AD-A042 686 COMPUTER SCIENCES CORP HUNTSVILLE ALA DEFENSE SYSTEMS DIV F/G 9/2
SOFTWARE PRODUCTION DATA.(U)
JUL 77 J DONAHOO, S CARTER, J HURT F30602-76-C-0163
UNCLASSIFIED RADC-TR-77-177 NL

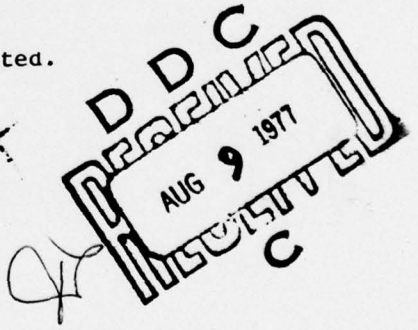
1 OF 2
ADA042-686

12

RADC-TR-77-177
Final Technical Report
July 1977

SOFTWARE PRODUCTION DATA

Computer Sciences Corporation



ROME AIR DEVELOPMENT CENTER
Air Force Systems Command
Griffiss Air Force Base, New York 13441

Some of the figures in this report are not of the highest printing quality but because of economical consideration, it was determined in the best interest of the government that they be used in this publication.

This report has been reviewed by the RADC Information Office (OI) and is releasable to the National Technical Information Service (NTIS). At NTIS it will be releasable to the general public, including foreign nations.

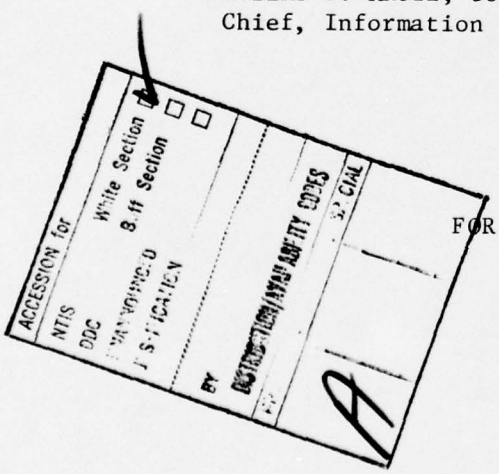
This report has been reviewed and is approved for publication.

APPROVED: Roger W. Weber

ROGER W. WEBER
Project Engineer

APPROVED: Robert D. Krutz

ROBERT D. KRUTZ, Colonel, USAF
Chief, Information Sciences Division



FOR THE COMMANDER: John P. Huss

JOHN P. HUSS
Acting Chief, Plans Office

If your address has changed or if you wish to be removed from the RADC mailing list, or if the addressee is no longer employed by your organization, please notify RADC (DAP) Griffiss AFB NY 13441. This will assist us in maintaining a current mailing list.

Do not return this copy. Retain or destroy.

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE *(When Data Entered)*

REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETING FORM

1. REPORT NUMBER: RADC-TR-77-177	2. GOVT ACCESSION NO.
3. RECIPIENT'S CATALOG NUMBER	
4. TITLE *(and Subtitle)*: SOFTWARE PRODUCTION DATA	5. TYPE OF REPORT & PERIOD COVERED: Final Technical Report. Feb 76 - Mar 77,
6. PERFORMING ORG. REPORT NUMBER: N/A	
7. AUTHOR(s): John Donahoo, Robbie Farquhar, Susan Carter, James Hurt	8. CONTRACT OR GRANT NUMBER(s): F30602-76-C-0163
9. PERFORMING ORGANIZATION NAME AND ADDRESS: Computer Sciences Corporation, Defense Systems Division/Huntsville Operations, Huntsville AL 35802	10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS: 62702F, 55810267
11. CONTROLLING OFFICE NAME AND ADDRESS: Rome Air Development Center (ISIM), Griffiss AFB NY 13441	12. REPORT DATE: July 1977
	13. NUMBER OF PAGES: 97
14. MONITORING AGENCY NAME & ADDRESS *(if different from Controlling Office)*: Same	15. SECURITY CLASS. *(of this report)*: UNCLASSIFIED
	15a. DECLASSIFICATION DOWNGRADING SCHEDULE: N/A

16. DISTRIBUTION STATEMENT *(of this Report)*

Approved for public release; distribution unlimited.

17. DISTRIBUTION STATEMENT *(of the abstract entered in Block 20, if different from Report)*

Same

18. SUPPLEMENTARY NOTES

RADC Project Engineer:
Roger W. Weber (ISIM)

19. KEY WORDS *(Continue on reverse side if necessary and identify by block number)*

Modern Programming Practices, Software Development, Real-time Command and Control Systems, Software Tools, Cost and Schedule Control, Reliable software, THREADS, Top Down Design, Progressive Testing and Integration, Comparison Methodology, Programming techniques.

20. ABSTRACT *(Continue on reverse side if necessary and identify by block number)*

The primary objective of this research task was to assess the effects of modern programming practices (MPP) on selected Computer Sciences Corporation software system development projects. Specific MPPs used on the projects studied were identified and defined. MPP are those engineering and management techniques that may be applied in an integrated fashion to influence software reliability, cost, and schedule during software system development. Once the assessment had been completed, the practices identified were compared with

over

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE *(When Data Entered)*

the techniques discussed in the IBM Structured Programming Series. The final technical report includes both the results of the MPP research and future research plans recommended. Section 3 includes the analysis of the practices used on the three projects plus comparisons. Section 4 discusses the CSC comparison methodology. Section 5 defines a recommended set of programming practices. The comparison methodology was designed to permit an unbiased assessment of programming practice effectiveness so that an optimum mix of practices may be selected for use on any planned software development project.

TABLE OF CONTENTS

TABLE OF CONTENTS (Cont'd)

EVALUATION

This report describes the software development technology and management practices employed on three specific developments by Computer Sciences Corporation.

The intent of the RADC program to which this document relates, TPO V/3.4, is to describe and assess software production and management tools and methods which significantly impact the timely delivery of reliable software.

The study contract is one of a series of six with different firms having the similar purpose of describing a broad range of techniques which have been found beneficial.

RADC is engaged in promoting utilization of Modern Programming Technology, also called Software Engineering, especially in large complex Command and Control software development efforts.

Roger W. Weber

ROGER W. WEBER
Project Engineer

SECTION I - RESEARCH TASK

1.1 OVERVIEW

The primary objective of this research task was to assess the effects of modern programming practices (MPP) on selected Computer Sciences Corporation (CSC) software system development projects. In order to do this it was necessary to identify and define specific MPP used on the projects studied. MPP are those engineering and management techniques that may be applied in an integrated fashion to influence software reliability, cost, and schedule during development of a software system. This definition is broad enough to include all of the diverse management and technical staff activities such as structured programming, design reviews, support software implementation and others. Each project was carefully researched to isolate the programming practices used. In some cases the project managers had adapted specific practices to meet unique requirements. These variants were also identified and defined. Using each complete list of project MPP as a guide, the individual projects were carefully examined in depth. All available information sources were used. This included project personnel, documentation, reports, products, etc. This research was directed toward assessing the impact of the programming practices on the conduct and completion of the system development. The product of this research is a complete analysis of the three CSC projects which served as the basis for the subsequent research tasks.

Once the assessment had been completed, the programming practices identified were compared with the structured programming technology (SPT) techniques discussed in the IBM Structured Programming Series. Before the two could be compared it was necessary to develop a format and criteria for completing the comparison. There was not a one-to-one correspondence of MPP to SPT techniques except in a few instances. In some cases, terminology was different where actual activities were the same or very nearly alike. In these instances the practice and technique were identified by their similarities. In every instance possible, correspondence between individual MPP and SPT techniques was established for comparison purposes. The comparison process was completed for each applicable computer program life cycle phase, and to the extent possible, was based upon quantifiable factors common to the activities under study. Where correspondence was not established the activity was evaluated and compared to other MPP and SPT techniques within appropriate phases.

For the purpose of this study the computer program life cycle as described in the AFM 800 series was used. This manual defines the life cycle as being composed of Analysis, Design, Code and Checkout, Test and Integration, Installation, and Operating and Support phases. The Design phase has been subdivided into Functional and Detailed Design subphases for the purpose of these discussions. The phases are shown on Table 1-1 along with phase activities, documents, and reviews.

Table 1-1. Computer Program Life Cycle

PHASES	ACTIVITIES	DOCUMENTS	REVIEW/APPROVAL
	Operational Requirement Identification	Data Automation Requirement	Staff Action
Analysis	Requirements Analysis	Development Specification	Preliminary Design Review
	Design Analysis		
Design	CPCI Design Definition	System, Functional Design	Critical Design Review
	Management & Test Plan Production	Development Plans (PMP, CM, QA, V&V, etc.)	
	CPC Design Definition	Detailed Design	In Progress Reviews
Code & Checkout	Computer Program Coding & Testing	Program Documentation	In Progress Reviews
	Progress Reporting	Milestone Reports	
Test & Integration	Subsystem Testing & Integration	Test Reports	In Progress Reviews
		Milestone Reports	
Installation	System Execution	Initial Operational Test & Evaluation (IOT&E) Report	
Operating & Support	Operational Test & Evaluation (OT&E)	OT&E Report	Full Operational Capability (FOC) Acceptance

Following the comparison of MPP and SPT techniques, a composite set of standard practices was identified. These standard practices were selected because of their positive effect on the acquisition process and their utility within the various types of system development environments. The factors influencing the selection of these practices are discussed, along with techniques for their use. This set of modern programming practices represents an optimum array of management and technical aids to be applied throughout the complete life cycle. However, these practices are not representative of the entire spectrum of available management and technical aids. They were selected from only three projects and the Structured Programming Series. It must be emphasized that there are other practices which should be evaluated in order to produce a comprehensive analysis.

The final phase of the Software Production Data project involved development of a plan for future MPP research. The research plan presents a phased approach to the study of MPP that is an extension and amplification of research conducted under this contract. The MPP study discussed herein produced significant insight into the identification and definition of effective programming practices. There remain, however, many areas of potentially productive research that can be pursued. The goal for this research will be to confirm the validity and usefulness of the MPP selection methodology. To accomplish this CSC proposes a research program that:

- Provides for systematic performance data collection from new software acquisition projects of varied types
- Defines an automated data reduction system to assimilate the research data
- Performs analysis of the reduced performance data.

The MPP research was hampered during its conduct by adverse circumstances which had some effect on the results. This is not to say that the conclusions drawn from the research are necessarily invalid. It is simply to point out that for conclusive proof that the research findings are true, more comprehensive and extensive research is necessary. The most significant problems encountered in this research project were twofold. First, none of the three projects' management controls or reporting produced the types of data necessary for a thorough analysis of the programming practices effectiveness. The data required for complete analysis must be defined at the project outset and then collected and verified throughout the life of the project. The performance measurement and reporting required for this goes far beyond that necessary for routine project management and will, in all likelihood, increase overhead costs considerably. Second, by beginning this study when the projects were completed or in the final development phases much of the information sought was not readily available. Key project personnel often had transferred, substantiating resource data was occasionally missing and some reports were difficult to analyze.

MPP data collection can only be effective if it is structured as a coordinated project task which is an integral part of the software development. This approach to software data collection offers significant benefits over that used for this study. First, by initially defining data collection requirements and procedures there is more assurance that they will be less disruptive of the software development process and will produce more definitive and accurate data. It is extremely important that data collection not inhibit the normal production cycle. Data accuracy must be maintained throughout the collection process to insure the validity of research findings. Next, software data collection must be carried out by an independent research group. To the maximum extent possible members of this group should maintain a disinterested status relative to the project in order to prevent any bias from appearing in the research report. This approach will be costly in terms of manpower required in excess of that required exclusively for program development.

1.2 REPORT CONTENT

This technical report includes both the results of the MPP research and the future research plans. Section 2 contains descriptions of the three CSC projects studied, including definitions of the MPP used on each project. Section 3 includes the analyses of the practices used on these three projects plus comparisons among related MPP from these projects. Section 4 discusses the CSC comparison methodology. Section 5 defines a recommended set of programming practices selected from MPP and SPT techniques. Section 6 is the plan for future research into MPP technology. Appendix A contains a list of terms common to the discussions of programming practices included in this report.

SECTION 2 - PROJECT DESCRIPTIONS

2.1 OVERVIEW

The three CSC projects studied had many similar characteristics, and yet they were diverse enough to have varied environments for employment of MPP. Each project involved either direct or indirect contractual relationships with the U.S. Navy. The software in all three projects was developed for real-time command and control systems for shipborne employment. The earliest, and largest (in terms of manning) of the three provided the nucleus of experienced personnel for the other two. Also, much of the support and test software was common to all three. This was due to the standardization of computer hardware and software for Naval tactical systems. The Univac AN/UYK 7 computers are used for Navy shipborne tactical systems and all software is developed using the CMS-2 compiler. The CMS-2 compiler is the responsibility of one organization that makes all modifications and enhancements to the compiler to ensure all Navy installations are operating with a current standard version. Nevertheless, development environments, management approaches and system requirements all were different, and the success achieved in each of these acquisition projects varied considerably. The following discussions will provide a basis for the MPP analysis and recommendations. Table 2-1 identifies the three projects and the programming practices they employed.

The subsections which follow include discussions of AEGIS (2.2), TRIDENT (2.3) and DD-963 (2.4). These discussions will cover requirements, environment, history and MPP.

2.2 AEGIS ENGINEERING DEVELOPMENT MODEL-1 PROJECT

The AEGIS Engineering Development Model-1 (EDM-1) project was a subcontract with RCA for the U.S. Navy to design and develop engineering models of the Command and Control (C and C) subsystem, Radar (SPY-1) subsystem, Operational Readiness Test System (ORTS), the real-time executive (ATEP) and the support software (i.e., Signal Processor Check Out (SIPCO), Utility software (ATUL) and data reduction programs (SMART and RAP)).

2.2.1 System Requirements

AEGIS is an integrated, computer-controlled shipboard weapon system composed of a guided missile, acquisition and guidance radar and an automated control system. EDM-1 is a proof-of-concept system now at sea on the USS Norton Sound. The computer selected for operational software was the UNIVAC AN/UYK-7 and the language selected was the CMS-2 high order language.

Table 2-1. Project Programming Practices

PROGRAMMING PRACTICES \ PROJECT	AEGIS	TRIDENT	DD-963
TOP DOWN DESIGN	X	X	X
THREADS	X		X
BUILDS	X	X	X
THREADS MANAGEMENT SYSTEM	X		X
PROJECT REVIEWS	X		
PROGRAMMING TECHNIQUES	X	X	X
PROGRESSIVE TESTING & INTEGRATION	X	X	X
SOFTWARE CONFIGURATION MANAGEMENT	X	X	X
STRUCTURED WALKTHROUGH		X	X
BUILD LEADER			X
CHIEF PROGRAMMER			X
SUPPORT PROGRAMS	X		X
VERIFICATION PROCEDURES			X
INDEPENDENT TEST & EVALUATION	X		X

2.2.2 Development Environment

The CSC contract performance period evaluated by this research project was from January, 1970, through April, 1973, for the EDM-1 version of the system. The data collected for this research only covers the EDM-1 software development, however, CSC also provided level of effort (LOE) support to RCA in the areas of system integration and testing of EDM-1 software.

The AEGIS technical staff average approximately one hundred personnel over the contract period. This staff had an average of ten years data processing experience and 90% had related experience prior to assignment on the AEGIS project. Related experience was determined to be experience in the subsystem to which the person was assigned (i.e., real-time executive, radar, command and control, etc.). The staff was comprised of 67% senior members of the technical staff or higher, while 70% of the staff possessed college degrees. The requirements for a senior member of the CSC technical staff are: BS degree, MS preferred. Minimum five years professional experience with supervisory or management experience. The average turnover rate for the technical staff during contract period was 21.6% per year.

2.2.3 Project History

The EDM-1 was an R and D effort with customer emphasis on quality. The hardware and software for the system were developed in parallel which resulted in several restarts and modifications to the original design. The project personnel interviewed during this research felt that the success of this project was due to the very close working relationship among all personnel and a good overall system coordination. The CSC developed software amounted to over 310,000 words of core in size at a cost of $7,850,000 for the development.

The first year of the contract, 1970, was spent in preliminary and critical design reviews. The executive design was accepted in March, 1971, the command and control design was accepted in May, 1971, and the radar (SPY-1) design was accepted in September, 1971. The build concept was used on the AEGIS project to permit demonstration of system capabilities to the customer as the capabilities were produced. The first capability of the command and control (C and C) system was demonstrated in November, 1971. The first SPY-1 radar capabilities were demonstrated in January, 1972, and C and C and SPY-1 interface capabilities were successfully completed in February, 1972. The final production version was tested and subsystem integration completed in early 1973 and delivered to the customer in April, 1973.

The major milestones were met early or on time. The development schedule was established by CSC in conjunction with the customer and monitored very closely. The demonstrations of partial system capability (builds) were the intermediate milestones and these were met on time or within a week or two of the scheduled date.

The CSC management monitored this progress very closely and redistributed resources when necessary to maintain the overall schedule.

2.2.4 Project MPP

The modern programming practices identified on the AEGIS project were:

- Top Down Design
- Threads
- Builds
- Threads Management System
- Project Reviews
- Functional Programming
- Progressive Testing and Integration
- Software Configuration Management
- Support Programs
- Independent Test and Evaluation

2.2.4.1 Top Down Design

The AEGIS designers decided on a top down approach to the software system design for several reasons, the most important being the system size and complexity. The AEGIS system was at that time one of the largest ever developed by CSC and the top down design was to provide a systematic approach to segment the system into manageable development units. The decomposition and refinement of functions down to the program level during a top down design requires the identification of the interfaces necessary to communicate between the levels of the system. Once these interfaces between levels of the system are identified the system becomes less complex as components become smaller. In contrast, the bottom up approach becomes more complex as the next higher level of a system is built and lower levels dictate what system interfaces are required. Thus Top Down Design was chosen for AEGIS.

The total system development concept for the AEGIS program was to develop the EDM-1 version for maximum functional flexibility and system maintainability to enable future enhancements to be made with a minimum effort and cost. The top down design provided CSC with a system architecture that was compatible with development of functional subsystem components.

2.2.4.2 THREADS

The THREADS concept, which is used by CSC on most of its current projects, was conceived during the AEGIS project. AEGIS was one of the largest real-time systems CSC had ever developed and a technique was needed to ensure that all system requirements were covered by the design specification. Some of the AEGIS staff

members developed this technique by identifying all system inputs and outputs in the design document and identifying the processes or paths through the system from input(s) to output(s). Then, these processes were mapped back to the requirements to verify that requirements were met by the design, thus insuring that the design was complete.

Once the data from this first threading process had been collected and documented the true potential of threads became apparent. The grouping of related processes or threads was used to identify functional system components that could be developed and demonstrated as a partial system capability; these groups were called Builds. Since the threads were paths through the system, the interfaces between modules had to be identified and the data base elements used for intermodule communications were defined early in the development cycle. The early identification and definition of module interfaces established the message patterns for all programs which eliminated programmer interpretation, thus reducing this type of integration problem.

2.2.4.3 Builds

The Build concept was primarily used on the AEGIS project as a means of incremental system development. A build, as stated in section 2.2.4.2, was a group of functionally related threads which could be developed and demonstrated as a partial system capability. By grouping the threads into controllable entities, the total software production could be scheduled and controlled at a manageable level. This technique allowed management to schedule the development based on hardware availability, critical areas of software and other constraints and to utilize development resources to the maximum degree. This technique provided visibility if replanning was necessary due to unavoidable problems, such as late delivery of a hardware component. The build completion dates were used as intermediate milestones by lower levels of management to measure progress.

The technique also promoted progressive testing and integration which is covered in section 2.2.4.7 of this report.

2.2.4.4 Threads Management System (TMS)

As discussed in Section 2.2.4.2 of this report, after the threads were defined and documented, other uses of this information were conceived and computer programs were developed to obtain the full benefits of the data. This collection of programs was called the Threads Management System (TMS) because many of the reports generated by the system were used for management control.

The data base for the TMS contained identifying information about each system thread, such as:

- Thread name
- Thread number
- Performance specification paragraph reference number
- Design specification paragraph reference number
- Modules required to process the thread
- Thread input source
- Thread output format
- Thread status such as designed, coded, tested and integrated
- Milestone reporting dates

The AEGIS management used the TMS report to monitor production progress of modules by threads within a module. The status is reported in a binary format, a module is either 100% coded or 0% coded. This type of status reporting eliminates estimates by the programmers and reports the true status of a software package. The system also reports missed schedules on an exception basis, as well as, current and projected activity status. Other reports include any changes to the data base, such as rescheduling of milestones and providing change impact analysis for proposed system changes.

This system has been enhanced since its development on the AEGIS project to provide extensive management reports. The system served AEGIS as a control tool and provided visibility to CSC management and the customer.

2.2.4.5 Project Reviews

Project reviews may not always be considered to be MPP, however, during this research all project managers interviewed felt that the use of in process reviews (IPR) kept the customer involved in the project throughout the development cycle and enhanced development progress. The normal sequence of events for most projects begins with the customer approving the system design at the Critical Design Review (CDR) and then departing until some time later when he returns to accept the software. This results in little or no input by the customer as the product evolves through the development phases. This sometimes causes system integration problems due to hardware or software changes or trade-offs being made without proper developer-customer coordination. CSC encourages customer involvement during all phases of the software development. This fosters awareness by all system development participants in problem areas thus promoting their expedient resolution and keeps everyone informed of the current status of progress on all project tasks. The AEGIS project staff had a monthly status meeting in which problem areas were identified, action items were assigned to the appropriate personnel and progress reported since the last meeting. CSC staff personnel enjoyed a very good working relationship with the other members of the AEGIS program team. This is not to imply there were no problems, however, a good overall team coordination assisted in quick solutions or work arounds to most problems.

The preliminary design reviews (PDRs) and critical design reviews (CDRs) are contract obligations, however, the format in which they are performed can have a significant impact on the system development. The schedule for PDRs and CDRs are normally milestones in a development cycle (and should be), however, the schedule should be flexible enough to be adjusted if the design is not 100% complete. To hold a review on a scheduled date without regard to actual design status, often results in reviewing an incomplete document which creates excessive questions and wastes many hours of valuable management time. If approval is given to an incomplete design document the result will probably be an incomplete or faulty system design.

The CSC approach to this problem is to insure the design is complete prior to the review or delay holding the review until it is complete. Copies of the material to be reviewed are sent to all parties attending the review early enough to allow them time to read the material and formulate questions prior to the meeting. This procedure expedites the review process and produces a solidified design.

2.2.4.6 Functional Programming

The AEGIS program was designed as a three phase procurement to be developed in engineering development models by adding enhancements and additional capabilities to each succeeding model. Therefore, CSC designed the software in functional modules to enable additional functions to be added or upgrade existing functions without affecting the other system components. The system design technique was top down, however, the module development was not restricted to hierarchical levels of the system.

By programming modules to perform a function or functions, conversion to the next phase or model in the AEGIS program required fewer development resources. This technique also allowed flexibility during development of EDM-1 since requirements changed rapidly as they normally do in an R and D environment. Another consideration with this approach was that it supported progressive testing and integration which is discussed in Section 2.2.4.7.

Structured programming (SP) was considered for this effort, but was not used. Several test programs were written using the SP constructs and the same programs were written in free form. The results of these tests revealed that SP written in CMS-2 (the high order language selected for this project) required 8% more core than free form. Since the core allocation for the system was limited, the decision was made not to use SP. CSC did enforce coding conventions on the tactical subsystem programs to ensure standardized module construction and to promote maintainability of these programs, while obtaining the most effective use of core.

2.2.4.7 Progressive Testing and Integration

The CSC AEGIS staff used the progressive testing and integration technique in conjunction with the MPP discussed earlier to support phased system implementation. Progressive testing and integration is the process of testing the builds as they are developed, demonstrating the capability performed in the build and baselining the programs after a successful demonstration. This process continues as each successive build is completed, with each build being integrated with the previously baselined builds and additional system capabilities being added until the total system is integrated.

The benefits of this tool make it very cost effective. For example, errors are detected and corrected earlier in the development cycle when the cost-to-correct is lower. By initiating the integration process early, any interface problems are easier to track and total system integration at one time is avoided.

Another benefit is that the customer can see success demonstrated early. During the AEGIS demonstrations, Navy personnel were used to operate the consoles to demonstrate that the software and the man-to-machine interfaces functioned as designed. This technique gave the Navy hands-on experience and their inputs and comments helped to improve areas such as display formats and operator key-ins. The build-a-little, test-a-little philosophy of incremental development and testing, used in conjunction with regressive testing to assure that previously baselined functions were not affected, provides a complete verification of the system functions prior to system testing and acceptance tests.

2.2.4.8 Software Configuration Management (SCM)

As stated earlier in this report AEGIS was a very large system and control of the software during the phased implementation process required a large SCM staff. The AEGIS SCM staff ranged between four and six personnel over the life of the project. The primary responsibility of the SCM staff was to control the baselined software from unauthorized modifications and provide documented verification of the various versions of the system. This group also tracked and controlled all changes to the computer programs which had been baselined and all system documentation.

The benefits of this function are normally dependent on the size of the SCM staff in relation to the size of the project. The AEGIS project had proper funding to allow for an SCM staff capable of controlling the software.

2.2.4.9 Support Programs

AEGIS support programs were developed to influence the design development, test, integration and validation of the operational software. They were not a part of the

operational software. Included were those programs used in the integration of hardware and computer programs, the installation and checkout of each system with every other system element. These programs along with the tactical executive program provide facilities and tools to determine product quality and acceptability. The programs used included the following:

- Test Support Programs
- Interface Simulators
- Operational Readiness Test System

2.2.4.9.1 Test Support Programs

The computer program development activity utilized test support programs as tools in the process of development of computer programs. One such test support program was the Software Test and Evaluation Program (STEP). STEP included the following functions:

- Driver-Control execution of STEP internal functions, together with module under test.
- Interpreter-Interpret input control card images and set up schedule table linking modules with each other and with their test data.
- Loader-Link-Editor - Load and patch the modules being tested and link them to any previously loaded modules. The loader shall maintain a memory map and make available the locations of globally defined symbols.
- I/O Control - Control the input-output section of the computer.
- Test Evaluation - Evaluate and print specified test results.
- SNAP/DUMP - List contents of selected areas of memory.

Other programs like STEP were developed, as needed, to serve as test tools to aid in the orderly development of computer programs for the AEGIS system.

2.2.4.9.2 Interface Simulators

Computer program debugging and testing, as well as system debugging and testing, were aided by use of interface simulation programs in the absence of complete system interfaces. These programs introduced external stimuli into the system, program or element under test, noting occurrance and validity of response; facilitated action/reaction criteria; and supported evaluation of timing considerations. The utility portion of STEP was used as a test tool along with simulation programs, as appropriate.

The complement of interface simulator routines included both special-purpose routines for directly simulating the interfaces to the routines being tested and the

service routines that were used to help set up and monitor the particular test. The former set was prepared individually for each autonomous system. Some of the service routines were shared by more than one simulator package.

2.2.4.9.3 Operational Readiness Test System (ORTS)

Although ORTS was configured as part of the operational software it was classified as a support program.

ORTS provided a system for performance and readiness monitoring, fault diagnosis and maintenance evaluation of function availability and operability level, and summary status reporting at the command level. Within this framework, the operational programs residing in the computers employed by the radar, command and control and weapon direction subsystems demonstrated the following capabilities:

- Fault detection by periodic monitoring and analysis of critical sensor test data and computer programs.
- Fault isolation by monitoring and analysis of discrete sensor test data and computer programs.
- System operability testing by processing and analysis of simulated radar return data.
- Controlled injection of simulated signals into monitored equipment for the purposes of fault detection, fault isolation, and operability testing.
- Control of ORTS displays for the purpose of reporting the results of testing and processing of ORTS operator inputs, for test initiation, and access of system operability and maintenance action information.
- System operability testing of the AEGIS equipment.
- System operability testing of the overall AEGIS system.

2.2.4.10 Independent Test and Evaluation

Computer program testing was conducted by the AEGIS Test and Evaluation (T&E) Department. This department operated independently within the project organization to prepare test plans and procedures and conduct the testing. Project staff programmers and analysts completed all computer program unit testing during development of the programs. When unit testing was successfully completed the programs were integrated into functional subsystems known as builds and released to the T&E Department for testing. Using prepared scripts and scenarios the T&E personnel evaluated the performance of build programs using a build demonstration as the vehicle for the testing. A build demonstration is a formal test of specific system functional capabilities. Normally, the customer observed each build demonstration as the vehicle for the testing. A build demonstration is a formal test of specific system functional capabilities. Normally, the customer observed each build demonstration and reviewed test documentation.

The T&E Department manager reported directly to the AEGIS project manager. Computer program testing was executed and documented under project guidelines using standardized procedures.

2.3 TRIDENT MONITORING/DATA ACQUISITION SUBSYSTEMS PROJECT

The TRIDENT Monitoring/Data Acquisition subsystems project was a subcontract with IBM for the U.S. Navy to develop the Monitoring subsystem (MS) and the Data Acquisition (DA) subsystem software for the TRIDENT submarine sensor system. CSC had responsibility for design specification, detail design, program generation, coding and checkout, and subprogram integration. However, all system testing and integration is being accomplished at this time by CSC personnel at the Navy Land Based Evaluation Facility (LBEF) in Rhode Island. The work is being done under a time and materials contract with IBM as the prime contractor.

2.3.1 System Requirements

The TRIDENT class submarines will be the largest of the Fleet Ballistic Missile Submarines (SSBN), and will carry a complement of 24 TRIDENT I missiles. Each submarine is designed for quiet operation, high systems reliability, and extended cruise endurance.

The MS and DA subsystems operate within the ship's central computer complex to provide automated sensor signal analysis, supporting quiet operation of the ship. The on board computers to support these programs are the UNIVAC AN/UYK-7 and AN/UYK-20. The computer programs are written in the CMS-2Y high order language.

2.3.2 Development Environment

The CSC contract performance period for this effort was April 1974 through August 1976. The technical staff consisted of approximately thirty personnel during the coding and testing phase, of these 60% were senior level personnel. The average data processing experience for the staff was 9.6 years, and 73% of the staff possessed college degrees. Some of the technical staff members were transferred from similar CSC projects, and of the total staff, 47% had directly related experience using the UNIVAC AN/UYK-7 and the CMS-2Y language prior to assignment to the TRIDENT project. The annual turnover rate was 24% during the contract period.

The MS and DA subsystems developed by CSC totaled 94K words in size and the development cost was $2.4M over the contract period.

2.3.3 Project History

This project encountered several problems during the development cycle. The major problem arose when a complete redesign of the subsystems was required after the sensor data collection hardware could not meet the performance requirements. The original hardware known as the Information Transfer System (ITS) was replaced by special purpose hardware using an AN/UYK-20 computer as a control unit. This involved a major change in the concept of operation of MS and necessitated development of the DA subsystem. The decision to make this change was made only after the first two levels of the monitoring subsystem were delivered to the customer.

The project also had a problem with program sizing caused in part by the added requirements that accompanied the replacement of ITS by DA hardware. Also the subsystems were coded using structured programming with the CMS-2Y language which further extended the core budget. Even after several test cases proved that structured programs required 5% to 8% more core than non-structured programs, the decision was made by the customer to continue with structured programming. A major effort was undertaken to reduce the size of the programs late in the development cycle and monitoring of the core budget became a high priority effort.

Hardware, compiler and facility problems further slowed progress and resulted in milestones being rescheduled. The customer became concerned and was given additional visibility into development activities. Additional controls, resources and techniques were applied to the project and most of the problems were resolved.

The problems resulted in the final product delivery date being rescheduled from May 1976 to August 1976.

This project did not use the Threads Management System and the milestone achievement data was not well documented. The first two major milestones, Level I and Level II of the original subsystem design were delivered on schedule. At this point the decision was made by the customer for a redesign of the monitoring and information transfer subsystems and all milestones had to be rescheduled.

2.3.4 Project MPP

The modern programming practices identified on the TRIDENT project were:

- Top Down Design
- Builds
- Structured Programming
- Structured Walkthrough
- Progressive Testing
- Software Configuration Management

The use of these MPP's and the impact they had on the CSC TRIDENT project will be discussed in this subsection.

2.3.4.1 Top Down Design

CSC was required by contract to develop the monitoring and data acquisition subsystems using top down structured programming techniques. The design of the subsystems was implemented using a top down approach, by designing the control architecture first, then the internal control logic within each module to a progressively lower level of functional detail, until the lowest functional level is complete.

The result of this technique was a design that segmented the software package into functionally unique elements that supported phased development and implementation using the top down structured programming techniques.

There was no data to indicate that this approach was more or less expensive than another approach, such as a bottom up design.

2.3.4.2 Builds

Late in the development cycle several problems began to surface and the customer required additional management visibility. A CSC technique known as "builds" was implemented. This technique groups related processes into system functions which can be demonstrated to the customer. This confirms that specified fully developed processes do perform their design function or functions.

The technique impacted the project in several ways. It provided the customer and CSC management with visibility into the true status of the project. The milestones were then rescheduled and lower level milestones were established to assist CSC management in tracking and controlling the system. This "build-a-little-test-a-little" concept promoted early subsystem integration and led to detection of interface problems early. This technique in conjunction with other tools assisted CSC management in resolving some problems and turning out good software within a reasonable time frame.

2.3.4.3 Structured Programming (SP)

The use of structured programming with the UNIVAC computers and CMS-2Y compiler was a contract requirement. This requirement created a problem, in that the structured code using CMS-2Y used 5% to 8% more core than free form code. However, the customer felt the benefits derived from structured programming justified the additional core required.

The CSC analysts interviewed on the TRIDENT project felt that program design for structured programming was time consuming because of the lower level of design detail required. But the coding phase was reduced to almost half the time of free form programs due to the design being developed at almost the code level. The personnel using SP for the first time thought the constraints of SP did not effect the productivity of the senior level personnel to any degree. However, a noticeable adjustment was required by the junior level personnel to develop logic in a structured manner. It was also felt that better quality programs were produced by the junior staff members using SP.

The total effect of SP on the TRIDENT programs could not be determined. The measures of its effectiveness are in program maintainability, reduction of the testing and integration effort and ease of transferring program responsibility. These measures could not be evaluated on TRIDENT since CSC was not tasked under this contract to perform system testing and integration and the product has not reached the operational phase to determine maintainability.

2.3.4.4 Structured Walkthroughs

This technique was used on the TRIDENT project for detailed program design review and code review. The detailed program design review was held after program design and prior to coding and the code reviews were held after a clean compilation was obtained and prior to unit testing. Structured walkthroughs were conducted by an evaluation committee. The evaluation committee varied in size from case to case but normally consisted of the analysts whose programs had interactions with the program under review and the primary programmer's immediate supervisor. The design or code was reproduced and distributed to the committee members for review prior to the evaluation date. The reviews were internal to the project and informal to a degree, to encourage constructive criticism from the participants. The dates set for the walkthroughs are part of the milestones for completion of program design and coding as scheduled by the programming manager. The evaluation team would look for misinterpretations of the data base structure, communication links, logic paths and the general approach to the solution of the problem.

The effect of the walkthroughs as expressed by the personnel on the staff showed them to be cost effective in both man-hours and machine time. The number of personnel involved in a walkthough ranged from three to seven and the time required, was about four hours per program. The direct benefit from these evaluations was in the test time being reduced by 50% to 80% because of errors which were detected prior to the test phase. The indirect benefits were (a) verifying program interfaces to alleviate integration problems, (b) spreading knowledge of system components among the staff, (c) accelerating the software production by reduction of test and integration time, and (d) detecting errors early in development cycle, thus reducing the cost of correction.

2.3.4.5 Progressive Testing

This technique was used in conjunction with the "build" concept for incremental system development. There are several advantages gained by using this tool rather than developing the entire subsystems, then testing and integrating all the functional components at the end of the development. This tool provides intermediate milestones for tracking progress. It also allows early visibility for the customer, as a build or capability is completed it can be tested and integrated with previously completed builds to test interfaces between segments and demonstrate multiple capabilities. If errors are detected during the build test, they will be easier to find than when the entire subsystem is integrated. Likewise interface errors are easier to trace during a phased integration process since the number of interfaces being added are normally fewer.

2.3.4.6 Software Configuration Management (SCM)

This development tool is not a new technique in the software development process. The impact it has on a given project depends on the size of the SCM staff in relation to the size of the program and the amount of control the SCM staff can apply over the software development. The TRIDENT project had 20 to 30 technical personnel and one person was responsible for SCM as a part of his duties. The prime contractor did not specify any configuration control requirements during development of the software subsystems, therefore, SCM was limited to recording current computer program positions for testing purposes. The subsystems were baselined when they were delivered.

2.4 DD-963 PROJECT

The U.S. Navy is constructing a new class of destroyers to assure protection of the fleet through the 1980s. This class, known as DD-963 will be equipped with the most sophisticated weapons and control systems available today. The heart of these systems will be a real-time, automated command and control system. CSC as a prime contractor to the Navy has developed the operational software for this advanced system based on a prototype system developed by another contractor.

The mission of the DD-963 class destroyers is to operate offensively in the presence of air, surface or subsurface threats as an escort for strike forces or antisubmarine warfare (ASW) aircraft carriers. In addition, it is designed to seek out and destroy enemy submarines. The system is also capable of providing friendly aircraft control during ASW patrol and surveillance, and search and air rescue.

2.4.1 System Requirements

The DD-963 project is a time and materials contract with the Fleet Combat Direction Support Services Activity (FCDSSA) at Dam Neck, Virginia. In May 1973, FCDSSA

requested CSC to assume responsibility for two activities associated with DD-963 development. The first activity was to monitor Litton Industries production of a prototype version of the DD-963 command and control system known as Model III. In this task CSC reviewed the computer program design specifications (CPDS), subprogram design documents, program test plans and program test procedures. Also, CSC personnel witnessed all levels of computer program testing. The objective of the second activity was to upgrade the prototype version of the command and control system. This involved developing a totally new system, designated Model IV. The following subtasks were completed for the Model IV development:

- Revise Computer Program Performance Specifications (CPPS)
- Write CPDS for this model
- Modify the real-time executive as required by the new CPPS and CPDS
- Code, debug, test and integrate the application programs
- Produce system test procedures and assist the Navy in conducting program acceptance and system integration testing

It was the second activity associated with development of the DD-963 command control system which was evaluated under this study. The MPP research was devoted exclusively to the system development efforts for Model IV of the DD-963 system.

2.4.2 Development Environment

The DD-963 development environment had a very positive impact on the conduct of system development activities. The project technical staff, which averaged 30 people through the life of the project, was located on site at Dam Neck, Virginia. These personnel averaged 10 years data processing experience and 46% had directly applicable prior experience. This included UYK-7 hardware and CMS-2 language experience. Half of the staff were senior members of the technical staff or higher, while 73% of the staff possessed college degrees. The average, annual personnel turnover rate was only 13.2%.

System development took place over a period of two and a half years at a cost to the customer of approximately $3.5M. The software system developed under this contract consisted of 14 modules comprising 250K lines of code. These modules form the ship operational program (SOP) which is an element of the Naval Tactical Data System (NTDS) command and control system supporting the ASW mission of DD-963 class destroyers. Processing is handled by an AN/UYK-7 computer system configured with three central processor units (CPUs), two input/output channels (IOCs) and ten memory banks. A fourth CPU runs with one IOC and two memory banks as a reserve processor. Scheduling of the modules, core allocation, timing, etc., is handled by the real-time executive program.

The primary output device is a Hughes display console, which displays a radar/symbology presentation and amplifying data. Track symbology for air, surface and subsurface tracks will be displayed, with modifiers showing velocity and direction,

assignment or engagement status, etc. Also special points such as reference points, sonobuoys and electronic warfare fixes will be shown.

2.4.3 Project History

Initial development of the DD-963 command and control system was begun by Litton Industries. Litton produced a prototype version of the system, known as Model III, which is now operational, at sea. Following production by Litton of the computer program performance specifications (CPPS) for Model III of the system, FCDSSA requested that CSC expand its work force at Dam Neck to include two activities connected with DD-963. CSC had a project office and staff at Dam Neck working on other related development tasks at that time. As discussed earlier, CSC assumed the role of program monitor for the balance of Model III development. Concurrent with this activity CSC assumed responsibility for development of Model IV of the DD-963 system beginning with rewriting the computer program performance specifications (CPPS). Following this CSC proposed a three year computer program development schedule for Model IV, but the Navy specified a two year requirement, which CSC accepted.

The CSC project office at Dam Neck then expanded its staff and began the task of rewriting the computer program design specifications (CPDS). As work began it became obvious that the CPPS would have to be revised in order to more accurately reflect modified performance requirements. After the CPPS was revised, the CPDS was rewritten and the threads for the new CPDS were developed. Threads were employed at this time to verify the design to performance relationship within the system. Once the detailed design was completed the code and checkout activity began.

During the code and checkout phase the DD-963 staff used the builds concept to establish system development milestones. By combining functionally related threads into groups known as builds it was possible to define an incremental development schedule for system capabilities. Production responsibility for each build was assigned to an individual task (build) leader. He assigned coding tasks and monitored progress through formal demonstration of the system capabilities represented by that build. This demonstration was a formal presentation to the customer of software functions which satisfied specific system performance requirements. Build milestone achievement through demonstrations is recorded in Table 2-2. This information was extracted from the Threads data base. Demonstrations were conducted by an independent, project test group assisted by Navy personnel.

System test and integration requirements were satisfied by the build demonstrations. Using this "build a little, test a little" concept the system was complete and ready for installation when the final build was delivered. The MPP analysis did not extend into the installation, and operation and support phases of the MPP project. These activities were not a part of the time and materials contract for DD-963.

Table 2-2. BUILD DEMONSTRATIONS

BUILD NAME	DATES SCHEDULED	DATES ACTUAL	COMMENTS
A	12/6/74	12/6/74	
B	2/24/75	3/10/75	System Hardware Modification Development Computer Out One Week Threads Added to Build
C	4/21/75	4/14/75	
D	6/16/75	6/27/75	Hardware Mock-Up Problems Computer Timing Problems Software Integration Task Underestimated
E	7/22/75	7/22/75	
F	9/8/75	9/22/75	Development Computer Out One Week Peripheral Simulator Not Fully Operational Job Turnaround Time Excessive
G	11/24/75		Rescheduled Due to Nonavailability of Supporting Hardware
H	1/15/76	11/24/75	Replaced Build G in the Demonstration Schedule
G1	3/1/76	3/3/76	Rescheduled as Noted Above
G2	3/1/76	3/3/76	Reschedule as Noted Above
G3	3/22/76	4/9/76	Rescheduled as Noted Above
I	3/22/76	4/15/76	Compiler and Hardware Mock-Up Problems
II	3/22/76	4/15/76	Compiler and Hardware Mock-Up Problems
J	5/19/76	5/19/76	
J1	5/19/76	5/19/76	
K	6/15/76	7/25/76	Restructuring Build Components Mock-Up Availability Problems
L	10/1/76		Delayed at the Request of Customer
L1	10/1/76		Delayed at the Request of Customer

2.4.4 Project MPP

The DD-963 project staff employed a variety of management and technical methods and tools to aid in the development of the command and control system software. The following is a list of those MPP identified during the research of this project:

- Top Down Design
- THREADS
- Threads Management System
- Builds
- Build Leader
- Structured Modular Programming
- Chief Programmer
- Peer Group Programming
- Support Programs
- Verification Procedures
- Independent Test and Evaluation
- Progressive Testing and Integration
- Software Configuration Management

The following discussions will highlight the impact of each of these MPP on this project including the benefits of their use.

2.4.4.1 Top Down Design

Top down design facilitated the consistent and complete definition of a system architecture which satisfied all performance requirements. This design technique was most effective for DD-963 because of the high degree of subsystem/module interaction required by this real time system. Definition of a system in a hierarchical manner facilitates the positive identification of system functional relationships and assures that internal data flow will be optimized. For the DD-963 command and control system there were a number of diverse operational capabilities which had to be linked through a single executive to a master display/control subsystem. This was a complex design problem with the level of intermodule message traffic as a paramount consideration. The functional capabilities had to be designed so that they maintained their unique characteristics, and yet communicated efficiently with each other and the display/control subsystem.

For DD-963, top down design was also compatible with the use of threads for design verification. The threading process involves decomposition of the system design proceeding from high to low level processes. Thus, threads definition logically followed the design methodology applied to the DD-963 system. Top down design and threads were compatible with the modular system architecture necessary for this real time system to provide the flexibility required in meeting the dynamic operational requirements placed on the diverse components of a command and control system, a modular system is the best design.

The top down design methodology would benefit any system which can be specified to operate with multiple functional levels. For example, if an executive function will exist which exerts operational control of system subfunctions then top down design would be beneficial. On the other hand, where a system is composed of autonomous functions with limited interfunction communication, top down design would not be called for. In fact, it might be counterproductive to attempt a top down system design.

2.4.4.2 THREADS

THREADS is a discipline developed by CSC which may be used to map internal system processes or operations. A thread represents a sequence of events or discrete operations that lead to satisfying a specific function within a system. This sequence encompasses one process from stimulus to response and may be defined at any functional level. Thus, threads may be specified at the system, subsystem or program levels.

The DD-963 staff threaded that system during the design phase of system development. These threads were used to verify the design to the requirements as the CPDS was being developed. In this way design flaws were detected early when there was more time to make design adjustments without significantly impacting system development. These threads assured that the performance requirements as specified in the CPPS had been translated in a consistent manner into system design.

This method of system definition also assured that module interfaces were compatible. Later, during the code and checkout phase the threads were used to monitor development progress. The Threads Management System (TMS) provided the vehicle for reporting and displaying threads completion data. This provided management visibility into the current status of system development. Finally, test scripts were developed using threads to specify the processes to be tested and to identify the required stimuli and responses.

THREADS is a design verification tool with wide application. It is a discipline which provides a straightforward method for relating design to requirements. As a result of this verification process system design flaws are more likely to be detected early, before coding begins. By employing the Threads Management System with the threads data, the management and the customer are given good visibility into the development process. Through TMS, management is able to develop greater flexibility in responding to requirements changes. TMS provides the types of reports which may be used to develop change impact assessment. THREADS is an excellent management tool for managers at all levels.

2.4.4.3 *Threads Management System*

The Threads Management System (TMS) is a software system developed by CSC that can provide automated threads data management services to threads users. With

this system users have access to threads analysis and progress information that may be displayed in a variety of formats.

The data base for the TMS as used on the DD-963 project contained identifying information similar to that described for AEGIS in Section 2.2.4.4 of this report. This information was gathered and entered into the TMS data base by a member of the configuration managent staff. Updates were performed regularly, usually on a weekly basis.

The DD-963 staff used the TMS to provide visibility to the customer during the code, test and integration activities for all subsystems. The customer was very concerned about receiving current information on the true status of system development. TMS reports provided an excellent vehicle for satisfying this requirement. For DD-963 the TMS was used in essentially the same way that it had been used on the AEGIS project. The primary difference between the two TMS applications was in the level of usage. The DD-963 staff relied on the TMS reports as the mainstay of their management reporting system. The customer was briefed regularly and frequently on production progress. Both the DD-963 staff and the customer depended very heavily on the TMS reports. These reports were readily accepted by the customer because of their clarity and accuracy. By using the TMS and its reports the DD-963 staff was able to maintain management control of the project and develop an excellent working relationship with the customer.

2.4.4.4 Builds

The build concept is a natural corollary to THREADS. Threads represent distinct processes within a system. By combining related processes into groups which represent demonstratable system functions, builds are created. A build represents a specified system capability such as target tracking, noise monitoring, data update, etc. Threads do not have to be defined for a system in order to use the builds concept. Any system capability which may be uniquely identified may be specified as a build. All software routines supporting that capability become a part of that build.

Builds were created for DD-963 by combining threads in order to define an incremental software production schedule. Through the builds, specific system capabilities could be coded, tested and demonstrated to the customer. This provided the desired customer visibility into the development process and permitted incremental production of each subsystem. Visibility was provided through the build milestones and TMS reporting. The customer was made aware of the current status of all threads and thus all builds in production. When production problems caused delays or external conditions halted effective program development on a given build, resources could be shifted to other builds and the customer rapidly appraised of the schedule changes.

The use of builds would be of benefit to any software development task where incremental production of system capabilities is planned and threads are defined. Builds are also useful in providing customer visibility into the development process and may be created for that purpose alone. A supplementary benefit is derived from the incremental production and integration of system capabilities. Through early, phased integration of system components many of the massive problems associated with integration of all subsystems at one time are avoided. Builds represent more management control and customer visibility throughout the development process.

2.4.4.5 Build Leader

The function of build leader was created to satisfy the requirement for a team chief to manage development of each system build. This responsibility was assigned to a senior member of the staff (usually a Chief Programmer). The build leader then assigned individual programming tasks for production of the build capability. This group, headed by the build leader, worked together as a team until the build was demonstrated. This team functioned in a manner closely resembling the Chief Programmer Team described in the Structured Programming Series. A significant difference was the absence of a team librarian function. This function was common to the entire DD-963 project. The build leader concept of operations proved very successful for this project.

The build leader and team concept could be very effective on any project using incremental subsystem production and integration through builds. It provides a focused management organization with coordinated accomplishment of limited objectives.

2.4.4.6 Structured Modular Programming

The programming technique used on the DD-963 project involved strictly enforced coding conventions similar to structured programming. Using this structured modular programming, DD-963 programmers developed short programs with standard data names, indentured code structure, and limited use of "GO TO" statements. Single entry/single exit logic for these programs was stressed, however since the high order language (CMS-2) did not have all of the structured programming constructs it was not possible to use pure structured code. The program code generated by project programmers was examined by the Chief Programmers prior to being submitted for initial compilation. This provided a degree of coding standardization enforcement and kept the Chief Programmers apprised of program development progress. The effect of using this programming approach was that all programs were easier to read and understand. Transfering responsibility for a particular program was facilitated. However, the Chief Programmers' work load was quite heavy because most production materials, including program code, were funneled through them.

Structured modular programming was simply a variant of more conventional programming techniques. It was used effectively on the DD-963 project, but is not uniquely better than any other structured technique. Perhaps its greatest worth was in the standarization that it imposed on the program development work.

2.4.4.7 Chief Programmer

The DD-963 staff organization included two Chief Programmer positions. However, there were no teams associated with these positions as in a Chief Programmer Team structure defined in the Structured Programming Series. The DD-963 Chief Programmers functioned more as section managers than as task oriented team leaders. Actually, a Chief Programmer's responsibilities included those of a section manager, team leader and technical advisor. The Chief Programmer function was critical to the success of the development effort. Each Chief Programmer supervised a group of 10-15 programmers and analysts. In addition he was assigned build leader responsibilities on a continuing basis. He monitored and examined program generation and assisted in subsystem testing. The Chief Programmers were the focus of program development efforts. They assured continuity and consistency in design to code translation for the entire DD-963 system. The efforts of these two Chief Programmers were complimented by the outstanding management support provided by the Department Manager. He assured that the Chief Programmers were relieved of all administrative responsibilities and provided support services such as configuration control, independent testing, program library management, etc. Under these conditions the Chief Programmers could devote their full attention to program development.

The benefits which accrued to this project from the use of a Chief Programmer function stem mainly from the individuals who filled the two positions. In essence the chief programmers were section managers. They were given more responsibilities than would ordinarily be given to section managers and more latitude to exercise their authority. Under these conditions the two individuals who were chief programmers, with the support of the Department Manager, did an outstanding job in directing the efforts of the technical staff. They made the Chief Programmer position an effective management technique.

2.4.4.8 Peer Group Programming (Structured Walkthrough)

Peer group programming is a less formal version of the structured walkthrough process. The DD-963 staff used this technique for detailed design and program code review of each program developed. These reviews were led by project task leaders, usually the Chief Programmers. Each review committee was composed of the primary programmer's peers, other staff programmers and analysts. This group would bench check the design or code in order to identify logic and coding errors. Project task leaders felt that with this review process test time was less than half that which would have been required without the reviews. For DD-963 this was a

significant factor because of the compressed schedule under which they were working. The indirect benefits derived from these reviews were confirmation of the compatibility of subsystem interfaces and broadened staff understanding of subsystem control logic.

The use of peer group programming is beneficial as a less formal alternative to structured walkthroughs. They both accomplish the same purpose, but peer group programming seems to be suited to small, informal project staff organizations. The cost in manhours to conduct these reviews seems more than offset by the savings accrued during program testing.

2.4.4.9 Support Programs

The support programs used on the DD-963 project played an important part in the successful production of the software system. Support programs were credited by task leaders with supplying added capabilities which assured that the technical staff could meet its production milestones. The support programs such as those producing core dumps, program compilations, message traps, selective memory register recordings, and flow chart documentation were tailored to conform to the requirements of this project. The modifications were completed by the staff to insure that the programs satisfied the specialized requirements of the DD-963 project. This was a continuing process as needs and requirements changed. Development of the support software was aggressively pursued throughout the project. For this reason the support programs did what they were supposed to do and all members of the project technical staff were able to use them with confidence.

Support programs can be an effective aid in promoting software development. It appears from data gathered on the DD-963 project that they are most effective when created or tailored for a specific system development. The DD-963 technical staff had confidence in their support programs and used them frequently. The programs were maintained and modified conscientiously. These factors combined to produce an aggressive development program whose success was due in no small part to the support software.

2.4.4.10 Verification Procedures

Verification procedures as used on the DD-963 project included all activities associated with confirming the consistency of computer program design and code. Consistency was judged against the Computer Program Performance Specification (CPPS) and the Computer Program Design Specification (CPDS). The confirmation process began with an analysis of the computer program design to assure that the requirements stated in the CPPS were satisfied. Once the program code had been developed the logic was compared with the design specifications to insure that it conformed to the CPDS. The key to successful implementation of these verification procedures was the use of the threads. Threads were a vital element of the top

down verification process. For bottom up verification the auto-flowchart documentation was used. This support program provided an automated flowchart generation capability to the project staff. This documentation was compared to the CPDS during program verification.

Verification procedures had a very positive impact on the DD-963 project. It was through these procedures that the department manager and the chief programmers were able to control and monitor the production of program code and the demonstration of system capabilities. The procedures established a formal set of rules or guidelines which governed the system acquisition process. They were instrumental in insuring that each system function developed was responsive to the performance specifications and adhered to the design specifications.

These procedures can be implemented on any type or size project. They would be most effective if used in conjunction with THREADS and builds. The procedures should be adapted to project development requirements and implemented early in the development cycle.

2.4.4.11 Independent Test and Evaluation

The primary function of an independent test and evaluation team is to develop test scenarios that will ensure all system functions are tested and that they perform in accordance with the requirements specifications. The group should be organizationally structured to report to the project manager rather than the programmer manager in order to provide unbiased evaluation of the test results.

The management personnel on the DD-963 project felt the independent test and evaluation teams contributed significantly to the success of the project. The impact of this approach is difficult to quantify. However, when the tester is not involved in the development of the programs he does not test the conditions which were coded, he tests the performance as it was designed. This approach is more comprehensive because more time will be spent on the test script since this is the primary function of the test group.

The testing on this project was conducted with the customer present and in some cases the customer personnel actually operated the consoles as they would onboard the ship. This technique instilled confidence in the product and provided training for the customer, as well as confirmed the actual "man to machine" interfaces that were required to execute the system.

An independent test and evaluation team should be used to produce the greatest benefit to the project. When this technique is not feasible, the tester should be someone not directly involved in developing the program(s) being tested. It is also recommended the test plans and procedures be developed early in the development cycle and baselined and controlled under the same conditions as the design document.

When possible, the customer should also be involved in the testing in order to insure that he understands what is involved in the operation of the system as early as feasible.

2.4.4.12 Progressive Testing and Integration

Progressive testing is the process of verifying subsystem capabilities on an incremental basis as the system is being developed. For the DD-963 project this technique provided excellent visibility for the customer during the life of the project. By integrating these capabilities as they are tested the massive integration problem caused by attempts to integrate all subsystems at one time is avoided. The progressive testing and integration process also facilitated detection of subsystem interface errors earlier. In conjunction with testing new system capabilities the previously integrated system capabilities were tested through regressive testing. This insured that the newly integrated software did not adversely affect the existing system functions. The use of this technique along with other MPP permitted the DD-963 staff to provide the customer with a current status report of system progress and to demonstrate system capabilities as they were developed.

The total effects of progressive testing and integration on the DD-963 project cannot be determined at this time as they have not reached the system installation phase, however based on the results of other projects using this technique the benefits will outweigh the effort expended.

The use of this technique will benefit any project where communication between system components is required. When used in conjunction with other MPP such as the THREADS/Builds concept the cost of applying the tool is minimal and the benefits derived make it very cost effective.

2.4.4.13 Software Configuration Management

Software Configuration Management (SCM) is the process of controlling the software system configuration after a specific configuration has been declared to be a baseline position. The baseline position establishes program structure for formal testing and demonstration. Once the baseline is established, it is the responsibility of the SCM staff to control program modification and prevent unauthorized changes. The DD-963 SCM staff consisted of one person who produced all system version positions for build demonstrations and who controled all changes to current system versions. He was aided by the Chief Programmers who reviewed program code and monitored production of new program versions. However, these new program versions were transferred to the system file without recompilation because of manpower and production facility constraints. Apparently this did not create any problems during testing and integration, but it was a potential problem area. Overall, SCM was vigorously applied on the DD-963 project and contributed to the successful demonstration of all system builds.

An SCM function is usually associated with all projects. The size of this staff function should be determined by the size of the project. The configuration control plan should be developed during the system design phase and the plan should be enforced to ensure the integrity of the system configuration throughout the life cycle of the project. The benefit derived from SCM is directly proportional to the conscientiousness with which it is applied.

SECTION 3 - MPP ANALYSIS

3.1 OVERVIEW

The modern programming practices that were identified during the research of CSC projects have been introduced as part of the project descriptions. Those discussions focused on how the practices were used on each project and what benefits were derived from their use. In this section each practice will be analyzed from the standpoint of its impact across the projects studied. In this way a more comprehensive and definitive analysis can be developed.

The format of these individual programming practice analyses was selected to facilitate the kind of critical review which will produce a clear definition of the MPP value. These discussions have been developed around four topics. First, each practice will be described as completely and accurately as possible. The descriptions will include all forms of each practice that were employed on the projects. Next, the effectiveness of each practice within applicable development phases will be discussed. Effectiveness will be measured using quantitative data such as error generation rates, code production rates, etc. when possible. Also, qualitative assessments will be used in lieu of or to supplement quantitative data. Computer program life cycle phases based on AFM 800 series definitions will be used in the discussions of MPP effectiveness. Then, a comparison will be made of the impact of each practice across each of the projects where it was used. This comparison will consider how each practice was employed on the projects and how the MPP influenced each individual project. Finally, recommendations for use of these MPP on future projects will be proposed. These recommendations will be based on the benefits and costs associated with use of the MPP.

3.2 TOP DOWN DESIGN

Top down design is the technique of design definition which proceeds from the highest functional level to the lowest level within the system. This process is accomplished by defining, at the highest level, a functional requirement to be performed, then refining that definition in terms of its functional components at the next lowest level and continuing the process until the most elementary level is reached. It may be distinguished from bottom up design where functional components at the lowest level, i.e. routine, program, module, are defined first. These components are then linked through further system definition to succeedingly higher levels until the highest or system executive level is reached.

3.2.1 Development Phases

This programming practice is applied during the design phase of the development cycle, however the design that results will affect all following phases. The value of this technique stems from its inherent ability to promote design completeness in

each level of the hierarchy by forcing each level to be logically complete within itself. The interfaces between the system functional components are identified and can be demonstrated after the top level program has been developed through the use of dummy execution routines. This process leads to detection of integration problems early in the development cycle when they can be corrected at a much lower cost.

The three CSC projects reviewed all used top down design and realized benefits in both the management and technical areas. This technique supports project management through its systematic, hierarchical design process and functionally consistent task and development milestone definition. Top down design promotes functional traceability as system requirements are translated into system components. The hierarchical approach insures clearly defined functional paths as progressively lower level system components are defined. This design process promotes optimization because functional paths are allocated based on the logical progression of system requirements down to the lowest level. Top down design supports system maintenance into the operation and support phase through the modular definition of system components. Software modules evolve as functionally complete components which facilitates isolation of malfunctioning system elements. These modules also support system portability by providing a capability for selective implementation of system functions based on hardware or operational limitations.

3.2.2 MPP Comparison

There were no research findings to substantiate that top down design was more effective on any one of the projects than another. The project which did not thread the design did encounter some problems, one being a redesign of the system due to a major concept change, however the problems could not be linked to the original design of the software. The top down design and modular approach to the development of this system assisted in salvaging a minor part of the original code.

3.2.3 Recommendations

Top down design is recommended for a software development where the total software system is to be developed. In some cases where a major part of the system might be "off the shelf" software packages used at lower levels of a system, it might be advantageious to use a bottom up design. When using existing software packages it could be easier to design a system to fit the packages rather than changing the packages to fit the system. The top down approach involves a more detailed functional design than is produced by other design techniques. This means greater initial cost in design development but this is more than offset by later savings in manhours during detailed design.

The benefits discussed in this report and the completeness normally found in the architecture of a top down design are the major factors used for the basis of this recommendation.

3.3 THREADS

The THREADS methodology is a design verification tool. A thread represents a sequence of events that contributes to the satisfaction of a specific system function. This sequence is an internal process and may be defined at any functional level. A single thread demonstrates a discrete series of operational activities from stimulus to response.

In threading the system design the following is accomplished for each operational process:

- Isolate inputs available (stimuli)
- Identify outputs to be produced (responses)
- Determine processing required to produce each output from the inputs (processes)
- Define the sequential processing path from each stimulus to its response (thread)

A complete set of threads defines all processing required to satisfy all system functional requirements. Thus, threading a system design is a process employing a discipline that is external to the system to verify design consistency relative to requirements. THREADS is a uniquely functional approach to design verification that may be used effectively in other roles such as test plan development and, in conjunction with the Threads Management System (TMS), management control.

3.3.1 Development Phases

The three projects which were subjected to MPP evaluation reflect three distinct management philosophies with respect to the use of THREADS. In the case of AEGIS, one subsystem was threaded and those threads were used to verify design specifications and to monitor software development progress. The TRIDENT software system was not threaded. For DD-963, the entire software system was initially threaded during the design phase, and in conjunction with the Threads Management System threads were aggressively used throughout the period of system development. It is worth noting that both AEGIS and DD-963 were successful projects in that they were completed on time and within budget. TRIDENT, on the other hand, encountered serious development problems and as a result was not as successful.

Threads are most effective when they are introduced early in the system acquisition process. Threads employment should be defined within the project management plan as an integral part of the total management effort. System level threads may be used in the iterative process of evaluating alternative architecture concepts preparatory to selecting the optimum architecture for system development. These system level threads must be defined based on system requirements specification. As system development continues, lower level threads support activities such as system and

subsystem design verification, interface definition, test plan development and system integration.

For the DD-963 project, analysis and design phase activities included revision of Computer Program Performance Specifications (CPPS) and writing the Computer Program Design Specifications (CPDS). During these phases, CSC personnel threaded the system. It was necessary to extensively revise the CPPS. The threading process was carried on in conjunction with writing the CPDS. This provided an extremely effective vehicle for linking the design document to the performance specifications. Developing threads during the analysis and design phases assured more effective use of available manpower in addition to improving the quality of the design. Also these threads, developed early in the acquisition cycle, continued to enhance the system development process throughout later phases.

The AEGIS Command and Decision subsystem was not threaded until after detailed design was established. Threads were defined after the fact to verify the design specifications. This was the first use of the THREADS methodology and thus involved a period of learning for those developing the threads. Nevertheless, they were effective in design verification and continued to be used in support of later development phase activities.

During the code and checkout phase for both the AEGIS and DD-963 projects, threads were used by project managers to monitor and report system development status. Thread development status reported through the TMS is a reliable indicator reflecting progress toward achieving project milestones. Completion of specific threads within each build was easily recorded and periodically this data would be reported in summary form for project monitors and managers to review. This provided all concerned personnel with current, easily understood project status information. Recording the thread data requires the expenditure of some manpower, but not a significant amount. For instance, on the DD-963 project approximately 50% of one person's time on a weekly basis was devoted to maintaining a threads data base.

Test and integration phase activities for the DD-963 project included developing and executing test plans built around the system, subsystem and module threads. Threads provided the basis for the stimulus/response type material included within the test scripts. This was a natural outgrowth of the development process to this point. Test team members could be assured of thoroughly exercising all software functions because of the one-to-one correspondence with system design elements. Later, the integration of system components was facilitated for both DD-963 and AEGIS personnel because of the close correlation among all system components achieved through threads definition. Threads provided the framework for positive control of the test and integration phase.

There is little research data to support positive conclusions concerning threads effectiveness in the installation, and operating and support phases. Research was limited

to preinstallation development for all the projects. It seems reasonable to assume, however, that the operating and support phase would benefit from threads documentation. With the system functions clearly defined, it should be easier to make the modifications and functional adjustments normally associated with this phase.

3.3.2 MPP Comparison

The AEGIS and DD-963 project staffs made effective use of the system or subsystem threads. Threads were employed with equal success on both projects as a design verification tool. Later, through the TMS, threads became an invaluable aid in maintaining management and customer visibility of software production status. Threads status reports were considered by the staffs to be one of the most current and definitive indicators of production progress. For AEGIS the reports were primarily used for internal management control and monitoring. The DD-963 staff employed the reports internally and also used them regularly to keep the customer informed. Once the build demonstrations and phased integration began on the DD-963 project the threads played a dual role. They continued to be used to report progress. In addition the test group used threads to develop and modify test scenarios. In general the DD-963 staff seemed to employ threads a bit more aggressively than the AEGIS staff did. Thus threads had greater impact on the DD-963 project.

3.3.3 Recommendations

The decision to use THREADS should be made early in the acquisition life cycle. Threads are most effective when introduced early in the analysis phase to support the design verification effort. Once the commitment is made to use threads and they are defined, their utility increases over the life of the project. In addition to providing traceability directly from requirements to all levels of design, threads support effective management of software development and maintenance. This includes management visibility into development progress, test scenario definition and execution, and system and subsystem integration. THREADS must be considered an important element of a coordinated software management plan.

3.4 BUILDS

A "build" is a collection of functionally related processes that is implemented and demonstrated as a set. System development employing builds includes early visibility of actual system functions, incremental availability of actual operating capabilities, and distributed integration. By its definition, a build is functional. Because a build is composed of individually operational processes, the integration of each build into the system supports progressive, functional level testing. When each build is operational, it provides a capability that is potentially totally operational, rather than pieces and parts that may be only partially operational.

3.4.1 Development Phases

The builds concept is applied in the design phase to support detailed design by identifying related threads or processes and combining them into functional entities. Actually, threads are produced using the functional design and builds are established prior to detailed design. During this same time frame the development schedule is determined and a completion date for each build is set. These completion dates are used as milestones. The benefit of this technique is realized during the test and integration phase. With each build being tested as it is completed and integrated into the system, any interface problems can be detected and corrected early in the development cycle. The other major benefits derived from this technique are an ability to measure the performance of the developer through the system capabilities being performed during the incremental demonstrations, as well as assuring the user that the system functions do perform in the manner in which he had envisioned.

When used properly this programming practice can be a very effective tool. By grouping related functions or processes together into demonstrable segments, the developer can control the system at a much lower level. This technique promotes incremental development with early integration and provides milestones, as well as, visibility of progress.

During the development of the AEGIS system, the need to segment and build the system in increments was realized. The threads technique was developed to segment the system and functionally related threads were grouped into builds to provide functional components. These tools proved very effective throughout the development cycle and were refined and enhanced for use on other projects.

The DD-963 project senior level personnel came from the AEGIS project and the techniques that had proven effective were brought along for use on DD-963. The threads and builds concepts were used and have been very successful. The builds on this project were used to demonstrate functional capabilities to the customer (U.S. Navy) who required a high degree of visibility during the development cycle. The builds were set as milestones for tracking development and were monitored very closely. Even changes in the schedule of builds because of unavailability of hardware or due to customer requirements were made with minimum impact to production efficiency.

3.4.2 MPP Comparison

For all three projects builds provided a means for expressing the functional decomposition of the detailed design. Thus, the project management staff of each of these projects was able to break down the system to be developed into functional capabilities. Each system level function became a build. This approach led quite naturally to the creation of system development milestones based on builds production. Thus, a production schedule was established that was realistic in its goals and flexible

enough to be adapted to changing circumstances. When threads were used to develop the builds the two MPP together became a powerful management tool. Under these conditions builds fostered the phased integration of system capabilities as they were produced. In addition, with or without threads, builds provided a medium for maintaining excellent management and customer visibility. The build milestone schedules clearly defined the system decomposition and presented an understandable plan for production and integration of all system capabilities.

3.4.3 Recommendations

Builds are an effective adjunct to threads and TMS for management control and monitoring of the software development process. Using threads as elements of the builds leads quite naturally to functional differentiation of the system. For instance in a real time, command and control system such as that developed during the DD-963 project, builds were defined for major subsystem functions - Manual Tracking, Beacon Video Processor, Underwater Fire Control System, etc. Build milestones should be established at regular intervals based on program development capabilities. Again, using DD-963 as an example, builds were demonstrated (milestones) on an average of every eight weeks. The schedule varied because builds varied in size ranging from 32 to 542 threads. The average size build contained 167 threads.

However, thread definition is not a prerequisite to the employment of builds. Builds may be defined based solely on functional decomposition of the system at a high level. In either instance builds provide a method for establishing development milestones and instituting incremental production and integration of software subsystems. The build demonstrations, where functional capabilities were tested to show the user that the software satisfied system requirements, also were vehicles for confirming that previously demonstrated capabilities still functioned as designed. This was accomplished through regression testing which was part of every build demonstration. Regression testing consisted of executing test scenarios from earlier build demonstrations. The test coordinator selected scenarios that would demonstrate key operations which were related to the current build's capabilities.

Therefore, builds are recommended for any large software system composed of multiple functional subsystems. For systems that are threaded the builds will be a natural evolutionary step in system definition. The incremental development approach which builds facilitate is a powerful management tool particularly for large, on-line systems.

3.5 THREADS MANAGEMENT SYSTEM

The THREADS Management System (TMS) serves to automate and simplify the process of creating, updating, interrogating and reporting THREADS information to management. The standard TMS output reports include build status, thread status, thread descriptions, threads schedule and build summary. Examples of these reports

are shown as Figures 3-1 through 3-5. The TMS operates on a variety of computers and provides a wide range of capabilities depending on the requirements of the user.

3.5.1 Development Phases

The TMS could be implemented at any phase of the development cycle. The maximum utilization of this tool is realized by implementing it when the system is threaded or immediately following detailed system design. The TMS can be an effective tool in all phases of the program life cycle once threads are defined if all capabilities of it are exercised. Project planning, configuration control, monitoring, testing and verification/validation are all supported by threads. The TMS is also applicable after the development cycle as a configuration management and change impact assessment tool during the maintenance phase.

3.5.2 MPP Comparison

The effect this programming practice had on the three projects varied greatly. The TRIDENT project was not threaded and had no TMS. The AEGIS project was partially threaded and the TMS was utilized as a monitoring tool. The DD-963 project was totally threaded and the TMS was a major contribution in all phases of the development cycle. The DD-963 project used the threads to verify the design back to the requirements, develop the builds, plan the development schedule, monitor the progress, and develop the test scripts; and they will be used as a software configuration tool during the maintenance phase.

The THREADS concept and the TMS were conceived on the AEGIS project and were used very effectively. However, the DD-963 project used the technique after it had been enhanced and the benefits derived from the TMS during this project was greater due to the additional capabilities. The effect of this practice is proportional to the degree of usage on the three projects. It had no impact on TRIDENT, it was considered a very effective tool on AEGIS and a key element on DD-963 project in providing management visibility. The TMS also supported other MPP used during the development of the DD-963 software system.

3.5.3 Recommendations

CSC recommends the usage of a Threads Management System, when threads are implemented, to receive the maximum benefits from a THREADS data base. There is some cost involved in maintaining and updating the data base; for instance, 50% of a non-senior, technical staff member's time. However, the TMS is an effective management tool that provides definitive data on program development status. The benefits gained are in improved management control and visibility.

THREAD MANAGEMENT SYSTEM RETRIEVAL

LEVEL: SUBSYSTEM
RUN DATE: 11/15/72

NAME	DESCRIPTION	REFERENCE	SUBSYSTEM ELEMENTS
CM01	ATC	3.3.2.2.1.2	CDI CMS CSN
CM02	TD	3.3.2.2.1.2	CDI CMS CSS
CM03	MSS/EC	3.3.2.2.1.2	CDI CMS CED
CM04	SS	3.3.2.2.1.2	CDI CMS CTS
CM05	DT	3.3.2.2.1.2	CDI CMS CDT
CM0E	IDLE	3.3.2.2.1.2	CDI CMS CIS
TC07	HOLD CLOCK	3.3.6.2.1.1A1	CDI CED
TD08	RUN CLOCK	3.3.6.2.1.1A1	CDI CED
TD09	SET CLOCK	3.3.6.2.1.1A1	CDI CED
TD10	CRT PRINT	3.3.6.2.1.1A1	CDI CED CPX
TD11	SYS STATE PRINT	3.3.6.2.1.1A1	CDI CED CPX
TD12	ALL SIMUL START	3.3.6.2.1.1A1	CDI CED CWI CMI
TD13	ALL SIMUL STOP	3.3.6.2.1.1A1	CDI CED CWI CMI
TD14	RECORD START	3.3.6.2.1.1A1	CDI CED CPX CWI CMI
TD15	RECORD STOP	3.3.6.2.1.1A1	CDI CED CPX CWI CMI
TD1E	LOCAL SEQ	3.3.6.2.1.1A1	CDI CED
AV17	AVAIL STATES	3.3.6.2.1.1A1	CDI CED CAS
AV18	DISPLAY	3.3.6.2.1.1A1	CDI CAS CDS

Figure 3-1. Threads Management System Retrieval Report

This report depicts selected data concerning all the Threads at a designated level.

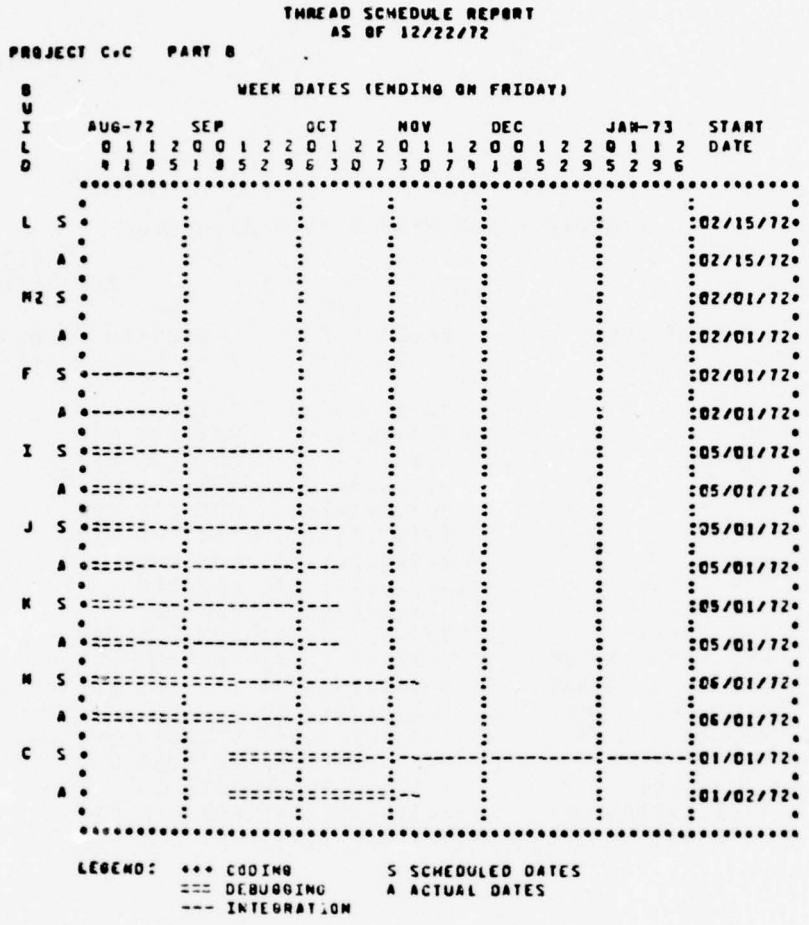


Figure 3-2. Thread Schedule Report

The Thread Schedule Report indicates the status of each Build. Both scheduled and actual activities for each development phase are shown. Also included are the Build start dates and a legend to assist the reader in understanding the chart.

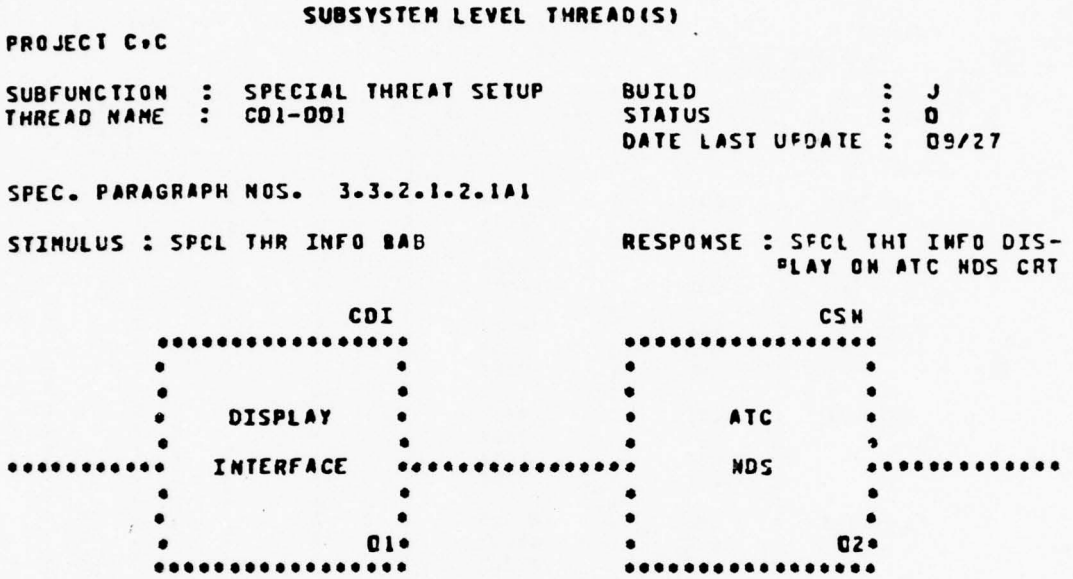


Figure 3-3. Subsystem Level Thread Report

This report presents a graphic representation of the Subsystem Level Threads. The boxes represent the subsystem elements and the processing required within each element. The Threads that require more than two boxes would display the additional boxes directly below the two shown.

PROJECT C.C SUBSYSTEM LEVEL THREAD STATUS PAGE 001
DATE 12/22/72 FOR BUILD C
TIME 17:00:00

THREAD NAME	SUBSYSTEM ELEMENT	DATE OF LAST STATUS CHANGE	DESIGN	CODE	DEBUG	INTE-GRATE
C01-093	CDI	12/04		X		
	CSL	12/04		X		
C90-093	CDI	12/04	X			
	CSL	12/04	X			
C03-025	CDI	9/27	X			
	CSM	9/27	X			
	COS	9/27	X			
C01-378	CDI	9/27			X	
	COS	9/27			X	
	CRI	9/27			X	
	CTC	9/27			X	
E99-007	CCX	09/23				X
	DOX	09/23				X
	COS	09/23				X

Figure 3-4. Subsystem Level Thread Status Report

The Subsystem Level Thread Status Report gives detailed information concerning each individual thread. The report is produced for any designed build. It lists each thread by name and shows the subsystem elements involved in each thread. The date indicates when the TMS data base was last updated for this thread information. The status is shown for each development phase and by subsystem element. It refers to that portion of the element that pertains to the thread shown. In the example, the CDI subsystem element has completed code for the first thread, but has only completed design for the second, and so on.

PROJECT C.C SUBSYSTEM LEVEL BUILD STATUS PAGE 001

DATE 12/22/72
TIME 17:00:00

BUILD	THREADS	DESIGNED	CODED	DEBUG	INTE-GRATED	BUILD STATUS	COMPLETION DATE SCHDL.	ACTUAL
A	10	100%	100%	100%	100%	100%	11/01/71	11/01/72
B	26	100%	100%	100%	100%	100%	1/03/72	1/03/72
D	25	100%	100%	100%	100%	100%	1/14/72	2/01/72
E	23	100%	100%	100%	100%	100%	2/01/72	2/01/72
G	5	100%	100%	100%	100%	100%	2/15/72	2/15/72
H	71	100%	100%	100%	100%	100%	4/04/72	4/04/72
M1	11	100%	100%	100%	100%	100%	5/01/72	5/01/72
PART A SUMMARY	171	100%	100%	100%	100%	100%	6/01/72	6/06/72
L	6	100%	100%	100%	100%	100%	6/01/72	6/06/72
M2	36	100%	100%	100%	100%	100%	7/03/72	7/03/72
F	102	100%	100%	100%	100%	100%	9/01/72	8/24/72
J	66	100%	100%	100%	100%	100%	10/16/72	10/13/72
I	7	100%	100%	100%	100%	100%	10/15/72	10/13/72
K	17	100%	100%	100%	100%	100%	10/15/72	10/13/72
N	42	100%	100%	100%	100%	100%	11/01/72	11/01/72
PART B SUMMARY	276	100%	100%	100%	100%	100%	11/16/72	11/18/72
PART A,B SUMMARY	447	100%	100%	100%	100%	100%	3/15/73	3/15/73

Figure 3-5. Subsystem Level Build Status Report

This report is designed to provide an overview of the build status. It includes the build name, the number of threads that are identified in each build, and the percentage of completion for each development phase (design, code, debug, and integrate). Each thread has either completed or not completed a particular phase. The overall build status is derived by the use of an algorithm that gives "weights" to each of the development phases. This is necessary since the coding of a thread requires less effort than the debugging phase. A comparison of the scheduled and actual build completion dates will give an indication of the project status. The sample shows a Part A and Part B summary. Each Part is a logical collection of builds and represents a significant milestone. The analysis of a particular system would determine whether or not this particular feature is applicable.

3.6 CHIEF PROGRAMMER

The Chief Programmer is the senior technical staff member assigned to a specified organizational level. For a project employing 20 to 30 technical staff members the Chief Programmer may direct the entire software development effort. On the other hand a project requiring a technical staff of 100 to 150 personnel may use two or three Chief Programmers. The span of control assigned to any Chief Programmer position depends very heavily on the technical and management expertise of the individual selected for the position. A Chief Programmer is the focal point for all software development within his organizational boundary. He must be expert in his mastery of all elements of program development, i.e., computer languages, hardware, JCL, etc. He is a manager also, assigning tasks, monitoring progress and adjusting resources to maintain schedules. Much of a Chief Programmer's responsibility is compatible with a Section Manager's. But, a Chief Programmer fills a position of broader responsibility than a Section Manager as the most senior or one of the group of most senior technicians on a project.

The Chief Programmer does not function in a team chief role as described in the Structured Programming Series. He has a task group of programmers and analysts involved in coordinated subsystem development. There is no group librarian or configuration management function. These requirements are satisfied by project staff personnel who perform these functions for the total development effort. Likewise, quality assurance and testing are project functions and thus outside the group's responsibility.

3.6.1 Development Phases

The two Chief Programmers for the DD-963 project were assigned project staff positions. Their basic responsibilities did not vary appreciably from phase to phase during the project. They were primarily concerned with technical direction for assigned software production. They did lead individual programming teams which were assigned specified builds to develop and test. The department manager provided management support which freed the Chief Programmers to direct software development with minimum distraction. Overall this was an extremely effective project organization. It was successful in part because the project staff was small so that the span of control for each Chief Programmer (10 to 12 personnel) and the Department Manager was within manageable bounds. Also, the superior professional and technical skill of both Chief Programmers and the Department Manager was a significant contributing factor.

During all phases of system development the Chief Programmers were focal points for software production. This began during the analysis and design phases when they were deeply involved in preparation of the performance and design specifications. They also participated in the definition of system threads during this phase. Later, during the code and checkout phase this experience was applied to the production and

demonstration of system builds. They assigned specific modules and segments to programmers within their groups and monitored the development and testing of the program code. As leaders of the build development teams they provided technical direction and guidance to team members through demonstration of the build. Throughout the development of this system the Chief Programmers were a key link between the project analysts and programmers, and the management staff.

3.6.2 MPP Comparison

Since the Chief Programmer function was applied to only one project there is no basis for comparison of its effects among the other projects.

3.6.3 Recommendations

The Chief Programmer position as used on the DD-963 project was successful primarily due to these factors:

- Size of the project staff was such that two programming groups provided optimum coverage of program development requirements.
- Programming groups were large enough to encompass a broad range of programming expertise without sacrificing supervisory control.
- Individuals assigned as Chief Programmers were able to provide outstanding technical direction and control.
- Department Manager provided administrative and program development support so that Chief Programmers could devote their full attention to software production.

It would be difficult to quantify all of the factors which led to the successful application of a Chief Programmer function because many relate to personnel interactions. It is possible that given the proper mix of personnel a much larger project could effectively use this staff structure. Certainly, a smaller project could easily implement a Chief Programmer organization such as the one DD-963 used.

It is possible that with the same project staff another type of staff organization would have worked as well or better than the one used on DD-963. To recommend using a Chief Programmer team structure one must consider all the factors mentioned in this discussion. There are no definitive guidelines. It is obvious, however that this organization can be used effectively and can enhance the technical direction and controls applied to a software acquisition project.

3.7 BUILD LEADER

A build leader is a specialized task leader whose sole responsibility is to supervise production of software to support a build capability. As in most task oriented organizations the build leader may have other non-related or compatible responsibilities.

As an example, Chief Programmers were build leaders for most of the system builds during one project. The build leader directed the efforts of the task group of programmers and analysts assigned to a particular build. This group functioned as a unit until the build was demonstrated. Build leaders were instrumental in timely production of system capabilities, and in the detection and correction of design and code errors.

3.7.1 Development Phases

Build leaders were used only during the Code and Checkout, and Test and Integration phases. They were used with equal effectiveness across both of these phases. Perhaps the greatest benefit to be gained from using build leaders is in having software development teams working with well defined, limited objectives. Work progress can be readily determined and personnel can be assigned to specific tasks. The build leader and his group functioned as an action team within the project staff. Once the build was demonstrated these people were reassigned to other tasks. In this way, manpower becomes a more flexible resource for use within the project. This was true of the way build leaders were employed through both development phases.

3.7.2 MPP Comparison

Data on the use of build leaders was only available from the DD-963 project. Therefore, no comparison of their effects among the projects can be made.

3.7.3 Recommendations

If a management staff elects to employ threads and builds to promote phased production and integration of system capabilities, then build leaders are an appropriate technique to assure their effective use. Build leaders can be used on projects of all sizes. The project staff organization must be structured so that the build leader has available to him the staff support he needs to accomplish his assigned tasks. For example the DD-963 organization included configuration management, quality assurance and program librarian functions that were common to all task groups. Thus, the build leaders could devote their full attention to program development. The total project staff size was small enough to accommodate this staff organization. Build leaders and their groups were able to have direct and frequent access to support personnel during build production.

3.8 PROJECT REVIEWS

The review process used by CSC on the projects under study included both informal and formal reviews. Preliminary Design Reviews (PDRs) and Critical Design Reviews (CDRs) were required by contract and were formal reviews. In Process Reviews (IPRs) were more informal and were held throughout the development cycle to provide the customer and CSC management, information concerning the current

project status.

3.8.1 Development Phases

For most projects, after the CDR is conducted near the end of the design phase, there is very little user involvement in the software development process. Once the detailed design is approved the user must content himself with patient anticipation while awaiting delivery of the final product. This approach can lead to development of software that does not satisfy user requirements. Also the developer does not have a convenient vehicle for advising the user of program development problems which might be resolved by modifying system design or requirements. The In Process Review (IPR) is an attempt to keep the user and/or program management staff involved in program development through the period of software production and testing. IPRs are conducted regularly and involve face-to-face contact between user and developer personnel.

The design reviews were conducted during the design phase of the project and in most cases after the CDR the customer involvement with the development is almost nonexistent. CSC attempted to keep the customer involved throughout the development phases by the use of regularly scheduled in process reviews.

3.8.2 MPP Comparison

The design reviews for the three projects were very much standard, but different in number and length because of the size and complexity of the projects. The IPR approach, however, had a very positive impact on the projects by providing management visibility and customer involvement.

The TRIDENT project had very few IPRs in the early phases of its development, however after various problems had been encountered and the milestones were not being met, the customer became more involved. This resulted in regular reviews. These reviews provided the customer with visibility into progress and into the problem areas and assisted in expedient resolution of problems.

The AEGIS project was conducted in a research and development environment with changing requirements and many contractors involved. IPRs with RCA and Navy personnel became a way of life early in the project and were continued throughout the project. The personnel interviewed during the research felt that the IPRs played a major role in the success and timely completion of the project. The data presented at these reviews were current and factual to show the current status of the overall project.

The DD-963 project held IPRs on a scheduled basis at the customer's request. This was a time and materials contract and the customer desired a high degree of visibility. The effect of this tool was much the same on all projects. The reviews

established a good rapport with the customers and instilled confidence in the product being developed.

3.8.3 Recommendations

Project reviews are essential for visibility and control of all software development projects. The following recommendations will improve the conduct of these reviews:

- The review dates should be scheduled as milestones.
- The review dates should be rescheduled if program development is delayed for any reason. The work must be complete in order for the review to be meaningful and not just a review for the purpose of meeting the milestone.
- The material to be covered should be sent to all attendees well enough in advance of the meeting for the attendees to properly review the material. This procedure will result in more pertinent questions into specific areas and reduce the time required to conduct the review.

The use of in process reviews is recommended for all projects.

3.9 PROGRAMMING TECHNIQUES

There were three distinct programming techniques used during development of software for these projects. Collectively they represented formal attempts to structure the programming process. Structuring the process is important because it provides a standardized product (program code) and introduces a procedural approach to what has a tendency to be a disorganized activity (programming). The name of the technique or the particular rules which support it are less important than the fact that a structured technique is used and enforced. This is the key to success in using any programming technique.

Structured programming is a method of writing programs with the logic flow always proceeding from a single entry through a single exit without arbitrary branching. To support this method, a system must be segmented into small programs that can be written with a limited number of logic structures to develop clear, readable and maintainable programs.

Functional programming involves the development of software based on functional routines that are integrated to create system and subsystem capabilities. This programming technique employes well defined coding conventions to produce routines in standardized formats for use within the system.

Structured modular programming is a modification of structured programming which is less restrictive in the area of structured constructs. For structured modular programming the technical staff is permitted to use "GO TO" statements and may deviate

from the single entry/single exit requirement of structured programming. However, the intent of structured programming is followed with respect to data names, code indenting and program length.

3.9.1 Development Phases

These programming techniques are applied during the program design and coding phases, with the benefits realized not only in the coding phase, but in the test and maintenance phases as well. For structured programming the detailed design will involve more intense activity than for other programming techniques, because of the level of detail required for structured programming. However, this detail will decrease the effort required to produce program code. All three of these techniques improved productivity in the test phase because of the smaller (usually one program per computer print out page), less complex programs. Also the improved readibility of the standardized coding structures will have a positive effect on program maintenance in the operation and support phase.

3.9.2 MPP Comparison

The comparison of these techniques over the three projects reviewed is difficult because only one project used pure Structured Programming (SP) and the other two used a modified form of SP. The TRIDENT project which used the pure form of SP had some problems during the development of the software. One of the problems was core usage under SP constraints. It was realized toward the end of the coding phase that the core budget would be overrun. This resulted in some reprogramming to optimize core usage while maintaining SP code. During interviews with the TRIDENT staff, the analysts stated that they felt the program design required a longer period of time than that for free form programs because of the need to define the program structure at a more detailed level. But, the coding phase was reduced to almost half the normal time due to the design being developed at almost the code level. The supervisors on these projects thought the constraints of SP did not effect the productivity of the senior level personnel to any degree, however a noticeable adjustment was required by the junior level personnel to develop program logic in a structured manner. It was also felt that the quality of the programs produced by the junior staff members using SP was higher than would ordinarily be expected.

The total effect of structured programming on the TRIDENT programs cannot be determined at this time since one of the measures of effectiveness is maintainability and the product has not reached that phase in its life cycle.

On the AEGIS project, as a test, several programs were written by CSC senior programmers using the SP constraints and then rewritten using a structured format with free-form code. The results of the test indicated that structured programs required 5% to 8% more core. Since the core allocation was a key factor on the AEGIS project a modified version of SP, functional programming, was selected for this project.

A similar technique was used on the DD-963 project were structured modular programming was employed. The modified versions of SP used on the AEGIS and DD-963 projects were successful in producing quality programs within the core limitations. The programs written on these projects did not adhere to SP constraints, however, both projects were developed under rigid coding conventions and used a structured format to ensure standardization of programs. The AEGIS EDM-1 system has been operational since 1974 onboard the USS Norton Sound with software trouble reports (STR) being sent to CSC for corrections.

3.9.3 Recommendations

Structured programming (SP) or a version of structured programming would be recommended for any system. The more complex and interactive a system is, the greater the requirement for a standard structure. The compiler being used will have some influence on any decision to use pure SP or not. In some cases a compiler without all of the SP verbs, or a preprocessor, will generate inefficient code and/or use excessive core. However, whatever technique is used, there should be specific coding conventions to dictate development of structured programs with straightforward logic flow which have the characteristics of clarity, readability and maintainability.

3.10 PROGRESSIVE TESTING AND INTEGRATION

Progressive testing and integration is the process of verifying and integrating segments of a system in a time phased schedule as the system is being developed. The purpose is to detect any errors as early as possible, provide the customer with early visibility and avoid the conventional process of testing and integrating the total system in one process at the end of the development phase.

3.10.1 Development Phases

This technique is applied during the code and debug phase with the benefits being realized through the system integration phase. The immediate benefit is the visibility of progress that is demonstrable to management and the customer. The prime benefit of progressive testing and integration is early error detection and correction. Figure 3-6 depicts the errors detected and documented by trouble reports during a three year span covering program design, program development, system integration and maintenance for a large system development project. The peaks on the graph during the first year represent the errors detected during the testing and integration on the builds for that year. There were 755 errors detected during the program design and program development phases. Without phased testing and integration these errors probably would not have been detected until system integration when the cost-to-correct is much higher. With phased testing and integration the overall development schedule is shorter, since the progressive testing and integration is accomplished in parallel with the coding of the next build.

CSC

SYSTEM DEVELOPMENT TROUBLE REPORTS

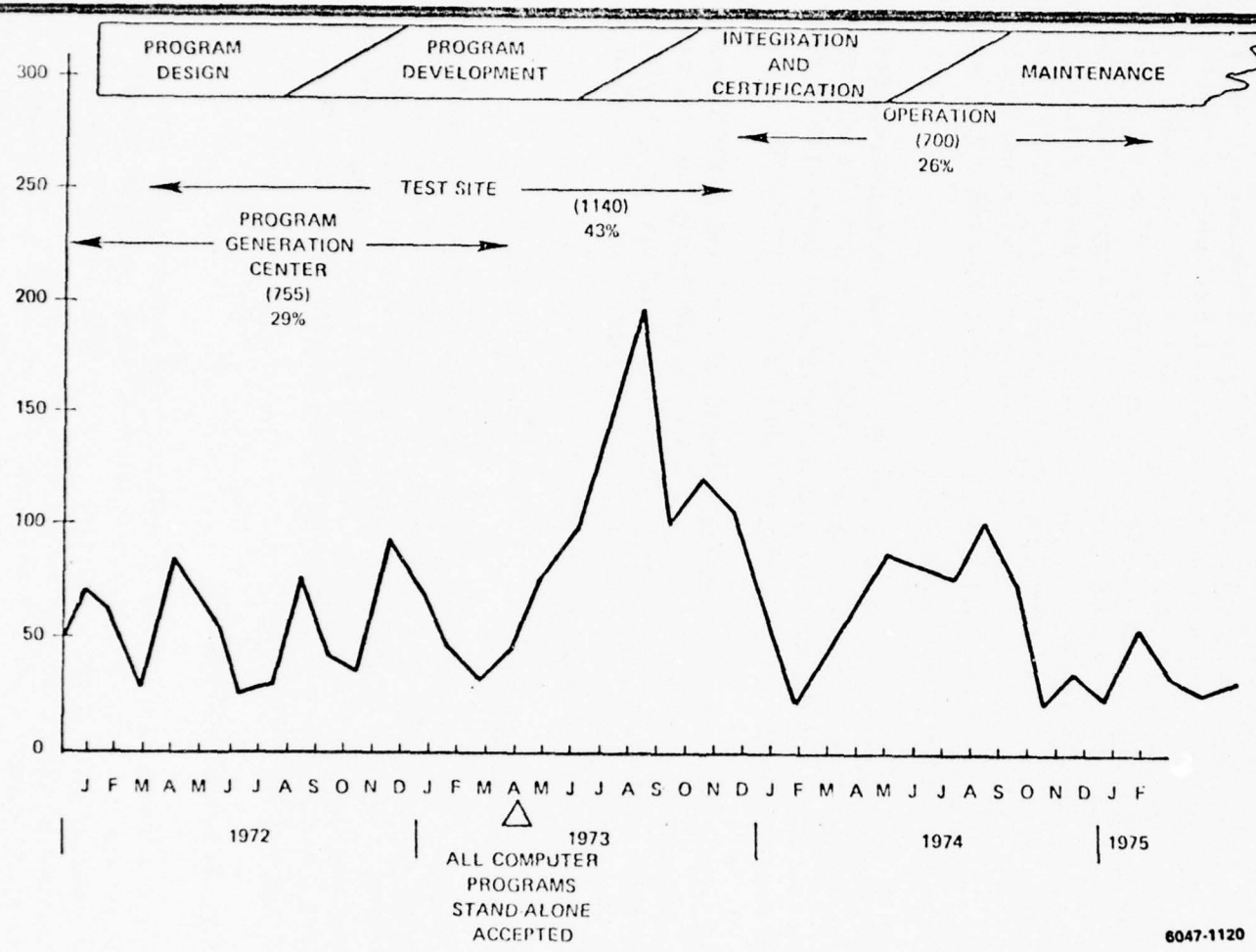


Figure 3-6. System Trouble Reports

3.10.2 MPP Comparison

This technique was used on all three CSC projects reviewed in conjunction with builds. The THREADS/Builds concept to identify functional components for development and test, an independent test group to ensure all system functions are tested, and a software configuration management group to control the baselined portion of the system are all supported by progressive testing and integration. During the integration of the builds into the baselined portion of the system, regression testing is performed to ensure previously implemented system capabilities were not affected. The AEGIS project used these techniques and the system integration phase was reduced due to many interface problems being resolved during the integration of builds. This does not mean that there were no problems encountered during system integration, but the impact of problems detected during this phase were greatly reduced by applying this practice during the previous phases.

The DD-963 project was similar to AEGIS in that the customer required a high degree of visibility. With the use of progressive testing in conjunction with other MPP, CSC was able to provide the customer with current status reports of the system progress and demonstrate the system capabilities as they were developed.

The total effects of progressive testing on the DD-963 and TRIDENT projects cannot be determined at this time as they have not reached the system integration phase, however based on the results of the AEGIS project and the benefits already realized on the DD-963 project, such as early error detection and early visibility this technique has proven to be a very effective tool.

3.10.3 Recommendations

The use of this technique is recommended with any project where communication between system components is required. A system which has a great amount of interactions, such as a command and control system, will benefit the most from progressive testing and integration. However, when used in conjunction with the other MPP discussed the cost of applying the tool is minimal and the benefits derived make it very cost effective.

3.11 SUPPORT PROGRAMS

Support programs comprise all software which is functionally external to an operational system, but executes in conjunction with that system to provide diagnostic, simulation and translation services. This includes computer programs and subsystems providing program compilations, core dumps, timing data, message traps, selected memory register dumps, interface simulations and interactive simulations and interactive instruction manipulation. In addition, on the DD-963 project program documentation was enhanced through the use of an automated program flow chart generator.

Support programs were used on all three projects studied. The AEGIS project was the source for many of the support programs adapted for use on both the TRIDENT and DD-963 projects. These programs were implemented as former AEGIS staff members recognized their utility under similar program development conditions on both TRIDENT and DD-963. In particular, the DD-963 managers initiated a comprehensive support software package supplementing the AEGIS programs with additional support systems to promote increased productivity from their limited staff. On both the AEGIS and DD-963 projects support programs played a significant role during program development and testing.

3.11.1 Development Phases

Support programs were used on all three projects studied. Their impact on the development process in each case was heavily dependent upon the level of application. These projects presented different management approaches toward their use.

This practice affected all phases in the program life cycle of the AEGIS project. The support software was defined during the design phase as to the types of software needed and what phase in the development cycle it would be required. It was realized early, that most of the support programs required to develop this large weapons system would have to be designed and developed by the project staff. This decision was influenced by the fact that the computer to be used was new and very few debug programs were available for testing real-time programs, also the hardware was being developed in parallel with the software which meant that simulation programs would be required for software checkout. These factors and the contractually required support software resulted in 63% of the total system software being support software. (See Figure 3-7.)

The support software employed on the TRIDENT project consisted of debug routines and simulators used during the test phase. For the most part these were programs developed for the AEGIS program and were resident in the Program Generation Center (PGC) libraries. Both AEGIS and TRIDENT shared the PGC development facility.

The DD-963 technical staff was able to increase productivity during the code and checkout phase through the use of the support programs. The technical staff was not large considering the size of the system to be developed. Also, the customer had specified a shorter development period than had been originally proposed. These factors combined to create a sizeable workload on all members of the technical staff. The department manager and supervisors stated that without the benefit of this support program package the delivery dates for system and subsystem components could not have been met.

For the DD-963 project the code and checkout, and test and integration phase activities overlapped. Through the build concept, selected system functional capabilities

TACTICAL			SUPPORT		
	NUMBER OF MODS	CORE (K)		NUMBER OF MODS	CORE (K)
SPY-1/ORTS			UTILITIES (ATUL/DMSO)	15	57
o CPU1	11	32			
o CPU2	19	35	TEST MODULES	10	45
C&C	36	83	SIPCO	20	69
ATEP	9		DATA REDUCTION	27	52
o KERNAL		8	BOSS-7	8	26
o TASK STATE		5			
			SIMULATORS		
COMMON SERVICE	12	8	o SPY-1	1	3
			o ORTS (TIFS)	6	4
			o C,C (LIFS)	13	39
(SUB TOTAL	87	171)	(SUB TOTAL	100	295)

TOTALS: 187 MODULES, 466K CORE

Figure 3-7. AEGIS EDM-1 Program

were developed and demonstrated on an incremental basis. In this way system functions were integrated while being tested in an orderly fashion. The support programs were equally important during this period when a great deal of individual program troubleshooting and code modification was taking place. Throughout this period programmer productivity remained high as the project technical staff made effective use of the support programs.

The study of support programs' effectiveness did not extend into the installation or operation and maintenance phases. The project had not entered these phases while the software production data research was being conducted.

3.11.2 MPP Comparison

Support programs were used very effectively on both the AEGIS and DD-963 projects. For AEGIS much of the program test software was developed under the contract for delivery to the user. Some DD-963 support programs were also delivered to the user. In obth projects the technical staffs made full use of all available software aids and modified the programs when necessary to make them more effective. This had a very favorable impact on the software development process for both projects. In particular, the DD-963 task leaders stated that they would not have been able to meet scheduled milestones had it not been for the program debug tools available in the support software.

The TRIDENT project staff did not appear to depend very heavily on the support programs which were available. As stated earlier, they used AEGIS support programs that were accessible to them through the PGC program library.

3.11.3 Recommended Use

Support software is required for most large software development projects. The type and quantity vary with the product being developed. The data collected during this research indicates that without the proper support tools the successful completion of these projects could not have been obtained in the required time frame. CSC recommends evaluation of existing support programs that could provide, with the least amount of software modification, the necessary support of system development efforts.

3.12 SOFTWARE CONFIGURATION MANAGEMENT

Software configuration management (SCM) is the process of controlling a system configuration throughout its life cycle to avoid unauthorized changes after the baseline has been established.

3.12.1 Development Phases

SCM is applied throughout the life cycle of a software system, from the time the design document is approved and baselined through the operation and maintenance phase. The benefits of this tool are realized when a change or an enhancement to the system is required. The documented or designed configuration must be the same as the operating configuration in order for the maintenance personnel to determine which portion of the system requires modification. If any part of a system has been "patched" or changed without updating the documentation, a modification or enhancement could effect the system performance by incorrect interface with the patched portion of the system.

This programming practice was applied to the projects reviewed and varied in relation to the amount of funds available for configuration control. The AEGIS project, the largest on the three studied had a configuration management section with five personnel to control the large volume of documentation and programs being produced during system development. During the acceptance testing phase this section controlled multiple versions of the system to conform to the various stages of hardware configuration.

The DD-963 project had a one man configuration control section. This project was smaller than the AEGIS project and control of the baselined system was not as difficult. All changes made to the baselined programs had to be approved by the chief programmer. All changes to the system required approval of the program Configuration Control Board.

The TRIDENT project was also a small project in terms of personnel (20-30 technical) and the configuration control was part of the duties of one person. The configuration control on this project was somewhat lax because under the conditions specified by the prime contractor the system wasn't considered baselined until it was completed. This version of SCM was not as desirable as that used other two projects.

3.12.2 MPP Comparison

The effect of SCM varied over the three projects in proportion to the amount of control applied through that function within the project. The TRIDENT project where the smallest amount of resources was expended in controlling the "current" version of system, encountered some problems. The problems were costly in man-hours and machine time but were not unsurmountable.

The other two projects which placed more stringent control on the baselined version of the system were more effective in controlling changes to the configuration and ensuring the system performed in accordance with the specifications.

In summary the effectiveness of any SCM function is determined by the amount of control applied within a given project.

3.12.3 Recommendations

The use of an SCM function is recommended for all projects. The size of this staff function should be determined by the size of the project. The configuration control plan should be developed during the system design phase and the plan should be enforced to ensure the integrity of the system configuration throughout the life cycle of the project.

3.13 STRUCTURED WALKTHROUGHS

A structured walkthrough is the review of a program design or program code by an evaluation committee. This review is conducted prior to coding as a design review and prior to unit testing as a program logic review. The primary programmer presents his design or code to the committee members who step through the logic of the program to detect errors prior to coding or testing. This technique is sometimes referred to as peer group programming.

3.13.1 Development Phases

The phase where walkthroughs are applied is the coding phase. However, a walkthrough of the program design prior to actual coding can be very effective in detecting misinterpretations of data base structure, communication links and the general approach to the solution of a problem. From this viewpoint, the phases affected by this technique could be described as the design phase through the test and integration phase. This technique is similar to a mini critical design review (CDR) except that it is normally an internal process and less formal than a CDR.

This technique was used on both the TRIDENT and DD-963 projects in a similar manner. The evaluation committee averaged five people, and normally consisted of the analysts whose programs had interactions with the program under review and the primary programmer's immediate supervisor. The design or code was reproduced and distributed to the committee members for review prior to the evaluation date. The reviews were internal to the project and informal to a degree, to encourage constructive criticism from the participants. The dates set for the walkthroughs were part of the milestones for completion of program design and coding as scheduled by the programming manager.

3.13.2 MPP Comparison

The effect of the walkthroughs on the projects reviewed was expressed by the personnel on the respective staffs as cost effective in both man-hours and machine time. The number of personnel involved in a walkthrough ranged from three to seven and

the time required, was about four hours per program. The direct benefit from these evaluations was in the test time being reduced by 50% to 80% because errors were detected prior to the test phase. The indirect benefits are, (a) verification of program interfaces to alleviate integration problems, (b) expanding knowledge of system components over the staff members, (c) accelerating the software production by reduction of test time and integration time, and (d) detecting errors early in development cycle, thus reducing the cost of correction.

3.13.3 Recommendations

It is recommended that this technique be applied to any project since the cost/benefit ratio appears to be very favorable. The indirect benefits derived from most well executed walkthroughs would offset the cost of the participating personnel. This technique is more beneficial to an interactive system where program interfaces are critical, but could be cost effective on stand alone programs as well, due to the cost of correcting errors that could be detected early in the development cycle.

3.14 INDEPENDENT TEST AND EVALUATION

Independent test and evaluation (IT&E) is a method for conducting software verification and validation. It involves designation of a group or team of technical personnel to form an IT&E unit. These personnel, who must be knowledgeable of system requirements and design, are responsible for conducting all system tests prior to delivering the software to the user. This group operates independent of the system development staff in order to provide an objective and unbiased evaluation of the software. The primary objectives of this group are to prove that the software performs as designed and that the performance meets or exceeds all system requirements. It must establish that the system can be implemented, error free, consistent with the performance requirements.

To accomplish these objectives the IT&E group performs the following functions:

- Compare computer program design to system design specifications
- Develop system test plans and scenarios
- Evaluate software performance against system performance requirements

3.14.1 Development Phases

To develop an effective test plan the preparation should begin during the design phase. The test and integration scripts for the AEGIS and DD-963 projects were designed to be executed in conjunction with build demonstrations. The unit testing was done by the individual programmers prior to the demonstrations and the formal testing was performed by test groups using the test scripts to ensure that the functions designed into a specific build satisfied the requirements for that portion of the system.

Using the build concept and integrating each build as it was developed and tested into the baselined system during the programming phase reduced the time required for the final system test phase.

3.14.2 MPP Comparison

The AEGIS project had a Test and Evaluation Department which reported to the project manager. This group generated the test plans and procedures for all tests, conducted and evaluated the tests and retested any programs which required corrective action from a previous test. All test plans and procedures were approved by the customer prior to the actual test and the customer reviewed test results after the test.

The DD-963 project staff had a test group which worked jointly with the Navy in writing the test scenarios from the threads data base. This group reported to the DD-963 department manager. The test scripts were the scenarios used during a build demonstration to ascertain if all functions in the build perform as specified in the requirements document. The results of the test were evaluated and software trouble reports (STR) were written on any problem detected during the test. The STR was tracked by this group until the error was corrected. The test script was executed again to confirm the correction and ensure the corrective code did not effect another part of the build. All tests were made with Navy personnel present and the programs baselined after the demonstration.

The formal testing of the TRIDENT project will be the responsibility of IBM and thus there were no data available during our investigation of this project.

Those involved in directing program development for both AEGIS and DD-963 stated that getting the user involved in the testing process proved to be very successful from two standpoints. First, this provided those developing the software with a first hand look at how the user intends to use the capabilities. Second, the user becomes more familar with the product and can provide comments to clarify any operational inconsistencies.

3.14.3 Recommendations

IT&E groups should be used on all software development projects when there are adequate resources to support them. The costs associated with employing an IT&E group can be as high as 10% of the total program costs. Independent test and evaluation is a valuable MPP and should be employed recognizing the following conditions:

- IT&E personnel disassociated from software development staff
- Test plans developed early using requirements documents and then baselined
- User involved directly in testing to the extent possible

3.15 VERIFICATION PROCEDURES

Formal verification procedures were used only on the DD-963 project. Computer program design and code were verified on all of the projects using MPP such as THREADS, Builds, PDR/CDR, etc. However, it was only on the DD-963 project that verification procedures were specified through documented project operations directives.

There were two verification processes used on the DD-963 project. In order to verify that the program design met the CPPS requirement a top down verification process was used. Then the program code was verified against the CPPS using bottom up verification procedures.

Top down verification was implemented through a multi-step process using threads and builds. That process proceeded as follows:

- Prior to each build demonstration each primary programmer reviewed the programmed threads for which he was responsible to verify their correctness against the CPPS.
- A build demonstration scenario was developed which assured execution of every thread within that particular build.
- The build demonstration was conducted to confirm specific functional capabilities associated with that build.
- Monitor runs (trial executions) were conducted for selected programs or modules to confirm questionable thread execution which occurred during the build demonstration.
- Post build testing was employed to insure that the final post build system configuration was functionally acceptable. It consisted of the following:
  - Exhaustive testing to execute functions over the entire range of values for input data.
  - Negative testing to evaluate system responses to erroneous input data.
  - Degradation testing to assess system operation under subsystem fault conditions.

Bottom up verification was achieved primarily through the use of support programs as program code was developed. The elements of this process were:

- Compare automated flowchart generator statements with the program or segment code to assure a correspondence.
- Create program or segment flowcharts using the automated flowchart generator.
- Verify that the program or segment flowcharts are consistent with the CPDS.

3.15.1 Development Phases

These verification procedures were first implemented during the code and checkout phase when threads were verified against the performance specifications in preparation for build demonstrations. They continued to be used through the test and integration phase as bottom up verification was conducted to confirm that the program code paralleled the design specifications. The net result of these activities was reflected in the following:

- Early definition of subsystem and module interfaces
- Comprehensive design and code analysis
- Error detection enhanced throughout entire period of development
- Subsystem functions proved through exhaustive testing
- Viability of system and subsystem threads confirmed

3.15.2 MPP Comparison

This programming practice was used exclusively on the DD-963 project, and therefore, no comparison can be made of its effects on other projects.

3.15.3 Recommendations

The programming directive that specified the verification procedures to be used by the DD-963 technical staff was a clear definition of the methods to be used in verifying the software. It was a document that all members of the technical staff could use as a guide during program production and testing. Perhaps its greatest benefit is that it offered a coordinated verification plan using the tools and techniques available to the staff. For this reason this practice is to be recommended as an example of a viable plan for software verification. This type of directive could be applied to any software development project.

SECTION 4 - COMPARISON PROCESS

4.1 INTRODUCTION

Comparison of CSC programming practices and structured programming technology techniques is an essential prerequisite for the definition of MPP standards. This comparison of practices and techniques as described herein is based upon analysis of programming practices used by CSC on the three projects researched and upon descriptions of SPT techniques found in the IBM Structured Programming Series. A methodology for comparing the practices and techniques has been devised. This methodology is founded upon the percent of time specific error types exist and the probability of detection or reduction of that error type by the practices and techniques within each life cycle phase. Using this data and statistical analysis techniques, each practice or technique can be assigned an effectiveness value for each life cycle phase. These values will be used later to support the selection of MPP that will be proposed as standards for Air Force software engineering projects.

Computer program life cycle phases provide the framework within which comparison of practices and techniques is conducted. Life cycle phase definitions contained in Air Force system acquisition management directives (AFM 800 series) are used for the comparison process. Those phases are: Analysis, Design, Code and Checkout, Test and Integration, Installation, and Operating and Support. These phase definitions differ from the definitions contained in the Structured Programming Series. Correlation between the two life cycle phase definitions is shown in Figure 4-1. The AFM 800 series phase definitions will be used throughout this section.

The MPP research and data collection did not extend into the Installation and Operating and Support phases. The reason for this is that the CSC project data did not cover those phases. Therefore, the discussions and analysis of practices and techniques performance does not include these phases. For two of the projects, CSC contractual support did not extend into those phases in the same form as that researched. The third project had not reached the Installation phase during the period of the research.

4.2 COMPARISON METHODOLOGY

CSC's comparison methodology is a stepwise process proceeding from identification of programming practices and techniques to calculation of their individual effectiveness values. The fundamental premise upon which this methodology rests is that the timely production of error free computer programs is the goal of software acquisition projects. Computer program errors are defined to include not only processing faults, but also failure to satisfy system performance requirements. This broad definition is meant to be inclusive of all aspects of computer program performance.

CSC SOFTWARE PRODUCTION DATA REPORT	COMPUTER PROGRAM LIFE CYCLE						
	ANALYSIS	DESIGN	CODE & CHECK-OUT	TEST & INTEGRATION	INSTALLATION	OPERATING & SUPPORT	
STRUCTURED PROGRAMMING SERIES	INITIATION & DEFINITION	DESIGN	IMPLE-MENTATION	SYSTEM TEST & INSTALLATION		MAINTE-NANCE	REVISED OPERATION
	SOFTWARE DEVELOPMENT CYCLE						

Figure 4-1. Life Cycle Definitions

In developing the comparison methodology four major assumptions were made. First, practices and techniques that promote the detection or reduction of computer program errors can be ranked according to their effectiveness with regard to that capability. Second, each practice or technique included in this study has been used to further the timely production of error free software. Third, it is possible to define an inclusive set of software error types that can be readily identified during any software development. Fourth, data can be collected that will support definition of probability of error type existence and probability of error type detection or reduction.

In the following discussions, values that are used in the matrices and in the calculations are not universally supported by project research data. There are several reasons for this. The Software Production Data project involved research of only three CSC software acquisition projects and the IBM Structured Programming Series. This meant that available data points to support conclusions were limited or nonexistent. Much of the available resource material was not designed to support research of this type. In many cases the material could not be substantiated or expanded because project personnel had transferred or left the Company. To counter these conditions the research staff used statistical techniques and CSC management staff surveys. Through these aids a consistent and reasonable data analysis was completed, although CSC admits that this data is by no means complete, it is felt that enough of a data base exists to make a first-cut rough estimate--at least to the extent of trying to prove the usefulness of the methodology.

CSC's comparison methodology is presented as a viable tool for the evaluation and ranking of all known MPP. It establishes an unbiased analysis process that may be applied to all types of software acquisition projects. While the data available for analysis of CSC software production data were limited, the conclusions drawn are believed to be realistic. Given a more comprehensive and extensive data resource, the validity of this methodology could be more fully substantiated. It is important that in the following discussions the comparison methodology itself be given consideration equal to that of the comparison results. The methodology is comprised of these activities:

- Identification of practice or technique phase applicability
- Definition of software error types
- Establishment of percentages of error type existence
- Establishment of error type detection or reduction probability
- Calculation of practice or technique effectiveness values

4.2.1 Phase Applicability

The CSC comparison methodology is a life cycle, phase-dependent process. Each life cycle phase will be a basis for conducting the complete comparison process. Only programming practices and techniques that are actively applied in a given phase are compared in that phase. Therefore, it is essential that the phase applicability

of each practice and technique be established.

The applicable phases for CSC programming practices comparison have been established through the research conducted by the MPP Project staff. For the IBM SPT techniques it was necessary to refer to the Structured Programming Series for this information. In Volume XIII of that Series the SPT techniques, related SPT techniques and non-SPT techniques with future potential are identified. They are listed on graphs depicting their life cycle phase applicability. These graphs were the source for their phase applicability definition in this report. The non-SPT techniques with future potential are included because they are discussed in the Series. In addition, one SPT technique was not listed in Volume XIII, but was discussed in the text of other Series volumes. That technique, Program Review, is included with the other techniques in this report and an applicable phase was established through reference to the text discussion.

Phase applicability of the practices and techniques is graphically portrayed in Figure 4-2. Practices and techniques are merged into one listing under the column heading of Programming Practices. Their applicability through the life cycle phases is shown on the bar graph. The programming practices have been grouped into two subsets: technical and management. This differentiation will be significant later in the comparison process, as will the ordering of the practices and techniques in the list. The Design phase of the computer program life cycle has been divided into Functional and Detailed subphases. This has been done to better define the phase applicability of some programming practices.

4.2.2 Error Types

The universe of software errors may be grouped into discrete subsets based on characteristics unique to the errors in each subset. These subsets, or error types, have been variously defined by different industry studies. The error types established for this study are adapted from those defined by TRW Systems Group for RADC and published in a Software Reliability Study report, RADC-TR-74-250, dated October, 1974. The type definitions used during the Software Production Data (MPP) research are generally more inclusive than those in the TRW report. This has been done so that the number of error types is more manageable. An additional error type was also introduced in this study. Timing errors were isolated as a subset because of their critical significance to real-time software development projects such as those studied during this research task.

The error type definitions which follow have been used throughout this study to establish MPP effectiveness values:

- Computational - inaccurate mathematical operations. Errors in this category occur when a wrong equation or convention is used causing erroneous mathematical output within a routine or program.

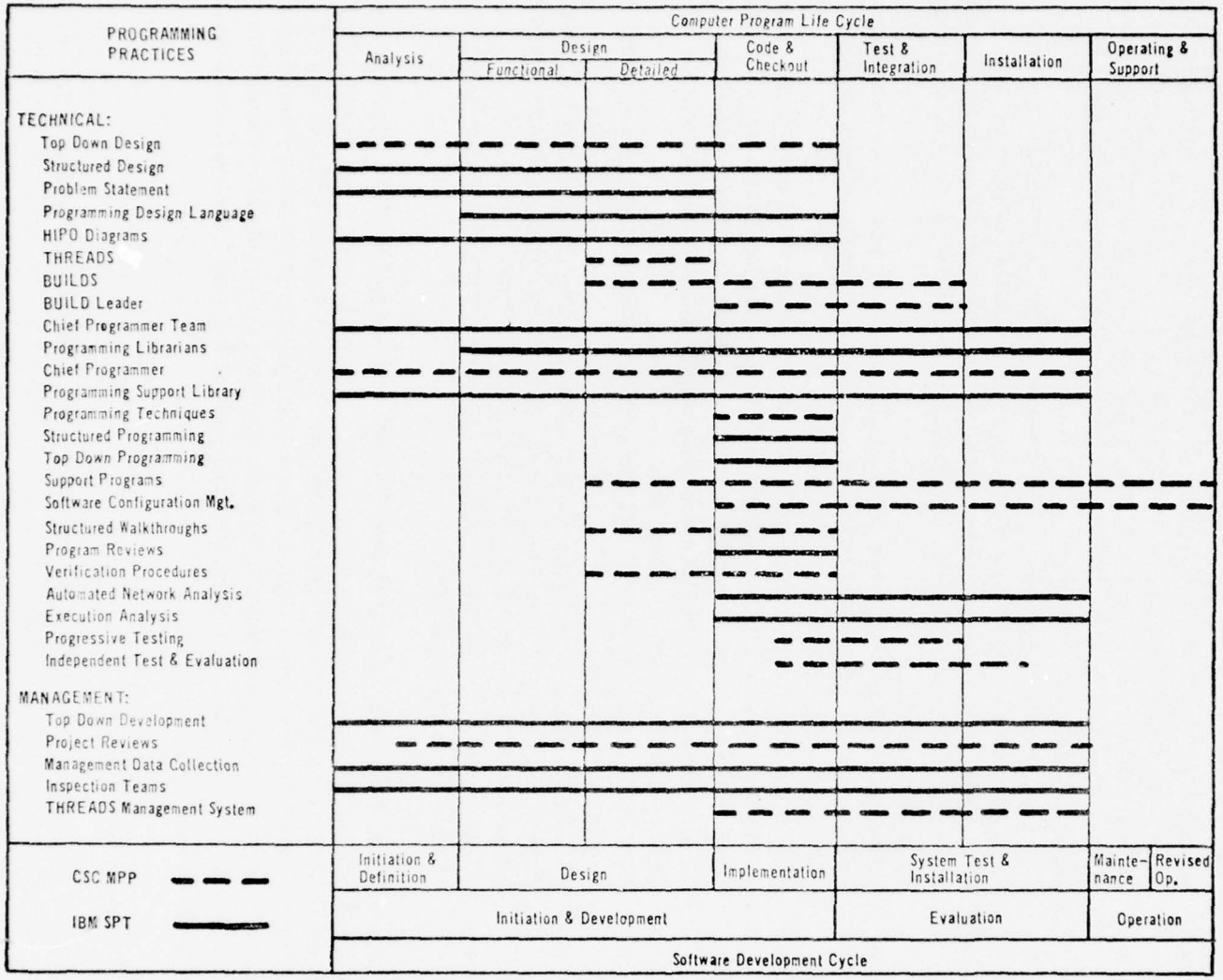


Figure 4-2. Programming Practices Applicability

- Logic - faulty routine or program decision flow. Included in this category are errors caused by incorrect condition tests, status checks, model definition and data storage references.
- Input/Output - improper data reception or transmission. These errors related to data format, field size, position, completeness and control.
- Data Handling - incorrect data manipulation within a routine or program. Errors made in reading, writing, moving, storing and modifying data items internal to a routine or program.
- Operating or Utility Systems - all errors attributed to these systems.
- Configuration - incompatibility of applications and operating system software. This category includes the catastrophic errors and unexplainable program halts caused when applications programs violate operating system conventions.
- Software Interface - incompatible data flow across routine or program boundaries. These errors result from inconsistent data transmission protocols between software components.
- Hardware Interface - incompatible hardware/software interaction. Any error that may be attributed to data flow between hardware and software.
- User Interface - incompatible dependent manual/automated functions. Errors at the user interface including the machine operator, manual or data card inputs, tape inputs, etc.
- Data Interface - incompatibility between data base structure and using routine or program.
- Timing - failure to maintain normal program execution or system operation due to data manipulation timing. Errors caused by the inconsistent operation of time sensitive processing components.
- Requirements Compliance - failure to provide a capability specified in the requirements document.

For the purpose of completing this research it was assumed that all software errors could be characterized as one of the above types. Certainly, there are errors that do fit more than one of the above type definitions. In the broadest sense it can be hypothesized that the error types defined above will include all known software errors. These type definitions are sufficient to support implementation of this comparison methodology. It is possible that more extensive research would result in refinement or addition to these types.

4.2.3 Error Existence

Throughout the computer program life cycle, as any software system is being developed, programming errors are generated and corrected by the development staff. These errors occur and are removed at rates that are dependent upon factors such as complexity and size of the software system being developed, and size, skill and organization of the development staff. The relationships among all factors (even their positive identification) and the error rates have not been clearly defined. There

simply is not enough empirical data to support definition of these relationships.

Information gathered during research of the three CSC projects was not definitive enough to substantiate any hypothesis relative to the rate of error occurrence or removal. However, through project staff interviews and surveys, the MPP research team gathered information which lead to definition of a prototype, error existence matrix. This matrix, shown as Figure 4-3, incorporates error type occurrence and removal rates in one set of values representing the percentage of all error types existing across the life cycle phases. The values reflect percentage of time that a given error type will be generated in a specific phase and remain in the software through subsequent phases. Matrix values for each phase (matrix column) will sum to 1.00 indicating that the error percentages are inclusive of all errors for each phase. The values in this matrix are representative of error type existence from the three CSC projects studied.

4.2.4 Error Detection or Reduction

Programming practices, in general, function to improve development staff performance, assure adherence to the production schedule, and insure maintenance of programmed production costs. Software error occurrence and removal represent a common denominator for practice performance measurement among the three practice functions. Performance, schedule, and cost are impacted by the presence or absence of software errors throughout the development process. Therefore, practice effectiveness in all functional areas is related to the detection or reduction of software errors. The operative relationship between practice performance and error detection or reduction has not been explicitly determined. However, a realistic characterization of the relationship can be hypothesized. This may be done by establishing probability values for the detection or reduction of error types by each programming practice in each of the life cycle phases.

A matrix containing probability values for the detection or reduction of error types by all programming practices is shown as Figure 4-4. The values in this matrix have been developed for the Code and Checkout phase as is indicated at the top of the figure. Because of the life cycle phase dependency of programming practice performance, values such as these must be defined for each phase. Therefore, a unique detection or reduction probability matrix will be necessary for each phase. This report will deal with only the Code and Checkout phase in presenting research findings and developing the comparison methodology. There are two reasons for this decision. First, the code and checkout phase activities involve use of more programming practices than any other phase. This means values for most programming practices appear in the matrices. Second, the resource material for this phase was more comprehensive than for any of the other phases. It should be pointed out that the values in the matrix were qualitatively extracted from the CSC projects reviewed.

ERROR TYPES	COMPUTER PROGRAM LIFE CYCLE						
	ANALYSIS	DESIGN		CODE & CHECKOUT	TEST & INTEGRATION	INSTALLATION	OPERATING & SUPPORT
		Functional	Detailed				
COMPUTATIONAL	.03	.03	.10	.11	.05	.03	.04
LOGIC	.08	.13	.19	.22	.10	.05	.08
INPUT/OUTPUT	.02	.04	.05	.08	.11	.09	.04
DATA HANDLING	.04	.07	.13	.17	.11	.08	.09
OPERATING OR UTILITY SYSTEMS	.02	.05	.04	.06	.03	.02	.03
CONFIGURATION	.05	.06	.06	.05	.06	.04	.04
SOFTWARE INTERFACE	.04	.11	.09	.06	.10	.06	.09
HARDWARE INTERFACE	.04	.08	.07	.07	.09	.11	.05
USER INTERFACE	.06	.07	.06	.05	.11	.17	.17
DATA INTERFACE	.03	.07	.06	.05	.08	.06	.03
REQUIREMENTS COMPLIANCE	.54	.21	.10	.03	.05	.11	.16
TIMING	.05	.08	.05	.05	.11	.18	.18

Figure 4-3. Percentaged-Error Type Existing in Each Phase

CODE & CHECKOUT PHASE												
Programming Practices \ Error Types	Computational	Logic	Input/Output	Data Handling	Operating or Utility Systems	Configuration	Software Interface	Hardware Interface	User Interface	Data Interface	Requirements Compliance	Timing
Technical:												
Top Down Design	.05	.10	.01	.05	.05	.15	.65	.20	.20	.50	.40	.01
Structured Design	.05	.10	.01	.10	.05	.20	.45	.05	.25	.40	.40	.01
Programming Design Language	.05	.05	.05	.10	.05	.10	.05	.01	.25	.15	.45	.01
Builds	.05	.05	.05	.01	.05	.25	.45	.05	.20	.35	.30	.01
Build Leader	.70	.60	.30	.05	.05	.30	.70	.15	.35	.50	.55	.10
Chief Programmer Team	.80	.50	.40	.40	.05	.60	.70	.25	.35	.60	.60	.50
Programming Librarians	.01	.01	.01	.01	.01	.01	.01	.01	.01	.05	.05	.01
Chief Programmer	.80	.50	.40	.45	.05	.65	.75	.50	.50	.60	.70	.50
Programming Support Library	.01	.01	.05	.05	.01	.10	.20	.01	.05	.01	.05	.01
Programming Techniques	.15	.35	.25	.25	.01	.50	.35	.05	.10	.35	.10	.05
Structured Programming	.15	.35	.25	.25	.01	.50	.20	.05	.10	.40	.15	.05
Top Down Programming	.15	.30	.20	.25	.01	.45	.45	.01	.10	.35	.10	.05
Support Programs	.30	.25	.15	.20	.05	.10	.15	.20	.35	.20	.10	.10
Software Configuration Mgt.	.01	.01	.05	.01	.01	.01	.10	.01	.05	.01	.05	.01
Structured Walkthroughs	.85	.50	.25	.30	.05	.20	.80	.65	.70	.15	.25	.50
Program Reviews	.75	.65	.55	.70	.01	.15	.70	.50	.65	.50	.50	.40
Verification Procedures	.35	.20	.30	.20	.05	.25	.45	.25	.35	.40	.35	.20
Automated Network Analysis	.10	.05	.25	.10	.05	.10	.20	.10	.20	.25	.01	.20
Execution Analysis	.65	.50	.50	.20	.01	.15	.35	.15	.10	.15	.05	.50
Progressive Testing	.55	.40	.60	.50	.05	.25	.40	.20	.15	.25	.05	.35
Independent Test & Evaluation	.55	.45	.50	.50	.05	.30	.40	.20	.15	.30	.10	.40
Management:												
Top Down Development	.30	.20	.20	.15	.05	.10	.25	.25	.20	.30	.70	.01
Project Reviews	.10	.01	.05	.10	.01	.05	.10	.30	.35	.45	.75	.15
Management Data Collection	.10	.01	.05	.05	.01	.15	.15	.10	.20	.15	.50	.01
Inspection Teams	.25	.05	.10	.15	.05	.15	.15	.20	.20	.40	.65	.10
Threads Management System	.05	.05	.05	.01	.01	.05	.25	.05	.05	.40	.40	.01

Figure 4-4. Probability of Error Detection or Reduction

Using Figure 4-4 the probability of error detection or reduction by any programming practice in the Code and Checkout phase can be determined. For instance, Top Down Design has a 5% probability of reducing computational errors in this phase.

4.2.5 Effectiveness Values

Programming practice effectiveness values may be defined for each practice relative to all error types within each applicable phase. Thus, effectiveness values are a function of the percentage of time error types exist and the probability of practice detection or reduction of those error types. The purpose of mathematical manipulation of the two values is to define a figure of merit or effectiveness value that can be derived for all programming practices. Using this value, programming practices can be compared quantitatively within or across all life cycle phases. Naturally, the effectiveness values are only as accurate as the probability values. The probability values presented in Figures 4-3 and 4-4 of this report are not based on quantitative measurement. They represent a consensus of experienced assessment by CSC staff members. However, given adequate source material and research tools the actual probability values could be derived. In any case, CSC's comparison methodology does offer a viable approach to programming practice evaluation and comparison. The values shown in Figures 4-3 and 4-4 are valid for the three CSC projects studied. Therefore, they support the comparison process to be described in the following discussions.

Development of effectiveness values is a two part process. First, the practice effectiveness values are calculated for each life cycle phase. Next, these practice effectiveness values are used to develop phase effectiveness values for each practice across all life cycle phases. In the following subsections each of these activities is described using probability values for the Code and Checkout phase.

4.2.5.1 Practice Effectiveness Values

Practice effectiveness values are calculated using the percentage of error types existing and error type detection or reduction probabilities. The calculation involves selecting a programming practice, error type and life cycle phase. Then, the percentage of error type existence is multiplied by the probability of error type detection or reduction. That product is multiplied by 100 to normalize the value. As an example, using the Top Down Design programming practice and the Computational error type in the Code and Checkout phase the effectiveness value calculated would be:

Existence Percentage	x	Detection/Reduction Probability	x	100	=	Effectiveness Value
0.11	x	0.05	x	100	=	0.55

Completing the calculations for all practices and error types in the Code and Checkout phase will produce the matrix shown as Figure 4-5. Practice effectiveness against any error type may be found by reference to the matrix. The values are significant only through comparison with other practice effectiveness values. The Total Effectiveness column in Figure 4-5 represents the sum of values for each row or programming practice. These totals will be used later in the comparison process. Completion of these calculations for all life cycle phases will result in a series of matrices that may be used in a life cycle comparison process.

4.2.5.2 Phase Effectiveness Values

Phase effectiveness values are designed to provide quantitative, comprehensive values that represent a measure of programming practice performance against individual error types. These values will serve as inputs to the process of developing phase effectiveness values. An algorithm has been defined that uses practice effectiveness values to produce a value that defines practice utility within a phase. These phase effectiveness values may be used to compare practices in an unbiased fashion.

In developing the phase effectiveness algorithm consideration was given to certain aspects of the practice comparison process and some assumptions were made. First, the *probability of error existence matrix* was considered to have established the relative distribution of error types across the life cycle. It represents an indirect statement about the volume of errors by type that will exist in any phase. This is an important factor when evaluating practice performance. Next, the phase effectiveness values, when calculated, must account for error types that practices are most and least effective against. In addition, the relative volume of those particular errors on a phase by phase basis is important. Finally, the goal for development of the algorithm was to create a procedure that would promote selection of an optimum mix of practices for use across all life cycle phases. It was assumed that all error types are of equal significance to the software development process. That is, a logic error is just as significant as a timing error. Error types were not given weighting factors. It was also assumed that practice effectiveness against the most and least frequently encountered errors in a phase does reflect the quality of that practice.

The phase effectiveness algorithm uses the following input data; practice effectiveness value for the error type with the greatest percentage of existence (labeled 'a'), practice effectiveness value for the error type with the least probability of existence (labeled 'b'), practice total effectiveness value (labeled 'c'), and the number of error types (labeled 'd'). The algorithm is expressed as follows:

$$(a + b)\ \frac{c}{d}$$

CODE & CHECKOUT PHASE														
Programming Practices \ Error Types	Computational	Logic	Input/Output	Data Handling	Operating or Utility Systems	Configuration	Software Interface	Hardware Interface	User Interface	Data Interface	Requirements Compliance	Timing	Total Effectiveness	
Technical:														
Top Down Design	0.55	2.20	0.08	0.85	0.30	0.75	3.90	1.40	1.00	2.50	1.20	0.05	14.78	
Structured Design	0.55	2.20	0.08	1.70	0.30	1.00	2.70	0.35	1.25	2.00	1.20	0.05	12.38	
Programming Design Language	0.55	1.10	0.40	1.70	0.20	0.50	0.30	0.07	1.25	0.75	1.35	0.05	8.32	
Builds	0.55	1.10	0.40	0.17	0.30	1.25	2.70	0.35	1.00	1.75	1.50	0.05	11.12	
Build Leader	7.70	13.2	2.40	0.85	0.30	1.50	1.20	1.05	1.75	2.50	1.65	0.50	37.60	
Chief Programmer Team	8.80	11.0	3.20	6.80	0.30	3.00	1.20	1.75	1.75	3.00	1.80	2.50	48.10	
Programming Librarians	0.11	0.22	0.08	0.17	0.06	0.05	0.06	0.07	0.05	0.25	0.15	0.05	1.32	
Chief Programmer	8.80	11.0	3.20	7.65	0.30	3.25	1.50	3.50	2.50	3.00	2.10	2.50	52.30	
Programming Support Library	0.11	0.22	0.40	0.85	0.06	0.50	1.20	0.07	0.25	0.05	0.15	0.05	3.91	
Programming Techniques	1.65	7.70	2.00	4.25	0.06	2.50	2.10	0.35	0.50	1.75	0.30	0.25	23.41	
Structured Programming	1.65	7.70	2.00	4.25	0.06	2.50	1.20	0.35	0.50	2.00	0.45	0.25	22.91	
Top Down Programming	1.65	6.60	1.60	4.25	0.06	2.25	2.70	0.07	0.50	1.75	0.30	0.25	21.98	
Support Programs	3.30	5.50	1.20	3.40	0.30	0.50	0.90	1.40	1.75	1.00	0.30	0.50	20.05	
Software Configuration Mgt.	0.11	0.22	0.40	0.17	0.06	0.05	0.60	0.07	0.25	0.05	0.15	0.05	2.18	
Structured Walkthroughs	9.35	11.0	2.00	5.10	0.30	1.00	1.80	1.55	3.50	0.75	0.75	2.50	45.60	
Program Reviews	8.25	14.3	4.40	11.9	0.06	0.75	1.20	3.58	3.25	2.50	1.50	2.00	56.61	
Verification Procedures	3.85	1.10	2.40	3.40	0.30	1.25	2.70	1.75	1.75	2.00	1.05	1.00	25.85	
Automated Network Analysis	1.10	1.10	2.00	1.70	0.30	0.50	1.20	0.70	1.00	1.25	0.03	1.00	11.88	
Execution Analysis	7.15	11.0	4.00	3.40	0.06	0.75	2.10	1.05	0.50	0.75	0.15	2.50	33.41	
Progressive Testing	6.05	8.80	1.80	8.50	0.30	1.25	2.40	1.10	0.75	1.25	0.15	1.75	37.40	
Independent Test & Evaluation	6.05	9.90	1.00	8.50	0.30	1.50	2.40	1.40	0.75	1.50	0.30	2.00	38.60	
Management:														
Top Down Development	3.30	4.40	1.60	2.55	0.30	0.50	1.50	1.75	1.00	1.50	2.10	0.05	20.55	
Project Reviews	1.10	0.22	0.40	1.70	0.06	0.25	0.60	2.10	1.75	2.25	2.25	0.75	13.43	
Management Data Collection	1.10	0.22	0.40	0.85	0.06	0.75	0.90	0.70	1.00	0.75	1.50	0.05	8.28	
Inspection Teams	2.75	1.10	0.80	2.55	0.30	0.75	0.90	1.40	1.00	2.00	1.95	0.50	16.00	
Threads Management System	0.55	1.10	0.40	0.17	0.06	0.25	1.50	0.35	0.25	2.00	1.20	0.05	7.88	

Figure 4-5. Practice Effectiveness

To demonstrate its use, given the Top Down Design programming practice and the Code and Checkout phase, the following data would be input:

Practice Effectiveness Values:

Logic Errors (encountered most often) - 2.20
Requirements Compliance (encountered least often) - 1.20
Total Effectiveness Value - 14.78
Number of Error Types - 12

The algorithm would then be executed as follows to calculate the phase effectiveness value:

$$(2.20 + 1.20)\ \frac{14.78}{12} \quad = \quad 4.19$$

This calculation sequence is repeated for each programming practice in the Code and Checkout phase. The resulting values are shown in Figure 4-6. Only the Code and Checkout column is completed because input data for the other phases has not been defined. These values establish the relative merit of the programming practices within the Code and Checkout phase.

4.2.5.3 Effectiveness Value Benefits

Certain benefits are realized through the use of effectiveness values in the programming practice comparison process. The most obvious benefit is the capability to develop discrete measures of practice performance through a consistent and comprehensive series of operations. The practice effectiveness value matrices permit the determination of programming practice voids, specific instances where no practice is effective against a particular error type. This situation would indicate an endemic deficiency within the program development process. The phase effectiveness value matrix provides the vehicle for optimization of error detection through identification of the minimum number of programming practices that generate a maximum set of values. Analysis of the phase and practice effectiveness values could provide a global measure of anticipated product reliability. By adding cost-to-detect and cost-to-fix data to these matrices it would be possible to develop minimum cost approaches to program development.

Programming Practices \ Computer Program Life Cycle Phases	Analysis	Functional Design	Detailed Design	Code & Checkout	Test & Integration	Installation	Operating & Support
Technical:							
Top Down Design				4.19			
Structured Design				3.79			
Programming Design Language				1.70			
BUILDS				2.41			
BUILD Leader				46.53			
Chief Programmer Team				51.31			
Programming Librarians				0.04			
Chief Programmer				57.09			
Programming Support Library				0.12			
Programming Techniques				15.61			
Structured Programming				15.56			
Top Down Programming				12.64			
Support Programs				9.69			
Software Configuration Mgt.				0.07			
Structured Walkthroughs				44.65			
Program Reviews				74.54			
Verification Procedures				11.74			
Automated Network Analysis				1.12			
Execution Analysis				31.04			
Progressive Testing				27.89			
Independent Test & Evaluation				32.81			
Management:							
Top Down Development				11.13			
Project Reviews				2.76			
Management Data Collection				1.19			
Inspection Teams				4.07			
THREADS Management System				1.51			

Figure 4-6. Phase Effectiveness Values

SECTION 5 - RECOMMENDED MPP

5.1 INTRODUCTION

Selection of the programming practices to be recommended as standards for Air Force software engineering projects is the culmination of this MPP comparison process. This process, which began with research of three CSC software engineering projects, has lead to the identification and description of all programming practices that were used. These programming practices include both management and technical disciplines applied in all phases of computer program development. A comparison methodology has been defined that permits unbiased and definitive comparison of the programming practices. The comparison methodology has been employed to establish a basis for analyses of CSC programming practices and IBM SPT techniques. Application of the comparison methodology as described in this report has produced a quantitative assessment of each practice and technique in the form of phase effectiveness values for the Code and Checkout phase. Now, MPP selection procedures will be used to identify those practices and techniques that will be recommended as Air Force standards.

On the surface it might appear that the CSC comparison methodology has reduced MPP definition to a purely mechanical process. In part this is true. When comprehensive performance and error data is available to support the definition of true, error existence and detection or reduction probabilities the identification of effective programming practices will be a highly structured activity. However, even under these conditions it will be necessary for a program manager to exercise professional judgement in selecting an appropriate mix of MPP to satisfy the particular software development requirements with which he is faced. The selection procedures to be defined depend upon practice effectiveness evaluation and development environment analysis. The comparison process will produce effectiveness values and the program manager will weigh these values and the MPP characteristics (as discussed in Section 3 of this report) against the software development environment. Using the CSC comparison process will provide program managers with more objective and definitive data with which to specify the project MPP in preparation for software system development.

Selection procedures are structured to promote definition of an optimum mix of MPP. In the following subsections selection procedures are defined within the context of this research project. Then, the MPP to be recommended as Air Force standards are identified.

5.2 SELECTION PROCEDURES

This discussion of selection procedures will focus on the quantitative aspects of MPP selection. Determining which programming practices have maximum phase effectiveness values within a life cycle phase or across the entire life cycle is a

fairly straightforward process. As will be shown, consideration must be given to the effects of interaction among the MPP selected and to their functional characteristics. On the other hand, defining the process whereby a program manager assesses the software development environment and his resources in order to identify the optimum set of MPP to use is another matter and will not be attempted in this report. Quantifying programming practice effectiveness has given the manager a valuable tool for the selection of MPP for his project. However, it is recognized that in using this tool his reliance on the phase effectiveness values must be tempered by his experience and good judgement.

In preparing for MPP selection it is important that the functional similarities of programming practices be considered. Functional similarity refers to the employment and performance characteristics of certain programming practices that make them functionally similar. For instance, top down design and structured design are both employed to perform the same design function and their effect on the design process is essentially the same. HIPO diagrams and THREADS also fit this description. In each case one of these practices may do the job better than the other, but in any case no one would consider using both practices on the same project. So, functional similarities among the programming practices are identified to aid in the selection process. Figure 5-1 shows the programming practices defined in this research project with the functionally similar subsets indicated by vertical bars.

Phase effectiveness values must be established for all practices in each applicable phase. As previously discussed, this research project has led to the definition of phase effectiveness values for the Code and Checkout phase. Selection of MPP for a software development project would require phase effectiveness values for all life cycle phases. Effectiveness values are assigned to programming practices in accordance with the phase applicability of the practices. Phase applicability was discussed in Section 4 and shown in Figure 4-2. Selection of the MPP involves choosing a practice (or practices) in each functional grouping which has the largest sum of phase effectiveness values across the applicable phases. In cases where two or more practices within one group have comparable effectiveness it will be up to the program manager to select the one he prefers (or perhaps use more than one if they are compatible). Also, a practice might be selected within a group because it is applicable in more phases even though its effectiveness value sum is low. This would be the manager's selection perogative. Also, proper selection would require optimization of MPP overall error types within each phase.

5.3 STANDARD MPP

A standard set of MPP's that would be applicable to all Air Force software engineering projects cannot be developed with the current data available. Each project must be analyzed in view of its characteristics and development environment, then a proper set of MPPs could be selected based on the effectiveness of those MPPs throughout the project life cycle. However, based on the data collected during this

FUNCTIONALLY SIMILAR PRACTICES

- TOP DOWN DESIGN (M)
- STRUCTURED DESIGN (S)
- PROBLEM STATEMENT LANGUAGE (S)
- PROGRAMMING DESIGN LANGUAGE (S)

- HIPO DIAGRAMS (S)
- THREADS (M)
- BUILDS (M)

- BUILD LEADER (M)
- CHIEF PROGRAMMER TEAM (S)
- PROGRAMMING LIBRARIANS (S)
- CHIEF PROGRAMMER (M)
- PROGRAMMING SUPPORT LIBRARY (S)

- PROGRAMMING TECHNIQUES (M)
- STRUCTURED PROGRAMMING (S)
- TOP DOWN PROGRAMMING (S)

- SUPPORT PROGRAMS (M)
- SOFTWARE CONFIGURATION MGT (M)

- STRUCTURED WALKTHROUGHS (M)
- PROGRAM REVIEWS (S)
- VERIFICATION PROCEDURES (M)

- AUTOMATED NETWORK ANALYSIS (S)
- EXECUTION ANALYSIS (S)
- PROGRESSIVE TESTING (M)
- INDEPENDENT TEST & EVALUATION (M)

- TOP DOWN DEVELOPMENT (S)
- PROJECT REVIEWS (M)
- MANAGEMENT DATA COLLECTION (S)
- INSPECTION TEAMS (S)
- THREADS MANAGEMENT SYSTEM (M)

(M) : MPP
(S) : SPT

Figure 5-1. Functionally Similar Practices

project the following set of MPPs would be applicable to most software development projects:

- Top Down Design
- Threads
- Builds
- Chief Programmer
- Programming Techniques
- Support Programs
- Software Configuration Management
- Program Reviews
- Independent Test and Evaluation
- Project Reviews
- Threads Management System

SECTION 6 - RESEARCH PLAN

6.1 MPP RESEARCH RESULTS

CSC's software production data research has resulted in definition of a methodology for identification and selection of a set of effective MPP. This research task began with an indepth review of three CSC software development projects. These projects were similar in many respects. They involved production of software systems and subsystems for command and control application. These were U.S. Navy shipborne systems using AN/UYK 7 and 20 processors with the computer programs written in CMS-2 high order language. Differences among the projects involved the sizes of the systems being developed and consequently the sizes of the project staffs. The contractual relationships between CSC and the U.S. Navy or the Navy's prime contractor varied among the projects. However, the differences most significant to this research involved the programming practices employed on each of the three projects. Each project staff used a different set of practices either by choice or by direction of the customer.

Once these programming practices had been identified and defined by the MPP research team, the projects documentation and reports were reviewed to identify performance data supporting analysis of the practices' effectiveness. This information included development milestone achievement, program test results, program trouble reports, management review documents and system size data. The documented information was supplemented through interviews with project personnel. All of this research material was analyzed to isolate the data to be used to determine practice performance criteria.

As the research data was reviewed it became more and more apparent that research of this type cannot be conducted effectively after the fact. Once the software development plan has been written and development has begun or been concluded it is not possible to produce complete and coherent, programming practice analysis data. There are several reasons for this, foremost of which is the fact that project managers are almost totally concerned with producing and delivering the software. They assign and adjust resources to satisfy the most urgent requirements with little regard for continuity of programming practice employment. Also management reporting and documentation is not structured to support programming practice analysis. All of this leads to discontinuous data for analysis, an unpredictable research environment, and marginally useful management reports. These factors prompted consideration and development of a new approach to evaluation of programming practices.

CSC's MPP comparison methodology is proposed as a comprehensive and universal approach to specification of project programming practices. The prototype process described in Section 4 of this report can reduce the identification and selection of effective programming practices to a straightforward, well defined procedure.

It has been used to produce the list of standard MPP that is shown in Section 5 of this report. This is still an unproved process, however. More research and testing will be required to establish the validity and reliability of the procedure. That research and testing is discussed in the following subsection.

6.2 FUTURE RESEARCH TASKS

Perhaps the most notable results of this research project is the insight that has been gained into the definition of effective programming practices. As with most learning experiences, this one has not ended. The research that has been completed was productive, but there is still more productive research that should be done. The following discussions will outline the requirements for future research into definition of MPP. Those requirements are:

- Design MPP research program in coordination with software engineering project planning
- Produce data management system to assimilate and reduce research data
- Conduct analysis of the practices performance data and recommend MPP for future projects.

6.2.1 Research Program

The MPP research program should be designed to complement planned software development projects where the research will take place. Research should begin during the planning phases for the software development projects. The research may be conducted by U.S. Air Force personnel or by a disinterested contract research group. In any case, it should not be conducted by the software development contractor. Every effort should be made to keep the research unbiased. The software development projects should be varied in type in order to produce programming practice performance data from differing environments. It is possible that practice effectiveness will, in part, be a function of the type of software system being developed. If so, this relationship should be established.

The research goals should be defined so that the research tasks support all data requirements. Hopefully the research tasks and software development tasks will operate independently with minimum interference. The research must not affect the conduct of software development. Research data should be collected from the Analysis through Installation Phases. In addition, procedures should be defined for continuing data collection by the user into the Operating and Support Phase. This data collection will produce a mass of programming practice performance data for reduction and analysis. The most effective way to satisfy the requirement for reduction is through the use of a data management system.

6.2.2 Data Management System

Performance data will be used to determine the probabilities of error detection or reduction and ultimately the practice effectiveness values. The probabilities of error existence must be determined independently by the research team during project research. All of this data will be cataloged by the data management system and manipulated to produce MPP recommendations and reports.

The programming practice performance data will be developed from program trouble reports. This will be supplemented by other reports such as those prepared for milestone achievement, unit test and core budget. The program trouble reports and other internal error reports will be used to determine the types and volume of errors being generated during software development and integration. Further reporting and analysis must be accomplished in order to determine the relationship between the occurrence of these errors and their detection or reduction by the programming practices. The other reports will be used to complete analysis of the practices' performance.

6.2.3 Performance Analysis

There are two aspects to the analysis of programming practice performance. One is the evaluation of the practices when they are not being used in a controlled environment. That is, a software development project is studied after it is concluded to see how well the practices performed. This was done for the Software Production Data project. The second aspect involves specification of the practices to be used in a development project and monitoring their employment.

Once the programming practices have been assigned effectiveness values, the validity of these values must be confirmed. Their validity can be confirmed by applying them on a software development project and analyzing their performance relative to error detection or reduction, production milestone achievement, program test results and project cost control. For subsequent projects the MPP can be specified using the results of the MPP comparison process. Then the performance analysis can be conducted as previously discussed. Thus, through this controlled process of experimentation and analysis a viable and efficient MPP definition technology can be developed.

APPENDIX A - GLOSSARY

AUTO-FLOWCHART PROGRAM

This is a utility routine that accepts as input, flowchart statements added to the program code. Using these statements the routine generates a standard flowchart of that program. Modification of program logic which necessitates changes of program code also will affect the corresponding flowchart statements. As the program code is updated so also are the flowchart statements, thus assuring production of current documentation through a dynamic, versatile process.

BUILD

A logical collection of functionally related processes that are implemented and demonstrated as a set.

BASELINE

A configuration identification document or set of such documents, and also the computer programs after the product content and structure has been formally designated and fixed at a specific time during the product life cycle.

Under baseline configuration management, there are usually these baseline positions:

Allocation - established at the end of the requirements/and performance definition phase.
Functional - established at the end of the conceptual phase, normally existing prior to the start of a software development project.
Product - established at the end of the test and acceptance phase.

CHIEF PROGRAMMER TEAM

A management technique that organizes program production staff with a Chief Programmer as a technical manager or 'super-programmer' responsible for the efforts of a program development team.

CHANGE STATUS REPORT (CSR)

A listing or report of all proposed changes and corrections to a configuration and their current disposition and implementation status.

DESIGN TROUBLE REPORT (DTR)

System design problems identified by anyone (CSC or Navy) associated with the DD-963 project are defined and documented through DTRs. DTRs are formally controlled by CSC personnel.

DESIGN, TOP DOWN

A methodology for system design whereby the high level, executive components of the system are designed first and the balance of the system is designed in an orderly progression from top to bottom.

DECISION TABLE

A table of information consisting of all contingencies and actions that are to be considered for each possible set of conditions in the description of a problem. Decision tables may have multiple dimensions depending on the complexity of the arguments and actions inherent in the problem. Decision tables are sometimes used in place of flowcharts for problem description and documentation.

DESIGN REVIEW

Detailed Design Review normally occurs at the completion of the design of a system, subsystem, or level of abstraction. Consists of a detailed examination of the software design and may be performed numerous times during development of the software. Successful completion of this review establishes the design baseline and program coding begins at this time. (Also called Critical Design Review.)

Preliminary Design Review normally occurs at the completion of the system design phase. May be formal or informal in structure. Successful completion of this review establishes the preliminary of system level computer program development specifications, interface specifications and data requirements specifications in the system design baseline.

DYNAMIC CODE ANALYZER

Software routine that analyzes computer programs for errors during execution of those programs.

ENGINEERING CHANGE PROPOSAL (ECP)

A formal request for alteration of the configuration of a computer program or other configuration item that is delivered, to be delivered, or under development. ECPs are required after formal establishment of the component configuration identification (baseline position).

INTERFACE

A common boundary between or among system components. Examples are a hardware device which links other system components or a specified storage area within the processor which is accessed by two or more computer programs.

INTEGRATION TEST

A test to determine the functional compatibility of two or more dissimilar systems or system elements (i.e., hardware and software, applications and system software, etc.). Also used to ensure the compatibility and proper functioning of a new or modified system element when it is placed within the existing system.

MILESTONE

A well-defined interim or final goal which must be achieved during the development of a hardware or software system. It usually establishes a baseline position or signifies the completion of a specific task. Completion of system Builds is one form of milestone achievement.

MILESTONE REPORTING

Milestone Reporting encompasses all levels of the managerial and technical staff. At the lowest level, the programmer milestones are established. As the chain of command progresses, milestones become broader and more complex, and they reflect the status of the entire project.

PROCESSING, COMPUTER PROGRAM

Batch Processing

(1) Pertaining to the technique of executing a set of computer programs such that each is completed before the next program of the set is started.

(2) Pertaining to the sequential input of computer programs or data.

(3) Loosely, the execution of computer programs serially.

Stacked Job Processing

A technique that permits multiple job definitions to be grouped (stacked) for presentation to the system, which automatically recognizes the jobs, one after the other. More advanced systems allow job definition to be added to the group (stack) at any time and from any source, while honoring priorities.

PROGRAMMING

Defensive Programming, is a concept closely related to functional and structured programming, in that emphasis is placed on avoidance of errors at the time of coding, rather than during program checkout. The idea is to visualize worst case situations and program accordingly.

Functional Programming emphasizes the creation of correct programs so that much of the program testing traditionally associated with building large systems will not be necessary. The methodology focuses on clearly describing an object in functional terms before it exists in final form.

Peer Group Programming is a concept used by CSC which focuses on the fact that it is often helpful and instructive for a programmer to have his program reviewed by a group of peers, both before and after coding and prior to checkout.

Structured Programming is a programming discipline providing a means of expressing a system design that ensures a testable and understandable implementation, and that enforces simple and well-defined connections between program modules. The programming discipline uses the repeated application of a small number of basic control statements to form simple program constructs that represent large and complex programs. The process of creating the program modules includes: making local or tactical programming decisions within the designed module; writing program segments that represent these decisions; and integrating program segments into a unit corresponding to a system module.

Modular Programming is a division of software into blocks of code called modules. Each module represents a specific subsystem and is developed as an entity.

"Top-Down" Programming is a technique of software development that closely parallels the natural approach conventionally used for system design. The control architecture, including the executive program, major executive interfaces (modules), and data base is put into place first. The internal control logic is then constructed within each module to a progressively lower level of functional detail, ultimately resulting in a completely coded software package.

PROJECT MANAGEMENT PLAN (PMP)

Project Management Plan includes complete development considerations which encompass the plans for phasing, organizing, testing, change control, documenting, training, and installing. The plan becomes the road map and is the key tool used for the successful implementation of all software development projects.

RESOURCE UTILIZATION (PLAN)

Referring to a plan for the assignment of resources (men, machine, money, time, and materials) to project elements (task, products, accounts and organization units), and the accounting for the actual expenditures of these resources in the accomplishment of the project mission.

SOFTWARE TROUBLE REPORT (STR)

A form used to define and document software failures and faults. TRIDENT project called the form itself an STR, on DD-963, it was known as CSC Trouble Report (CTR). In both cases it served the same purpose. Through this form project managers are able to track the actions taken to dispose of software problems. It also gives some measure of programming effectiveness. Trouble reports provide visibility to the customer concerning development activity.

SYSTEM VALIDATION DIAGRAM (SVD)

A block diagram portraying finite processes at their lowest subdivisions. Each block (stimulus-response element) depicts input (stimulus), process (operation) and output (response). A complete function may be represented by a series of blocks.

AD-A042 686 COMPUTER SCIENCES CORP HUNTSVILLE ALA DEFENSE SYSTEMS DIV F/G 9/2
SOFTWARE PRODUCTION DATA.(U)
JUL 77 J DONAHOO, S CARTER, J HURT F30602-76-C-0163
UNCLASSIFIED RADC-TR-77-177 NL

2 OF 2
ADA042 686

END
DATE FILMED
8-77
DDC

SYSTEM

(1) An assembly of methods, procedures, programs, or techniques united by regulated interaction to form an organized whole.

(2) An organized collection of men, machines, and methods required to accomplish a set of specific functions.

(3) A structured combination of interacting parts satisfying a set of functional objectives and performance objectives.

STUB

A minimal piece of code used to simulate an operation. In top-down program development, stubs are the first realization of a program's functional specification. They are expanded into programs when the development effort moves to the level of abstraction to which the stubs belong.

SOFTWARE FAULT

A departure from correct, specified program execution which is due to error(s) occurring during the translation of the original specification of an algorithm to the program being executed.

SOFTWARE ERROR

Any discrepancy between a computed, observed or measured quantity and its true, specified, or theoretically correct value. Errors are introduced into software by human mistakes, that is: (1) deficiencies or misinterpretations of design criteria, (2) logical mistakes, (3) a syntatical mistake and (4) mistakes made in transcribing program statements into the input data. Contrast with fault, malfunction, and mistake.

SOFTWARE DEVELOPMENT LIFE CYCLE

The process by which user requirements are translated, via software, into a functioning system. The actual steps involved may differ according to the size, purpose, and end use of the software. For the formal development of a large program the following steps are involved: system requirements definition, software requirements definition, preliminary design, analysis and detailed design, coding and checkout, development and system testing, delivery, and maintenance.

SOFTWARE

Computer programs, procedures, data, and associated documentation concerned with the operation of a data processing system. The term software includes (a) computer program products in the form of card decks and magnetic tapes, and (b) all documentation associated with computer programs, including specifications, listings, manuals, flowcharts, version description documents, test plans, and test procedures.

SIMULATOR

A software/hardware system whose primary function is to emulate the behavior or operation of another system. Simulators are normally employed to provide realistic inputs for a target system or device which is being evaluated. They may also be used in pure research to mimic other systems' responses to programmed inputs when the real systems cannot be made to respond under laboratory conditions.

SEGMENT

Portion of a larger system entity, such as a subsystem, program, or data segment. Segment dimensions may be somewhat arbitrary since a segment usually encompasses a single functional operation. The term segment is most often applied to individual subroutines or subelements of system modules.

SOFTWARE CONFIGURATION MANAGEMENT (PLAN) [SCMP]

Software Configuration Management (SCM), is a tool that assures the integrity of implemented systems. SCM locks out the programmer from the implemented data files and programs by establishing controlled program versions, and it allows only authorized changes to those versions.

STRUCTURED PROGRAMMING

(See Programming)

STRUCTURAL ANALYZER

Software routine that analyzes structured code to assure that the basic structured patterns are adhered to.

TEST SCRIPT

A procedural scenario using Threads to ensure all requirements of the program specifications are met.

TOP-DOWN DEVELOPMENT

The process of designing a software system through a sequence of step-wise refinement rhough successive levels of abstraction; and then implementing the design by giving development priority to control structures and module interfaces. Application routines (i.e., bottom-level modules) are first represented by program stubs simulating the control and interaction interfaces. The actual programs are subsequently integrated into the control structure such that system level testing for the control structure and interfaces are performed first and continually verified as application routines are substituted for their stub representations.

TEST CASE GENERATORS

Test Case Generators are utility routines that generate sample input data as part of the checkout activity. Emphasis is on producing data which will cover all conditional paths.

THREADS

The THREADS methodology is based on defining a system in terms of processes specifically keyed to required system capabilities and then organizing all system development activities around the processes. A Thread is a functional path or sequence of activities that results in the production of a required response from a set of inputs or stimuli. Each requirement places on the system can be expressed as a Thread.

VERIFICATION, BOTTOM-UP

This process was used on the DD-963 project. It is accomplished by first verifying that the program auto flowcharter statements match the program logic. The auto flowchart routine will then be executed to generate program flowcharts. These flowcharts are then compared to the design specification to ensure conformance. This process is usually accomplished by someone other than the responsible programmer.

VERIFICATION & VALIDATION (V&V)

Verification

(1) The process of testing designed software to ensure that the system and/or system components are logically correct.

(2) The process by which the contents of a computer program's physical medium (tape, deck, etc.) are authenticated.

Validation

The process of ensuring that specific program functions meet requirement specifications.