers and acquisition engineers to determine documentation requirements, based on a system complexity analysis.

In addition to this traditional kind of documentation, the advancement of modern programming techniques has engendered the increased use of program support libraries (PSLs). The PSL is a repository for development data which is stored in two forms—machine readable form and hard copy notebook form.

Support of the programming process involves support of the design, coding, testing, documentation, and maintenance of computer programs and the associated data-base definitions. A PSL provides this support through:

- Storage and maintenance of programming data.
- Output of programming data and related control data.
- Support of the compilation and testing of programs.
- Support of the generation of program documentation.
- Collection and reporting of management data related to program development.
- Control over the integrity and security of the data stored in the PSL.
- Separation of the clerical activity related to the programming process.

The PSL is of special value in large, complex systems where the work of individual programming teams on distinct modules must be carefully managed.

Configuration Management

Configuration management (CM) accomplishes the following tasks:

- Identifies and documents the physical and functional characteristics of the software
- Controls any changes to those characteristics in accordance with MIL-STD-480* and the software portion of MIL-STD-483**
- Verifies that approved changes have been made
- Maintains a complete change status record

*"Tactical Digital Systems Documentation Standards," SECNAVINST 3560.1, Office of the Secretary of the Navy, Washington, DC, 8 Aug 74.

**"Automated Data System Documentation Standards Manual," DOD Manual 4120.17M, Office of the Assistant Secretary of Defense (Comptroller), Washington, DC, Dec 72.

*"Configuration Control—Engineering Changes, Deviations and Waivers," MIL-STD-480, Department of Defense, Washington, DC, 30 Oct 68.

**"Configuration Management Practices for Systems, Equipment, Munitions and Computer Programs," MIL-STD-483, Department of Defense, Washington, DC, 31 Dec 70.

Under the "modern programming practices" umbrella, configuration management is a most vital means of insuring adherence to hardware/software interface specifications, documented programming standards, and particularly software documentation requirements. Because of the ease of making changes in software, perhaps nowhere else is configuration management so important.

Software Testing

There are three categories of software testing that are appropriate to the software reliability effort of a typical acquisition:

1. **Subprogram/Module Tests.** This testing is done to determine compliance with technical, operational and performance specifications. As a minimum, each subprogram or module should be tested to:

   - Ensure that it actually reflects the performance and design specification requirements.
   - Ensure error-free compiling and assembly of the coded subprogram or module.
   - Exercise the subprogram or module in terms of input and output performance with results reflecting the applicable performance and design specification requirements.

2. **Function Tests.** Once the subprogram or module testing has been satisfactorily completed, the subprograms are combined into subsystem programs and similarly tested.

3. **System Performance Tests.** The subsystems are then integrated and tested to:

   - Verify the total man-machine interface.
   - Validate system initiation, data entries via peripheral devices, program loading, restarting, and the monitoring and controlling of system operation from display consoles and other control stations as applicable.
   - Conduct an error-free retest of any system failures, following complete analysis and understanding of the causes of those failures, and appropriate corrective action to preclude their recurrence.

WHAT REMAINS TO BE DONE

The many current actions and approaches to improving software reliability have been demonstrated effective in recent programs in which they have been implemented. But they beg the question which remains: What specific factors or disciplines *influence* software reliability, and to what extent? The extent to which reliability is influenced is important only insofar as it would enable a ranking of such factors on which priorities and limits could be established; any attempt to quantify software reliability would be misguided effort.

In order to convey to the reader a better understanding of this critical need, some potential factors which could or are known to influence software reliability can be listed. During the development of the software (and the hardware in the case of special-purpose tactical computers), some examples include:

- Number of demand interrupts
- Number of polling interrupts
- Number of input/output message types
- Number and types of data categories
- Software partitioning scheme
- Core utilization efficiency
- Utilization of each peripheral device
- Number of conditional branches per module
- Number of instructions per module
- Number of arguments per conditional branch
- Coding language selection
- Number and location of program restart points

The reader familiar with computer operating system design and applications programming will recognize immediately that each of these factors influences software reliability in some way. What are the many other factors missing from this list? Which are the most critical, and how could their use be specified and controlled?

During the software verification activity, the accuracy and throughness of each of the activities

listed in Figure 5 again influence the ultimate reliability of the software. Likewise, the extent to which the validation activities listed in Figure 6 fully represent the eventual operational environment or scenario, is crucial to reliability. Which activities are most critical, if prioritization is necessary? How can the Navy and the contractor specify and control them? By this is meant setting standards and requirements, thresholds and limits, software design review and audit criteria, etc.

When it comes to software maintenance for whatever purposes (more effective output mechanisms, changes in the operational scenario, threat characteristics, or environment being controlled, et al), the stability of the software design affects its reliability. The fraction of modules requiring change, the conditions under which the change must be made, and the mechanism for controlling changes add their effects to software reliability. What other maintenance factors are critical, and again, how can they be prioritized, specified, and controlled?

Figure 7 illustrates conceptually the nature of these as yet unknown relationships which the Navy is seeking to identify and quantify to the extent necessary to specify and control their use in software programs. For example, the best language for the application should be specified; limits on demand versus polling interrupts should be established, where applicable; module size and complexity should be limited according to application and operating environment; also, utilization limits on core, input/output registers, channels, and peripheral devices should be specified on the basis of carefully estimated worst-case operational conditions. It should be possible to identify selected areas where poor software/hardware structures may be tolerated to accommodate a specific mission, as well as selected areas of highest potential danger in which to concentrate test planning.

This is the critical and as-yet-unmet need in order to achieve software reliability by design.

RECENT NAVY INITIATIVES

The Naval Material Command is actively pursuing software reliability by design. This is supported by many recent studies which consistently confirm that the overwhelming percentage (as high as 70 percent) of software errors occur early in

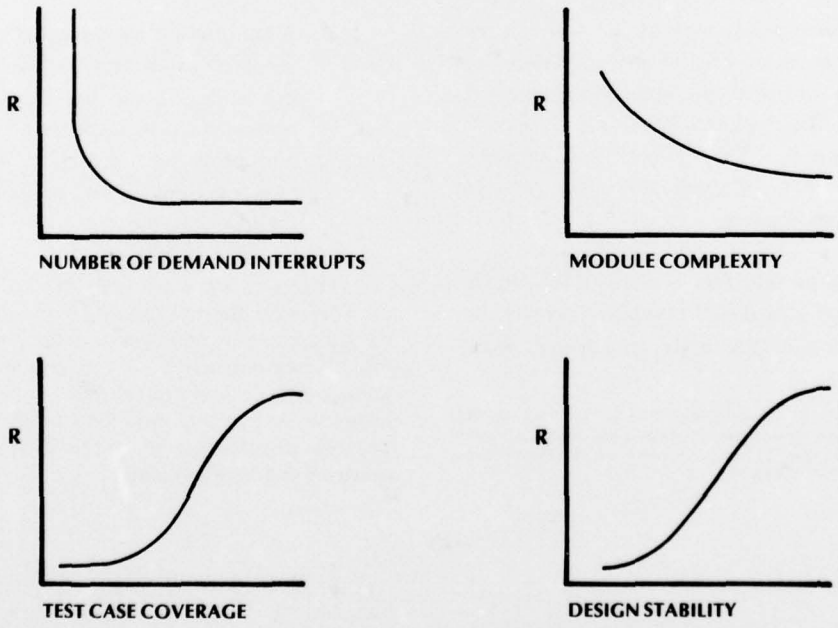


Figure 7. Software Reliability Influance Factors (Conceptual)

system development, and that such software design practices have a far greater impact on error reduction than post-design debugging efforts.* Armed with the results of such studies, the Navy has taken several steps to mandate the use of proven software design approaches, while at the same time stimulating the development of improved methods. An office for tactical digital systems (Naval Material Command Code 09Y) has been established, and the office for reliability and engineering (Naval Material Command Code 08E) is coordinating its efforts to upgrade software reliability in order to expedite improvement. The first five references in the following section are representative of recent Navy activity in software reliability improvement.

In the meantime, the Naval Material Command has as its central theme the checking and rechecking at every level of design, coding and testing to ensure that the most error-free product possible is created—whether that product is a requirements specification, a module design, lines of code or a test specification. The software reliability requirements imposed on a system developer are intended to ensure that:

1. Procurement and planning documents associated with the acquisition of software systems specify appropriate and enforceable software reliability requirements for both executive and applications-type software. This includes Request for Proposals, proposals, T&E plans, specifications, statements of work and other program documentation.

2. Software reliability is planned for and designed into digital computer systems beginning in the early conceptual phase, whether the systems are contractor-furnished or government-furnished.

3. Software evaluations, reviews and audits are conducted on a regularly scheduled basis. Software reliability should be addressed at design and program reviews.

4. Procedures are defined that will be followed when modifications are introduced following test and checkout. The validity of previous testing must be evaluated and a retest carried out if needed.

5. A problem reporting and corrective action system is required to provide an effective means of identifying problems encountered and prescribing follow-up action to assure positive resolution of problems.

6. The system is designed to accommodate a minimum of 120 percent of expected utilization in *all* its components, including memory capacity.

7. A comprehensive test program is required to demonstrate the quality, integrity and correctness of all the software. It must ensure that previously tested subprograms and programs interact as intended and that specified system availability requirements will be met.

Efforts to date are only the beginnings of a concentrated attempt to bring software design into the sphere of disciplined system reliability engineering. If they are to avoid the crippling effects of software failures in critical multi-million dollar defense systems, ways must be found to hasten the progress of software design reliability from a black art to a disciplined science.

*For example, see J. R. Brown's "Impact of Modern Programming Practices on System Development," TRW Final Technical Report 29115-6001-RUOO, Jan 77, and Air Force Rome Air Development Center RADC-TR-77-121, May 77.

REFERENCES

1. Dept of Navy, "Tactical Digital Systems Documentation Standards," SECNAVINST 3560.1, Washington, DC, Aug 74.

2. Naval Material Command, "Reliability of Naval Material," NAVMATINST 3000.1A, Washington, DC, Apr 77.

3. _____, "Configuration Management of Computer Software Associated with Tactical Digital Systems and Other Technical Computer Systems," (Draft Revision) NAVMATINST 4130.2A, Washington, DC, 19 Jul 76.

4. _____, "Standard Tactical Digital System Software Quality Assurance Testing Criteria," (Proposed) TADSTAND X, Washington, DC, 17 Aug 77.

5. Dept of Defense, "Tactical Software Development," (Draft) MIL-STD-1679, Washington, DC, 1 Aug 77.

6. Gene F. Walters and James A. McCall, "The Development of Metrics for Software R&M," General Electric Company, Sunnyvale, CA, presented at the 1978 Annual R&M Symposium, Los Angeles, Jan 78.

7. "Reliable Computer Software," prepared for the Naval Air Systems Command Applied R&M Seminar by the General Electric Company, Arlington, VA, 1977.

8. B. C. DeRose, "Software Reliability Survey and Analysis," Technical Report TR-01917.30–01, Vitro Laboratories, Silver Spring, MD, Oct 73.

9. International Business Machines, "Software Reliability," Report 76-PS2–002, Manassas, VA, Dec 76.

10. _____, "Software Reliability," a presentation to the Naval Material Command (Code 08E), IBM, Manassas, VA.

11. Dept of Defense, "Defense Management Journal," Volume II, Number 4, a special issue devoted to software, Washington, DC, Oct 75.

12. International Business Machines, "Structured Programming Series, Volume V, Programming Support Library (PSL) Functional Requirements," Gaithersburg, MD, Jul 74.

13. Logicon, Inc., "AEGIS/CSGN Design and Implementation Considerations for Computer Program Verification and Validation," AEGIS Technical Instruction, Number 1728, Merrifield, VA, Sep 75.

14. Dept of Defense, "Software Quality Assurance Requirements," MIL-S-52779(AD), Washington, DC, Dec 75.

15. Naval Material Command, "Computer Resources Management Manual," (Draft) Washington, DC, 6 Dec 76.

16. _____, "Standard Shipboard Tactical Digital Processors and Program Language," TADSTAND 1, Washington, DC, May 73.

17. _____, "Standard Specification for Tactical Digital Computer Program Documentation," TADSTAND 2, Washington, DC, Nov 74.

18. _____, "Standard Requirements for Inter-Digital Processor Interface Documentation," TADSTAND 3, Washington, DC, Nov 74.

19. _____, "Standard Definition of Tactical Digital Systems," TADSTAND 4, Washington, DC, Apr 72.

20. _____, "Standard Reserve Capacity Requirements for Digital Combat System Processors," TADSTAND 5, Washington, DC, May 72.

21. _____, "Combat System Designs Employing Multiple AN/UYK-7 Processors," TADSTAND 6, Washington, DC, Jun 72.

22. _____, "Standard Shipboard Digital Display Consoles, "TADSTAND 7, Washington, DC, Jul 74.

Mr. Willis J. Willoughby, Jr., is Assistant Deputy Chief of Naval Material for Reliability and Maintainability. He came to the CNM command in August of 1973 at the request of ADM Isaac Kidd, to establish the policies and programs necessary to bring about an improvement in fleet hardware reliability and maintainability. Prior to joining the Navy, Mr. Willoughby was the Director of Apollo Reliability, Quality, and Safety for the National Aeronautics and Space Administration's Office of Manned Space Flight, Apollo Program Office. He has 22 years of experience in basic engineering, systems R&D, engineering research, and engineering and program management.

As part of NASA's program management, he was responsible for the establishment, coordination and integration of policies and plans for both the Apollo Program Office and the NASA/OMSF field centers. His organization was responsible for overall safety and reliability of the Apollo program. This responsibility was carried out through the review and integration of space vehicle system designs which encompassed 9 major propulsion stages, 106 subsystems and 6 million components.

Mr. Willoughby was also associated for a number of years with a consulting engineering firm as Program Manager of Special Projects Programs and Senior Engineer. He holds membership in ASME and AIRE and is the author of numerous technical papers and reports. He is also the recipient of the Apollo Group Achievement Award and the NASA Exceptional Service Award.

He received a B.S. degree in mechanical engineering from the University of South Carolina in 1952.

DEFENSE SYSTEMS MANAGEMENT REVIEW

The Defense Systems Management Review is published quarterly by the Defense Systems Management College, Fort Belvoir, VA 22060. Publication of the Review was approved by OASD(PA) May 18, 1976.

The views expressed in the Review are those of authors and not necessarily those of the Department of Defense or the Defense Systems Management College. Expression of innovative thought is encouraged. Unless copyrighted, articles may be reprinted. When reprinting, please credit the author and the DSM Review. Two copies of reprinted material should be forwarded to the Editor.

Distribution of this publication is controlled. Inquiries concerning distribution, or proposed articles, should be addressed to the Editor.

The Review is available in microfiche or paper copy from the National Technical Information Service, 5285 Port Royal Road, Springfield, VA 22151, and the Defense Documentation Center (DDC), Cameron Station, Alexandria, VA 22314. When ordering from DDC specify the volume and issue number of the DSM Review desired and the date of the issue.

CALL FOR MANUSCRIPTS

Manuscripts in the following general areas are of particular interest to the Review's readership and are invited at all times for editorial consideration:

- Views of professionals on current and pertinent defense systems acquisition and program management
- Problems confronting Program and Systems Acquisition Managers
- Analysis of approaches to problem solution
- Past experience of responsible authorities
- Defense systems management perspectives of the US Congress, the military services, the media and multinational programs

Within these contexts, the next few issues will emphasize two respective themes from which perspectives manuscripts will therefore be especially welcomed:

Technology Transfer and Government Patent Rights

Joint Program Management

To share your knowledge and expertise, contact the Managing Editor, Defense Systems Management Review, Defense Systems Management College, Fort Belvoir, VA 22060. Telephone: (703) 664–5082; AUTOVON 354–5082.

DEFENSE SYSTEMS MANAGEMENT REVIEW

DISTINGUISHED ASSOCIATE EDITORS

The Honorable Norman R. Augustine
Vice President for Technical Operations
Martin-Marietta Aerospace

General Jack J. Catton, USAF (Ret)
Vice President, Operations
Lockheed Aircraft Corporation

Professor John W. Fondahl
Stanford University

Mr. Eric Jenett
Vice President
Industrial Civil Division
Brown & Root, Inc.

Mr. John M. Malloy
Vice President, Administration
Teledyne Ryan Aeronautical

General Samuel Phillips, USAF (Ret)
Vice President and General Manager
TRW Energy Production Group

The Honorable Leonard Sullivan, Jr.
Consultant

Mr. John J. Welch, Jr.
Senior Vice President
Vought Corporation

Mr. J. Ronald Fox
Lecturer, Harvard University
Graduate School of Business Administration

DEFENSE SYSTEMS MANAGEMENT COLLEGE

Rear Admiral Rowland G. Freeman III, USN
Commandant

Colonel John B. Hanby, Jr., USA
Deputy Commandant

Captain Ronald M. McDivitt, USN
Director, Department of Research and Publications

DSM REVIEW EDITORIAL STAFF

Managing Editor
Vacant

Mr. Robert W. Ball
Senior Editor

Mr. John A. Waring
Writer-Editor

Mrs. Susan Pollock
Ms. Jeanette Tippie
Editorial Assistants

Mr. Richard B. Brown
Graphics Chief
Nicholas W. Kunz, SP-5, USA
Eduard H. Boyd, SP-5, USA
Graphics Designers

PHOTOGRAPHIC CREDITS

Robert F. Holzhauer, PH1, USN
Official DSMC Photographer

☆ U. S. GOVERNMENT PRINTING OFFICE : 1978—724-048

END

DATE
FILMED

8-80

DTIC