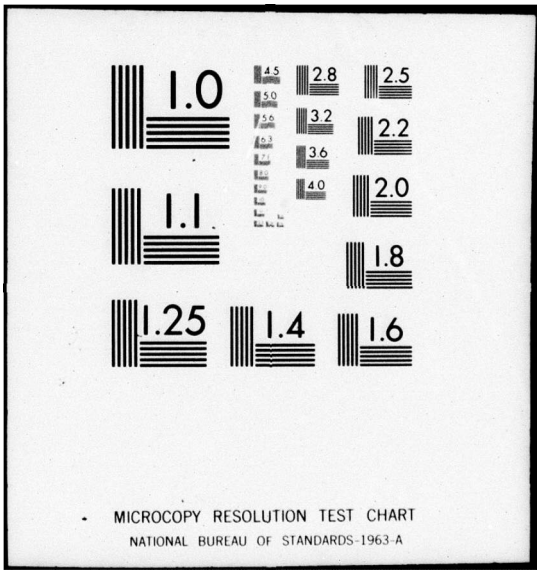
AD-A041 678 SYSTEM DEVELOPMENT CORP SANTA MONICA CALIF F/G 9/2
AN INVESTIGATION OF PROGRAMMING PRACTICES IN SELECTED AIR FORCE--ETC(U)
JUN 77 G H PERRY, N E WILLMORTH F30602-76-C-0180
UNCLASSIFIED SDC-TM-(L)-5839/000/00 RADC-TR-77-182 NL

1 OF 3
ADA041678


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

RADC-TR-77-182
Final Technical Report
June 1977

AN INVESTIGATION OF PROGRAMMING PRACTICES IN SELECTED AIR FORCE PROJECTS

System Development Corporation

Approved for public release; distribution unlimited.

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

This report contains pages which are not of the highest printing quality but because of economical consideration, it was determined to be 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.

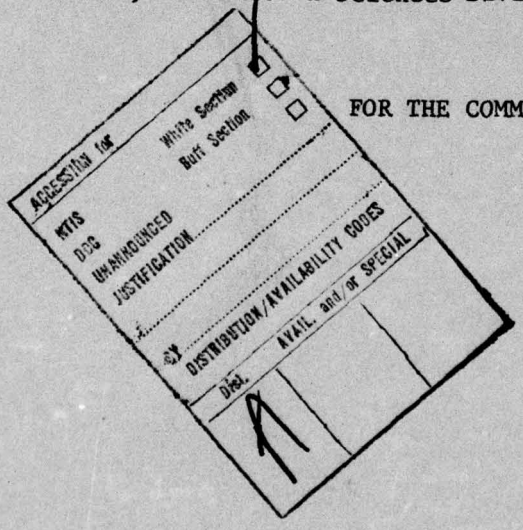
This report has been reviewed and is approved for publication.

APPROVED: Roger W. Weber

RODGER 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

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-77-182	2. GOVT ACCESSION NO.
3. RECIPIENT'S CATALOG NUMBER	
4. TITLE (and Subtitle): AN INVESTIGATION OF PROGRAMMING PRACTICES IN SELECTED AIR FORCE PROJECTS	5. TYPE OF REPORT & PERIOD COVERED: Final Technical Report, Apr 1976 - Apr 1977
6. PERFORMING ORG. REPORT NUMBER: SDC-TM-(L)-5839/000/00	
7. AUTHOR(s): Grover H. Perry, Norman E. Willmorth	8. CONTRACT OR GRANT NUMBER(s): F30602-76-C-0180
9. PERFORMING ORGANIZATION NAME AND ADDRESS: System Development Corporation, 2500 Colorado Avenue, Santa Monica CA 90406	10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS: 62702F, 55810266, 02
11. CONTROLLING OFFICE NAME AND ADDRESS: Rome Air Development Center (ISIM), Griffiss AFB NY 13441	12. REPORT DATE: June 1977
	13. NUMBER OF PAGES: 233
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	Data Definition Library
Chief Programmer Teams	System Data Control
Configuration Management	Programming Style
Program Definition Library	Environment Analysis Tools

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

The frequent high cost and low quality of many command, control, and communications systems provide a strong impetus for the study of Modern Programming Practices that might bring costs and quality under control. In this study the standard practices for two USAF software projects were evaluated. The first, the COBRA DANE missile intelligence system, instituted a broad spectrum of standard practices for program production and management. The second, the Air Force Satellite Control Facility (AFSCF), faced with serious problems in co-

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

UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

339 900

ordinating the data processing efforts of many diverse system users and software suppliers, implemented a compool-sensitive system to help integrate the varying needs and efforts of the project.

The COBRA DANE evaluation concluded that elements of programming style, program and data definition libraries, configuration management practices, and chief programmer teams are effective in improving productivity and quality. Relaxation of standards was associated with project difficulties. Of those, not baselining the Computer Program Development Specification was judged most serious, inhibiting the firm definition of the system and the control of changes.

The AFSCF compool investigation concluded that utilization of a compool, data change control procedures, and environment analysis tools (1) contributed to the simplication of the programming job, (2) improved the efficiency of utilization of computer resources, and (3) helped solve the configuration management and interface control problems that accompany coordination of the activities of many users and suppliers of software.

TABLE OF CONTENTS

TABLE OF CONTENTS (con't)

TABLE OF CONTENTS (con't)

TABLE OF CONTENTS (con't)

TABLE OF CONTENTS (con't)

TABLE OF CONTENTS (con't)

LIST OF FIGURES

LIST OF TABLES

EVALUATION

This report describes and evaluates the software development technology employed on two Air Force software projects by System Development 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. It is believed that formal production and management discipline will enable cost and quality control as well as schedule adherence.

Roger W. Weber

ROGER W. WEBER
Project Engineer

PART I - OVERVIEW

1. INTRODUCTION

The overall objective of the Evaluation of Standard Programming Practices study is to assist RADC in the assessment of the effects of Modern Programming Practices (MPPs) on the development of large systems produced for the Department of Defense (DoD).

The objective is met here by performing case studies on two projects. First a broad band evaluation is made of a variety of techniques applied to a high technology space surveillance system (COBRA DANE). Second, the results obtained in applying a very powerful development and management tool (the Air Force Satellite Control Facility's COMPOOL-Sensitive System) to a very complex and difficult software integration task are described. These are two entirely different levels of detail in assessing MPPs but both serve to illustrate the close relationship between applied software development technology and proper software project management. Both are essential if goals of increased programming productivity and improved software quality are to be achieved.

1.1 DEFINITION OF THE PROBLEM

The development of computer software systems all too frequently requires unexpectedly large expenditures by the Air Force as well as by other branches of the military establishment and the government. Frequently the delivered software fails to perform at the specified levels and is poorly documented and difficult to use and maintain. In many instances, high costs are attributed to the slippages in schedules that result directly from inadequate developmental methods and poor management practices. To select the most effective methods in achieving high quality software products, comparative evaluations of software development methodologies in actual practice are required. Although many of the current developments in computer programming technology are intellectually appealing, the claims of beneficial results to be expected from their utilization are largely based on personal opinion and small projects. Large projects are more complex, less amenable to close control, and more resistant to easy transfer of new technology. First, managers are unwilling to risk the success of their projects upon relatively

untried methods, and, second, appropriate time and money is seldom available for the development of procedures and the massive reindoctrination of system analysts and computer programmers in the new techniques.

Nevertheless, a great deal of experience in the development of command, control and communication systems has accumulated over the past two decades. Software projects have tried a wide variety of tools and techniques and have encountered and overcome an equally wide variety of problems. If this experience can be distilled and quantified, valuable guidance for software acquisition by the Air Force and for software development by software contractors can be extracted. Using this guidance it is possible that significant reduction in production costs and improvements in software quality can be attained by employing the best available software engineering tools, techniques and methods. The studies reported here, in combination with many other such studies, will provide the basis for developing that guidance.

1.2 OBJECTIVES

RADC's objectives in evaluating MPPs include:

- Evaluation of the effectiveness of various MPPs.
- Determination of the costs and benefits of acquiring and using MPPs.
- Determination of the conditions under which an MPP is applicable and most useful.
- Placement of MPPs within the context of the total software acquisition and development process.
- Determination of those MPPs that should be required and encouraged in Air Force software development efforts, including justifications and conditions for their use.

These studies will help further those objectives by detailed evaluations of developmental experience in using software development techniques.

1.3 SELECTION OF CRITICAL STUDIES

SDC selected two of its most recent experiences with MPPs for inclusion in this study. The first was the COBRA DANE project, selected because initial planning for the project laid down a broad set of standard practices. These practices were representative of both new technology and accepted practices. A mix of both technical and managerial standard practices was used. Since the effectiveness of one class of techniques may be dependent upon the effectiveness of the other, the combination is highly desirable for study. Furthermore, at the time of selection, the COBRA DANE project was nearing completion and offered unique opportunities to estimate how well the standard practices were observed and to assess the impacts of levels of observance on productivity and quality.

The second instance selected was the COMPOOL-Sensitive System employed by the Air Force Satellite Control Facility (SCF) to define and control data structures. It was selected because this tool was specifically developed to solve an extremely difficult set of problems that faced the SCF. These problems involved such conditions as encountering a large number of interface problems in integrating the software products of a variety of independent software developers, redundancy and inefficiency in the definition and use of data structures, and severe limitations on computer capacity. The system has met its original objectives. This study provides an opportunity to describe how well it has done so.

1.4 ORGANIZATION OF THE REPORT

This report is organized into three parts and five appendixes. This first part provides an overview of the studies and summarizes their results and recommendations. Part II, COBRA DANE Standard Practices, reports the results of evaluating the impacts of levels of observance of standard practice upon programmer productivity and software quality for the COBRA DANE project. Part III, SCF COMPOOL-Sensitive System, evaluates how well the SDC COMPOOL has resolved the problems it was designed to solve. Appendixes A and B provide detailed data base information relevant to understanding the scope of the SCF COMPOOL-Sensitive System. Appendix C is a glossary of the major

terms used in this document. Appendix D provides references to technical documentation used in the preparation of Part II. Appendix E provides references to technical documentation used in the preparation of Part III.

The remainder of this overview presents:

- A discussion of MPPs.
- A description of SDC's approach to the studies.
- A summary of results from the two studies.
- A list of recommendations for adoption and/or further study of the MPPs discussed herein.

2. MODERN PROGRAMMING PRACTICES

MPPs include all those software engineering tools, techniques and methods currently employed in the development and maintenance of software and the management and administration of software projects. The effectivity and usefulness of any particular MPP and its relative desirability as a tool will depend to a considerable extent upon its conditions of use. For instance, although a small system needs precise definition of requirements and designs to guide its development and a reasonable procedure for requesting and evaluating changes to these, it does not necessarily require the formal configuration management procedures that are necessary to control the content and structure of a large command, control and communication system. While a Chief Programmer Team may be very effective at developing even a moderately sized software system without elaborate organizational structure or managerial overhead, such teams may not be entirely adequate to meet the coordination and communication needs entailed in producing a large number of interrelated modules.

2.1 A TAXONOMY OF MPPs

Although many MPPs support both management and technological activities, they are usually classified as providing either technological support or managerial support. The technical tools and techniques may be further classed according to the sort of software development activity they support. These include analysis, synthesis, production or composition, diagnosis and

evaluation, and maintenance techniques. Management tools and techniques are often grouped as management or administrative, depending upon whether they are aimed at planning and directing activities or maintaining accounts and issuing reports. Management techniques may be further classed according to the aspect of a project they are intended to govern such as personnel, work activities, configuration management, quality assurance, information or document control, and data control. Any management technique intended for control purposes requires record keeping, activity monitoring and report issuing.

MPPs may also be classed according to their level of abstraction, ranging from a concrete tool, to a technique, to an abstract methodology. A methodology is an overall way of proceeding, that may incorporate several or any one of a number of techniques or tools. It is a particular arrangement or organization of work, such as a "software factory," "software engineering," "structured programming," or a strategic approach rather than a particular tactic. A technique is a tactic, a specific way of solving a problem or doing a task. If configuration management is a methodology, particular ways of defining configurations and/or controlling change are techniques. A tool is the specific implementation or embodiment of a technique or a device used in the discharge of a technique. If higher order languages are methodologies, a particular language might be a technique and its compiler a tool.

Tools and techniques are often referred to as being automated (computer programmed), mechanized (a mechnical device), manual (a clerical or physical activity), or conceptual (a mathematical or logical algorithm). Many MPPs are mixtures of these. Methodologies are almost always conceptual. In software, tools are almost always at least partially automated. However, one may use a manual, automated or mixed Program Evaluation and Review Technique (PERT) technique (or tool if network scheduling is the technique) or a completely manual or almost totally-automated program library operation.

As with most taxonomical groupings, the classification of MPPs is not always clear or precise. A technique in application may have manual and automated parts. It may have both technical and management objectives. It's classifi-

cation may vary with context or usage. For instance, simulation may be a methodology in an overall conceptual context, but a technique if applied to the solution of a particular problem. However, if these are looked upon as characteristics rather than classifications, MPPs may be described, if not classified precisely.

2.2 THE SOFTWARE DEVELOPMENT CONTEXT

Another important characteristic of MPP is its software life-cycle phase applicability. These aspects of its applicability are pertinent to its evaluation:

- When it is used
- When it is acquired
- When major costs are incurred
- When major benefits are realized

The phase of use is the period one tends to consider first because that is where the activities are that are important to the developers in getting the job done. However, the major costs of an MPP may be in its acquisition rather than in its operation. In the case of some manual precedures the major costs may lie in training project members to use the technique rather than in its acquisition or operation.

Similarly with benefits; the major savings in costs or the maximum yield in product quality may be in a different phase than its maximum cost or the point of use. For instance, design techniques leading to easier maintainability pay off long after the techniques are used. Thorough preparation, review and baselining of a Development Specification may incur significant costs in the definition phase, but save much lost time and money in avoidance of necessary changes, correction of logical errors, and resolution of uncertainties in later phases.

The principal life-cycle phases that have been recognized are:

- Definition or validation

- Design
- Implementation
- Qualification
- Operations and maintenance

Each of these may be divided into finer steps.

- Definition
  - Preliminary functional analysis and design (normally ending in a contract proposal)
  - Detailed functional analysis and design (normally ending in a Development or Performance Specification)
- Design
  - Preliminary or system design (normally ending in a Preliminary Part II or system design specification and an "allocated baseline").
  - Detailed or component design (normally accompanied by a Computer Program Component (CPC) Design Specification)
- Implementation
  - Coding and debugging
  - Computer program development testing
  - Unit testing
  - Software integration testing
- Qualification
  - Preliminary Qualification Testing (PQT)
  - Formal Qualification Testing (FQT)
  - System Development Testing (Validation Testing)
  - Operational Test and Evaluation (OT&E)
- Operations and maintenance

2.3 MEASURES OF MERIT

Stated simply, the merits of an MPP are evaluated by determining rates of gains over costs. The difficulty lies in assigning a value to the benefits gained. Costs are measured in terms of the resources consumed directly in acquiring and using the MPP and indirectly in losses due to not using the resource for another purpose or in negative impacts on other activities. The major resources include dollars, elapsed time, effort, manpower, machine time, materials and facilities. The scarcity of any of these resources may place a premium on its use for MPP acquisition and operation. Manpower, for instance, is not a simple resource, it covers a range of personnel skills and levels. Availability of particular expertise may limit the amount that can be allocated to MPP development and operation or prohibit acquiring and using the MPP at all. Similarly with machine time and capability. The resource may simply not be there to use.

Benefits are normally evaluated in terms of reductions (savings) in the amount of resources consumed, or in value added to products and services. Both benefits and costs may be complex. That is, several resources may be consumed or saved by an MPP. Cost savings and benefits like decreased effort or understandability are often difficult to establish, having different values for different persons. Benefits like increased management visibility and control may or may not be reflected in dollar or time savings or in better products and the costs of achieving them may not be trivial. Hence, benefits may vary in value with the situation. Selection of appropriate tools depends in part on perceived needs and limitations.

Management-oriented benefits to be gained by using MPPs include increased visibility and control, traceability and auditability, and accountability. Some of the product quality benefits sought are increased reliability, maintainability, understandability, readability, modifiability, robustness and efficiency in speed and storage. Systems in which computer capabilities are limited or whose functions are time-critical may place a very high value on efficiency and reliability. Systems with a high rate of change may value traceability and maintainability. Such benefits are also difficult to quantify. It is difficult to say that one MPP yields twice as much management

visibility or twice as much modifiability as another. For many MPPs, a taxonomy will be able to cite only relative costs or benefits. The individual user will have to assign more concrete value to using the technique.

3. TECHNICAL APPROACH

Although the evaluation of each of the projects in the study went through the general steps of a preliminary analysis to establish the desired information, a data collection phase, an analysis of the results and the writing of a report, each study had somewhat different requirements.

3.1 COBRA DANE

The basic premise of the COBRA DANE study was that on a project of that size there would be variability in the level to which proposed standards would be observed and that the level of observance would be reflected in impacts upon productivity or product quality.

The preliminary analysis examined the published plans and proposed standards for the project to extract and formulate statements of the standards. The literature was examined for theoretical formulations in the area of the standards. Data parameters necessary to establish the level of observance of standards and the impact upon project productivity or product quality and potential sources of these data were identified.

The data collection phase examined source listings of programs, written software specifications, configuration management records, program library records (including records for global labelled common), and software problem reports and records. Personal interviews were conducted with SDC COBRA DANE managers, administrators, programmers, and specification writing and integration and test personnel.

In the evaluation of results, estimates of the degree of observance of standards were made from the records and the interviews. Special emphasis was given to the reports of specification writers and test personnel in evaluating the understandability and maintainability of programs in reference to the

degree of observance of rules for programming style and to managers and programmers on estimating the impacts on productivity.

The interpretation of the results and the conclusions and recommendations therefrom are reported in Part II of this report and are summarized below.

3.2 SCF COMPOOL-SENSITIVE SYSTEM

The SCF COMPOOL-Sensitive System was developed by SDC to solve certain management and technical problems that the SCF had encountered in modifying and maintaining its satellite ground support system. The overall objective of the study was to evaluate experience with the system in terms of its contribution to:

- Configuration control
- Reduction of software errors
- Simplification of software development
- Enforcement of design and programming standards
- Increased efficiency and effectiveness in utilization of computer capabilities

The preliminary analysis concentrated on establishing a set of project objectives for implementing the COMPOOL and locating documentary sources describing pre-COMPOOL conditions. The data collection phase examined configuration control records and change and error accounts and determined current conditions of operation in the SCF system.

The evaluation of results compared the pre-COMPOOL situation with current conditions and derived evaluations of how well the stated objectives had been met. Finally, conclusions and recommendations were drawn and documented.

4. SUMMARY OF RESULTS

4.1 COBRA DANE STANDARD PRACTICES

The following five areas of proposed standard practices were evaluated for the COBRA DANE project:

- Elements of programming style
- Program library operations
- System data control
- Configuration management
- Chief programmer teams

The primary objectives of the project in adopting these standard practices were to ensure the high quality of the software and to reduce the costs and difficulties of producing and maintaining the software. Following the rules for programming style was expected to improve the understandability of programs. The procedures for program library operations were expected to increase the effectiveness of programming support, control program quality and change, and increase management visibility of the program product and its status. The procedures for system data control were intended to enhance the quality and efficiency of data structures and handling in the system and provide improved visibilty of data structures. Configuration management provides a stable, organized environment for software development and management visibility of the developing product. Chief programmer teams provide an effective, efficient organization for programming. In general, it may be concluded that observance of standard practices contributes to improved software quality, to improved management visibility and to increased project productivity.

4.1.1 Elements of Programming Style

The COBRA DANE project established programming conventions for program commentary, naming, paragraphing and modularization. Based on the experiences of the members of the specification writing and integration and test team, observation of the practices made the programs easier to understand and to maintain.

Some extra effort is entailed in preparing commentary and in modularization of the code, but much of this effort is recovered in a greater ease of debugging. When a major redesign of some programs was required, the modularity of the code made the reconfiguration job easier than would have otherwise been expected.

4.1.2 Program Library Operations

The rules that were established for program library operations included the definition of responsibilities and procedures for the System Librarian and for interacting with the Program Library. They also included procedures for program change control and the handling of system builds. An alternative to team librarians, the System Librarian and Library serve as points of coordination for the project and regulate and control changes to the software product.

Program library operations ensure better program quality by inspection of source code for adherence to the established conventions for programming style and by ensuring that unit testing is complete before inclusion of the program or any changes to programs in the controlled or 'system master' portion of the library. Management visibility is enhanced by the inclusion of the total set of programs in the library via completed modules or program stubs, maintenance of status information and issuance of regular reports of status and activity.

Except for the record keeping and report activities, program library operations directly support programming and are tasks that must be done with or without a librarian. Library maintenance procedures (include quality assurance and change control) organize and regulate operations upon the software product and ensure the quality and currency of the working and tested portions of the library. These activities require manpower and computer time to support them, but the cost is regained in fewer problems in integrating and testing the software and in improved software quality.

4.1.3 System Data Control

The rules proposed for system or global data control established the same level of control over data structures as did the Program Library rules for programs. Quality inspections ensure the observance of structural rules for data elements and the avoidance of redundancies and inefficiencies in data organization and definition. Data control may be expanded to cover system resource utilization accounting.

Since most programs interface through passed or posted data and since it is data that adds meaning to the processing, control of data content and structures strongly influences the quality of the system. Efforts expended in enforcing quality standards and coordinating definitions of data elements result in fewer interface problems, more efficient utilization of storage and data handling capabilities, and management visibility for interfaces and resource utilization.

4.1.4 Configuration Management

Following procedures which lead to firm definition and authentication of the successive representations of the software and the critical review and control of all changes to the definition provide a firm basis for work planning and control of project costs. Maintaining status accounts helps to track developmental progress, helps to ensure that all approved changes are made and precludes unauthorized changes.

There are costs associated with the operation of the configuration control board, the maintenance of status records, and the critical review and costing of proposed changes and suspected errors. Experience indicates that formal configuration management runs about one to three percent of contract costs. Avoidance of one major unnecessary change or of one major integration problem would more than offset costs. Avoidance of confusion and provision of a standard decision procedure save effort and time even though the 'red tape' involved may seem time-consuming and irksome.

SDC's experience on COBRA DANE re-emphasizes the importance of having a firm agreement between procurer and producer on the exact functional and structural characteristics of the system. A 'baselined' Development Specification is essential to control change and to direct design activities.

4.1.5 Chief Programmer Teams

COBRA DANE used an alternative organization for Chief Programmer Teams. While the teams were built around a highly proficient programmer, program and data librarians operated on a project rather than a team basis. The Chief Programmers formed the basic membership for the Internal Configuration Control Board that determined Program Library updates and controlled software problem evaluations and corrections.

Building a program development team around a highly proficient Chief Programmer results in high code productivity. On a large project with several teams, having a Chief Architect to coordinate the overall design facilitates design decisions and helps avoid interface incompatibilities. On COBRA DANE, program and system data librarian functions were performed on a project rather than a team basis and promised to provide excellent support to a Chief Architect centered organization.

If the team follows rules for design and/or code walkthroughs and provides other checks for program quality, the cohesiveness provided by the Chief Programmer in designing and implementing the programs enhances quality. Chief Programmers are such exceptionally good coders and diagnosticians that it is difficult to avoid loading them down with these duties to the neglect of other functions of a team leader.

4.2 THE SCF COMPOOL-SENSITIVE SYSTEM

SDC has been the Computer Program Integrating Contractor (CPIC) for the SCF since 1963. The decision to implement a COMPOOL-sensitive system was made in the mid-sixties as a solution to several very serious problems. First, the system was in a state of rapid growth and constant change. There were numerous users of the system with user-specific programs and data as well as many common data processing needs and there were numerous contractors and programming groups. The system was large with many programs and a large amount of data and numerous interfaces and interdependencies among the program and data elements. These conditions made for much redundancy in data, little standardization of data and interface definitions, often inefficient usage of storage and numerous interface incompatibilities. Finally, there were stringent core and other computer resource limitations which, despite computer upgrades, continued to be serious.

The SCF needed to enhance the efficiency of programming and data handling and improve the effectiveness of system management. Although such other actions as increasing the stringency of integration procedures and a broad standardization program were also undertaken, the principal tool for attaining these goals was an SCF support system that was COMPOOL-sensitive. Such a system represents a considerable investment, but the seriousness of the problems and the advantages offered by the system were thought to justify the expense. Some of the alternatives that could have been adopted included:

- Adopting only a manual data definition, control and coordination operation.
- Using a COMPOOL-sensitive compiler with COMPOOL generation and listing capabilities to support it.
- Adding symbolic debugging and test tools to augment the compiler.
- Adding program and system environment analysis capabilities to the above.
- Adding symbolic data management capabilities.

- Adding a COMPOOL-sensitive, dynamic loader operating system.

Each of these levels of sophistication provides additional increments of control and capability. It was judged necessary to implement all of them if the SCF objectives were to be met.

The SCF had three major objectives in implementing a COMPOOL-sensitive system:

- Simplification of the tasks of design, development and maintenance of the control computer software.
- Improvement in the efficient utilization of existing hardware resources.
- Facilitation of configuration management and interface control.

4.2.1 Simplification of Programming

Program production and maintenance were improved by providing better communication between system elements, simplifications in programming, introducing guarantees of data integrity, and working through the COMPOOL to incorporate data changes rather than reprogramming.

Communication between system elements was improved by adopting standard calling sequences, standard data structures, standard predefined communication buffers and standardizing data blocks of varying levels of control.

The programmer's job was simplified by the reduction in the number of data declarations he had to make, by predefinition of working and communication space, by automating load and link operations, by providing symbolic corrector capability and by providing automatic units conversions for input and displays.

Better guarantees of data integrity were obtained by central definition and distribution via library tapes of constants and global data definition, and by distribution of standard data bases. In operation, the saving and

automatic updating of specified data files is guaranteed. All global data definition is centrally controlled and standard data formats and units of measurement must be used.

The COMPOOL-sensitive system not only supports program maintenance through the automatic updating of programs impacted by global data changes but through well-defined interfaces and a variety of analytic aids. The aids include symbolic and binary compool listings, program characteristics listings, program environment listings and core maps of every dymanic load. Symbolic correctors are also an aid to program maintenance.

4.2.2 Resource Utilization

Control over the utilization of very limited computer resources in the face of a system that has grown very dynamically in size, sophistication and throughput load has been very effective. This control has been accomplished through a strict program of storage on an as-needed basis, purging of redundant and obsolete data and an active program of control and reduction of program environments to ensure efficient storage utilization and data handling.

Storage is conserved by the commonality of data structures, utilizing extensive overlay capabilities, providing hierarchical compools to reduce executive overhead, using an efficient segmentation algorithm and automatically policing current core allocations.

Using the compool on-line to determine and adjust storage allocations at load time improves the flexibility of operations by making all programs and data relocatable, providing dynamic linking and loading, and allowing symbolic correctors to be inserted with any core load. A push-pull capability exists that will avoid the dynamic allocation overhead if a core load is invariant from run to run.

The procedures employed to review and control additions and modifications to the compool and the analytic aids provided contribute to the purging of obsolete and redundant data from the system. This keeps storage requirements down. Programs are also reviewed for standard functions and software suppliers are encouraged to use standard routines wherever possible.

Analytic procedures are extended to indepth analysis of program environments to detect inefficiencies in data storage and handling. Commonality of data items is searched for and suppliers are encouraged to use common data elements. The compool change procedures flag inefficiencies, including inefficient use of routines and segmentation procedures. Separating control data from application-specific data encourages the use of overlays and tables of constants.

4.2.3 Configuration Management

In an environment of constant change, multiple configurations, and many users and suppliers, close control over software configurations is necessary if problems are to be kept to a minimum. The improvements realized in implementing the compool included much enhanced configuration control, improved coordination of interfaces both between programs and with external agencies, better control over the sources and the quality of data in the system, and a general improvement in overall management.

Configuration control profits from precise change evaluations and configuration accounting procedures. Standardization simplifies evaluation not only of compool changes, but of all new programs and modifications.

Coordination of interfaces is improved by the change evaluation procedures, by the standardization of calling sequences, communication buffers, and block structures, and by regular analyses of the software system requirements.

Control over the quality and sources of data is established by providing library tapes and data bases, and by providing project-specific and private data bases for the protection of sensitive data and project-peculiar requirements.

5. CONCLUSIONS AND RECOMMENDATIONS

5.1 CONCLUSIONS

The general conclusions that may be drawn from this study are:

- Observation of standard programming practices does contribute to software quality.
- Observation of standard practices for software management increases the efficiency of programming, simplifies decisions, and improves the visibility and manageability of the software product.

5.1.1 Elements of Programming Style

Using the elements of programming style increases the understandability and maintainability of software. Deviations from rules for commenting, naming, paragraphing and modularizing programs lead to more cryptic, more complex, less easily diagnosed and verified programs.

5.1.2 Program Librarian

Using a program library places the evolving software system under control and helps make its content and status visible to management. In conjunction with software problem reporting and change request procedures, it ensures the evaluation of changes and corrections, and makes sure that no unauthorized changes are made or approved changes overlooked. It improves the efficiency of programming by keeping programs readily accessible for further work and for use in integration testing.

It is concluded that a System Librarian is a viable alternative to Team Librarians, but additional help may be required to perform quality inspections and maintain hard copy materials. For most effective operation, a strong and knowledgeable person is required to be System Librarian.

Pitfalls exist for the Program Library in the relaxation of quality assurance criteria and the reduction of maintenance lead time under the press of time and workload. These permit errors to creep into controlled software and may degrade the delivered product through substandard observance of programming style conventions. Providing a stable environment for programming, integration testing and equipment checkout is essential to efficient operations. Frequent updates of the Program Library may be a disruptive factor unless accompanied by a maximum amount of communication. The integrity of delivered versions of the system should be kept to fall back upon in case of failure. Version control and accounting are particularly important where a series of successive system builds are used as a development strategy or in order to attain an early initial operational capability.

5.1.3 Global Data Control

A system data definition library provides the same element of management control and visibility over data as the program library exercises over software modules. It simplifies the programming task, reduces data redundancies and inefficiencies, facilitates program maintenance, and simplifies both program and people interfaces. In a complete system, such as that used by the Satellite Control Facility, the system may be used to monitor the utilization of computer resources and can provide many system environmental analysis tools.

The Data Base Coordinator must also be a strong and knowledgeable person in order to enforce standards of data structure and efficiency and oversee resource utilization.

Pitfalls exist in treating data elements less seriously than program modules, in not thoroughly coordinating the impacts of data changes among users, and in relaxing quality assurance rules under pressure. Data changes often have far more impact upon the system than program changes.

5.1.4 Configuration Management

Observance of configuration management practices is essential to the orderly development of a software system. These practices ensure the clear definition and review of the system at several critical points in the developmental process. They ensure the orderly evaluation and incorporation of changes to the system and the maintenance of the system definition for all approved changes. They provide status accounting and traceability of requirements and changes.

Configuration Management procedures should include both external and internal configuration control boards. The internal board should work in close conjunction with Program Library and Global Data Control, which support configuration management via contents inventories, change control and accounting, and published activity logs.

Pitfalls exist in configuration management in the failure to baseline system representations as they appear. The most serious pitfall is the failure to baseline the Functional Requirements Specification. Baselining provides a firm basis upon which to evaluate, plan, and manage change. Not doing so provides no basis upon which to ground software change control procedures.

Pitfalls also exist in not implementing software change (ECP) and software problem (SPR) processing procedures and accounts and in not implementing configuration definition maintenance procedures (usually maintenance of specification documents).

Not having formal procedures permits informal, uncontrolled changes and also allows undocumented changes to be forgotten. Document maintenance is necessary to keep both procurers and producers aware of what is to be produced and communicates change to all affected participants.

5.1.5 Chief Programmer Teams

Building a programming team around an exceptionally able programmer appears to be an effective way to increase the volume and quality of code produced.

On a project with several teams working on interrelated software, a Chief Architect to oversee the whole system, provide direction and ensure the integrity of the system may be a desirable organizational feature.

Pitfalls to Chief Programmer Teams exist in the overcommitment of the Chief Programmer to module production at the expense of guiding and training his team members, critiquing their designs and code, and handling supervisory problems. The Chief Programmers may have slightly different interpretations of programming standards and the importance of enforcing them leading to some variability in quality.

5.2 RECOMMENDATIONS

On the basis of these studies, it can be recommended that all of the techniques studied be adopted for all major software development projects. Additionally, further development of several of the standard practices is desirable.

5.2.1 Elements of Programming Style

Although standard rules for writing programs are becoming more generally accepted, a minimum acceptable set should be extracted and stated as verifiable standards.

5.2.2 Program Libraries

A program library should be employed on every project to store the product representation, ensure its quality, protect it from unauthorized change, and provide configuration and change status accounting and reporting.

Further development should be done on program library support systems both in batch and interactive environments. A minimum set of functional requirements needs to be established and further research on the effectiveness of library tools needs to be done.

5.2.3 Global Data Control

A project should exercise even closer control over the configuration and contents of a global data definition file than it does over the program file. A viable approach to this control lies in the COMPOOL-sensitive system, but unless such a system exists, it is recommended that only the portion of it that can be demonstrated to be economically cost-effective be implemented for any particular project.

5.2.4 Configuration Management

All projects should follow configuration management practices. Configuration identifications should be established and baselined and formally maintained for change. Procedures for handling both changes in design requirements and errors in controlled programs, including accounts for status and traceability, should be used. Regular configuration, change and problem status reports should be maintained.

5.2.5 Chief Programmer Teams

Although organizing a programming team around a very proficient programmer seems an effective technique, no recommendation for any specific team organization is made. Projects vary a great deal in conditions and available personnel so that the organization that satisfies one set of requirements might not satisfy another.

System Librarians and System Data Controllers seem a viable alternative to Team Librarians, although some person should be assigned to keep team records.

If there are several teams engaged in a project, it is recommended that a Chief Architect be appointed, in addition to the Project Manager, with responsibility for the overall technical direction of the software system. The Chief Architect, the program and data librarians and a configuration control librarian may form a Configuration Management Office with the internal and external configuration control boards as the coordinating mechanism.

PART II - COBRA DANE STANDARD PRACTICES

1. INTRODUCTION

The COBRA DANE project was chosen as the basis for an analysis of MPPs because:

- It represented a major Air Force hardware/software development program.
- A broad set of standard programming practices was proposed for the project.
- Development of the system was nearing completion.

1.1 OBJECTIVES OF THE STANDARD PRACTICES

The primary objectives of the COBRA DANE standard practices were to improve the quality of the software and to reduce the costs and difficulties of producing and maintaining the software. The standard programming practices were expected to lead to more modular, more easily read and understood programs. This was expected to ease the integration task, make error diagnosis and correction much simpler, and result in more easily maintained and modified programs. Although the cost of producing and executing better structured, better annotated and more modularized programs might be expected to be greater, the costs of integration, test and maintenance should be reduced.

The management practices should ensure the compatibility of modules for integration, the control of the identification, content and structure of the system configuration and the final product, the control of system resources, the regulation of programming procedures and the assurance of product quality.

1.2 BACKGROUND

The COBRA DANE system is a phased array radar system procured by the Electronics Systems Division (ESD) of the Air Force Systems Command (AFSC) to be deployed at Shemya AFB, Alaska, in support of the Air Defense Command (ADCOM) and the Foreign Technology Division (FTD) of the Air Force. The primary mission of the system is intelligence data collection on long-range missiles launched from Soviet facilities on the Kamchatka Peninsula. Determining the characteristics of these weapons early in their development will provide profiles for later identification and an assessment of their capabilities. The secondary mission of COBRA DANE is to provide satellite tracking to the Spacetrack System and tactical warning to NORAD on CONUS-impacting missiles.

The Prime Contractor for the system was Raytheon. The System Development Corporation (SDC) subcontracted with Raytheon for the design and implementation of the COBRA DANE software. The work was done in Raytheon facilities in Wayland, Mass. The computing facility was managed by Raytheon, who also used the facility as a test bed for radar, communication and display equipment. Raytheon operators manned the facility but after the initial project phases, the bulk of program development, integration and test was accomplished through personally attended block time. SDC also supported Raytheon in the integration, installation and validation of the system at Shemya AFB.

The computer equipment configuration for the COBRA DANE system consisted of a CDC CYBER 74 computer with a 131K, 60-bit word main memory, ten peripheral processors, and 12 I/O channels which also interfaced with radar and real time controllers. The display subsystem consisted of five CDC 243 consoles with CRT screen, light pen, programmable function keyboard, and alphanumeric keyboard. There was also a strip chart recorder and a printer. The communications subsystem comprised three full duplex synchronous lines of 2400 bits-per-second to The Foreign Technology Division (FTD), Dayton, Ohio, and NMCC (NORAD) and/or the Alternate Space Defense Center, Eglin AFB, Fla. There are two full duplex synchronous lines of 160 bits per second (teletype) to Cobra Ball and to Autodin.

The operating system (CDOS) for COBRA DANE was a CDC SCOPE 3.4 system modified to include system specific commands. The languages used were FORTRAN IV EXTENDED and COMPASS. The SCOPE system included a program library capability (UPDATE) and a subroutine library, but no programmer macro capability. Any shared bits of common code had to be a manual exchange, either listings or cards; an inconvenience; but not limiting.

The COBRA DANE software was divided into a Mission CPCI and a Simulation CPCI. The development tasks consisted of eleven application and five executive and support functions. The application functions included:

- Radar Scheduling
- Radar Command Generation
- Radar Return Processing
- Tracking Selection and Process Control
- Object Classification
- Spacetrack Tasking
- Display Processing
- Calibration
- Orbital Mechanics
- Satellite File Maintenance

The executive and support tasks include:

- COBRA DANE Operating System
- Real Time Control
- Communications
- Post Mission Processing
- Simulation

Although the COBRA DANE software system was very complex (perhaps among the top half dozen ever produced by SDC) and contained some very high-level technology areas; there were no programs that were not state-of-the-art (i.e., that similar programs had not been developed previously). During the course of the project the Radar Command Generation function was expanded from a mere passing of general commands to the detailed control of the antennas. The displays were expected to be simple and straightforward and they were, as far as individual displays were concerned. However, the number of displays and their sizes and complexity exceeded expectations and the display processors provided with the display equipment were not as powerful or reliable as anticipated.

The COBRA DANE system was delivered in a series of seven builds, each build being planned to support the checkout of some item of COBRA DANE equipment. Additionally, the executive functions (CDOS, RTC) were required early as the matrix to embed the other programs. The simulation programs were delivered in three incremental builds to support the testing of the application builds and allied equipment.

The seven builds were implemented and tested in three major phases. In the first phase, the CDC-supplied SCOPE 3.4 Operating System for the CYBER 74 computer was modified to operate in real-time and to facilitate interfaces with radar command and control equipment. A Real Time Control (RTC) Executive was also developed and system analysis and design work on application programs was accomplished. In Phase II, the application programs which consisted of the real time programs (launch and space tracking), post-mission analysis and simulation programs were produced and verified in the CONUS test bed. Phase III consisted of providing support to the final system integration and validation in its operational environment at Shemya AFB.

An extensive package of standard practices for both programming and project management was proposed for COBRA DANE. Both practices that were proven and used extensively on previous projects and practices that have received endorsement in the current literature were included. Among the proven techniques were configuration management procedures, system (global) data control and information (document) control. The new techniques included the elements of programming style and structure, program libraries, and chief programmer teams (all practices subsumed under structured programming).

Actually, the standard practices proposed were mixtures of accepted practices at SDC and re-emphases derived from the tenets of Modern Programming Practices. For instance, program segmentation as a device to divide programs into manageable portions has long been an accepted practice, but the simple control structures and the principles of modularization provide logical justification and direction to the practice. SDC normally controls programs under construction by a "two-tape" system, a Test Master used to build and unit test programs and a System Master containing internally verified

programs ready for formal qualification and final integration. To this, the Program Library concept adds emphasis in terms of activity logs, library listings, and stronger, formalized librarian procedures. System or global data control has long been a feature of SDC JOVIAL-based programming support systems, an area not yet adequately covered by modern programming practices.

Configuration management and document control are matters of great concern to military contracts that tend to not be emphasized by the advocates of other modern programming practices. These practices are intended to establish a precise definition of the software to be produced and to prevent unauthorized changes to the approved configuration. They ensure that the producer knows precisely what is to be produced and that he will be reimbursed for any perturbations in his contractual responsibilities. They guarantee that the customer knows what he has ordered and provide a means for him to control the structure and content of the system.

In general, SDC, decomposes a system to be produced into segments that can be designed and programmed by small teams. The Chief Programmer Teams (CPT) formalize this concept and provide a specific structure. The teams organized for COBRA DANE emphasized the role of the lead programmer as the Chief Programmer. They varied from the suggested CPT organization by appointing a single project program librarian rather than several team librarians.

There are other MPPs such as quality assurance and software testing procedures that the COBRA DANE project followed, but that were not specifically covered in the proposed standard practices and that were not, therefore, included in this study.

1.3 ORGANIZATION OF PART II

The remainder of Part I covers the findings and recommendations of the COBRA DANE study. Section 2 discusses the technical approach taken in the study. Section 3 covers the objectives of the standardization program, both as they were stated in the COBRA DANE plans and as they have been explicated in the literature. Section 4 describes the results of the study. In this section, the proposed standard practices are stated, the evaluation of the enforcement of the practices is given, and an assessment of the impact of observing or not observing the standard on project performance is made. In Section 5, conclusions are drawn about the implications of the study for MPPs and recommendations concerning these practices are made.

2. APPROACH TO STANDARDS EVALUATION

The evaluation of the impacts on project performance and product quality of varying levels of standards observance was performed in these logical steps:

- Preliminary analysis
- Data collection
- Data categorization and analysis
- Evaluation of the standards program

2.1 PRELIMINARY ANALYSIS

First, the published plans and standards proposed for the project were examined to extract and formulate statements of the standards. Additionally, COBRA DANE project members were consulted for clarification and formal notes were used to determine details on the intent and implementation of the standard practices.

To establish a baseline for the theoretical understanding and evaluation of the standards, numerous books and articles in the literature were surveyed.[1] These included, for programming standards, Kernighan and Plauger [1], Van Tassel [2], and Meyers [3]. For Programming Library Operations, an overview was obtained from Baker [4, 5] and finer detail was obtained from two program library users' manuals and librarians' guides used on SDC projects. For system data control, reliance was placed on information gleaned from the SCF COMPOOL study separately reported here. For configuration management, the MIL-STDs 480,483, 490, and 1521 and AFRs 800-14 and 65-3 were reviewed, but general reliance was placed on the standards produced for this and other projects by L.V. Searle [6]. Documentation standards were taken from the same sources, plus the ANSI flow chart standards. CPT descriptions we e taken from Baker [5], and from Brooks [7].

Also as part of the preliminary analysis, the COBRA DANE project environment was examined. The intent of this investigation was to place the COBRA DANE project in a context that enables one to understand the size and scope of

[1] Bibliographic citations for these documents are numbered and listed in Appendix D. References to these citations (e. g., [7]) are used throughout Part II.

the project and the conditions under which it was performed. The environmental factors* that were examined, as summarized in Section 1, included:

- System type, size and complexity
- Computer characteristics
- Operating system characteristics
- Programming tools available
- Computer facility

These factors have been found in the past to affect project performance and the effectiveness of standards must be evaluated in light of the conditions under which they are applied.

On the basis of the list of standard practices and their objectives, tentative hypotheses were advanced concerning their affect on performance. Additionally, the data requirements necessary to verify the hypotheses were identified. These included:

- Source program listings
- Specification documents
- Program library logs and records
- Global data library logs and records
- Configuration indexes and status reports
- Software problem reports and status reports

2.2 DATA COLLECTION

Source program listings were examined to determine the level of observation of the elements of programming style in program construction. A unique opportunity to evaluate the effectiveness of the practices vs understandability and maintainability arose in two special project activities. First, a group of experienced programmers was assembled in Santa Monica to write final Part II Product Specifications for the program modules. Source material for this activity was largely source code listings. The Preliminary

*As a checklist for this investigation, the Project Environment forms proposed to RADC in TM-5542/007/01, "Software Development Data Collection: Compendium of Procedures and Parameters," SDC, June 7, 1976, were used but expanded to include explanatory detail.

Part II's were also used, but were of limited usefulness due to the many changes that occurred. The second activity was the team that was assembled at Shemya to support final integration and test of the system. Both groups were interviewed to determine how well the standard programming practices were observed and what their impacts were on the understandability and maintainability of the software.

A large sample of the documentation for the project was scanned for observance of the large practices for information control. Programmers and specification writers were queried concerning the impacts of documentation practices upon their work.

Program library logs and listings were inspected, but were too extensive to analyse in the limited time available. An attempt was made to write a program to summarize program library historical records. but proved infeasible, given the available computer time and programming costs. The library records had been reconstituted several times during the course of the project in order to provide additional management information and assembling a summary listing over the dissimilar files was a more complex task than originally estimated.

Listings of global labelled common were were also scanned. Configuration status records such as configuration control board minutes and letters in response to Configuration Control Board (CCB) actions were examined. The files of software problem reports were also examined as were the regularly published, problem status reports.

In addition to the interviews with the specified writers and test team members, interviews were also held with a number of project members and managers. These interviews proved most fruitful in gaining insight into project environment as well as the impacts of standards observance.

2.3 ANALYSES

Evaluations of the observance of the standards were derived based on the reviews of software products and records. In general, the available information was inadequate to perform statistical analyses concerning exact observance and performance impact and reliance had to be placed on overall assessments and "critical incident" reports from project members and managers. Responses to interview questions were compiled by standard practices area and evaluated for seriousness of impacts.

2.4 EVALUATIONS

The evaluation of observance of the elements of programming style resulted in a series of estimates of the degree of conformance: (1) when first submitted to the Program Library, and (2) as the Product Specifications were written [i.e., during Integration and Test (System Validation Testing) phase.] Impacts on performance were gathered from programmers, specification writers and test team members. However, overall programmer performance was so good that it is difficult to assess the impact of the standardization program on productivity. Whereas theoretically code production might be expected to be depressed by following stylistic rules, the COBRA DANE project produced code at a lower cost per statement than originally estimated. This productivity rate was achieved despite a large volume of changes including at least one major redesign.

Observance of program library and system data control standard practices was evaluated from library logs and the comments of the System Librarian. The impact on performance was assessed from comments made by programming and test personnel.

Observance of configuration management practices was evaluated from the existing records, including CCB minutes, the Software Problem Reports and Program Library records, and the comments of the project managers. Impacts on project performance were evaluated through comments by programmers, and by administrative and managerial personnel.

The evaluation of the use of CPTs is based on the project productivity record and the special emphasis placed upon the Chief Programmers in controlling and performing the work. Special consideration was given to deviations from the industry-proposed team composition, specifically, System vs Team Program Librarians.

3. OBJECTIVES OF THE STANDARDIZATION PROGRAM

The primary objectives of the COBRA DANE standardization program were to increase software quality and intercompatibility of modules and to reduce the cost and difficulty of producing and maintaining the software. To attain these objectives, it was planned to adopt standard practices for several aspects of the project. These included:

- Programming Style
- Program Library Operations
- Global Data Control
- Configuration Management Practices
- CPTs

3.1 PROGRAM STYLE

The elements of programming style include rules for:

- Commentary
- Paragraphing
- Naming
- Modularity

The objectives of these rules are to increase the legibility and understandability of the program, to contribute to the control of program quality, and to enhance the ease of maintenance and modification of the program.

Commentary, both explanatory prologues to modules and internal annotations of instructions, is intended to explain the intent of the module and to clarify processing steps. By identifying the author of the module, listing its data and subroutine environment, and estimating sizing and timing parameters, programming control operations are enhanced.

Paragraphing, which may be extended to punctuation and formatting rules as well as blocking and indentation, is aimed at increasing the legibility of the program. Paragraphing helps reveal the internal structure of the program and organize its contents as well as make the program easy to read.

Naming conventions for program and data elements are aimed at increasing the semantic content of the program through nmenonic associations and construction rules. Legibility may also be increased by arranging names in lists and calls in sequential order, by adopting standard abbreviations for common words, and by stating simple rules for the hyphenation and continuation of expressions.

Modularity has several functions beyond breaking a large program into more easily understood partitions. Rules for modularization include (1) the minimization of interaction among modules, (2) making module subordination apparent, (3) singularity of function for a module, (4) reducing entry and exit points, and (5) decoupling modular interdependence. The objectives are to increase the ease of understanding through greater structuredness of the program, simplification of functional performance and decreasing complexity of interaction, and to enhance maintainability by providing replaceability of modules and isolation of the impacts of change.

3.2 PROGRAM LIBRARY

The objectives of a program library are largely those of management control and accounting, but a program library also provides some programming efficiency in terms of organization and knowledge of the exact status and content of modules. Keeping program stubs in the library not only provides a "tickler" to tell management where planned elements are missing, but, where a stub simulator is present, will support an artificial environment for testing completed modules. Having an expert librarian and/or a broad set of canned job control and linkage procedures is a convenience to programmers and helps avoid the run and link errors that frequently accompany the use of complex operating systems.

3.2.1 Management Objectives

The managerial objectives of a program library include:

- Configuration identification and status accounting
- Design change control and accounting

- Software problem control and accounting
- Program and operating statistics
- Historical records of library operations

If the Program Library is initiated as soon as the system design is firm, with program modules represented by program stubs, an identification of all the elements of the system configuration is available and may be displayed for configuration management purposes. This makes the program production process much more visible to management.

As program elements are completed and tested, the module code replaces the program stubs and a record is kept of each change of state of the elements either in manual or machine storage. Part of the configuration identification and accounting is keeping track of the versions and modifications to a program and of the contents of builds, adaptations, and sets of modules of differing baselined or tested status (as Test Masters and System Masters) usually to facilitate the exercise of more stringent levels of control. Version or mod numbers or identifiers may be added to the module ID or separate library sections may be used. In short, this is an inventory control problem with each module of the product separately identified and/or stored.

In addition to keeping track of versions of the system as modified, changes to the modules are themselves controlled. No unapproved changes to controlled modules are permitted, although the stringency of control varies greatly with the status of the modules.

Keeping records of all changes to modules helps identify the contents of the various versions and modifications of the system and, by keeping test records and change status, an accounting of whether and when a change has successfully been accomplished. Changes may either incorporate altered requirements or correct detected errors and deficiencies. The incorporation procedures are quite the same, but with different impacts on status. Separate accounts are normally kept of design changes and error corrections. Since the bulk of

ECP and SPR processing occurs prior to the entry of a change into the Program Library, correlation must be made between program change identifiers and ECP and SPR identifiers for full accounting, all of which should be part of the Configuration Management Standard Practices for the project.

All activity in the program library is usually recorded, either by a Program Librarian logging jobs in and out and recording statistics, or by the operating system. Additionally, both source and object program size and operating time statistics may be kept. Although not strictly part of the Program Library system, some Program Library systems have access to analytic programs that can derive quite elaborate statistics on program makeup. For instance, analysis may find that a load module differs significantly in size from the sum of individual member sizes due to additional space for linkages and reductions in space for overlays. Program and operations statistics are useful to management in detecting current over-utilization of computer resources, in making future estimates of resource costs for new systems and similar analytic applications.

The objectives for keeping historical records lie in the evaluation of how well the project did and in the derivation of guidance for future projects. Unfortunately, to date very little use of such data has resulted. Much of the data is stored in manual form so that analysis requires a major effort. It may also require correlation of data from configuration management, resource utilization records, and the program library logs, a correlation made difficult by not recording against the same criteria. A complete program library system simplifies these analyses.

3.2.2 Programming Efficiency Objectives

Improving the efficiency of programming operations is the second main objective for program libraries. This is accomplished by:

- Improving the convenience and ease of use of the computer and the operating system.

- Keeping computer programs in readily accessible storage
- Providing a back-up capability in case of failure of new versions or modifications.
- Providing access to various analytic aids.
- Facilitating organization of the programming process.
- Making other programmers' modules available for interface tests.
- Providing a simulated environment via stubs for simulation and test runs.
- Providing information on the state of interfacing modules to programmers and test teams.

Operating systems are normally written in a very general fashion that provides a maximum of power and generality of application. However, the generality results in a complex job control language that must be used to direct the performance of each computer run. The intricacies of the operating system and the job control language may require a level of sophistication beyond the journeyman programmer. Most projects have a number of standard jobs such as loading a source program or data deck, compiling or assembling, link editing, or module modification that vary only in the module IDs and job parameters. One of the functions of the Program Librarian is to be an expert on the operating and utility program systems and in the job control language, first, as a consultant to individual programmers in accomplishing the tasks they desire to perform, and, second, in providing standard job control procedures for library use. This practice seeks to avoid many job control language errors leading to aborted jobs and lost time, and to select the most efficient and effective ways of performing jobs. It relieves the programmers of an onerous task and speeds up turnaround times and reduces irksome reruns due to errors not caused by the source program.

Readability and easy modification of one's program should increase programming efficiency. So also should knowledge of the status and ease of access to the programs of others that interface with one's own module. Knowledge of

status can avoid abortive runs using bad modules and hence may save computer resources. This knowledge is helpful both during unit testing and for integration testing.

The Program Library operation normally keeps program correctors in a separate file until they are checked out. When a new version is constituted, the library may keep at least one previous version to provide a backup version in case the modification or new version fails to perform correctly. While such operations are common under non-library operations, easy version accountability under a Program Library makes such fall-back operations very easy.

The Program Library also provides a service to the programmer in keeping things organized for him. He knows what versions or copies of his modules exist, which one is the latest version, and, in general, keeps things straight, simple, and uncluttered. A militant librarian can help in keeping bad or obsolescent materials purged, in getting storage space and computer time, and in interpreting the system to the programmer.

Use of stubs for system simulation and tests requiring module interactions before the actual modules are available may be of assistance to both system analysts and programmers. Providing such capabilities does mean extra tools in the Program Library or operating system which are among the many other analytic and service routines that can accompany a Program Library operation. Having a library makes program environment analysis (set-used, memory maps, etc.) easier.

3.3 GLOBAL DATA CONTROL

The objectives for the control of global data are much the same as for programs, i.e., management control and accounting and programming efficiency. However, control and accounting for system resources may also be added. That is, visibility of the structure and logical content of data elements, accountability and control of data changes, ease of access and use of global data and an organized way of doing business.

To these must be added the desire to control the assignment of computer resources (most especially storage capacity) to system elements and operations. The use of efficient data types and structures, the creation of the most efficient data modules to meet required program operations, and the avoidance of redundant data items are the corollaries to this objective.

Since many system elements normally interface through global data elements, changes to data often carry more adverse impact than do changes to programs. In fact, if proper modularization procedures have been followed, most modules will be decoupled and interdependencies isolated and relegated to data and passed parameters.

Standard practices for global data control include:

- Data naming conventions (including program and routine names)
- Allowable data structures
- Commenting requirements
- Central control of global data contents and changes to them.
- Quality inspections for redundancies and duplication.
- Analytic procedures for ensuring efficiency of data manipulation and storage utilization.

Naming conventions often present rules for the formation of mnemonics, for identifying the subelements of files and tables, and for indicating function or usage. Allowable data structures may prohibit the use of various structures and usages. Commentary usually includes a decoding of the mnemonic names and requires a short statement of data element usage. Additional descriptors (such as size, range, etc.) may be added (manually or automatically) to permit editorial checks on data values and the use of analytic routines for reports.

Data Library operations are much like those for Program Libraries, including if necessary, version and modification identifiers. The most important function is the control of data changes including the evaluation of the impacts of changes in the system and ensuring that each change is communicated to all users of the data element. Efficient data structures are just as important as efficient program structures and may communicate as much understanding of a program or system as does knowledge of the structure and functioning of modules. Control of them is as important as Program Library control.

3.4 CONFIGURATION MANAGEMENT PRACTICES

Configuration management is aimed at the precise definition and accounting for system structure and content and the control of and accounting for deficiencies and changes thereto. Many of the operations of a program library are of a configuration management nature. Configuration management includes:

- Configuration identification
- Configuration authentication
- Configuration control
- Configuration accounting

The objective of configuration identification is the precise definition and description of the system elements. This includes the assignment of names and identifiers, descriptive titles and descriptions. It may include identification of models and versions, responsible agencies, important dates, and descriptive statistics associated with the system element. Normally, through the system life cycle, a system has several representations. That is, it is represented first as a set of requirements, then as a series of one or more designs, then as software products (as program, global data description and data files) in various stages of integration and verification. It may also have several versions or models, several of which may be under development, operation and maintenance at the same time.

The objective of configuration authentication procedures is the review and auditing of a particular configuration identification to ensure the complete-

ness of its definition and the adequacy of its quality and attesting to, authenticating or approving the identification as a baseline for further work.

The objective of configuration control is to control all changes to a baselined configuration, i.e., to prevent all unauthorized changes and to identify and account for all requested changes and suspected discrepancies and ensure that all approved changes and corrections are made.

The objective of configuration accounting is to keep an inventory file of all system elements and their contents, including all approved changes to them, and their current state of development and to report this information periodically or as required for configuration management and control purposes. In this regard, if configuration management records are kept separately from the program library, listings of internal library contents may be compared with external listings to ensure their compatibility and currency.

3.5 CHIEF PROGRAMMER TEAMS

The purpose of any personnel organization is the mobilization of human resources to accomplish a task. The Chief Programmer Team is based on the premise that an experienced programmer supported by a small team of programming assistants can produce a computer program faster than programmers working under a more ordinary line and staff organization.* At a minimum, the team will consist of a back-up programmer and a librarian. Other programers may be added as required, but the team should not exceed ten members. The Chief Programmer is responsible for all overall design, for writing the main program, and for reviewing and critiquing all subprograms designed and written by his assistants. He directs the activities of his assistants but may or may not be responsible for personnel administration.

*Claims of productivity of more than 10 times the normal rates have been made for this approach [9].

The Back-up Programmer is responsible for one or more submodules and for reviewing and critiquing the programs written by the Chief Programmer. He acts as an advisor and sounding board for the Chief Programmer and, most importantly, is in training to take over from the Chief should that become necessary.

The Librarian, in a batch environment, is responsible for receiving and logging of jobs to be submitted to the computing facility and for preparing job control and linkage instructions and submitting jobs to the computing facility. He also receives jobs back from the computing facility, logs job statistics, files copies of listings and distributes the results to programmers. He also preserves documentation of other kinds. He is the custodian of the Program Notebook or Project File and of the Program Library. He may also be the custodian of subroutines, procedures and macros, making sure that they are documented and that the documentation is disseminated or that the programmers are aware of what is available in the common pool. He is also responsible for issuing Program Library reports and Run Log reports and is keeper of status information on the Library. In an interactive environment, much of the Librarian's record keeping functions may be taken over by the operating system, but he retains responsibility for reports, library maintenance, configuration records, documents and files.*

The remaining programmers work under the direction of a Chief Programmer. It is recommended practice that programmers work as pairs to review and critique one another's work and act as back-up. Structured walkthroughs are also recommended to get critiques from the whole team on designs. (Customers and other non-team persons may participate in walkthroughs.)

It has been suggested that the team may also incorporate, or have available to it, technical writers, administrators, system program experts, and even application area and special skills persons.

*In actual practice, Librarianship seems to range from being the team secretary and control clerk to a full-fledged operating and support system expert.

Although not a feature of the original Chief Programmer Team Concept, it is now recommended that when a project exceeds the capabilities of one or two teams, that a Chief Architect be appointed to determine overall designs of the system and to provide technical direction to the team. If all technical control activities are assigned to this person such as Configuration Management, Program Library Operation, Data Library Operations, Information Control and Quality Assurance, the Chief Architect Team is, in essence, the Program Management Office for the project. This concept assures that there is an administrative head and staff in addition to the technical head of the project.

4. COBRA DANE STANDARD PRACTICES

COBRA DANE set forth standard practices for each of the following standards whose objectives were described previously:

- Elements of programming style
- Program library operations
- Global data control
- Configuration management
- Chief Programmer team

In this section, the specific practices prescribed for each area of standardization will be described followed by an assessment of how closely the practices were observed and the perceived impact upon performance.

4.1 ELEMENTS OF PROGRAMMING STYLE

COBRA DANE had stated standards* for:

- Program commentary
- Indentation and paragraphing
- Naming conventions
- Module construction

4.1.1 Standards and Conventions for Programming Style

4.1.1.1 Commentary

The standard practices for program annotation are given in Table 1. Although not part of the initial standard, a specific requirement to annotate iterations and conditionals was added later. All programs were to be annotated. Changes to controlled programs were to be accompanied by comments giving the reasons for the change and citing the problem report resolved by the change.

*TM-(L)-356/300, "COBRA DANE Programming Practices Manual," SDC, Oct 12, 1973.

Table 1. COBRA DANE Commentary Practices (1 of 2)

A. Identification (PROLOGUE)

- Program name
- One line functional description and/or indication if this is a dummy or partial program for build test purposes.
- Name and phone number of responsible programmer.
- Name of latest update ident. This ident will include the two letter program ID and a two letter MOD starting with AA and working sequentially toward ZZ.

B. Status Information (PROLOGUE)

- Schedule date for code completion.
- Current state of code completion (e.g., none, partial, etc.) and date of status.
- Scheduled date for completion of unit testing.
- Unit testing status and date of status.
- Program under change control (yes/no).
- Scheduled date for completion of integration.
- Integration status and date of status.
- Date of last program update.

C. Descriptive Information (Prologue and Internal)

- Common cards describing program purpose and usage.
- Subroutine calling parameter descriptions.
- *CALL statements obtaining global COMMON blocks.
- Local data definitions and descriptions.
- DIMENSION, and EQUIVALENCE statements.

D. Return shall be made to the main program

EXAMPLE

```
*DECK INLOOP
      SUBROUTINE INLOOP (IN)
C     PROG = J. CODER  PHONE = 617-862-0050
C     INLOOP IS AN INFINITE LOOP FOR EXAMPLE PURPOSES
C     INCL
C     CODING COMPLETION - - APR 75
C     CODE STATUS       - PARTIAL
C     UNIT TEST COMPLETION- MAY 75
C     UNIT TEST STATUS  - UNTESTED
C     INT TEST COMPLETION - DEC 75
C     UNIT TEST STATUS  - UNTESTED
C     LAST PGM UPDATE   - 16 JAN 76
C
```

Table 1. COBRA DANE Commentary Practices (2 of 2)

```
C     INLOOP = INFINITE LOOP PROGRAM IS CALLED WHEN
C     EVERYTHING IS GOING TOO WELL.  IT WILL HANG IN AN
C     INFINITE LOOP UNTIL RTC TIMES THE CALLING PROGRAM OUT
C
C     INPUT PARAMETERS
C     IN - THIS ITEM IS TOTALLY UNNECESSARY
C
  **********. . . .
*CALL CAL
*CALL ROB1
***********. . . .   :
C
C     LOCAL DATA
         TYPE INTEGER ITEMPA; ITEMPB
C             ITEMPA AND ITEMPB ARE TEMPORARY STORAGE
C
C     IF THERE WERE ANY DIMENSION OR EQUIVALENCE
C     STATEMENTS, THEY'D GO HERE
C
C     HOUSEKEEP TEMPORARY STORAGE
      ITEMPA = 0
      ITEMPB = 0
C     LOOP
C
      100IF(IUA) 200,200,300
      200 GO TO 100
      300 GO TO 100
      RETURN
      END
```

4.1.1.2 Indentation and Paragraphing

Programs are to be segmented into small, single function modules. Subordination and ownership will be clearly indicated.

Coding will be clearly indented to show subordination and grouping of the instructions. Executable statements are to be restricted to one instruction per card. More than one data declaration per line of code is permissable. Wherever possible, lengthy statements such as complicated conditional branches should be broken down into a sequence of shorter code. Blank comment cards should be used to block the listing and enhance readability.

4.1.1.3 Naming Conventions

The conventions of CDC FORTRAN EXTENDED shall be followed. Module and data names are to be nmemonic tags up to eight characters in length. Special prefixes and suffixes may be appended but the remainder of each name must be descriptive of its function.

Reserved prefixes for tasks, programs and subprograms were:

- BD - Block Data Programs
- CD - Main task entry point programs

Reserved prefixes for variables for FORTRAN standards were:

- B through H - Real
- O through Z - Real
- I through N - Integer, octal or logical
- G - Real variable in Global Labelled Common
- IG - Integer, octal or logical variable in Global Labelled Common

Using the prefixes avoided declaring the item type in FORTRAN. Type could be declared or redefined.

Reserved suffixes for variables were:

- xF - Base address of an allocated file
- LF - Forward link pointer for a threaded list
- LB - Backward link pointer for a threaded list

Tags within a program are to be numbered sequentially.

Table item tags are to be composed of a two or three character table name followed by a nmemonic item tag. One of the reserved variable prefixes may be appended to define the item type.

4.1.1.4 Module Construction

Programs shall be segmented into relatively short, single function modules. Recommended segment length is one to four pages of executable code (i.e., excluding commentary and data declarations). Segmentation should not be arbitrary but based on such logical grounds as singularity or commonality of functions, operational interaction, and temporal occurrance.

Modules shall have single entry and exit points in so far as is possible. Ownership of submodules shall be clearly indicated. Upon completion of submodule processing, submodules shall return to the main routine, not exit to another module. Submodules may pass information and control among themselves, but may not do so outside their master module. Interfaces (entry and exit conditions and parameters) shall be clearly indicated and defined.

4.1.2 Observation of Style Conventions

Observation of the standard practices proposed for programming style was very good, perhaps best for the mission support programs, somewhat less for the executive programs and post mission processing and least for the simulation programs. The standards were enforced by the Chief Programmers, reviewed and updated by the System Librarian, and polished during integration and test. While not perfect, observance was well above average for known projects.

In forming this opinion source listings for approximately 60 percent of the COBRA DANE program modules were inspected, consisting of some or all modules associated with 50 of the 90 CPCs (Computer Program Components.) Additionally, the survey considered the System Librarian's assessments, Production Specification writer's evaluations, and Integration and Test Team comments. Of particular interest are the evaluations of the Product Specification Writers. In the Fall of 1975, the programmers on the project in Wayland were augmented by a group of approximately ten experienced programmers in Santa Monica who, on the basis of materials furnished from Wayland, wrote the bulk of the Part II Product Specifications. The materials furnished included source deck listings, the Part I Development Specifications and a description of data defined in Global Labelled Common*. The work was performed on a 'minimum' interference basis. That is, the writers were to call the COBRA DANE programmers only if they could not retrieve the information in some other way. Since much of the Development and Preliminary Product Specifications were obsoleted by many changes and redesigns of the system and of the programs, the writers had to rely almost entirely upon the source listings as the basis for producing the final Product Specifications. This afforded a unique opportunity to evaluate the effectiveness of the elements of programming style in contributing to the ease of understanding of programs.

The backgrounds of the Integration and Test Team members who supported the integration and evaluation test phase at Shemya, Alaska, are basically different. These programmers were well indoctrinated in the system having helped produce it, and were largely engaged in maintenance activities, the removal of latent defects and production of minor modifications to accommodate the desires of users. Their evaluations address how well the conventions were observed and their utility in maintaining the system.

*A note of interest: The writers invariably referred to this latter document as 'A COMPOOL Description' although there was no COMPOOL in fact associated with the project - i.e., the data declarations were available only to the Compiler.

Upon delivery of source programs for entry into the controlled Program Library, the System Librarian reviewed the programs for proper annotation, structure and tagging. If the standards were not observed, the Librarian attempted to obtain conformance, largely by rewriting prologues and indicating substandard blocking and naming.

4.1.2.1 Commentary

The System Librarian estimated that 70 percent of the modules were annotated when first submitted to the Program Library for entry into the System Master library.* At the time of the review of the source listings and the preparation of final Part II Specifications, the proportion was nearer 90 percent. At the end of integration and test at Shemya AFB, it is estimated that 95 percent of the modules were commented.

The specification writers who had the best opportunity to evaluate the commentary stated that it ranged from "absolutely nil" to "fantastically good". Overall, the specification writers felt that COBRA DANE Commentary was well above average compared to other projects they had worked on.

Although not an area covered by the proposed standard practices, one deficiency noted in the source listings especially by the specification writers, was the lack of maintenance of annotations for program changes. Every change made to a controlled program had to be covered by commentary cards justifying the change and citing the SPR the changed resolved. (See Section 4.2.). However, the changed processing was not often commented and old annotations were not always purged.

*Library files contained old compilation listings that could have verified this estimate, but the difficult task of researching extensive old listings was not attempted.

4.1.2.2 Indentation and Paragraphing

Proposed standard practices for blocking, indentation of code to show subordination, and use of spacing and outlining to show program structure were almost invariably followed. Except for data declarations, programs were always written one statement per line and conditionals were broken down. The specification writers did not report a single instance where the rules were violated.

4.1.2.3 Naming Conventions

Although there were slight variations in interpretation of the naming rules by different Chief Programmer Teams, the rules for creating program and data tags were quite consistent. Violations of the rules did occur in the later stages of the project when a large number of changes were being made, especially in table items. The pressure to implement changes and corrections was high and, perhaps, the persons making the changes were no longer acquainted with the original design of the tables.

4.1.2.4 Module Construction

Segmentation rules were quite well followed in COBRA DANE. Except for some programs in the executive area, where desire for efficiency may overrule modularization overhead, module size averaged three or four pages of executable code with very few exceptions.

Observation of standards for hierarchical structure and ownership was harder to verify. FORTRAN does not lend itself to showing structure and there were no environment analysis tools, such as a system set-used, that would reveal the structure of the system. The programmers all claimed that ownership rules were followed. Hierarchical structures were specified in designs and accounts were kept of which modules were part of which CPCs.*

*During the later phases of the project, a feature of SDC's project monitor, IMPACT, was being used to regularly report a breakout of the structure of the mission support programs with sizing counts, but the structure was declared and not derived from the Program Library records as were the counts.

Rules for single entrance, single exit, and simple control structures were very closely followed with only minor exceptions. There were some violations of the flow rules (calls outside a program's area of control) that resulted from removal of redundant code to save core as the system became core-critical. Such convolutions were very rare, however.

4.1.3 Impacts of Programming Style

Following the rules for programming style made the programs quite easy to read and understand and made modification and maintenance easier. Deviations from the rules created difficulty in understanding and in program maintenance.

4.1.3.1 Commentary

The specification writers found the commentary of greatest importance in preparing the Final Part II Product Specifications. Unannotated modules or less well annotated modules were harder to flow and to follow and the intent of the module was hard to derive.

The writers also had the Preliminary Part II Specifications to help them, but usually preferred to depend on the source listing for information about the modules they were describing. The Preliminary Part IIs were useful in expanding upon the intent and functional requirements of the modules, but so many changes in designs and requirements had occurred that the previous specifications could not be trusted for program details.

The specification writers reported that annotation of data items was more important than of processing steps in contributing to their understanding of the modules. As programmers they had no difficulty interpreting the logic of a mathematical or logical expression, but could not interpret the intent or meaning, "what it really does," without knowing what the data items meant. One of their most useful documentation aids was a well-

annotated listing of the global data items*. A listing of all tags by class would have also been of some further assistance.

The commenting of flags gave the specification writers difficulty that had not been anticipated in the proposed standards. Several COBRA DANE programs, largely those interfacing with radar controllers and other equipment, would pass or receive long registers of flags or control bits. The entire register was handled and described as a single data element in the Global Labelled Common, but a great deal of processing involved building or parsing the registers. To truly understand the programs, annotation for every flag is required**.

A similar situation is found in program calling parameters. Where the rule for defining these had not been observed, the specification writers had difficulty in evaluating both module process and interactions between modules. In FORTRAN, where the item type declaration is implicit in the item name, a little extra effort is required to list and annotate parameters. Calling parameters also are not normally the sort of items that are globally defined. Hence, if the parameters are not annotated, it is very difficult to tell what meaning is associated with the call. That is, it is difficult to evaluate what the called subroutine has been asked to do.

Subroutine calls themselves also profit from annotation. The subroutine names usually have some mnemonic value and normally there is a list or catalog of available common routines that may be consulted if the function of the subroutine is obscure. However, it is much more convenient if the description is immediately available in the source code.

*The specification writers invariably called this a "COMPOOL listing" although FORTRAN Global Labelled Common is only the first step toward a COMPOOL as reported in Part III of this study.

**In this instance, descriptions of most, but not all, the data in the registers were found in Raytheon equipment manuals.

Since it is somewhat tangential to the main task, maintenance of commentary for correctors and changes is also easily overlooked. The specification writers said that encountering a section of unannotated code was a clear signal of a trouble spot in a program since no commentary meant the program had been amended. If obsolete commentary had not been purged, it could be quite misleading. Most correctors and changes are introduced either during integration and testing or during the Operations and Maintenance phase when the pressure to get the program back into service is very high. In a crisis situation, one tends to concentrate on solving the problem and not upon the niceties of maintaining commentary. Hence, this is a matter requiring some quality assurance attention and perhaps a cleanup pass at the source code after the crisis is over.

Although the System Test Support Team members working at Shemya AFB were much more familiar with the software than were the specification writers (many of the former were Chief or senior programmers from the developmental effort), they still found the commentary of real use in understanding and maintaining the programs. In fact, the programmers reported that if commentary were missing or inadequate they would insert it for their own use in understanding the module's function and in diagnosing its flow. Reportedly, as many as 30 to 100 comment cards could accompany 2 or 3 correctors. Obviously, the commentary was prized as an aid to maintenance.

The impact of extensively commenting their programs upon the productivity of the programmers is not negligible, but does not consume a major portion of their time. Programmers report that producing good commentary does require some thought. Some programmers find it irksome to keep explaining the code when its meaning is clear to them and they want to get on with the job. However, the actual composition on paper (or other media) of the program is only a small part of programming. Most of the time and effort is expended on working out the details and the effort of recording the lines of code and the accompanying commentary is small. If modules are small, the lines of commentary may equal or exceed the lines of code, largely due to the explanatory material in the prologue. Interior annotation should appear

only when data elements are declared, routine calls are made, or a logical code grouping occurs (a conditional, a loop, or a step in an algorithm). The comments should be made as the code is written when the thinking that leads to the code is fresh and familiar.

Commentary is useful in program debugging and testing as well as in later maintenance. If a programmer is handling several modules, as is usually the case, he may easily forget prior thinking or confuse one module's contents with another. For some of the programs in COBRA DANE, the programmers were so familiar with the applications that writing the modules was almost second nature, but the value of commentary for maintenance work was universally recognized.

4.1.3.2 Indentation and Paragraphing

Blocking and formatting the code made it easier to read. The specification writers reported that they never had any difficulties perceiving the structure or internal subordination of the code. Blank spaces, punctuation and outlining with asterisks emphasized the logical blocks and broke up the code into easily grasped segments. Breaking down conditionals and restricting code to one statement per line revealed internal structure and made following the logic easy.

Programmers reported that indentation and paragraphing took little effort. After a while it became almost automatic and required little thought.

4.1.3.3 Naming Conventions

Following the naming conventions contributed to the understandability of the items. It helped locate programs and data and gave some idea of data type and usage. However, unless very long, tags have limited mnemonic value and are most useful if accompanied by commentary.

When the tags did not conform to the naming conventions, as occasionally happened when the burden of changes and production pressure grew, difficulties resulted in a lack of understanding. This was especially so in tracing

table items to their source when the table identification was omitted from the tags.* When the conventions were followed, the global data definitions did not have to be referenced so often since better inferences could be drawn concerning the source and meaning of the items.

Following conventions does require a little more care in programming, but does not consume a great deal of time. In the case of common data that is used by several programs, the care readily pays for itself in increased understanding.

4.1.3.4 Module Construction

Modularization helped to make the programs readable. It also achieved the objectives of making modification and maintenance easier. The simplicity of function and the brevity of the COBRA DANE modules made interpretation very easy.** Single entrance and exit rules, and rules for not calling outside the parent program, also made the flow easy to follow.

The System Test Support Team reported that the modularization aided maintenance a great deal. In diagnosing problems it helps localize and pinpoint errors by restricting the area of search. It breaks the program into small pieces that are easy to understand especially by a person unfamiliar with a program as is likely to be the case in the operational maintenance situation. If modularization is done correctly, interfaces are well-defined and it is clear what one module passes to another. In making changes or corrections, if neither inputs to, nor outputs from, a module are changed, the change is easily checked and understood, and should not impact the rest of the system. Modularization localizes the impacts of corrections. A whole module may be replaced by another without undue

*If an alphabetical listing of names such as that produced by SCF COMPOOL had been available this difficulty would have been alleviated. The FORTRAN compiler alphabetical listings were not exceptionally helpful and are not normally available as part of a source listing.

**Short modules were encouraged by the COBRA DANE requirement that programs be interruptible at intervals of milliseconds to monitor missile launch surveillance. There were, however, a few long modules.

labor. The modules reported to give the most trouble during final integration were one or two rather large, monolithic programs. The programs were reported to be well written, but the greater complexity and volume were harder to handle and understand. A greater amount of information had to be kept in mind when tracing through the code, for instance. Debugging dumps were always large, requiring more searching to diagnose errors and trace logic.

As reported, efforts to reduce code did lead to structural violations - largely calls to subroutines in other modules and some modules combined to conserve code. The collapsed code was more complex logically and more difficult to understand. The calls outside the parent area of control not increased the difficulty of following the flow. In sum, the redesign to accommodate core limitations disrupted what had been a very clean, tight design.

Difficulty in following the flow was also caused by the FORTRAN IF statements not falling through into the main sequence. If the programmer was not aware of this idiosyncracy of FORTRAN, the flow jumped back and forth in a sometimes confusing fashion.

Complexity in flows resulted in a few cases where there were multiple entrances and exits in programs. This is partly due to the obscurity of secondary entrances (one wastes time looking for a module rather than an entrance or trying to find the module the entrance is in) but partly it is due to complications in the information flow. The writers recommended that modules that were umbrellas for several entrances either be broken apart or that the module have a single interface to which parametric values determine the case to follow rather than multiple entries.

The specification writers frequently indicated that they needed more structural information on the system to do a good specification job. Although top level structure was available, internal structure could be obscure and FORTRAN does not represent ownership well. The writers would have liked a

system set-used matrix like that normally available from a COMPOOL system. Modularization for COBRA DANE was quite extensive and, although a module was immediately easy to read and understand, it was hard to grasp the larger picture if one did not know the relationship of that one miniscule module to its parent and sibling modules.

The COBRA DANE modularization was effective in increasing the ease of modification. When the system had to be redesigned, flows and interfaces had to be redefined and adapted to the new design, but many modules could be moved almost intact into the new configuration. Small, self-contained modules with single, well-defined interfaces were noticeably easier to modify than conversion of more monolithic code would have been. While rewriting code is always easier than the original creation of it, modularization mitigated the worst of an onerous conversion job.

4.1.3.5 Summary

In general, it may be concluded that the rules for programming style were effective in making the programs easier to understand and to maintain. Specification writers' complaints centered on those instances where commentary was missing and where structural information was inadequate. Although the Integration and Test Team members did not stress commentary, they supplied it for their own use when it was missing. The programs that gave trouble during integration were those that were monolithic, not because they were badly written but because their length and complexity made them difficult to understand and difficult to handle. Multiple entry points did contribute to confusion. Data annotations were judged more important than process annotations. COBRA DANE software was technically and structurally complex, much complicated by the interruptability and reentrancy requirements. Its modularity makes its maintenance feasible.

4.2 PROGRAM LIBRARY CONTROL

The COBRA DANE project had established standards* for the Program Library including definition of responsibilities for the System Librarian, for interacting with the library, for controlling change, and for quality assurance. The composition and status of seven builds were processed through the library during the project.

4.2.1 Program Library Operations Standards

The standard practices for Program Library operations included the establishment of a System Librarian with specified responsibilities and the definition of rules for interacting with the library.

The System Librarian's responsibilities included:

- Generating, updating and maintaining Program Libraries
- Maintaining a software problem status data base
- Monitoring update decks submitted by programmers
- Producing computer generated software and software problem status reports.

Interaction rules specified that:

- The System Librarian shall have sole responsibility for maintaining the Program Library. Individual programmers are forbidden, under all circumstances, to directly modify a Test Bed Program Library.
- The System Librarian shall establish a dummy deck (i.e., a program stub) for every anticipated program. The dummy deck shall include the program ident, an end, and such control information as program status and significant developmental due dates.

*TM-(L)-356/300, COBRA DANE Programming Practices Manual, SDC, Oct. 12, 1973, and assorted internal notes.

- The programmers shall work from the Test Bed Program Library while coding, all code to be turned over to the System Librarian upon completion as an UPDATE deck.
- Although the System Librarian may inspect program decks to ensure that programming practices are observed, it is the responsibility of the Chief Programmer to see that the practices are enforced.
- The System Librarian will perform no quality assurance other than ensuring that control information is correct and that other modules are not inadvertently changed.
- Once program checkout is complete, the program will be placed under deck control. That is, no changes may be made to a controlled program without specific authorization.
- For programs under control:
  - A list of all known errors and omissions will be kept.
  - All code must be certified as tested (but not necessarily all paths).
  - Test drivers, if necessary, shall be supplied by the programmer.
  - The System Librarian will update control information (status, due dates), resequence the program, compile the code, and report errors noted by the compiler.
- To modify controlled programs:
  - Submitted changes must be identified by deck control numbers.
  - Changes must be accompanied by short explanations of what the changes do.
  - Any errors corrected must be identified.
  - Relations to other previous or anticipated changes must be noted.

- Programmers shall prepare properly commented and structured program modules and shall provide proper control cards for library operations.

Library control procedures* defined a two stage interaction: A Unit Test phase prior to placing a module under control and an Integration/Verification phase after the module was placed in the Program Library. Programs coded and unit tested were submitted to the System Librarian who gave them an "uncontrolled" status and identification and placed them in the Upfile. The modules were kept in Uncontrolled status in the Upfile until the CCB (Configuration Control Board) authorized inclusion in an UPDATE cycle. While still "Uncontrolled," changes could be made to the module without authorization. Once the Update cycle was run, the problem shifted to "Controlled" status and all changes had to be authorized. Any problems uncovered were described in an SPR. The Chief Programmer responsible for the module assigned a control number and initially assessed the urgency of the problem. A library identification number was obtained from the librarian and the change coded and tested. When completed the change was added to the Upfile, but was not added to the library until CCB approval was given. The CCB reviewed problem status weekly. A status of "Open" was given to problems filed and being worked on or "Deferred" if filed, but awaiting manpower. A status of Resolved indicated that a solution had been found but was not yet verified and approved by the CCB. A status of "Closed" was assigned to rejected SPRs, to SPRs combined with others and to those settled by allied changes or later redesign. Changes could remain in the Update file for some time pending CCB approval if their test status was doubtful or if they depended on other changes to be valid. The CCB could also adjust the priorities of changes assigned by the Chief Programmers. Problem Status Reports were limited to those SPRs Open, Deferred, Resolved and Closed by the current update cycle. Build and modification status accounting was required.

*N-LX-771/100, "COBRA DANE Change Procedures", SDC, 24 March 1975.

4.2.2 Observation of Library Standards

The standard practices for the Program Library were instituted and were observed very well in most particulars. A System Librarian was appointed who stayed with the project throughout. The library update cycle was observed regularly. All proposed builds were created and tested as required. All changes to the System Master or controlled portion of the library were very closely controlled.

However, a complete set of program stubs was not created, the CPTs created their own copies of the library rather than working out of the Test or System Master, and the rules for quality assurance were relaxed as the pressure to deliver successive builds became intense.

4.2.2.1 System Librarian

There was a single System Librarian for COBRA DANE rather than Team Librarians. The librarian was an accomplished programmer charged with generating report programs and library utilities as well as maintaining and controlling the Program Library. Consequently, his skill level was well above the clerical level often used to support CPTs and he accepted a broader range of responsibilities. He was capable of performing quality assurance inspections of programs submitted to the library and of managing the library.

4.2.2.2 Library Maintenance and Control

Library operations were quite regular and controlled. Prior to the release of the system to the Integration and Validation Test Phase at Shemya AFB, 36 update cycles of the Program Library were completed. (Update cycles refer to the System Master controlled library; changes were made to the developmental or Upfile portion of the library on a daily basis.) Updates were continued on the delivered system during integration and validation, resulting in nearly as many update cycles during that period as during Full-Scale Development. The update cycle varied somewhat from the estimated once-a-week frequency, being somewhat longer initially but somewhat less than a week during the peak effort of build deliveries and slacking off again during the CONUS test bed integration and qualification period.

The timing and content of library updates were controlled by the internal Configuration Control Board. On the basis of its instructions, the System Librarian composed the update instructions and maintained the System Master library. Control and accounting for the System Master contents was very close. Each update resulted in a new library report with updated status records so the internal CCB always had current information on the state of the developing system.

There was no set leadtime for submission of materials to be included in an update. Submitted mods to the library were to be inspected, compiled and checked out prior to loading them into the library. However, as the pressure to produce became acute, the library lead-time became very short. The decision on which modules were ready to load rested largely with the Chief Programmers who comprised the bulk of the CCB. In a pressure situation, whenever there is a choice between risking delivery of a questionable product and the onus of missing a schedule, production-oriented people almost always opt for taking the risk. This undoubtedly happened in COBRA DANE for programmers reported that programs that were not adequately unit tested appeared in the System Master library from time to time.

The rule that programmers were to work out of the central library in developing and testing their programs was observed well until tests involving interactions with other program modules were underway. With more than one library update occurring weekly, which resulted in many corrections and changes being made to the master, plus some faulty programs being loaded, the programmers had to work in an unstable environment. That is, a feature checked out one day might not work the next due to changes in other modules. It was deemed less frustrating to work around known errors than to contend with unknown errors and the instability of a constantly changing environment. The programmers created for themselves copies of the library that they could control and maintain. In essense, instead of a single central library, each Chief Programmer Team had its own.

4.2.2.3 Change Control

Control over changes to the System Master Library was very tight. All changes to the library had to be approved by the CCB and no change was considered that did not reference an SPR or a requirements change directed by the external CCB. Hence, all errors noted in a controlled program resulted in an SPR being filed that was reviewed closely by the CCB. Priorities were assigned and corrections were directed. When the changes were programmed and checked out, the CCB identified the update cycle in which they were to be included. In actual practice, errors were often reported informally to the responsible programmer who would code and check out correctors and submit the SPR and change package to the CCB simultaneously. Depending upon the priority of the change, an update might be immediate, scheduled in the next update, or deferred to a more convenient cycle.

Library records of all mods were kept filed in the update and in the System Master so that strict accounting was kept on the status of all submitted changes.

4.2.2.4 Quality Assurance

For the most part, all modules submitted for inclusion in the System Master were inspected by the System Librarian for conformance with the programming standard practices. Library instructions submitted with the modules were evaluated to be sure no conflict with existing modules would result. The submitted modules were certified as being unit tested by the programmer. If the librarian discovered deviations from the rules, he sought corrections, contacting programmers for input and clarification. The librarian supplied as much additional commentary as he could and produced many of the library instructions personally.

4.2.2.5 Builds

Seven builds were created and delivered during the developmental period. Build content was also closely controlled. Build functional content was determined by the external CCB, but the internal CCB could include other non-deliverable modules in the System Master containing the deliverable build

functions. All build functions delivered were thoroughly tested and demonstrated. Non-deliverable functions might be tested, but were not officially demonstrated.

Changing requirements and the vagaries of development both of programs and equipment impact the content of builds but despite these, only four build plans were issued, attesting to the degree of control and conformance to the plans.

4.2.3 Program Library Impacts

There seems no doubt that the Program Library met its main objectives of providing close management control over the developing product and of improving programming efficiency and convenience. For instance, although the rule that all programming should be done out of the central library was abrogated due to the instability of the System Master, the CPT-specific libraries served the same purpose and achieved the stability required for efficient program checkout. COBRA DANE management was thoroughly convinced of the effectiveness of Program Libraries, declaring that the difficulties facing the project would have been insurmountable without them.

4.2.3.1 System Librarian

The effectiveness of the Program Librarian has a definite impact upon program quality and the effectiveness of Program Library operations. To be librarian to 50 strong and capable programmers and to enforce conformance to programming style conventions and library interaction rules requires a very strong, very able person. In the case of COBRA DANE, the single librarian became overloaded during peak periods and was not able to perform all his quality assurance tasks as effectively as he might. In some instances, updates were not completed exactly on time, partially due to limitations on computer time. The librarian's inability to convince the Chief Programmers and the CCB that enforcement of quality assurance and library interaction rules is necessary for effective library operations is evidence that a very forceful and aggressive person is required to carry System Librarian responsibilities.

The COBRA DANE System Librarian was an accomplished and able person. However, in view of the stressfulness of the situation, an even more forceful and aggressive individual seems to be required.

4.2.3.2 Library Maintenance and Control

The regularity of updates and the closeness of control was quite beneficial in promoting a smooth programming operation. While there were some deviations from the stated standard practices, these did not greatly detract from the effectiveness of the library operation.

Since manual records and control was exercised by the internal CCB over library content, not having a complete set of program stubs in the library did not detract from management effectiveness. By the time integration activities arose for a build, all required modules would be present in sufficient detail to support testing. Having the Program Library listing would have helped, but it was not essential.

Although the creation of CPT-specific libraries solved the problem of the instability of the System Master to a fair degree, it did exacerbate problems with storage. The growth of the system not only led to its exceeding core capacity, but the saving of copies and versions of the library used up the available disc and tape storage. The problem was resolved by stepping up storage media policing activities to purge obsolete and redundant files (another task for the System Librarian). The amount of delay resulting from not having storage immediately available was probably very minor, but some additional work was added to the librarian's task.

4.2.3.3 Change Control

Change control operations were very effective in organizing and controlling that very crucial aspect of program development. Regular issuance of SPR status reports Open and Closed gave high visibility to the amount of changes and made sure no reported errors were lost or remained unresolved. Considering the number of changes in requirements augmented by the amount of work occasioned by the redesign of portions of the system, undoubtedly a great many more lines

of code flowed through the sytem than the over 300,000 ultimately delivered. Control of that volume of activity represents no small task.

During the Full-Scale Development period, nearly 2500 SPRs and 2150 corrector decks were processed at Wayland and, during the integration and validation period, approximately 1100 correctors were filed at Shemya AFB. There were a few SPRs still outstanding when ESD conditionally accepted the COBRA DANE system from Raytheon at the end of 1976.* These were largely associated with the handling of "free space" (working common) and recovery after failures, and all were expected to be resolved in the clean-up period.

Although not elegant, the manual problem status reports worked smoothly. Such minor confusions as the same problem being reported from more than one section or an unresolved SPR being inadvertently dropped from the report were cleared up promptly.

4.2.3.4 Quality Assurance

The relaxation of quality assurance standards during peak operations did impact the efficiency of operations. More errors did slip into the System Master and these, as well as the instability occasioned by frequent updating changes and corrections, gave programmers difficulty in checking out their programs. In essense, time was spent chasing bugs in other people's programs as well as one's own. Uncertainty over where a fault may lie is irksome, so it may be that more time was perceived as lost than was actually spent. Further, since the Chief Programmers and the CCB were taking a calculated risk in relaxing the rules, the risks may have paid off in meeting schedules although some inadequately checked modules were entered in the library.

*An interesting, if trivial, phenomenon was noted in the pattern of problem reporting and resolution. In any month, approximately the same number of problems would be resolved as reported (around 200-250) so that the number of outstanding problems remained relatively constant (around 80). Activity peaked just before a build delivery and dropped back after delivery. No particular meaning can be placed on the phenomenon and perhaps it is what one would expect.

4.2.3.5 Builds

The succession of builds helped to organize and motivate the software development. The builds represented short range goals that provided a series of highly visible milestones to management and the customer. The high pressure and difficulty in meeting the milestones was more a function of the underestimation of the amount of work required and the number of changes accommodated than any inherent pressure to make schedules, although that force may have also been present.

For most of the course of the project, a delivered build was not held separate from the current System Master. That is, the deliverable build functions were verified in the current version of the System Master which could then change with the next library update. This led to instability in the builds which were being used to check out Raytheon equipment. The changing library prevented checks being rerun without adaptations to changes and the impacts of change were not always communicated or understood. Also, the equipment checks sometimes tried to use non-certified functions that rendered the system inoperative. Later in the project, protection over the integrity of delivered builds was established to prevent these problems.

4.2.3.6 Summary

In general, the Program Library operation met its objectives. It provided regular and organized support for programming, established control over program changes, and provided the desired visibility into the developing system. On the basis of the COBRA DANE experience, it would seem that if the System Librarian approach, as opposed to Team Librarian approach, is to be used, a very strong and able person is required with complete backing from project management. Stability and reliability are desirable in the library if it is to serve effectively for integration of programs. This places a premium upon the enforcement of quality assurance measures. In COBRA DANE, the System Master versions had to be built, integrated and tested to meet the build deliveries. Perhaps less frequent updates would have yielded more time for inspection and unit testing, but that is not certain. Version protection, both for delivered builds and developmental models, is needed to provide a

stable environment for testing and debugging. Although the SCOPE UPDATE function satisfied Program Library requirements after a fashion, the project could have profited from a more powerful Program Library support program. UPDATE was alleged to be inefficient in computer utilization and not powerful enough for accounting and reporting operations. (For instance, there was no symbolic reference capability for generating new reports.) Overall, however, the Program Library performed well for the project.

4.3 GLOBAL DATA CONTROL

The proposed standards for global data control were very similar to those used with most COMPOOL-sensitive systems at SDC and paralleled the standard practices specified for Program Library operations. Global data were defined as global labelled common for compiling purposes and maintained in their own section of the Update files. Naming and commentary practices are described in Section 4.1. Standard practices proposed for the Data Base Library covered:

- Responsibilities of the Data Base Coordinator
- Data base library maintenance and control
- Data base library change control
- Quality assurance
- Resource utilization

4.3.1 Standard Practices for Data Control

Standard practices for data control operations stated:

- Global data naming and definition of the global data base contents will be coordinated through a Data Base Coordinator.
- The Data Base Coordinator will be responsible for naming global data base entities, for naming programs, for providing initial values of constants in the global data base and for maintaining Global Data Base files.
- The Data Base Coordinator will be responsible for coordinating usage of the data manager and the system resources (i.e., tapes, discs, printer, etc.) used in the software released to Raytheon/ Air Force.

- Utilization of the Data Manager and computer resources for test purposes will be left to the discretion of individual programmers.
- Naming conventions will be established that (1) differentiate between local, global and test data and (2) that provide structural information on global names. (See Section 4.1 for established conventions.)
- Individual programmers are responsible for adhering to these conventions for local and test code.
- The Data Base Coordinator will ensure that these conventions are followed for global names.
- The Data Base Coordinator will issue periodic listings of the global data base contents.
- The global data base will contain all data for communication between all deliverable programs, i.e., all COMMON block data plus all information transferred to or from peripherals.
- Individual programmers will be responsible for defining sizes, lengths, data types and purpose to the Data Base Coordinator who will assign names and prepare and load data definition cards into the global data base.
- The Data Base Coordinator will:
  - Ensure that the initial contents of global data blocks are properly initiated.
  - Ensure that all necessary physical and mathematical constants are set up. [All constants in internal data store shall be metric unit (meters, kilograms, etc.).]
  - Design and implement data base initialization software.
  - Develop and disseminate data base interaction and initialization procedures.
- Programmers shall not declare global data internally to a program.

- Programmers shall use the physical and mathematical constants (such as pi and metric-to-English-system conversion factors) provided by the global data base and shall not hard code such values into their programs.

4.3.2 Observance of Data Control Standard Practices

Although the standard practices were established, they were not enforced as stringently as those for the Program Library.

4.3.2.1 Data Base Coordinator

The Data Base Coordinator's responsibilities were more of a clerical nature than were the Program Librarian's. He received the data definitions, made up the input decks with control cards, and loaded the decks into FORTRAN labelled common blocks.

4.3.2.2 Maintenance and Control

The Data Base Library was maintained on the same cycle as the Program Library. Listings of the global Labelled Common Blocks (LCB) were made regularly.

4.3.2.3 Data Change Control

The configuration control procedures applied to programs operated for data control as well. The LCB were treated just like program modules, but the control procedures were not so tightly enforced. Each programmer declared his own data blocks. Initially, individual programmers were permitted to submit changes as they felt necessary, but later the Chief Programmers each controlled and coordinated those LCB common to their functional areas. SPRs filed against programs frequently resulted in data changes, but SPRs were not issued specifically against LCB and changes to data definitions were not required to reference a justifying request for change or correction.

4.3.2.4 Quality Assurance

The Chief Programmers and their subordinates observed data naming and data structure conventions very well. Commentary rules were not as well observed as they were for programs, but some LCB were extensively annotated. A special task was undertaken near the end of Full-Scale Development to annotate the Data Base Library that resulted in excellent prologues for the LCB and annotations for the items.

Initially, the Data Base Coordinator inspected inputs for clerical accuracy, but redundancy and structural inefficiencies were controlled by the Chief Programmers. Since the LCB was largely local to a functional area, once the Chief Programmers adopted control procedures, a satisfactory level of quality assurance was attained. However, so long as individual programmers were permitted to introduce data changes without coordination, errors and incompatibilities arose.

4.3.2.5 Resource Utilization

Initially the proposed procedures for the control and accounting of computer resources were not instituted. However, control procedures were begun when

it became apparent that the system was running out of storage and channel capacity and cycle time. A report of the storage budget was created using a FORTRAN routine to scan the Program Library and calculate program and data element sizes. The counts were converted into inputs to SDC's project monitor, IMPACT, to which the system structure had already been described. IMPACT would then calculate and print out counts and subtotals for system elements. (See Figure 1 for a sample output of this listing.)

4.3.3 Impact of Data Control

Using Global Labelled Common blocks provided a measure of control over system data, but not to the extent that a full COMPOOL-sensitive system would have. (See Part III of this report for a complete description of COMPOOL-oriented programming support systems.) Additionally, the proposed standard practices were not applied as strongly as was originally recommended. Consequently, both project productivity and product quality were impacted to a degree.

4.3.3.1 Data Base Coordinator

Not having a strong data base coordinator whose authority and responsibility was equivalent to the Program Librarian's meant that the opportunity for redundant definitions and data element inefficiencies was increased. Although not an overriding problem, not having the Data Base Coordinator also responsible for the enforcement of standards and the coordination of data changes meant that programmer time was lost diagnosing and adapting to uncoordinated changes.

4.3.3.2 Maintenance and Control

Strong support for system data control procedures was expressed by both SDC programmers and Raytheon engineering personnel. Although easy access to the data definitions and frequent updates made programming easier in one sense, short update leadtimes and rapid reviews had some impacts in terms of errors and redundancies. Redundant definitions did not constitute a big problem but did occur in all functional areas causing some redefinition and recompilations.

DATE: 10/16/74, TIME: 19:04:22 QUANTIFIED ATTRIBUTE SUMMARY REPORT FORM

CONFIGURATION ITEM	UPDATED	PROGRAM DATA COMMON+BLANK	CALCULATED PROGRAM SIZE	GIVEN SIZE OF PROGRAM
TOTAL TO MCT		3038.00	378.00	2000.00
RERR	10/05/74			374.00
RFSCHED	10/05/74	711.00	118.00	
RESCFCB	10/05/74	335.00	40.00	
GETSPAC	10/05/74	569.00	58.00	
CDRERR	10/05/74	221.00	158.00	
		1836.00	374.00	
TOTAL TO RERR		1836.00	374.00	374.00
RRPL	10/05/74			1364.00
WSRP	10/05/74	71.00	36.00	
VRP	10/05/74	536.00	481.00	
UVBEAM	10/05/74	1899.00	130.00	
TSRP	10/05/74			
TRP	10/05/74	797.00	198.00	
TIRP	10/05/74	248.00	72.00	
SRP	10/05/74	560.00	301.00	
RCSBEAM	10/05/74	1924.00	82.00	
NSRP	10/05/74			
CRP	10/05/74	1899.00	20.00	
CDRRPL	10/05/74	708.00	160.00	
SUBTOTAL		6642.00	1480.00	
TOTAL TO RRPL		8642.00	1480.00	1364.00
RRPH	10/05/74			20.00
CDRRPH	10/05/74	512.00	20.00	
SUBTOTAL		512.00	20.00	
TOTAL TO RRPH		512.00	20.00	20.00
RRPI	10/05/74			61.00
CDRRPI	10/05/74	2217.00	65.00	
SUBTOTAL		2217.00	65.00	

DATE: 10/16/74, TIME: 19:04:27 QUANTIFIED ATTRIBUTE SUMMARY REPORT FORM

Figure 1. An IMPACT Resource Utilization Report

4.3.3.3 Change Control

The temporary incompatibilities created by uncoordinated changes to data elements were irksome if not frequent. Both SDC programmers and Raytheon engineers reported program crashes brought about by these. That is, a checked out program or a delivered build would fail after a period of successful operation. Usually, the failure would be found to be due to a change in an LCB. However, data changes tended to ripple through the system, impacting in unexpected places. These unexpected failures were reduced when closer control of the system data was instituted. Frequent data changes did not greatly complicate Program Library updates, but cannot be separated from other program modification actions.

4.3.3.4 Quality Control

As indicated in Section 4.1, data naming and structuring standards helped make the programs easier to read and to diagnose. Where the standard practices were not followed, some difficulty in tracing the item to its source was encountered. Data definition errors were usually filtered out during compilation and debugging and did not tend to show up in the controlled library.

4.3.3.5 Resource Utilization

Once instituted, resource accounting procedures kept track of core usage and time quite adequately. While maintaining utilization accounts did not prevent system growth, it made the project aware of exceeding the budget and encouraged economies. Redesigns, if necessary, could be accomplished in a more orderly fashion.

4.3.3.6 Summary

While COBRA DANE profitted to a certain degree in greater ease and efficiency of programming and in system data management, it did not realize the full potential of such a system. Close control over data library content and access, data element changes and resource utilization was not established until well into the project. Before the controls were instituted, data defi-

nition errors and uncoordinated changes caused program failures that required quite a bit of programmer time to diagnose. Integration of software and checkout of equipment was made more difficult. Early warnings of computer resource budgets being exceeded was also initially inadequate. Having as close control over system data as was exercised over program modules would probably have prevented or reduced much of the difficulty encountered.

4.4 CONFIGURATION MANAGEMENT

Standard practices for configuration management for COBRA DANE were proposed in TM-(L)-352/300/01, "Configuration Management Plan." This plan set forth a full set of configuration management procedures in keeping with the provisions of MIL-STD-483 and AFSCM/AFLCM 375. However, these practices were never effectively implemented. As the prime contractor, Raytheon was responsible for providing configuration status reports to the Air Force. Consequently, SDC contributed to these reports but was not under contract to maintain and publish the full set of configuration management accounts normally required for a major software development effort. Nevertheless, despite the failure to implement a formal configuration management system, COBRA DANE did have established configuration management practices at both the total system and the software segment levels.

4.4.1 COBRA DANE Configuration Management Practices

The COBRA DANE project had two configuration control boards, a Joint Configuration Control Board that handled Engineering Change Proposals (ECPs) and interface problems and an internal Configuration Control Board that resolved SPRs and software design matters.

The Joint Configuration Control Board (JCCB) was responsible for:

- Evaluation of all ECPs
- Determination of build contents
- Resolution of interface problems
- Major configuration decisions
- Issuance of the Configuration Status Report
- Issuance of JCCB minutes and directives

The Internal Configuration Control Board (ICCB) was responsible for:

- Evaluation of all SPRs
- Referring external interface and major development problems to the JCCB
- Control of Program Library contents
- Resolution of design problems
- Issuance of software status reports
- Maintenance of SPR accounts and reports
- Issuance of ICCB minutes

In support of configuration identification, documentation followed the procedures specified in MIL-STD-483, plus corporate format standards and ANSI standard flowchart symbology. However, the Change Page/Specification Change Notice procedure in the proposed standard was not instituted. The Program Development Plan provided a documentation tree of the planned and deliverable documents for the project.

The quality assurance procedures to evaluate configuration content at the JCCB level included:

- Preliminary Design Review (PDR)
- Critical Design Reviews (CDR)
- Functional Configuration Audit (FCA)
- Physical Configuration Audit (PCA)
- Preliminary and Formal Qualification Tests (PQT and FQT)

At the ICCB level, quality assurance procedures included:

- Design walkthroughs
- Program Library inspections

The JCCB was chaired by Raytheon and included members from both Raytheon and SDC. The ICCB was chaired by the technical director of the project and included the section supervisors (i.e., the Chief Programmers and the Software Test Director) as members.

The principal task of the ICCB was the control of changes to the Program Library. The configuration control procedure included:

- Submission of all SPRs to the CCB for consideration
- Evaluation of each SPR
- Determining the disposition of the SPR
- If a program change (error correction) was approved, establishing priorities and schedule target dates for the change
- Directing its inclusion in a specific library update, i.e., determining the content of each update cycle.

The JCCB met on a demand basis. The ICCB met weekly with occasional skips. Both CCB groups participated in the design walkthroughs.

4.4.2 Observation of Configuration Management Practices

Both the JCCB and the ICCB met regularly with prepared agenda and issued CCB Minutes. All planned specification documents were issued and all specified reviews and audits were held. The JCCB made major decisions on whether or not major features would be implemented in the system, defined Build contents and assumed responsibility for interface control. The ICCB exercised close control over the design configuration, the Program Library, SPRs and program changes.

4.4.2.1 Configuration Identification

All the planned specification documents were issued. Part I Computer Program Development Specifications were prepared for the mission support and executive and utility CPCIs. Preliminary Part II Computer Program Product Specifi-

cations were also produced on a CPC level. Test Plans and Test Procedures were also produced. Final Part II Computer Program Product Specifications were written, as described previously, near the end of the project.

The Build Plans were important documents in defining the functional contents of the successive partial deliveries of the system. The Build Plans, four in all, grew in detail and specifically as system versions were released. The Build Plan operated as a major control document for the system as it evolved.

Except for the major redesign of the Mission Support Program, the overall design configuration presented in the Preliminary Part II Specifications survived for the length of the project. There were many changes within the CPC-level modules, but departures from the overall design were not permitted.

As indicated in Section 4.2, the Program Library representation of the system was quite closely controlled. The proposed representation of the system via program stubs was only partially instituted, but manual records were kept of intended library content.

4.4.2.2. Configuration Authentication

The Joint Configuration Control Board exercised control over the functional content of the system and made all major decisions concerning the inclusion of features in the system and in defining interfaces with equipment and external systems. However, the Part I Development Specification that defined the functional requirements for the software was never baselined. For the bulk of the project, the Part I was in a regular state of revision, resulting in nine versions reflecting major and minor changes. While the Air Force was apparently controlling the total system and Raytheon's performance from the System Specification level, no solid basis for governing the software configuration at the Development Specification level was reached.

All the proposed reviews and audits of the software were held. However, the PDR and CDRs were scheduled four months and eight months into the project respectively and before Part I and Part II Specification delivery. (No SDR

was scheduled or held.) At best, the reviews were made on the basis of presentations and draft materials and did not result in approval or authentication of a baselined specification.

The design of program modules was controlled and approved by the ICCB. Design walkthroughs were held that were also attended by members of the JCCB. Initially, the walkthroughs were quite thorough but became somewhat more cursory as the developmental pressure increased. Before all programs were critiqued, reduced interest and time pressure caused the walkthoughs to be abandoned. The bulk of the module review and approval responsibility then fell on the Chief Programmers, both as the heads of their teams and as members of the ICCB.

The ICCB decided the contents of the controlled or System Master versions of the Program Library within the dictates of the Build Plans. It decided when a module was ready to be brought under control and scheduled the modules and changes to them into the appropriate Update cycles of the library. Although as reported in Section 4.2 some risks were taken in scheduling potentially unreliable modules into the library, exact knowledge and close control of the current contents of the Program Library existed at all times.

4.4.2.3 Configuration Control

The JCCB met regularly to consider proposed functional and interface changes. The ICCB met regularly to consider proposed changes to software designs and controlled modules. Both CCBs issued regular minutes of their meetings.

Although the formal ECP (Engineering Change Proposal) process proposed by the initial Configuration Management Plan was not implemented, all no-cost functional changes were directed in the JCCB minutes. Cost impacting changes were followed up by SDC letters verifying analyses and impacts and requested reimbursement. However, since the Part I Development Specification had not been baselined and was still open to interpretation and clarification, a great many changes were approved on the basis of informal or JCCB request until well into the project. After the crush to produce became crucial and any change threatened slippage and overrun, more formal change procedures were adopted.

The ICCB controlled changes to the Program Library very closely. All problems and errors encountered in controlled programs were reported on SPRs; all changes to modules had to be supported by an SPR citation and be approved by the ICCB.

Maintenance of the current configuration identification reflected in the specification documents was accomplished by the release of revised versions of the documents. (The proposed formal SCN/Change Page process was not instituted.) The size of this task is impressive since well over 10,000 pages* of software-related documents were produced, some in more than two versions.

4.4.2.4 Configuration Accounting

Although formal configuration and change status reports were not issued, regular configuration status reports were made to Raytheon and discussed at JCCB meetings.

Internal configuration accounting was quite strictly observed. Program Library listings were prepared for each library update. A log was kept of all SPRs and a listing for all open and just closed SPRs was issued with each update. The ICCB also issued Update Content directives. Listings of module changes in both the System Master and Upfile were made.

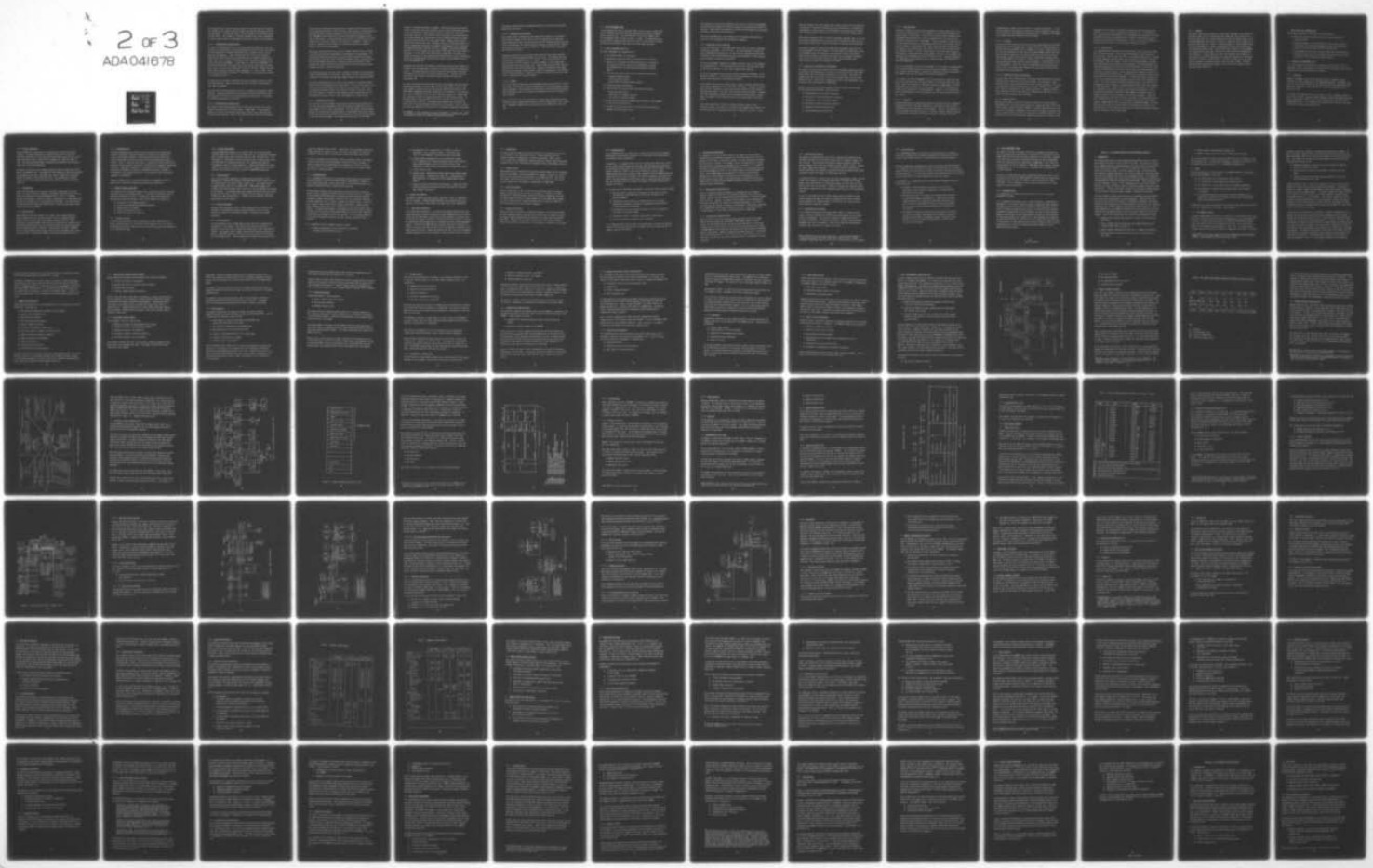
4.4.3 Impacts of Configuration Management

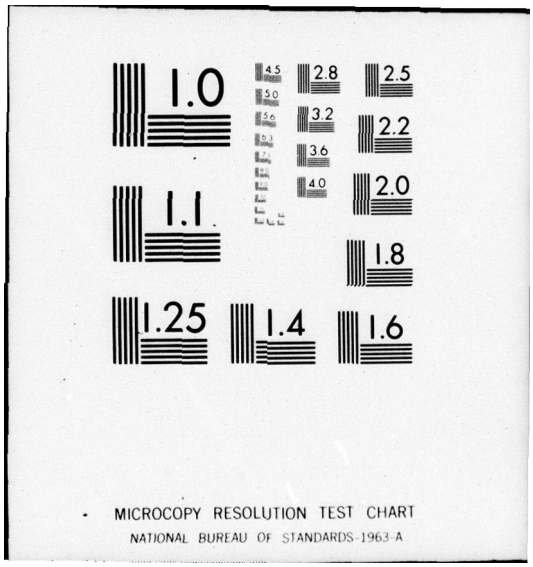
The impacts of not having rigorous configuration management procedures can pervade all aspects of software development. While COBRA DANE was far from having no configuration management, the fact that there were neither formal

*Final revision of specifications was not complete as of the end of this study. The Part I Specification exceeded 400 pages, Part II 8000, Test Procedures 1500, Test Reports 500, Standards, Plans and Manuals 400. Informal notes for coordination and design purposes resulted in an unknown but substantial number of pages. Processing 3200 SPRs and program changes also resulted in at least twice that many pages of written materials. Almost all documents were at least published in draft and final form. Part I Specification had several versions and Final Part II Specification had two or more.

AD-A041 678 SYSTEM DEVELOPMENT CORP SANTA MONICA CALIF F/G 9/2
AN INVESTIGATION OF PROGRAMMING PRACTICES IN SELECTED AIR FORCE--ETC(U)
JUN 77 G H PERRY, N E WILLMORTH F30602-76-C-0180
UNCLASSIFIED SDC-TM-(L)-5839/000/00 RADC-TR-77-182 NL

2 OF 3
ADA041678




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

ECP procedures nor a central agency charged with coordinating and accounting for change at the project level must impact productivity and product quality to some degree. To offset the lack of formal procedures for external configuration control, the firmness of internal procedures was cited by management as being a major factor in guaranteeing the overall quality of the software.

4.4.3.1 Configuration Identification

In view of the uncertainties of a fluid Developmental Specification and the difficulty of maintaining a current configuration identification, some reports of using obsolete or transient information that resulted in errors and rework must be expected. It was reported that very little faith could be placed in the Preliminary Part II Specifications as a guide to program design, code or test. Design information was best acquired by personal interaction with other programmers. To mitigate this, the Chief Programmers remained with the project from beginning to end, adding a great deal of stability to the situation and avoiding most of the ill effects. Despite this and other team assistance, there was often something of a "sink or swim" condition facing replacement programmers. Nevertheless, the programmers performed in an exemplary fashion.

At least one test designer reported writing test procedures against an obsolete specification. Some of the plans had to be revised at the expense of lost time and manpower.

The Build Plan did provide some firm basis for immediate configuration identification. Scheduling priorities and CCB decisions were influenced by these and they gave a firm basis for the construction of build tests. The impact on project performance was salubrious.

4.4.3.2 Configuration Authentication

Not having a baselined Development Specification was the factor most often cited as the major source of difficulties in COBRA DANE. It made changes difficult to control and created uncertainty at every level concerning precisely what was required. Some of the uncertainty was offset by the regular

operation of the JCCB and the publication of minutes clarifying many points. It was reported that quite a lot of time was spent pursuing clarifications and decisions and that confusion sometimes arose from conflicting directives. SDC programmers were working quite closely with Raytheon engineers during most of the project. It was consequently difficult to differentiate official requests for a particular feature from informal arrangements. Answers acquired by legwork could not always be trusted and could not be checked against a set of firm requirements.

The formal reviews held initially were of somewhat doubtful value in establishing the accuracy and completeness of the designs and requirements. Although preliminary analyses made to support proposal preparation had been very thorough, there was little leadtime provided to draft a complete statement of requirements and derive a software system design before the PDR. Although the overall design that was derived survived later analyses, many changes were made to the detailed designs.

The ICCB acted quickly and decisively in matters concerned with the Program Library representation of the system. The CCB was zealous in protecting the integrity of the design although many changes internal to modules were made.

The design walkthroughs were of value in the detection of potential design pitfalls and the solidification of vague areas. This was true, especially at first, when the walkthroughs were serious and conscientious. Later, the walkthroughs were among "shows" than serious critiques of designs. While no special formats or checklists were employed, design guidance did result that prevented serious logical errors from appearing in later periods.

4.4.3.3 Configuration Control

The fluidity of the Development Specification coupled with the need to rectify the simplifying assumptions made during contract negotiations lead to many changes in the original requirements. The CCB minutes indicate several changes of direction and reinterpretation of requirements that lead to additional functions being performed by the software and to a considerable growth in the size of the software. The revisions did not result in any formal

changes in projected schedules or budgets. Without exacting configuration control procedures, many minor changes as well as major shifts were admitted to the system without a good grasp of the total impact upon the project. The regular meetings of the JCCB gave some visibility to the change process, but it would appear that the pressure to solve immediate technical problems detracted from visibility into long range trends. Changes were not made without due consideration, but a threefold increase in projected system size without commensurate adjustment to schedules and budgets is not in keeping with good configuration control practices. However, individual programmer productivity for the project was also nearly three times the rate originally estimated. In fact, given that there were at least one or two major redesigns of portions of the system and many modifications as well as additions, and taking scrapped designs and code into account, actual productivity was triple the expected rate.

Management believed that the Internal Configuration Control Board showed remarkable skill and intellectual integrity in handling internal configuration control. The people doing the job were technical practitioners (the Chief Programmers) and not clerical personnel keeping books. Internal configuration worked very smoothly and helped control and make efficient the programming task.

As noted, 2150 corrections were made during CP DT&E (CONUS) and 1100 during System DT&E (SHEMYA), a rate of about 12 errors per thousand lines of code. At an industry average of one to five bugs per hundred instructions*, the COBRA DANE programming performance would seem to be adequate if not spectacular. This is especially true in view of the high pressure to produce and the relaxation of quality assurance procedures for the Program Library. On the other hand, if only the SPRs filed during integration are considered (since most error statistics are gathered on programs after delivery) the error rate of five per thousand instructions is quite good. Such a rate may still be high compared to claims as low as five per ten thousand instructions

*E. Yourdon, in "How to Manage Structured Programming," (Yourdon, Inc., 1976) cites several studies that lead to these estimates of the industry average and reports the claims of structured programming advocates.

reported by some structured programming advocates, but even so such performance seems excellent.

4.4.3.4 Configuration Accounting

Not issuing formal configuration and change status reports on the project level had some impact on the visibility of the developing configuration. JCCB minutes and letters of configuration provide voluminous records and some level of accounting, but tracking specific changes, tracing specific impacts and monitoring overall status are difficult. The audit trail is present, but difficult to follow.

Internal accounting for the Program Library and SPR status gave excellent visibility into configuration control at that level. No problems were lost and there was accountability for all changes. There were some minor difficulties such as an SPR resulting in several changes, a given change deck modifying more than one routine, and a single change resolving more than one SPR but these are minor considerations. Regular ICCB Meetings and regular SPR status and Program Library reports reflect the smoothness of the configuration control operation. Internal configuration accounting was quite effective in promoting efficient performance.

4.4.3.5 Summary

It is felt that not baselining the Development Specification and not instituting formal configuration control procedures had a significant impact on project performance. Some confusion and uncertainty existed concerning requirements throughout most of the project and changes were hard to evaluate, control and trace.

On the other hand, the firm configuration control and accounting practiced internally had an equally good impact on internal change control and accounting. Programmer productivity was excellent and the module error rate was low.

4.5 CHIEF PROGRAMMER TEAMS

It was proposed that Chief Programmer Teams (CPTs) be used in COBRA DANE. Chief Programmers were appointed and stayed with the project throughout design, code and test. The approximately fifty project members (not all programmers) were divided into about 10 teams. The COBRA DANE teams varied from the suggested organization in that a central System Librarian and Data Base Coordinator were used rather than Team Librarians.

4.5.1 Chief Programmer Practices

The Chief Programmer was responsible for:

- An assigned number of subroutines.
- Delegating coding and testing of subroutines to assistants
- Review of all programs and modifications prior to release to the System Librarian to ensure that fundamentally incompatible modules have not been coded and tested by different people.
- Providing the System Librarian with information concerning:
  - Software checkout status
  - Error correction status
  - Descriptions of update package contents
- Conducting design walkthroughs.
- Coordination with the Data Base Coordinator concerning:
  - Global data definitions
  - Utilization of computer resources
  - Utilization of the data manager
- Ensuring that programming standards were followed in the programs for which he was responsible.
- Providing any test cases and test drivers used for testing his programs to the Program Library.

Sole authority for directing conformance was vested in the Chief Programmers for ensuring the adequacy of internal annotation and blocking and naming conventions. There were no procedures for review and enforcement of the standard practices outside the line organization.

The chief programmer was also responsible for the normal administrative duties for the programmers working under his guidance.

4.5.2 Observance of CPT Practices

It was the stated goal of the COBRA DANE project that all Chief Programmers be working programmers. This was true. The lead programmers assumed personal programming responsibility for the principal subroutines among those allocated to them and performed the design and programming. Other routines were assigned to subordinate programmers.

The Chief Programmers promoted the standard practice rules for the elements of programming style. Since the observance of these rules was quite good, as was seen in Section 4.1, their daily guidance was effective.

The Chief Programmers held and participated in design walkthroughs. As reported in Section 4.4, this was done very conscientiously at first but the reviews were decreased as the workload increased.

The evidence would seem to indicate that a similar situation occurred with other review responsibilities of the Chief Programmers. Modules and changes were initially inspected in depth but with less severity as work pressure increased and greater risks were taken to meet delivery schedules. Although the Chief Programmers devoted time to training and advising their subordinates, the amount of time they were able to give diminished as they grew busier.

Due to the operations of the ICCB, software checkout status, error correction status, and control of Program Library updates were responsibilities that the Chief Programmers were able to discharge very well. It was

reported, however, that the configuration status reports were not widely distributed and if the Chief Programmer was not conscientious in informing his subordinates of status matters, the team members had to obtain the information through personal contact and as a result of using the Program Library.

Among other responsibilities, the Chief Programmers somewhat later in the project assumed responsibility for controlling the integrity of team copies of the Program Library and for controlling inputs to the Data Base Coordinator.

The greatest complaint expressed by COBRA DANE programmers concerning the Chief Programmers was their neglect of administrative and supervisory duties. All programmers were too engrossed in the technical work to pay heed to administrative matters, but someone must take care of personnel complaints, resolve conflicts and obtain decisions. When tensions become high, small personal and technical problems loom large.

4.5.3 Impact of CPT Approach on Performance

The impacts of having a few highly proficient programmers performing the bulk of the design and coding work is reflected in the high quality of the COBRA DANE software and the excellent productivity of the teams. The Chief Programmers were some of the best at SDC and the overall quality of the team members was high. While direct comparison of CPT vs other approaches is not available, CPTs were effective on COBRA DANE.

Based on interviews with project members, there are several factors that affected the team performance. Among these are:

- Overcommitment of the Chief Programmers
- Variability in the enforcement of standards
- Underemphasis upon training of members
- Relaxation of quality assurance measures
- Discharge of administrative duties
- Team coordination and direction

4.5.3.1 Overcommitment

There was a tendency for the Chief Programmers to assume responsibility for designing and coding modules at the expense of not discharging supervisory duties as well as they might. Since the Chiefs were expert programmers this arrangement made maximum use of their programming skills but may have depressed the productivity of team members. There are dozens of minor technical decisions to be made on a programming project that junior people are reluctant to make or need guidance on. Many of the decisions are arbitrary, but some are not and the junior person may not be able to distinguish which are and which aren't. Facing a steady stream of decisions that he cannot easily make is frustrating and time consuming. Not only is his productivity impaired, but he becomes anxious and irritable as well. Feeling frustrated and inadequate, he may leave or shirk his work or neglect to research his decisions adequately.

One of the advantages of the Chief Programmer is his breadth of understanding of the system and the speed with which he grasps the essence of a problem. He is usually able to resolve a question in minutes that would take another team member hours just to acquire the relevant information.

It is part of the Chief Programmer's job to be available to make decisions and to sort out those that do require further investigation from those that don't. While this was not an intolerable problem for COBRA DANE, all Chief Programmers were overcommitted to some degree. In view of the unexpected growth of the system, this is understandable. However, it is alleged that work of team members was impaired somewhat and that some turnover did occur as a result.

4.5.3.2 Standards

Although observance of programming standards was generally good, programmers indicated that there were some differences in the emphasis that the Chief Programmers placed on adhering to the standards. This is to be expected, of course, whenever more than one group is involved. However, despite the quality inspections performed by the Program and Data Librarians, the Chief

Programmers were almost the sole arbiters of standard enforcement. If the relative emphasis was enough to be noticed by the team members, some reduction in quality of some programs probably resulted. Independent enforcement of standards might have evened out the differences.

4.5.3.3 Training

As the person with the best grasp of both system requirements and team methodology, the Chief Programmer had a responsibility for indoctrinating and guiding not only new members but all members of his team. Additionally, he must arrange assignments so that more experienced members monitor and help the more junior people and so the junior people can profit experientially from acting as back-up to the senior person. Both these techniques were used on COBRA DANE, but the general overcommitment of all programmers, plus some alleged personality conflicts between "buddies", inhibited their overall effectiveness. While the juniors on COBRA DANE were most productive, quite a few of the modules they produced had rough spots that the Chiefs polished out during integration.

4.5.3.4 Relaxation of Quality Assurance

As noted above, both Program Library and design walkthrough quality assurance procedures were relaxed as the workload climbed. This resulted in some additional errors that required later work to isolate and remove. It is also true that some of the "calculated risks" paid off. Whether or not a beneficial trade-off was achieved is not readily assessible. However, evidence which indicates that the later an error is detected the more costly it is to correct militates against such relaxations.

4.5.3.5 Administration

Administrative details are often irksome to technically oriented people. Nevertheless, for an effective team operation, someone must take care of the "paper-pushing" and the personnel problems. No matter how much of the clerical work is automated or delegated, the decisions on budgets, schedules, vacations, computer time, and salary increases remain with the supervisor. The supervisor must also settle personal problems and conflicts, intercede with

management for his people, and handle relationships with other groups and agencies if he is to have an effectively operating team. On COBRA DANE it was alleged that the overcommitted Chief Programmers were reluctant to divert attention from the main programming task to handle administrative details. How much this detracted from performance, if any, is difficult to determine. Some disruption to plans, records and morale probably occurred.

4.5.3.6 Coordination

The COBRA DANE Chief Programmers were exceptionally able, technically competent, strong individuals. Unless given firm guidance, they tended to go their own way. They did have the ICCB as a coordination point, but it tended to settle immediate problems rather than coordinate activities or maintain an overview of the total system. Several of the project members felt that there should have been an equally strong technical individual separate from the project manager who could serve as Chief Architect for the system. Such a person could maintain an overview of the system, make configuration decisions from a system viewpoint, ensure components are compatible, better control and coordinate modifications, and better monitor computer resource utilization than any single Chief Programmer. In doing applications system work, it is very important that everything works correctly and fits together precisely. In a more research-oriented environment there is more room for experimentation, more tolerance of change and modification, and less necessity of exact interfaces. System failures are not so catastrophic and incompatible interfaces can be made to make do with some extra attention. In an operational environment, operations cannot depend upon having a super programmer available to solve stoppages or hang things together. Hence, the Chief Architect who sees that all teams are going in the same direction and that incompatibilities are not being introduced could serve a very useful purpose. COBRA DANE had some very technically competent managers, but there seemingly was a need for a person whose sole responsibility was the technical overview of the system who could arbitrate technical disputes and maintain the integrity of the system design.

4.5.3.7 Summary

Individual productivity, especially of the Chief Programmers, was magnificent. Considering that the final system was three times the size originally estimated and that there was a major redesign of the Mission Support CPCI as well as numerous changes, each of which resulted in some work being scrapped, productivity was excellent. There is a definite danger in the CPT approach that the Chief Programmer will become overcommitted to coding and not have time to discharge his other duties. He must have time to advise his team members, train them, monitor their work, and care for their personal and administrative problems. A large project can profit from having a Chief Architect to give overall direction to the design, to protect the integrity of the design from the ill effects of modifications, to resolve disputes and to see that quality standards are enforced evenly across the total system. The Chief Architect would receive major support in this role from a System Librarian and System Data Controller.

5. CONCLUSIONS AND RECOMMENDATIONS

The general conclusions to be drawn from this study are:

- Observing standard programming practices does contribute to software quality.
- Observing standard software management practices does organize and regulate the software development process, making decisions easier and better and reducing effort and confusion.
- Environmental factors such as working conditions and availability and power of tools and facilities does affect the difficulty and efficiency of the software development process.

5.1 ELEMENTS OF PROGRAMMING STYLE

It is concluded that the elements of programming style such as rules for commentary, naming, paragraphing and modularization contribute to the ease of understanding and maintainability of software.

5.1.1 Commentary

Based on statements from the specification writers, commentary greatly improved the understandability of the COBRA DANE programs. The modules that gave the writers most difficulty were those few where data annotations were deficient, especially in the description of flags and registers of bits passed as flags.

It has been estimated that, for small modules, lines of commentary ought to equal lines of code. Composing proper commentary takes some thought and care, but its performance requires only a small proportion of the total coding and debugging task. It is estimated that commenting the code took less than 5 percent of the code and checkout time.

5.1.2 Naming Conventions

Naming conventions provided help in organizing and classifying program elements. The names reported by the specification writers as giving difficulty were those in which the organizing elements were missing, such as table items which weren't identified as elements of a specific table or file. The naming system decreases debugging and diagnostic time by making items easy to find and understand.

Following a consistent set of naming conventions adds no significant increment to programming time. Enforcement of the standard via quality inspections by program and data control personnel does entail a small cost, perhaps as much as 20 percent of the librarian's time. The contribution to better communication and to improved maintainability totally justifies the effort required.

5.1.3 Paragraphing

Except for providing lines of special characters (asterisks) to set off blocks of code or blocks of commentary, programmers reported that no special effort was required to follow paragraphing rules. That is, it became habitual and automatic, and was as useful in debugging as in later legibility. The impact on readability of the programs was excellent. The specification writers universally reported no difficulties in following the overall organization and flow of programs.

5.1.4 Modularization

Modularization broke the programs into simple, easily understood units. Single entrance and exits and restrictions on calls outside the parent module's area of control kept flows simple and easy to follow. Where exceptions to program length, entry points and simplicity of flow did occur, both specification writers and test personnel noted that these modules were harder to understand and to maintain. Inversions in flow due to FORTRAN IFs not falling through into the main flow resulted in jumping about in the code, but that is a recognized FORTRAN structured programming deficiency.

5.1.5 Recommendations

Although standard practice rules for writing programs are becoming more generally accepted, a minimum acceptable set needs to be extracted and stated as verifiable standards. Although the principles involved are better established than they once were, detailed implementation remains a problem. That is, technical proficiency is required to either perform or to evaluate skilled technical tasks such as modularizing a program or annotating code well. Additional formulation is required for the principles and additional development of technical implementation guides is needed if enforcement of the standard practices is to be based on other than expert judgment of conformance.

However, adopting rules for using the elements of programming style is beneficial no matter how unsophisticated they may currently be.

5.2 PROGRAM LIBRARY OPERATIONS

Instituting a well-controlled Program Library operation provides a nexus of coordination for a programming project. It provides configuration control and status information as well as a vehicle to control and account for program corrections and changes. Some of the elements to consider are:

- The nature of the Program Librarian
- Library structure and maintenance procedures
- Change control procedures
- Quality assurance provisions
- Integrity of builds and versions

5.2.1 System Librarian

A System Librarian seems a viable alternative to Team Librarians, but a System Librarian must be a skilled and forceful person if he is to enforce programming and library interaction rules and gain the support and cooperation of the programming teams.

5.2.2 Library Maintenance

Although COBRA DANE suffered no ill effects from not initializing the Program Library with a full set of program stubs, it is believed that management visibility goals would be better served if that had been done. Firm procedures for interacting with the Program Library, including access rules and lead times for submitting materials, help regulate and organize programming as well as library operations. Regular maintenance and strong control has a stabilizing effect on the project. Where access rules were relaxed, inefficiencies were seen to occur in COBRA DANE operations.

5.2.3 Change Control

One of the principal advantages of Program Library operations is the opportunity it affords for placing a completed module under control so that no unauthorized or unverified change is made to it. The change control procedures worked very well for COBRA DANE. COBRA DANE managers felt strongly that program change control was one of the most beneficial practices of the project. It prevented unnecessary change and ensured that all proven errors were corrected. Control of changes has high impact on project performance and upon program quality. Program Library procedures contribute significantly to this control.

5.2.4 Quality Assurance

Program Library operations offer a unique opportunity for verifying the observance of programming standards and the adequacy of unit testing. Further, the quality of controlled software is assured by the change procedures.

5.2.5 Build Integrity

The build approach used in COBRA DANE was very effective in meeting the requirements of the project. Build deliveries keep the motivation to produce at a high level by providing short range goals with a high specific content. The approach helps organize the work and established priorities. However, COBRA DANE experience indicates the need to protect the integrity of the delivered builds. Both programming and integration personnel need a

stable environment to work within. Instability in the programs, whether the change corrects or creates an error, was really the impetus for the Chief Programmers making copies of the Program Library that they could control.

It must be concluded that providing a stable and known software environment during an evolving software development situation presents a problem. A means of protecting and guaranteeing the integrity of a build or version or modification of the system when these configurations must support other activities is highly desirable.

5.2.6 Recommendations

It is recommended that some sort of Program Library operation be instituted on every project to store the software product representation, protect it from unauthorized change, ensure the quality of the delivered product, and support configuration status accounting and reporting.

A wide range of Program Library operations running from almost completely manual to highly automated systems have been discussed and tried. Expansion of the Program Library operation to incorporate project monitor functions (schedules, budgets and personnel records as well as a software inventory) has been suggested, plus a variety of support and analytic tools. Obviously, small projects cannot afford highly complex and powerful Program Library support programs unless there is a high volume of such projects using the library. To achieve some balance in standard practices for Program Libraries a minimum set of functional requirements needs to be established. Additional functions may be suggested, but not required. It is not judged feasible at this time to develop a standard tool, but standard operations seem essential.

It is suggested that the standard functions include:

- Separate accountability for delivered and controlled versions, builds and modifications.

- Accountability for configurations in differing stages of development. (At a minimum, 'master', 'fall-back' and 'test' or 'developmental' versions should be maintained.)
- Program change accountability for any controlled program. (It is suggested that status accounting records cross-reference every program change to an ECP, CR, SPR, or Action Item identifier or other suitable authorization for making a change.)
- Library activity logs.
- Program Library configuration status reports and program change status reports. (Note that ECP, SPR, etc., accounting may be done separately or integrated with the program module change status.)
- Program Librarian responsibility definitions. (Note that "team" Librarian should have responsibilities and authority different from "System" Librarians.)

5.3 GLOBAL DATA CONTROL

Instituting a good system data control operation is just as important as that for programs. Although COBRA DANE began with relaxed system data controls, by the end of the project, stringent control was implemented.

5.3.1 Data Base Coordinator

If the Data Base Coordinator is to enforce programming standards for data naming conventions and data structures, prevent redundancies in definitions, consult on the efficiency of data structure and handling, and monitor the utilization of computer resources, a skilled and forceful person is required. The coordinator and the data control procedures must have strong management support. Initially on COBRA DANE the Data Base Coordinator operated on a clerical level and exercised no control over the content or operation of the global data definitions. Before the project was over, the Chief Programmers assumed responsibility for the integrity of data in their areas of purview.

5.3.2 Maintenance

The same access and update criterion as applied to the Program Library need to be applied to the System Data Library. Regular updating and firm control procedures contribute to efficient programming. Where control was not exercised in COBRA DANE, uncoordinated data changes caused unexpected program failures that interfered with program testing and equipment checkouts.

5.3.3 Change Control

Data change control procedures that parallel program change control procedures should be instituted. Ensuring evaluation of changes to detect all "ripple" effects and to ensure data efficiencies is a valuable service. Not having data change status accounting may be one reason the effects of data change were not always well coordinated or communicated in COBRA DANE.

5.3.4 Quality Assurance

Quality inspections are part of the access and change control procedures. Changes to data ought to be checked as thoroughly as changes to programs. Although not a high percentage, violations of naming conventions and usage did create some difficulty in understandability of the COBRA DANE programs. The specification writers thought good annotation of data more important than the annotation of instructional statements.

5.3.5 Resource Accounting

Tracking resource utilization is especially important in real-time programs. Not having adequate advance warning of over-utilization of computer storage and channel capacity aggravated the redesign crisis when this occurred for COBRA DANE. Although not as crucial for non-real time systems, resource budgets and accounts are desirable, if not essential, records for all systems.

5.3.6 Recommendations

It is recommended that the same level of control be exercised over globally defined data and data base structures as is exercised over program modules and software system configuration.

In operation it is suggested that the file of global data definitions should be one of the Program Library files and any tools provided for maintaining and analyzing the data definition file should be part of the Program Library support repertoire. While it may not be necessary to have as highly automated and controlled a system as that enjoyed by the SCF COMPOOL-Sensitive system (see Part III of this report), the control procedures used there and some of the environmental analysis tools such as system set-used and memory maps are desirable tools for tracking the impacts of change and monitoring resource utilization.

As a minimum set of functional requirements, system data control should include:

- The same level of control and accounting for versions and other structurally or developmentally defined data elements as for program elements.
- Enforcement of data definition standard practices (including programming standards, avoidance of redundancies and efficiency of data manipulation standards).
- Accountability for data changes equivalent to the accountability and control of program changes.
- System data control coordinator responsibility definitions.
- Data configuration and change status reporting.

It is suggested that one of the data files described in the data definitions be the file of program modules and its members and that accounts be kept of computer resource utilization.

5.4 CONFIGURATION MANAGEMENT

The intangibility of the software product, the degree to which software incorporates the operational procedures of the using organization, and the apparent relative ease of modifying software, all make configuration management particularly important for command, control and communication systems. The objectives of configuration management are to achieve a firm, complete, baseline of the system, to control and account for all approved changes and to provide management visibility into system composition and developmental status. While COBRA DANE had the machinery in the Joint and Internal Configuration Control Boards that partially satisfied these aims, the failure on the project level to institute formal configuration definition, change control, accounting and document maintenance procedures undoubtedly impacted project performance.

5.4.1 Configuration Identification

While all planned program and program test documentation was produced, frequent change not reflected in the documents impaired their usefulness. Internal control over the Program Library representation of the system helped offset the lack. However, the impacts of not having a firm configuration definition can pervade all aspects of software development. A firm basis for evaluating the impacts of change is not provided. Uncertainty concerning the responsiveness and firmness of designs and test criteria must follow. To some degree, all these effects were experienced in COBRA DANE.

5.4.2 Configuration Authentication

Not baselining or otherwise authenticating the specifications for COBRA DANE, and thus bringing the performance requirements under as tight a control as the Program Library, was felt by many observers to be the most serious factor in the difficulty of controlling and accounting for changes in COBRA DANE and the confusions and uncertainties that accompanied development to some degree. Without an authenticated specification it is difficult to differentiate between a "clarification" and a "modification" and it is difficult to verify or prove that a specified level of performance has been attained.

5.4.3 Configuration Control

The impacts of change and more specifically uncontrolled change have been frequently declared to be the most serious cause of schedule slippages and cost overruns. Having formal configuration change procedures might not prevent change, but it makes the changes and their impacts visible. Changes did impact COBRA DANE performance. The project did have configuration control procedures, but probably could have profited from having more formal procedures to make the impacts of changes more obvious.*

In contradistinction to control over functional requirements change, internal control over change to the software product once it entered the 'system master' controlled portion of the Program Library was very tight and observance of the rules was excellent. Control of change in the Program Library contributed a great deal to the organization and stability of the developing software.

It must be concluded that while formal change control procedures do not prevent change, they make the reasons for change and the costs of making changes apparent and indeed provide a rational basis for rejecting less cost effective or important changes.

5.4.4 Configuration Accounting

It is through configuration and change status reports, including status reports, that the configuration identification and changes to it are made visible to management. The volumes of proposed changes and suspected problems are indicators of potential difficulty. Without these indicators, management decisions are hampered. There is no doubt that configuration accounting does contribute to project efficiency.

*Just evaluating proposed changes represents a significant expenditure of resources and usually some lost time on the principal tasks. Formal procedures should make these costs as well as direct revision costs apparent.

5.4.5 Recommendations

It is recommended whether the software be acquired directly or through a prime, that configuration management procedures which may be used for both external and internal control, be established for all software systems.

Although configuration management practices are normally performed manually, it is suggested that consideration be given to the definition of a software monitor system to be used both for external and internal configuration accounting and reporting. It is possible to combine the software monitor with Program Library operations or to share responsibility with the library.

At a minimum, it is suggested that the configuration management practices ought to provide:

- Complete, firmly baselined, configuration identifications especially at the functional requirement or Software Development Specification level.
- Configuration control for changes to baselined specifications or controlled software configurations. These should include formal procedures for requesting, evaluating and disposing of modifications to requirements, designs or controlled products for any reason that arise either externally or internally to the project.
- Configuration accounting records and status reports to provide visibility to software configurations as amended by approved changes and the status of suggested changes. Separate accountability should be provided, as appropriate, for builds and versions of the system.

5.5 CHIEF PROGRAMMER TEAMS

The impact of having a few highly proficient programmers performing the bulk of the design and coding work is reflected in the high qualty of the COBRA DANE software and the excellent productivity of the teams. However, since there were substantial differences between the way the COBRA DANE teams were organized and the organization recommended in the literature, strong support for CPT is inhibited. COBRA DANE project members did make recommendations for a Chief Architect, a concept that is gaining favor in the literature.

The use of a System Librarian and a System Data Controller, which is the principal way in which the COBRA DANE CPTs differed from the organization suggested in the literature, is deemed a viable alternative to Team Librarians. This is especially true if these controllers are in direct support of the Chief Architect.

5.5.1 Recommendations

No recommendation is made concerning adoption of CPTs as a required management technique.

It is recommened that projects have a "Chief Architect", "Configuration Management" or "Principal Software Engineer" charged with final authority for making technical decisions and commitments and responsible for the overall structure and design of the system. The most efficient software development organizational structure depends very much on the individual company and individual system requirements and should not be dictated by the procuring organization. Whether or not those charged with technical software engineering decisions should also be charged with administration of the project is a moot question, but the chief technical person should be charged with work sizing and scheduling estimates whether or not he administers these aspects.

PART III - SCF COMPOOL-SENSITIVE SYSTEM STANDARD PRACTICES

1. INTRODUCTION

SDC has been the Computer Program Integration Contractor (CPIC) for the Air Force Satellite Control Facility (AFSCF) since 1963. The decision to implement a COMPOOL-sensitive system was made in the mid-sixties as a solution to several very serious problems. First, the system was in a state of rapid growth and constant change. There were numerous users of the system with user-specific programs and data as well as many common data processing needs and there were numerous contractors and programming groups. The system was large with many programs and a large amount of data and numerous interfaces and interdependencies among the program and data elements. These conditions made for much redundancy in data, little standardization of data and interface definitions, often inefficient usage of storage and numerous interface incompatibilities. Finally, there were stringent core and other computer resource limitations which, despite computer upgrades, continued to be serious.

The AFSCF needed to enhance the efficiency of programming and data handling and improve the effectiveness of system management. Although such other actions as increasing the stringency of integration procedures and a broad standardization program were also undertaken, the principal tool for attaining these goals was an AFSCF support system that was compool-sensitive. Such a system represents a considerable investment, but the seriousness of the problems and the advantages offered by the system were thought to justify the expense. Some of the alternatives that could have been adopted included:

- Adopting only a manual data definition, control and coordination operation.
- Using a compool-sensitive compiler with compool-generation and listing capabilities to support it.
- Adding symbolic debugging and test tools to augment the compiler.
- Adding program and system environment analysis capabilities to the above.

- Adding symbolic data management capabilities.
- Adding a compool-sensitive, dynamic loader operating system.

Each of these levels of sophistication provides additional increments of control and capability. It was judged necessary to implement all of them if the SCF objectives were to be met, i.e., the COMPOOL implementation.

1.1 SCOPE

The overall objective of this study was to attempt evaluation of experience with the AFSCF/COMPOOL* in terms of:

- Its effectiveness as a configuration control device.
- Its contribution to the reduction of software errors.
- Its contribution to the simplification of software development.
- Its usefulness in supporting the establishment and enforcement of standards.
- Its cost-effectiveness in supporting increased throughput, e.g., assimilation of heavier SCF workloads without requiring additional computing hardware acquisition.

The data for this study was almost entirely derived from inspection and analysis of SDC documentation and records. (See Appendix E.)

1.1.1 The COMPOOL Concept

A compool (Communications Tag Pool) is essentially a centralized file of data descriptions and declarations maintained independently of any one program in a system. A compool-sensitive system is a software system whose programs access the compool tables to determine the structure, size, location and nature of the program and data elements it deals with. In some instances, compools have been restricted to global or common data definitions, including

*It is important to state at the outset that throughout this study the term, "COMPOOL," will imply the compool itself, the compool-sensitive J4 JOVIAL compiler, and the SYMON (System IIB) operating system.

constants, needed by a compiler to generate and assemble each program. In other instances, compools contain the program and data descriptions and sequence parameters required to load and link programs for core loads and overlays. The compool may be used by test data generation programs to produce simulated test data by using:

- Debugging programs to locate and print out symbolically desired portions of programs and data.
- Patch programs to insert or load symbolic correctors into programs.
- Information retrieval and data management programs to retrieve data and manipulate data files.

Compools normally exist in two forms, a symbolic or source form for use by humans and a binary form for use by machines and programs. The symbolic compool is normally commented and annotated and provides automatic documentation of data (and program) elements. The binary compool may contain information supplied to it by the compiler and other programs that enable other "environment analyses" to be performed. These include system and program set-used, program environment, and core map programs. Compool listings of communication tags usually appear in both structural and alphabetic orders to enable ready cross-reference between items and the tables (or other structures) in which they are located.

In addition to its machine use, the compool is used as a management tool to support control and coordination of the data configuration for a system. A compool (or system data) control office is established to manage the compool. In addition to maintaining the stored compool, it receives, evaluates, and coordinates all additions and modifications to the compool and issues periodic listings and analyses on demand. The review looks for duplicate, redundant and obsolete items and evaluates data structures for inefficiency in storage and handling. The Compool Office advises on the use of the compool and the efficiencies of structures and ensures that standards for data naming and structures are observed. The Compool Office, using the compool and its analytic programs as tools, is a very powerful mechanism for supporting

software-interface compatibility and system efficiency and supporting configuration control and accounting functions for a system.

In general, although it may contain values of constants, sequence parameters, and directives, the compool contains descriptions of data and not data themselves. Although a compool is a data base of sorts, conceptually it describes data bases rather than being one. It may describe not only data bases, but the system itself and contain a variety of environmental data. In short, it is a very powerful tool for the coordination of both programs and data interfaces.

1.2 SUMMARY AND CONCLUSIONS

Twelve specific objectives for the implementation of the system were initially identified. These were to:

- Provide better communication between system elements.
- Simplify the programming task.
- Guarantee data integrity.
- Facilitate program maintenance.
- Control storage allocation.
- Eliminate fixed-address communication.
- Facilitate purging redundant and obsolete data.
- Reduce and control program environments.
- Support configuration management.
- Simplify interfaces.
- Centralize sources of data.
- Improve overall system management.

As will be clear in the following paragraphs, these objectives were all accomplished. The AFSCF Flight Support Computer (FSC) Segment provides a degree of cost-effective support to its many users that is probably matched nowhere else, given the stringently limited hardware resources.

1.2.1 Communication Between System Elements

Better communication between system elements was provided by adopting:

- A fixed, standard calling sequence.
- Defined data blocks of varying levels of control.
- Standard data structures.
- Standard, compool-defined I/O buffers.

While it does not make a great deal of difference in system efficiency which calling sequence is used, adopting a standard sequence simplifies program interactions and reduces coordination requirements. Standard data blocks, classified by the usage priority, increase efficiency of use and aid communications. Standard definitions of items, tables and arrays simplify data structures. Standard buffers defined in the compool greatly simplify interface interactions.

1.2.2 Programming Simplification

The programmer's task was simplified by:

- Reducing the number of data declarations required.
- Predefining working and communication space.
- Automating load and link operations.
- Providing a symbolic corrector capability.
- Providing automatic units conversions.

Data elements, defined centrally in the compool, reduce the number of data declarations the programmer must make. The number of declaration errors committed is also reduced.

The dynamic linking and loading capability of the operating system using compool definitions of program and data blocks eliminates loading and linking instructions and provides a fast load and go capability during program development.

The symbolic corrector capability not only eliminates much clerical work in preparing program correctors but provides an efficient alternative to frequent recompilations.

The automatic units conversions permits data to be displayed or entered in any mode desired while preserving standard units internally. Programmer effort in creating and using conversion routines is greatly simplified.

1.2.3 Data Integrity

To guarantee the quality and integrity of data, the compool control operation provides a centralized source of data and data definitions. Actions to guarantee the integrity of data include:

- Distribution of library tapes containing constants and global data definitions to all users.
- Distribution of data bases with standard data.
- Guaranteed saving of specified data blocks.
- Central, controlled definition of all global data.
- Standard, controlled data formats.
- Standard units of measurement.

Central definition and control of all constants guarantees the accuracy and precision of the values of the constants and avoids redundant, individual definitions with attendant opportunity for error. Central control and distribution of standard data prevents insertion of non-standard, inadequately inspected data blocks being used to support development or, more importantly, real operations.

Guaranteed saving of designated data blocks avoids the inadvertent loss of data between programs due to program error or failure.

Central control of global data definitions permits stringent quality assurance measures to be applied to ensure efficiency and proper classification and to avoid redundancies and errors. Standard formats simplify programming and quality assurance measures as does standard (metric) internal units of measure.

1.2.4 Program Maintenance

Program maintenance is facilitated by:

- Using a compool-sensitive compiler.
- Well-defined interfaces.
- Numerous analytic aids.

The compool-sensitive compiler permits updating all programs impacted by a data change by redeclaring the data element once in the compool and recompiling the affected programs. Individual program changes, with attendant opportunity for error, are avoided.

The establishment of standard calling sequences and data interfaces tends to reduce the number of programming choices, make program interactions easier to understand and maintenance simpler. The simpler interfaces have reduced the number of errors.

Analytic aids, such as compool listings, listings of the characteristics of a compiled program, system and program set-used tables, environmental listings and core maps make it easy to locate data errors, avoid duplicate data and program names, and understand the structure and operation of the programs.

1.2.5 Storage Control

The control and conservation of storage, a very important objective in view of the very limited capacity of the Flight Support Computers (FSC), was improved by:

- Commonality of data structures.
- Extensive overlay capabilities.
- Compool hierarchies.
- Efficient segmentation algorithms.
- Policing current core allocations.

Insisting upon maximum utilization of common data and eliminating all unnecessary private data keeps the demand for storage down. The overlay capability makes efficient use of storage, keeping the amount of resident, but inactive core at a low level.

The hierarchy of compools (system compool, project compools and temporary compools) ensures that only those data definitions necessary to the current task are in core while it is performed.

Data blocks are segmented to fit into available core and the dynamic loading and overlaying provides a near 'virtual storage' capability.

Dynamic loading enables the regular policing of core loads to ensure the elimination of redundant and obsolete data. Core loads for a task may be fitted to the specific needs of the run and not to fit all options for a task. The dynamic loading makes the core reallocations easy and core maps permit monitoring loads for efficient utilization.

1.2.6 Fixed-Address Communication

The elimination of fixed-address communication, made possible by the compool defining of data elements, improves the flexibility of operations by:

- Making all programs and data relocatable.
- Permitting dynamic linking and loading.
- Allowing symbolic correctors.

Although the dynamic reallocation of core each time a task is loaded provides great flexibility and gains efficient utilization of storage, it demands a price in computation time. However, if a task is to remain invariant between successive passes, the system provides a push-pull capability that will save and restore an allocation without dynamic address computation.

The symbolic corrector capability may operate during reallocation to make corrections, adjustments or adaptations to programs to fit current needs.

1.2.7 Redundant and Obsolete Data

The purging of redundant and obsolete data from the COMPOOL is enhanced by the procedures employed to review and control additions and modifications to the COMPOOL. These procedures include:

- Procedures for reviewing proposed changes and maintaining the compool
- Analyses of current content of the COMPOOL.

Close review of all compool-changes resulting from adding new programs or modifying existing programs keeps data redundancies to a low level, ensures the observance of data standards (including the isolation of constant, guaranteed and transient data) and efficient handling and storage utilization. Considered is the impact of changes on other users, the coordination of changes with users and the restructuring of the system.

Analytic listings are used to detect redundancies and impacts and evaluate segmentation and efficiency. The system, set-used listings are specifically used to detect obsolete items, those neither set nor used by the current system.

1.2.8 Program Environment Control and Reduction

As a corollary to the control of storage utilization, the control and reduction of program environments keeps storage requirements at a minimum. The features of the compool operation aimed at control of program environments are:

- Environment analysis and display capabilities.
- Commonality.
- Compool change procedures.
- Data grouping by use.

In depth analysis of the program environment helps detect inefficiencies and in combination with set-used listings suggests trade studies of more efficient configurations. Compool change procedures flag inefficiencies and detect improper use of routines and inefficient segmentation. Separating control from application-specific information provides guarantees of integrity and proper operation.

Just as compool organization of data into small aggregations tend to reduce the size of the program's data environments, it also encouraged further modularization of the programs themselves. Thus, the size of a program's instruction environments was effectively reduced as well.

1.2.9 Configuration Management

The standardization of data structures and handling coupled with compool-procedures simplifies the configuration management task. Some of the ways in which configuration management is supported are:

- Change evaluations.
- Configuration control and accounting.
- User control of system structure.

Standardized data structures simplify evaluation, not only of compool changes but of all new programs and modifications. The COMPOOL and its procedures provide a channel of communication among all contractors and users concerning the impact of changes. The reviews, in turn, encourage standard treatment of data.

Configuration control is simplified by easy accounting for data elements and all changes to them. Standards help control unnecessary change.

The visibility into data and system structure provided by the regular issuance of compool and compool change reports plus the involvement of user personnel in review procedures give the user a level of control over system structure and content not realized by most systems. This is a feature upon which the AFSCF places high value in helping it to control the data requirements generation and data usage of individual contractors and users.

1.2.10 Interfaces

Simplifying interfaces not only between programs but between developers and between users was a prime objective of the COMPOOL system. Simplification is attained by:

- Compool change control.
- Standardization of calling sequences
- Standardized intercommunication buffers.
- Standardized block structures.
- Analytic listings.

The change procedures and regular analytical reports keep all developers, users and system managers abreast of the current state and composition of the system. The standardized calling sequences, intercommunication buffers and block structures keep program interfaces standard and simple, and preserve their validity.

1.2.11 Data Source Control

Control over the source and quality of data quite apart from the data definition in the compool was an important objective of the COMPOOL implementation. The COMPOOL supports such control through:

- Providing library tapes.
- Permitting project specific data bases.
- Permitting private data bases.

Supplying controlled and validated data avoids unreliable data and is a service to users and developers alike. Permitting project-specific data bases, with values that may override the generally common information in the compool, adapts control to the peculiar needs of projects, yet economically supports the needs of most users. Having private data bases satisfies the sensitivity and security requirements inherent in the AFSCF users community.

1.2.12 Overall System Management

By these actions, the overall objective of improved manageability of the AFSCF system has been achieved. In summary: The COMPOOL system has contributed to:

- Enhanced configuration control
- Improved control over data bases.
- Much improved interface control
- Encouraging the use of common (GFE) subroutines as well as common data.
- Enhanced security for sensitive data.
- A measure of independence from machine limitations.

Orginal expectations have been met and in many instances exceeded. The return on the investment has far outweighed the costs.

2. AFSCF ENVIRONMENTAL CHARACTERISTICS

To make a proper evaluation of the problems and conditions that lead to the decision to implement the COMPOOL, it is necessary to understand the nature of the system and some of the changes that were made in it as a result of the COMPOOL implementation. The AFSCF data system has been supporting satellite command and control functions for Air Force and other agency satellite programs for over 15 years. These have been years of dynamic growth and change in the programs that are supported and the data system that supports them. The forcing function through the years has been the need to provide an AFSCF data system that was:

- Generalized enough to support the command and control needs of all Air Force satellite programs.
- Specialized enough to support program-peculiar requirements of a diverse set of users.
- Flexible enough to permit users to share the limited computer resources (see Figure 2) at rates varying from once an hour to once a month.

The rapid growth in size and sophistication of the support provided to the various satellite programs makes exact comparisons of throughput and support levels difficult. However, examination of the support provided and the relative amounts of resources utilized provides an excellent phenomenalogical evaluation of the relative merits of the pre-COMPOOL and COMPOOL data systems. For example, one satellite program formerly required the 24-hour-a-day dedication of three CDC 1604s for the duration of its flights; it now operates with only one dedicated CDC 3800, plus part-time support from a second. In addition, the support requirements have increased to include multiple rather than single reentries, attitude determination and control, rev-by-rev orbital correction, and much more complex command repertoires.

The aspects of the AFSCF data system that must be examined are the characteristics of:

- The satellite support provided.

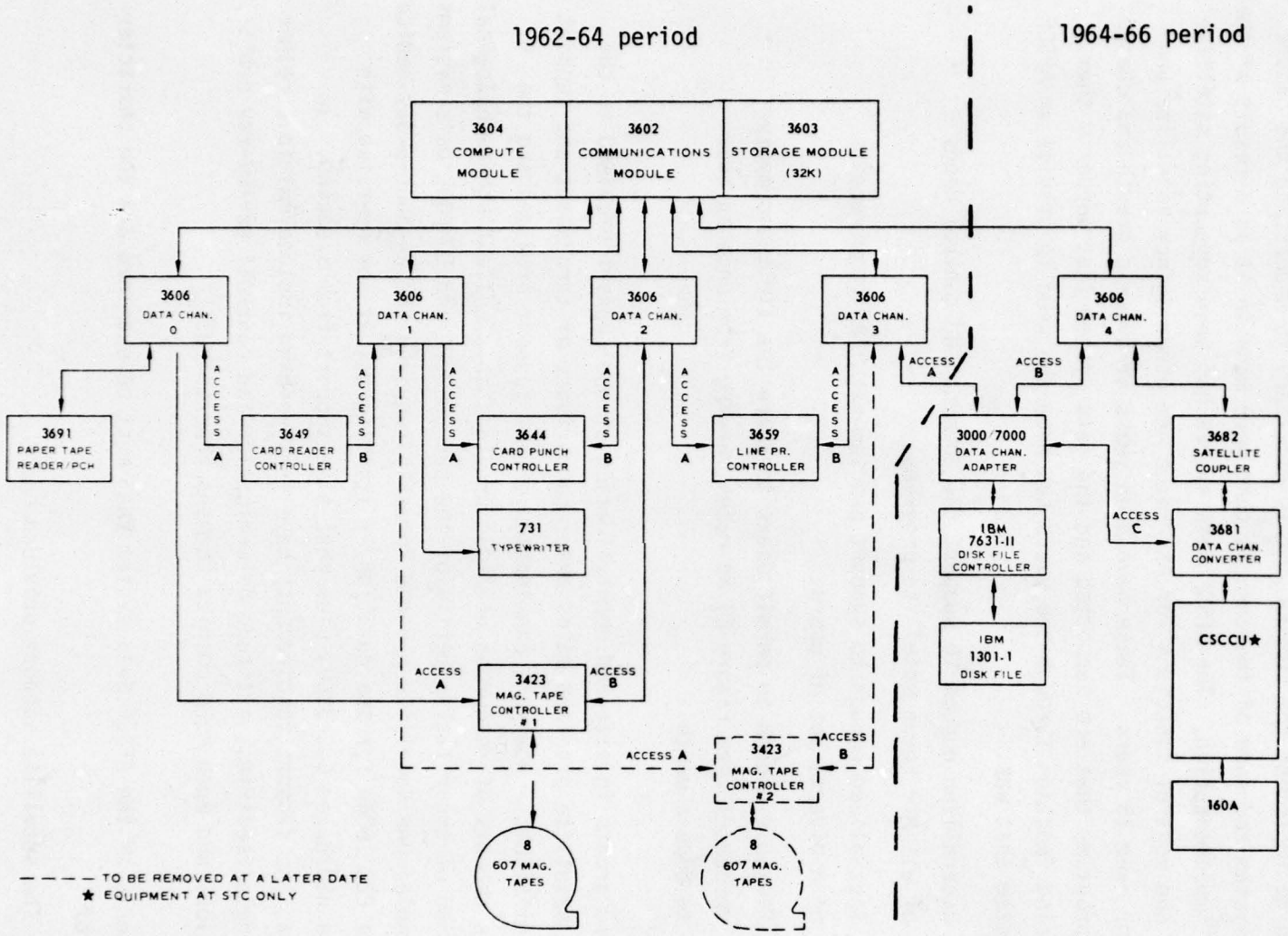


Figure 2. Central Computer(s) Configuration, 1962-64 and 1964-66

- The central computer
- The operating system
- The programming and utility tools
- The applications programs

2.1 SATELLITE PROJECT SUPPORT

Since 1961, the number of program offices supported has more than tripled, while the number of satellite missions has increased over tenfold [33]*. While these numbers are impressive in an absolute sense, the nature of the support provided has shown an expansion that goes far beyond these raw numbers. Programs have been increasing both in the number of satellites and the types of programs supported, ranging from sub-orbital (ballistic) trajectories to geocentric orbits up to 60,000 nautical miles. On-orbit maneuvers have included earth-reentry of both recoverable and non-recoverable vehicles, as well as staging, attitude adjustment, and orbital modification events. Requirements for ephemeris prediction (to support command-sequences generation) and ephemeris history (to support post-flight analysis) have increased greatly. Not only has the number of satellite programs involving reentering (i.e., recoverable) vehicles increased, but reentry has concerned multiple recoveries per launch rather than one.

Again, program office requirements have escalated from orbit-correction and command generation-sequences after each three or four revs (about 90 minutes each) to their being performed after each rev, i.e., every 90 minutes during a flight. Further, the duration of flights has increased from an average of 4 or 5 days to 30 or more days.

As the assortment of satellites, orbital characteristics, and maneuvers expanded, the requirements for accuracy, precision, and response-time laid upon the AFSCF data system also became more stringent. Table 2 shows the dramatic growth in the numbers of executable instructions in the FSC software system.

*Indicates source document No. 33 from reference list in Appendix E. The documents listed in Appendix E are referenced throughout Part III. References always appear in brackets, i.e., [ ].

Table 2. CDC 3800 Flight Support Computer Executable Instructions ($X10^6$)

TAPES	1970	1971	1972	1973	1974	1975	1976	1977
SST	1.02	1.03	1.14	1.31	1.50	1.52	1.58	-
AMT(Non-SPO)	.039	.054	.060	.258	.373	.496	.573	-
AMT (SPO)	e .175	e .200	e .225	e .250	.325	.462	.615	-
TOTAL	1.234	1.284	1.425	1.818	2.198	2.478	2.768	e 3.272

Key

e = estimated
SST = System Support Tape
AMT = Auxiliary Master Tape
SPO = Satellite Program Office

Each one of the satellite programs supported is considered an independent user of the system, with its own particular needs and requirements. Rapid advances in satellite and telemetry technology implies rapid advances in matching AFSCF capabilities. These advances also mean a high volume of new programs and program modifications, not only on the part of the generalized satellite command and control support system but in flight-specific software procured by the satellite program offices. Therefore, not only are there many users of the system, but many software suppliers who use the AFSCF data system both on their own premises and in the Software Development facilities provided by the AFSCF for development, integration and test of software.

2.2 CENTRAL COMPUTER CHARACTERISTICS

Between 1964 and 1968, the central computer hardware configuration underwent three significant changes*. The five CDC 1604s (1961--1964) comprised 32K of 48-bit words in core, with sixteen CDC 607 magnetic tape drives, and four data channels. In 1964, the CDC 1604s were upgraded to CDC 3600s, at which time they each had exchanged eight of the tape drives for an IBM 1301 disk file (used as eight pseudo tapes, i.e., serial files), and another data channel that helped accommodate a direct data line to another computer as shown in Figure 3 [12]. In 1967, the CDC 3600s were upgraded to CDC 3800s, and core was increased to 64K. The IBM 1301 disk was replaced by five CDC 854 disk (pack) drives.

The central FSCs, at the Satellite Test Center in Sunnyvale, now comprise six CDC 3800s, each with eight 607 tape drives and five 854 disk packs. Each 3800 system is free-standing, and can communicate with other elements of the AFSCF data system (see Figure 2) only through hand-carried magnetic tapes or card decks.

*For the sake of clarity, other minor hardware changes of inconsequential effect upon this study have been purposely ignored.

**Although the disk packs are nominally interchangeable, operating experience seems to indicate magnetic tape is the preferable medium for transfer of data and/or data bases between CDC 3800s.

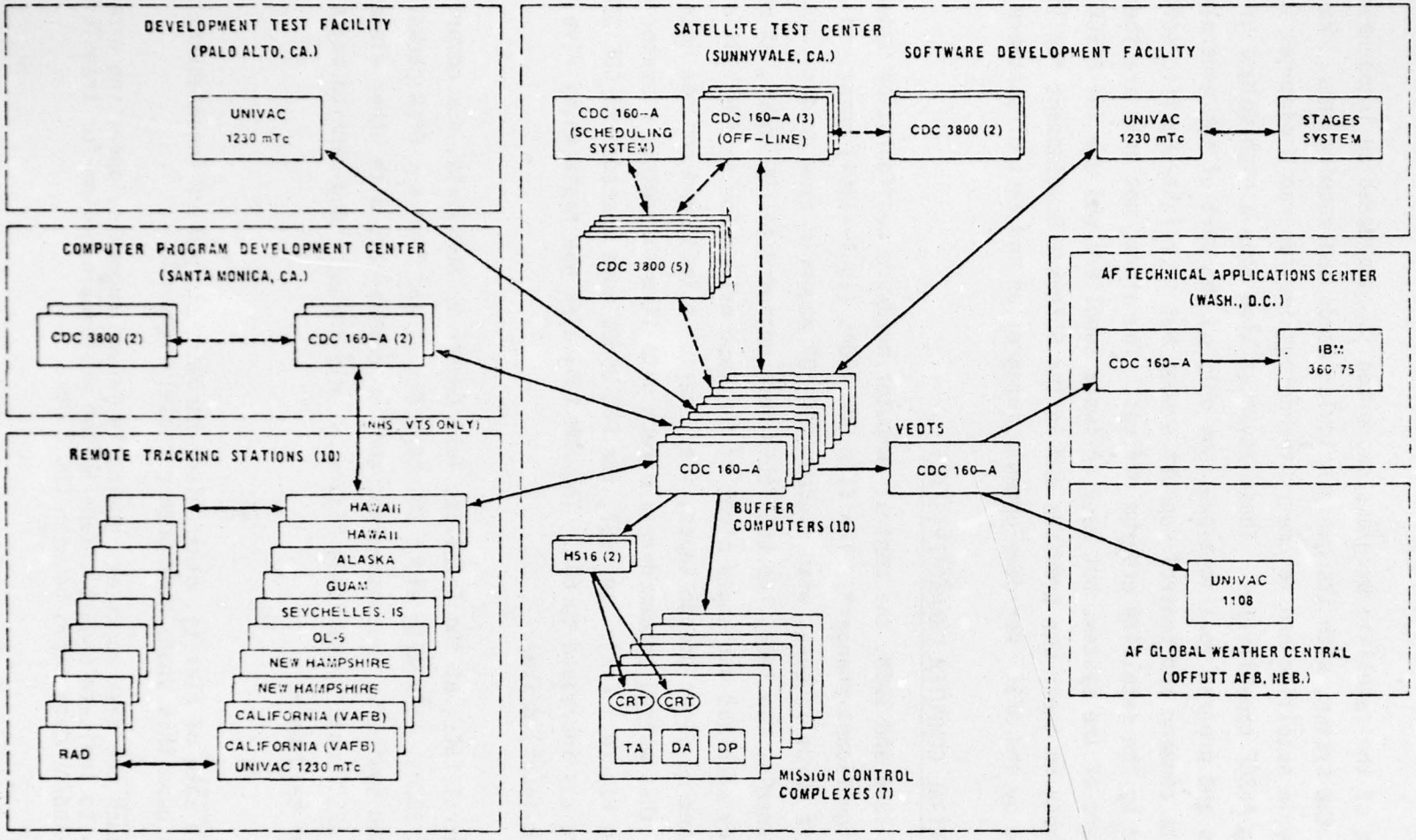


Figure 3. AFSCF Computer Resources, 1968

Each CDC 3800 has 65K of 48-bit words in core; most instructions are half-words (24 bits), while data and certain instructions, e.g., double-precision or chaining, occupy the full 48-bit word. As a result, 32K is the limit of addressing capability, but 64K of virtual core is made available through the address modification (AM) function performed by the CDC 3811 Relocation Module (see Figure 4). A price is paid in cycle-time when a program and its data exceed 32K words: the 0.8 user cycle time is increased almost 50 percent when the AM is used, i.e., "segment switching" occurs.

2.3 OPERATING SYSTEM CHARACTERISTICS

The pre-COMPOOL System Support Tape (SST) resident monitor, SMTC, was an operating system developed especially for the AFSCF command and control applications on the CDC 1604/3600 computers.

The Satellite Control for Operational Programs and Environment (SCOPE) system comprised the computer programs and which supported CDC 3600 Control Center Computer activities. The major components of SCOPE were the SCOPE Master Tape Control (SMTC) program which was the system executive program; the master tape, which contained the various elements of the system, including SMTC, and the operation requests which caused system actions.

SMTC processed system function requests, controlled the loading of required program elements, set up initial operating environments, controlled the operation of the requested function, provided input and output capability, provided data conversion routines, processed system interrupts, provided timing of functions, and performed miscellaneous utility operations (pseudo functions).

The master tapes were storage media for the elements in the system. There were two types of master tapes, the SST and the Flight Support Tape (FST).

A master tape, either an SST or FST, consisted of one file of records terminated by a logical end-of-tape mark (two ends-of-files). (See Figure 5.)

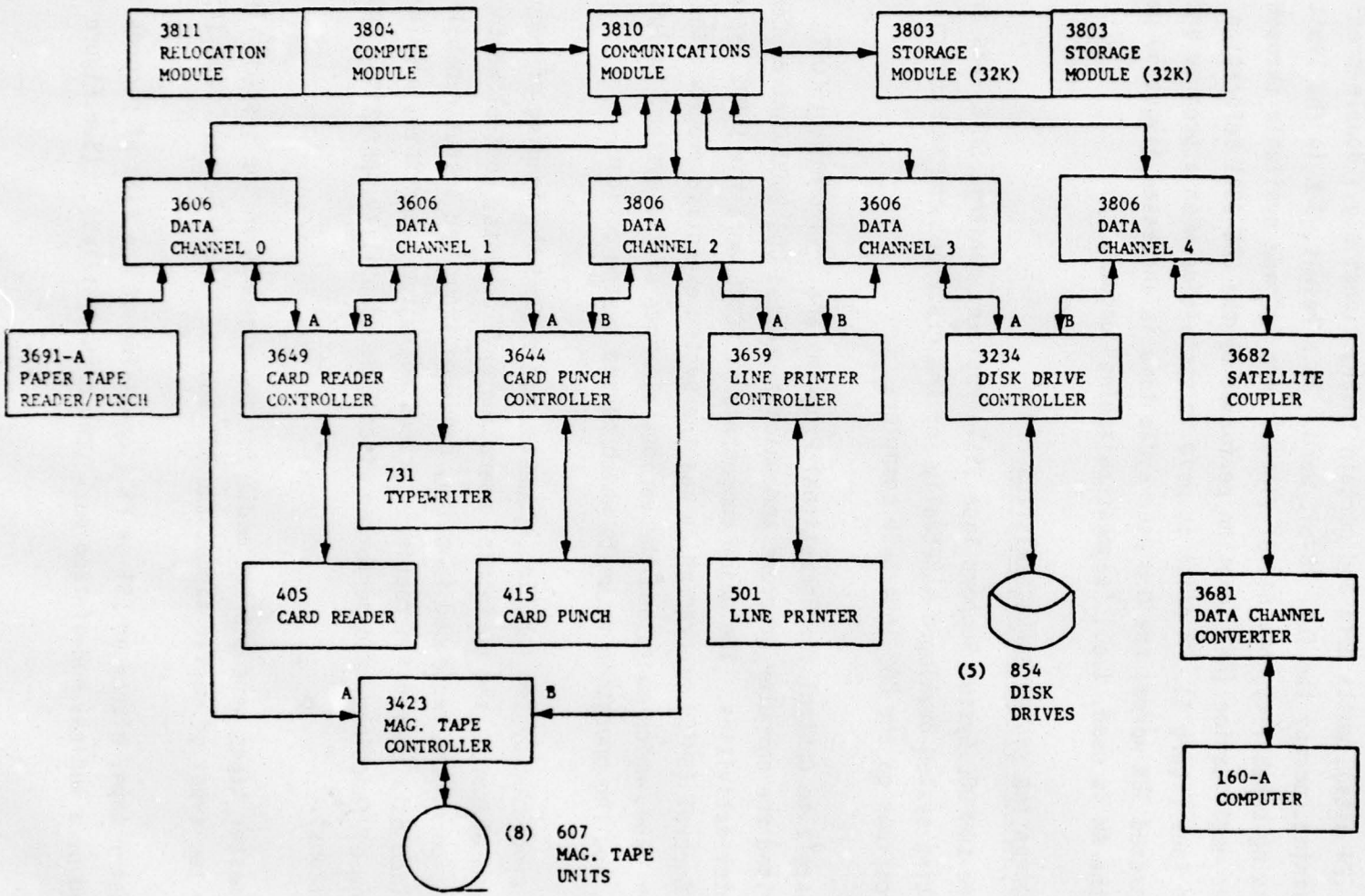


Figure 4. Central Computer(s) Configuration, 1968

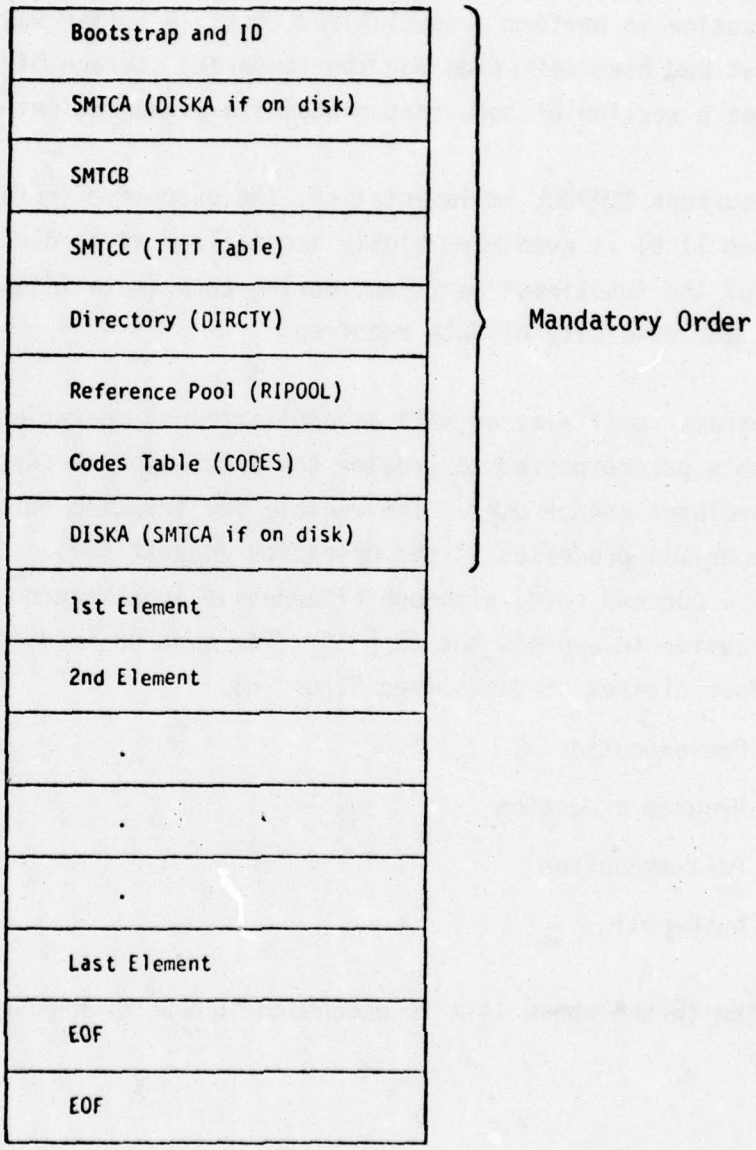


Figure 5. Format of Master Tape (SST or FST)

This file contained functions, subroutines, tables, and buffers, which were termed the elements of the system. A function was a computer program which could be called and initialized using the SMTC function request; it also could be operated by a calling sequence compatible with the one constructed by SMTC. A subroutine was a program which was called by an operating function or subroutine to perform a specialized task. A buffer was a section of core that had been set aside for the temporary storage of information. A table was a section of core that contains a permanent set of data [6].

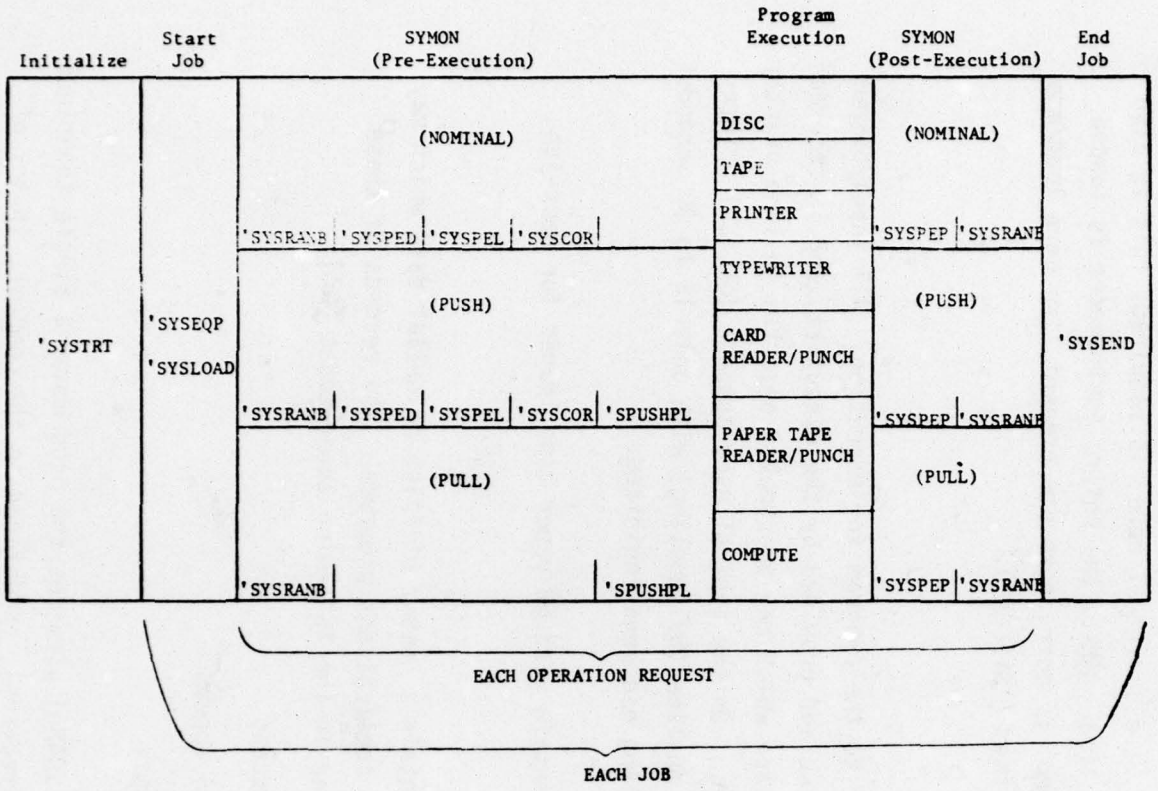
In the current COMPOOL implementation, the executive (referred to as SYMON or System II B) is even more highly specialized to handle the consecutive nature of the functions* performed during each (satellite support) job, as well as the diversity of data required.

The programs (utilities as well as applications) operating under System IIB are highly parameterized to provide the exact type of support required by each developer and/or user. The vehicle for invoking this precise subset of a program's processes is the Operation Request (OR). The OR is usually input via punched card, although alternative input sources may be used, and the executive interprets the OR [35]. For each OR in a job [35], SYMON performs four classes of tasks (see Figure 6):

- Pre-execution
- Program execution
- Post-execution
- Push-pull

Each item in the above list is discussed in the following paragraphs.

*Although the programs and their names are different in the COMPOOL implementation, the functional flow of satellite support remains largely the same as the pre-COMPOOL system.



Key

'SYSTRT = System Start (Initializer) Program
'SYSEQP = System Equipment Program
'SYSLOAD = System Load Program
'SYSRANB = System Request Analyzer Program
'SYSPED = System Program Environment Determinator Program
'SYSPEL = System Program Environment Load Program
'SYSCOR = System Corrector Program
'SPUSHPL = System Push Pull Program
'SYSPEP = System Post Execution Processor Program
'SYSEND = System End Program

Figure 6. Time-Line for 3800 Operations.

2.3.1 Pre-Execution

After interpreting the OR ('SYSRANB*) and setting up control/data tables for the called program, the total environment of subroutines and data is mapped into core. If segmentation, i.e., virtual core, is required, this is also computed ('SYSPED*) at this time. Next, the entire environment is loaded ('SYSPEL*) from disk to core. If correctors are present, the core load/map is modified to account for them ('SYSCOR*).

2.3.2 Program Execution

Control is then turned over to the program for execution. All input/output (I/O), however, is controlled and provided by the executive, and is recorded on the System Output (SO) tape which may be spooled onto the on-line printer or processed off-line later. On the typical user-support job, all printer data on the SO is produced on-line (by spooling) with only 15 to 30 seconds more job-time than if spooling had been inhibited.

However, the SO may also contain card or paper tape images for post-job, off-line production.

User data bases usually contain a number of files of serial data which may be input to or output from applications programs. The records of these files may be stored/accessed in the following three modes [34]:

- Random fixed (RF) access
- Random variable (RV) access
- Sequential (SA) access

The System Data Handler ('SDAHA) provides the programmer a simple technique of handling data, blocks (records), as defined in the compool, in any of these three modes.

*See "Key" in Figure 6 for program titles.

2.3.3 Post-Execution

General housekeeping chores are performed by the post-execution processor ('SYSPEP). One important task is the writing onto disk of any Guaranteed Data Blocks (GBLKs) whose contents were changed, i.e., updated, by the program which just operated. Another is setting all of core (except a small area of permanently-dedicated core) to zeroes.

2.3.4 Push-Pull

If the program is in the push-pull table, its entire subroutine environment is written and saved on disk before control is turned over to it. Most of the data environment is similarly copied*. Thus, when the program operates again during the same job, the environment determination and corrector handling is already done.

2.4 PROGRAMMING/UTILITY TOOLS

In addition to the usual assortment of dump, trace, list, etc. programs, the pre-COMPOOL configuration included both an assembler, SABER, and a limited JOVIAL (J3) compiler capability.

The key characteristic of all programs (whether SABER-assembled or JOVIAL-compiled) was relocatability, i.e., they were all compiled for a common starting address and dynamically relocated.

Although most applications programs were coded in SABER assembly language, the J3 JOVIAL compiler offered two programming modes: (1) an extensive procedure-oriented language, and (2) the incorporation of direct code for all or part of a program compilation.

The COMPOOL implementation has facilitated the inclusion of some important programming and utility tools. These can be classified in the following three categories:

*The exceptions, GBLKs and Index Data Blocks (IBLKs), are loaded anew for each operation, to ensure availability of the latest, updated data.

- Compiler-related tools
- Compool-related tools
- System analytic tools

2.4.1 Compiler-Related Tools

The J4 JOVIAL compiler contains enhancements that permit the use of external subroutines with both input and output parameters, as well as provision for alternate entrances and exits from programs. These enhancements provide a high degree of commonality of code to exist while enabling extensive parameterization of applications programs.

A complete set-use listing is developed for internal and external (compool) references by the program.

After each compilation, a "billboard" is presented which provides important statistical information and errors, as well as computer usage (see Figure 7) [34].

2.4.2 Compool-Related Tools

An important programming adjunct is the availability to development programmers of the compool-generating program, 'COMPOOL. This availability permits the applications programmer to generate his own override compool, thus relieving him of dependence upon the CPIC for updating the compool during program development. Of course, once the applications program is ready for delivery to the CPIC for integration, the override compool entries are represented by Compool Change Requests (CCRs) and are implemented into the official compool by the CPIC.

In addition, the program, 'LOOPMOC, will disassemble a binary compool into symbolic compool declarations, thus providing an important analytical tool during the development cycle.

Finally, the COMPOOL implementation supports the production of symbolic

```
                    SUMMARY OF PROGRAM  ≠SERRHST   MDGCA

SST USED.

COMPOOLS.ACTIVE.      ** SYS   COMSYSAF      ** AUX    NONE         ** DVR    NONE
  COMPOOLS USED.      ** SYS   COMSYSAF      ** AUX    NONE         ** DVR    NONE

JOVIAL COMPILER MODS.   * JGEN1 AM00   I JPOOL AM00   * JGEN2 AM00   * JTRAN1 AM00   *JTRAN2 AM00   *JLIST AL00
       ERRORS FOUND.      NONE             1           NONE           NONE           NONE          TOTAL ERRORS.        1
  NO. UNDEFINED TAGS.     NONE

*****************************************************************************************************************
*                                        *                                        *                              *
*NO. OF CARDS.            318             *UPDATE CARDS.         NONE            *DIRECT CODE CARDS.   NONE     *
*                                        *                                        *                              *
*NO. OF DEFINES.         NONE             *DECLARATIONS.          47             *STATEMENTS.           163     *
*                                        *                                        *                              *
*SYMBOL DESCRIPTIONS.     128             *STAT/DIMN TABLE.       15             *                              *
*                                        *                                        *                              *
*DECLARED DATA.           165  (00245 OCTAL)*JOVIAL DATA.         186  (00272 OCTAL)*TOTAL INSTRUCTIONS.   600  (01130 OCTAL)*
*****************************************************************************************************************
*                                        *                                        *                              *
*INSTRUCTION CELLS.       300  (00454 OCTAL)*TOTAL DATA CELLS.     351  (00537 OCTAL)*PROGRAM LENGTH.       651  (01213 OCTAL)*
*****************************************************************************************************************

                    ELAPSED TIME = H 0 M  0 S 46.984
```

Figure 7. JOVIAL Billboard

dumps which provide important time-savings in the debugging phase of program development [34].

2.4.3 System Analytic Tools

In addition to the symbolic core-dump capability, set-use and environmental listings, down to the item level, are available for entire master tapes [SST or Auxiliary Master Tape (AMT)].

Upon request, the executive also produces a core-map for each program as it is about to be loaded during a job [34].

2.5 APPLICATIONS PROGRAMS

2.5.1 Model Deliveries

Programs are delivered for applications use as a series of models and sub-models. Several iterations of the software system occurred during the transition between the pre-COMPOOL, "System I" system and the COMPOOL-sensitive "System IIB." (See Table 3 for a complete list of models during these years.)

Operating in the CDC 3600, the System I General Purpose Program Library was contained in an SST designated BESSTx, where x was the model, e.g., BESST11.5, which was superseded by Model 13, i.e., SST13.

In addition to the BESST Master, an FST provided program-peculiar software capabilities, on a satellite-specific basis, which interfaced with the BESST-resident monitor, SMTC. Although issues like sensitivity of data, uniqueness of requirements, and development/maintenance resource availability, sometimes modified it, the general policy was to incorporate existing general-purpose FST capabilities into a new BESST model in time to support a new Program Office or phase (launch series) within an existing program. FSTs, on the other hand, tended to be scheduled on a launch-by-launch basis.

The move into System IIB was gradual. First, System IIA software was introduced which used the CDC 854s to emulate the IBM 1301 which was, in turn, emulating the eight CDC 807 tape drives [3]. Finally, in 1968, the CDC 3800s

Table 3. Central Computer Masters and Dates of Delivery of AFSCF

MODEL	DATE	MODEL	DATE
5	3/31/65	SST13 (2)(3)	10/29/69
6	6/15/65	SST13A	12/10/69
7	9/30/65	SST13B (4)	-
7.1	1/12/66	SST13C	2/11/70
8	3/15/66	SST13D	3/10/70
8.1	4/15/66	SST13E	5/6/70
9	7/15/66	SST13F	7/24/70
10	10/14/66	SST13.1	9/25/70
11.0	4/15/67	SST13.1A (4)	
11.1	10/31/67	SST13.1B	12/17/70
11.2	3/12/68	SST13.1C	5/14/71
11.3	8/1/68	SST13.1D (5)	10/29/71
11.4	12/4/68	SST13.1D	1/7/72
BESST11.5	3/20/69	SST13.1E	6/28/72
BESST11.5A	-	SST13.1F	12/12/72
BESST11.5B (1)	6/5/69	SST15.0	7/13/73
BESST11.5C	8/15/69	SST15.0 (6)	10/8/73
BESST11.5D	12/12/69	SST15.0A	3/1/74
BESST11.5E	3/10/70	SST15.0B	11/14/74
BESST11.5F	9/25/70	SST15.0C	4/11/75
		SST15.1	8/6/75

(1) BESST11.5A and BESST11.5B delivered as one package.
(2) First operational COMPOOL model.
(3) Model(s) 12 were developmental models, not delivered for operational use.
(4) Not delivered for operational use.
(5) Pre-delivery directed (PRE13.1D).
(6) Re-delivery directed.

were put into (essentially) their present configuration. This consisted mainly of adding the CDC 3811 Relocation Module (AM)* [4]. System IIB software was now ready to be introduced and the CDC 854 desks were, at last, to be used as intended, i.e., as random-access storage devices (see Figure 4) [2].

2.5.2 Applications Program Characteristics

As was implied in Section 2.3, all functions, i.e., programs invoked by the user, operated in a consecutive (batch) mode, initiated by the SMTC-processing of an operation request input, usually by punched card(s). Some functions need be operated only once per flight, while others required numerous reiterations through the satellite life-cycle which itself might vary from a few days to several years, depending upon mission, orbital characteristics, etc.

The pre-COMPOOL general purpose functions fell into these categories:

- Pre-flight initialization
- Orbital/ephemeris computations
- Reset tape(s)
- Reset data in core
- On-orbit operations

In the COMPOOL implementation, the applications programs fall into two groups: (1) those that are general-purpose and can be used, through parameter-selection, by most or all Program Office users, and (2) those that are satellite or Program-Office unique; these appear on the SST or AMT, respectively.

---

*The CDC 1604/3600/3800 machines use numerous half-word (24-bit) instructions which place a 32K limit on the address portion. The AM permits the use of 64K of virtual core by an indirect address-modification technique.

The general-purpose programs perform a set of functions as follows [29, 35]:

- Prepass planning (see Section 2.5.2.1)
- Data base initialization (see Section 2.5.2.2)
- Orbit determination (see Section 2.5.2.3)
- Ephemeris generation (see Section 2.5.2.4)
- Orbit planning (see Section 2.5.2.5)
- Trajectory, mathematics, and general purpose (see Section 2.5.2.6)

Appendix B provides a detailed description of the Advanced Orbital Ephemeris System (AOES) programs and techniques that support these functions [29].

The project-specific functions usually include AMT programs for:

- Command generation (see Section 2.5.2.7)
- Attitude determination and control (see Section 2.5.2.8)

2.5.2.1 Prepass Planning

The interface between the FSCs (CDC 3800s) and the satellites is via the buffer computers at Sunnyvale, CA., and the Remote Tracking Station (RTS) computers (see Figure 2).

Data are transferred from the FSCs via a Prepass Tape which contains RTS-specific antenna-pointing, telemetry modes, command, and free-text messages (see Figure 8) [24]. The antenna-pointing data are computed from a vehicle ephemeris (V/E) which is generated from nominal flight data preflight, and from the actual orbit during orbital operations (see Section 2.5.2.4). The telemetry modes data are also generated by the FSC, using a special System I subsystem, MSTAC, on the TUX master tape (see Figure 8). Command data are generated by AMT programs on the FSC (see Section 2.5.2.6).

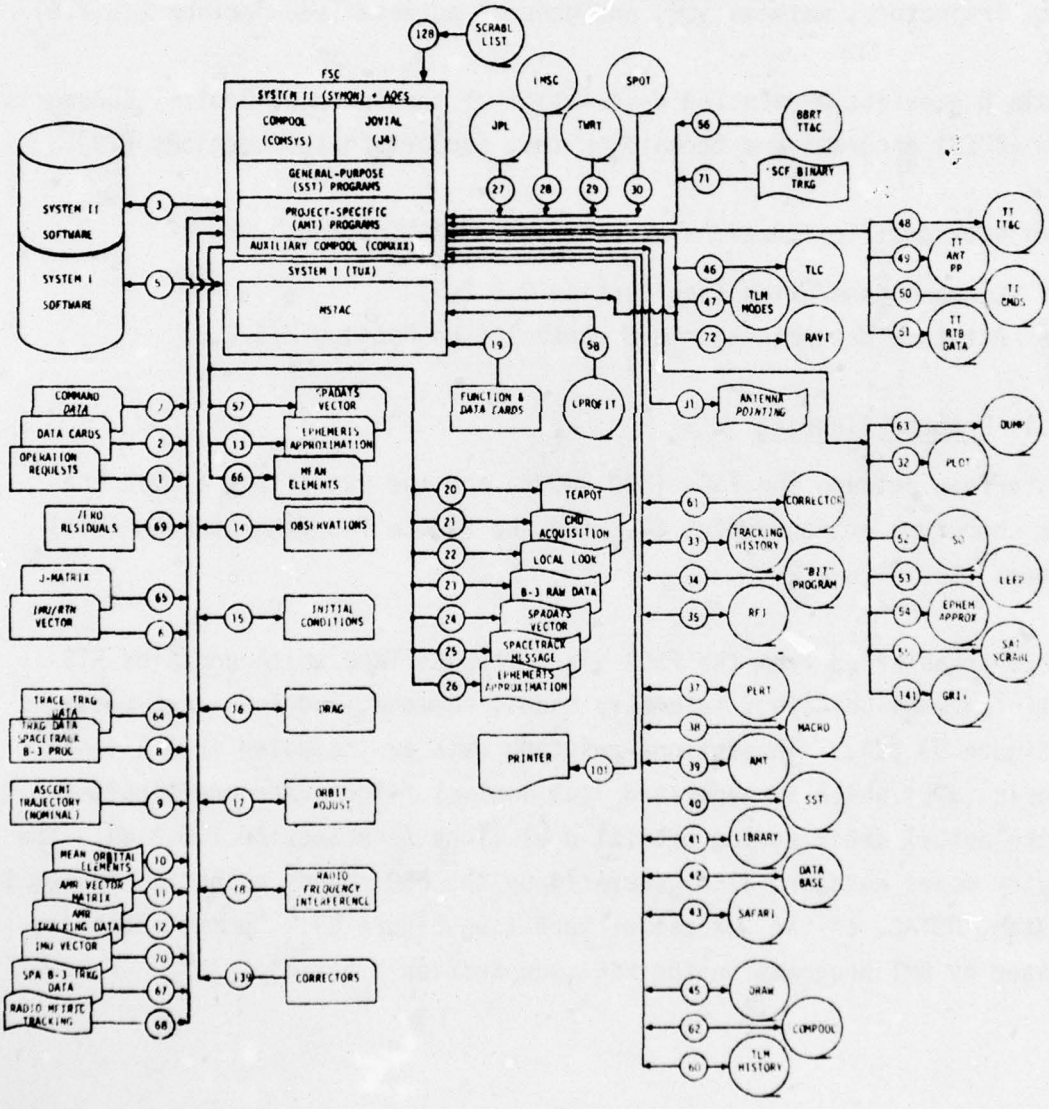


Figure 8. CDC 3800 Flight Support Computer (FSC)

2.5.2.2 Data Base Initialization

Although data base discipline is discussed in detail in Section 3.5, there are a set of applications programs that support the building and initialization for flight of the user's (satellite) data bases. Available programs will build a data base from cards, Library Tape, and/or an earlier Data Base Tape, list it, and make an item-for-item comparison of all the values in two data bases. The data in this case includes station coordinates, vehicle, fuel, and engine parameters, geophysical and atmospheric modules, special-purpose macros, etc.

Another class of data is then introduced to complete the preflight initialization. Supported by a series of special-purpose SST programs, these data include initial orbit conditions (state vector), drag and orbit adjust tables, and tracking data editing parameters (see Figure 9). It is to be noted that these same programs provide the capability to modify these data at any time after launch, as well.

2.5.2.3 Orbit Determination

This set of programs is by far the most mathematically sophisticated set on the SST. They fall into two functional groups as follows (See Figure 10) [29, 34]:

- Tracking data handling, including processing, editing and compacting.
- Orbit determination/differential correction.

2.5.2.3.1 Tracking Data Handling

Tracking data reports are accepted from any of the diverse types of tracking antennas and locations. These are normalized to common units and corrected for biases, refraction, etc.

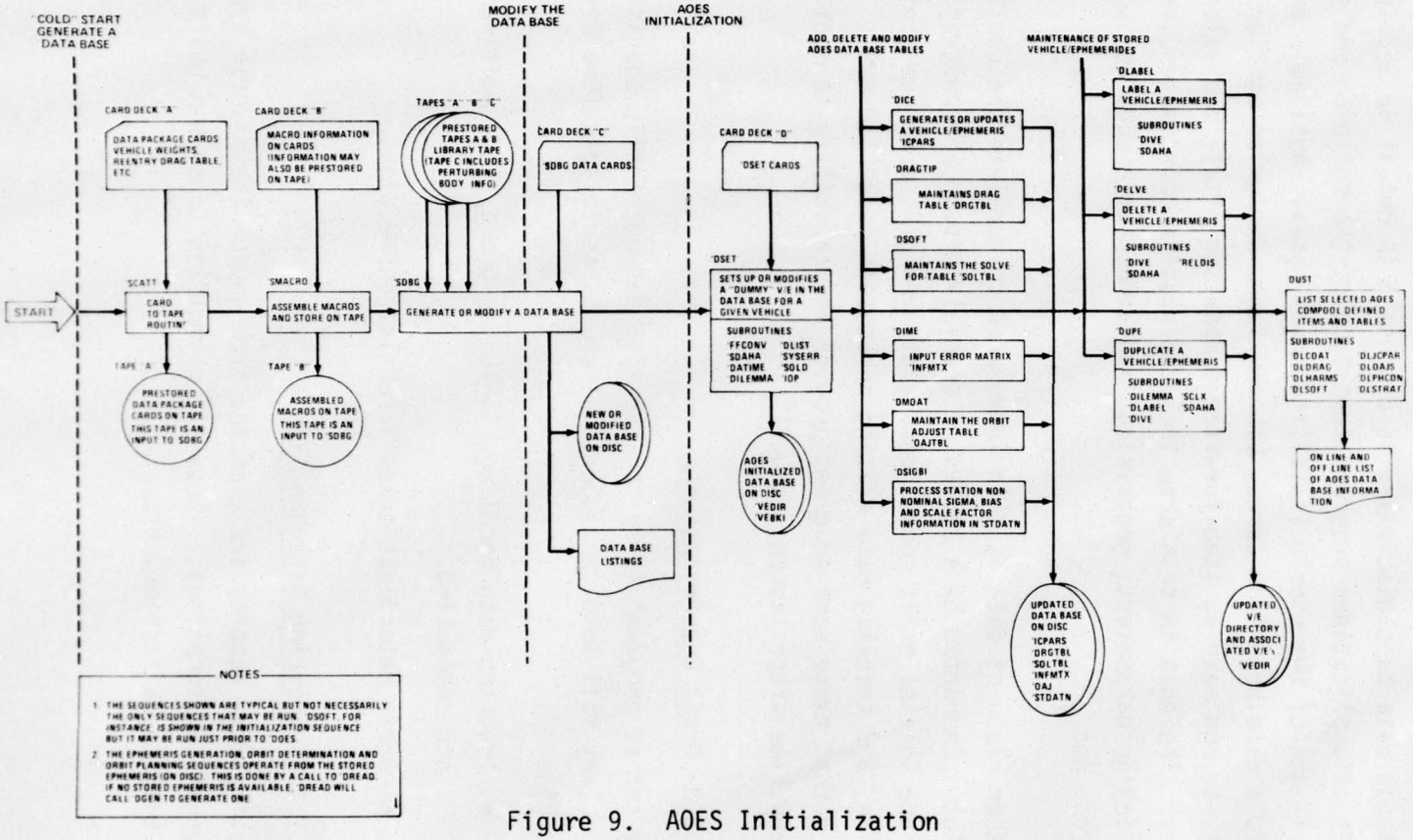


Figure 9. AOES Initialization

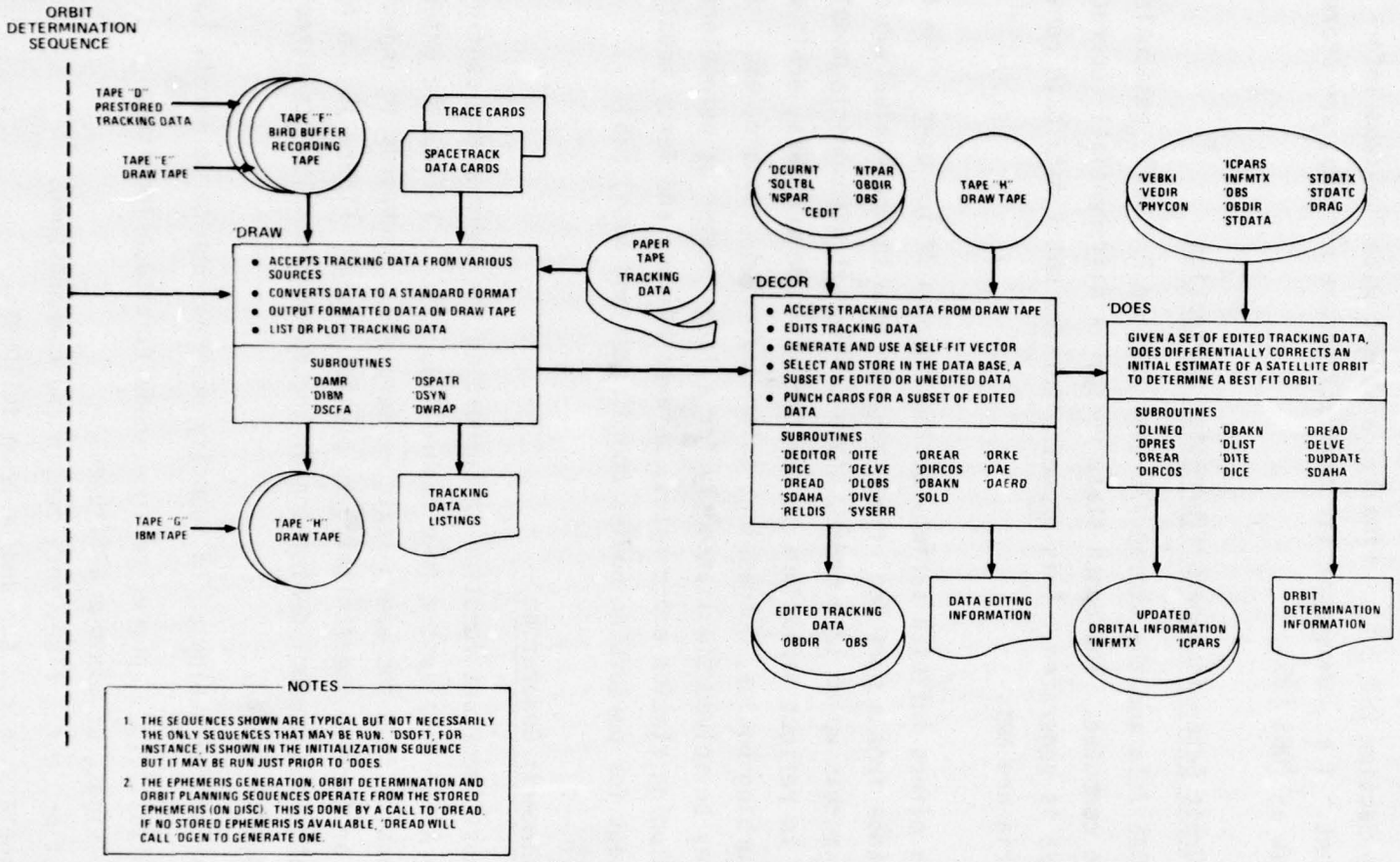


Figure 10. Orbit Determination Sequence

They are then edited for internal consistency based upon data base standard deviation (sigma) parameters. Next, they are compacted by a minimization of residual differences betweeen the actual and predicted values, i.e., from the V/E (see Section 2.5.2.4). Finally, a subset with "maximum information content," i.e., whose partial derivatives are extrema, are stored in the data base as 'OBS [35].

2.5.2.3.2 Orbit Determination/Differential Correction

Using the 'OBS file and the stored V/E (see Section 2.1.5.4), residuals and partials are computed. The epoch state vector is differentially corrected and a new V/E is generated. This process iterates until data base convergence criteria are met.

This process offers infinite variety to the Program Office user. The force model can range from a pure Keplerian two-body case to the evaluation of geophysical models up to 12th order, state-of-the-art atmospheric models, drag tables to reflect variations in vehicle attitude and mass, orbit adjustments, perturbing bodies, outgassing, solar radiation pressure, etc. Further, the model may be solved simultaneously for a large number of parameters (up to 24), including station coordinates, antenna biases, and force model parameters (except for perturbing bodies coordinates and mass) [35].

2.5.2.4 Ephemeris Generation

This is a most important function in supporting the command and control mission of the AFSCF data system (see Figure 11) [35]. A V/E can be extended bi-directionally from the epoch state vector, i.e., backwards to injection into orbit or forward indefinitely, to orbital decay. Figure 15, in Appendix B, illustrates the compool definition of the ephemeris file, its directory, and working-span buffer.

The V/E may then be evaluated to identify numerous ephemeris events [35].

- Orbital, e.g., apogee, perigee, nodal/latitude/longitude crossings, or minimum altitudes.
- Station, e.g., rise, set, zenith and command-span.
- Planetary, e.g., sun and/or moon eclipses.

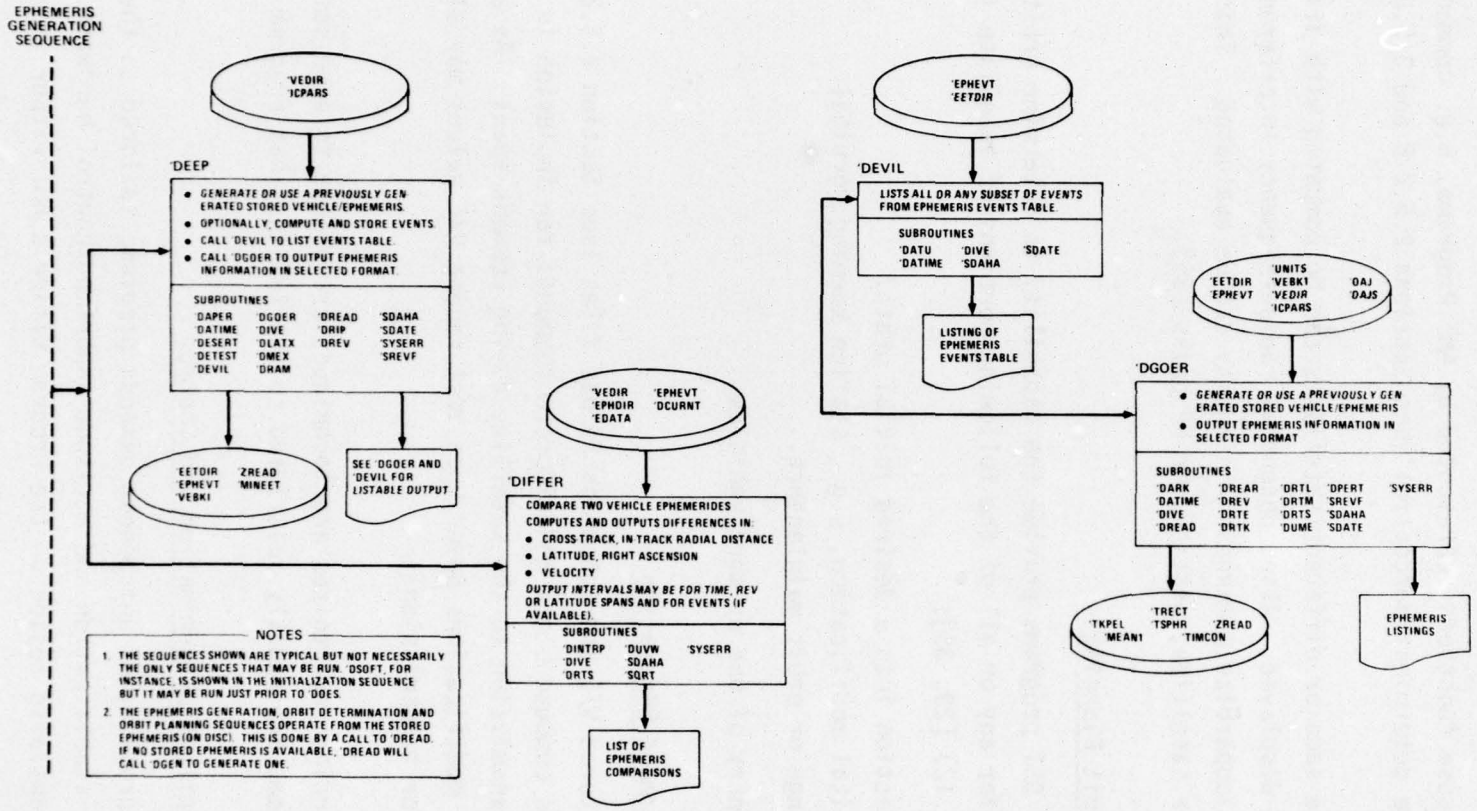


Figure 11. Ephemeris Generation Sequence

The V/E will also be used for orbit-planning (see Section 2.5.2.5) and for special-purpose functions represented by AMT Programs, e.g. command-generation and attitude determination/control (see Sections 2.5.2.6 and 2.5.2.7).

V/Es for the same or different satellites can be compared with differences in position displayed. Also, potential radio-frequency interference (RFI) that would jeopardize commanding integrity can be evaluated. This may be computed for satellite, station, or RF pairs [35].

2.5.2.5 Orbit Planning

This set of SST programs provide the capability to determine orbit adjust(s) parameters for any or all of the following portions of satellite flight (see Figure 12) [29, 35]:

- Injection into a desired initial orbit.
- Orbital modification, e.g., station keeping, orbital change or orbit maintenance.
- Reentry of one or more vehicles.

2.5.2.6 Command Generation

Using the stored V/E and Ephemeris Events files (see Section 2.5.2.4), these AMT programs compute block and discrete commands for inclusion in a Prepass tape and transmission to the satellite during command spans. As a flight progresses, modifications based upon most recent V/E values may also be generated for transmission.

Since commanding repertoires and techniques are unique to each satellite, these programs are usually restricted to the AMT for that Program Office.

2.5.2.7 Attitude Determination and Control

Again, the property of uniqueness demands programs tailored to the particular satellite's stabilization and attitude reporting/control hardware. Thus, these programs also reside on the Program Office's AMT rather than on the SST.

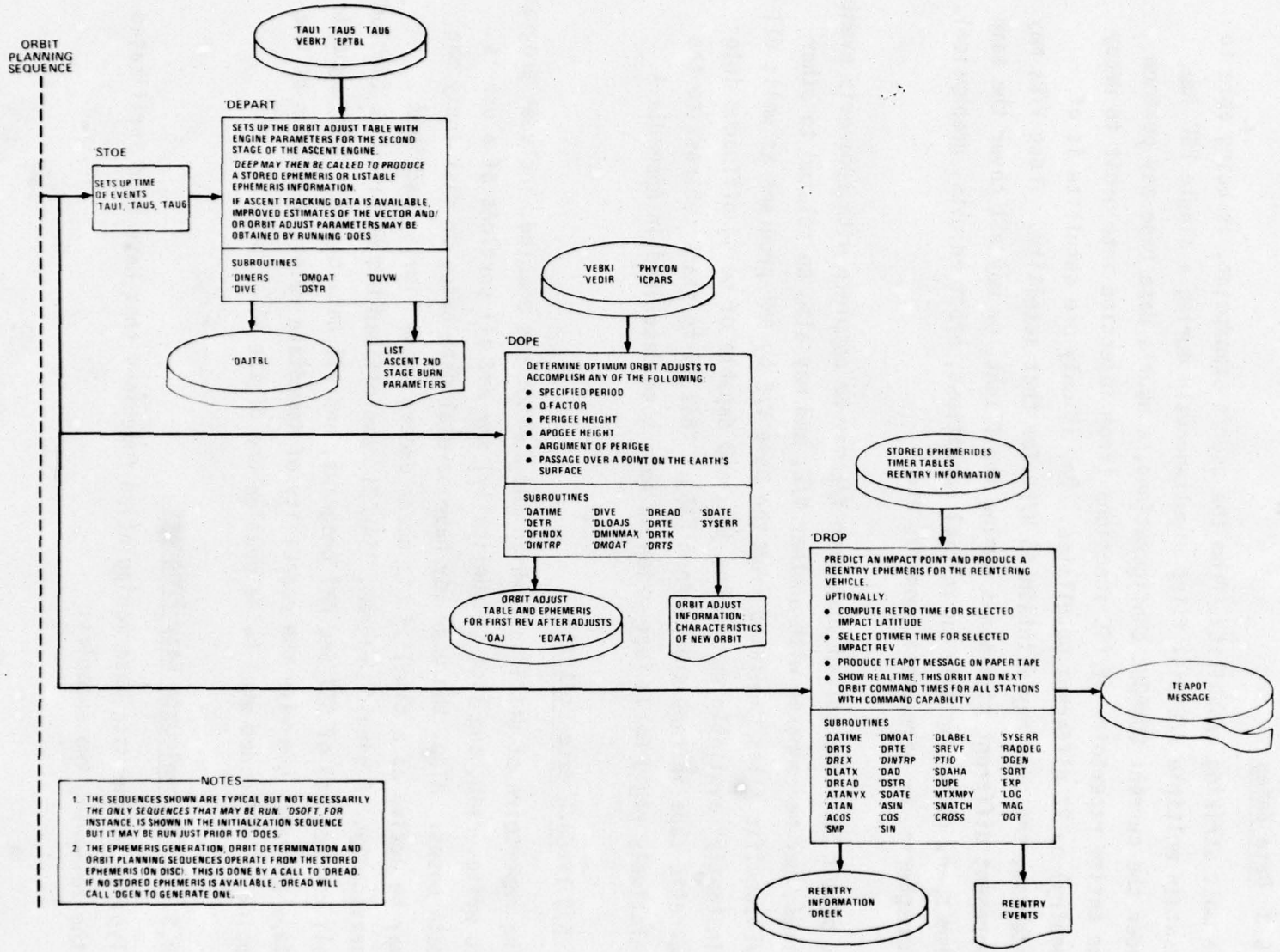


Figure 12. Orbit Planning Sequence

2.5.3 Data Bases

The most striking capability, from the user's standpoint, is being able to maintain multiple V/E disk files simultaneously during a single FSC job. Under the current COMPOOL configurations, a user's data base may perform the entire repertoire of FSC functions (from injection into orbit to decay reentry) on 22 different satellites. Or, if only one satellite is of interest, the user may maintain 45 V/Es for that satellite. These V/Es may represent different time-spans during the flight, or may all cover the same time-span, differing only in initial conditions, orbit adjusts, geophysical, atmospheric, or geomagnetic models, etc.

It should be reemphasized that each V/E may be complete with ephemeris events files, may be compared with another V/E, and may also be related to other V/E-specific files generated from the base V/E by AMT programs as well; all "instantly" available during the job, and capable of being offloaded into magnetic tape in recoverable form [35]. This is in stark contrast to the relatively rigid Reset Tape-oriented data base described in Appendix A.

2.5.3.1 Data Base Utilities

The repertoire of utilities, which the COMPOOL has enabled, includes programs to write, read, and list (symbolically) any and all portions of a user's data bases. Also, the user may have several data bases on disk (only one may be active at a time) if his needs exceed those limits described previously. A single program, 'SAFARI, can offload his entire disk (including all correctors) of SST and AMT programs, and all data bases. This magnetic tape provides him with the capability of immediate reinitialization at the point he offloaded when he is next on one of the FSCs [34].

2.5.3.2 Special Data Base Programs

Typical of the data base manipulation functions that have been facilitated are the following two examples:

- Any two data bases may be compared to the item (word) level. This comparison may be performed upon selected blocks or on the entire data base.
- V/Es which are disjointed, e.g., from successive orbit determination processes, may be conjoined ("splined") so as to eliminate the discontinuities.

3. COMPOOL IMPLEMENTATION HISTORY

The SCF COMPOOL was implemented between 1965 and 1970. This was a particularly dynamic period with respect to AFSCF data system changes that affected the FSCs. Unfortunately, the costs for the COMPOOL implementations are mixed with the costs for all the other changes and it has not been feasible to identify costs related specifically to the COMPOOL. The numerous changes fell into the following six basic categories:

- There were 27 operational SST models developed, integrated, and delivered.
- The computers were upgraded from CDC 1604s to 3600s, to 3800s, and finally to 3800s with a second 32K bank of core.
- The executive, SMTC (tape-oriented), was replaced by System I (pseudo-tape on disk orientation), and then by System II (random-access disk-orientation).
- The applications packages were changed from BMT based (closed-form computation), to General Purpose Tracking Programs/Orbital Ephemeris System (GPTP/OES) (numerical integration of variational equations/ephemeris based), to Advanced Orbital Ephemeris Subsystem (AOES) (analytic partials closed-form differential correction/ephemeris based).
- The data base discipline, as indexed by the CODES and TTTT tables and maintained on the Reset Tape, was changed by incorporating those tables into a compool with data maintained on disk pseudo-Reset Tapes. Finally, the entire data base was redefined into the current compool format with data randomly stored and accessed on disks.

- The SABER assembler was replaced by a SABER/JOVIAL joint capability, and finally by a machine-independent, compool-sensitive JOVIAL upgrade, J4, and all FSC programs were rewritten in J4 JOVIAL.

During this six-year period, the SDC CPIC effort encompassed the software products developed by a number of Computer Program Associate contractors (CPACs). Programs were being developed and maintained for the SST by numerous CPACs, including Mellonics, Laboratory for Electronics/Data Dynamics Inc. (LFE/DDI), Lockheed, and SDC. Programs were being developed and/or maintained for Flight Support Tapes (FSTs) by additional CPACs, e.g., General Electric, TRW, and Planning Research Corporation.

3.1 OPERATIONAL FSC MODELS

In general, since SST models could not be synchronous with all FST models, an "upward compatibility" was maintained. For example, an FST operating with SST Model 13 could also operate (interface successfully) with SST Model 13B. Of course, certain new capabilities in SST Model 13B might be unavailable to the FST user; this would be the Program Office's tradeoff decision regarding the cost of updating their FST. This decision would be conditioned by the operational life-cycle of the Program Office's satellite(s) as well as the importance to its support by the new SST features.

3.2 COMPUTER HARDWARE UPGRADES

The CDC computer upgrades which occurred between 1967 and 1968 fell into two phases. The upgrade of tape-oriented CDC 1604s to tape/disk-oriented CDC 3600s was reflected mainly in expanded instruction repertoires and faster cycle times. Achieving compatability in the existing software, while pervasive, was relatively simple and straightforward. This was not so, however, when the CDC 3600 to CDC 3800 upgrade took place. The CDC 3800

was not only a faster computer, but it had a number of instructions and capabilities that dictated a wholesale redevelopment of all the software, e.g., full-word, double-precision commands and chaining I/O techniques. In addition, the decisions to double the core storage from 32K to 64K, and to take advantage of the random-access nature of disk storage reinforced that necessity. The implications of these changes for executive, applications, data base, and programming language appear in the following paragraphs.

3.3 Executive System Modification

There were three identifiable phases in the modifications/development of the executive, as follows:

- System I pseudo-tape modifications
- System IIA disk file development
- System IIB disk file development

3.3.1 System I

A major modification comprised the ability to organize and access the disks as pseudo-tapes, i.e., serial data files. When the decision was taken by AFSCF not to redevelop the MSTAC telemetry-modes generation subsystem to operate under the COMPOOL, i.e., System II, System I remained viable for that use as modified for the CDC 3800s.

3.3.2 System IIA

Originally scheduled to support the full-blown COMPOOL, but with only 32K of core storage in the CDC 3600s, System IIA operated only in Model 12 which was declared to be a "developmental" model.* However, the stated requirements of the user community dictated the upgrade to CDC 3800s, the increase of 32K more core storage, and the redesign of the executive system into System IIB.

*One operational use was made of Model 12 and System IIA, known as the "'SIMPLE Model." In this case, an Ephemeris tape was generated on disk under System I (Model 11.5) and an FST was developed to read the file tape (under System IIA and the program 'SIMPLE.) The FST programs then employed the data on the Ephemeris tape in the System II manner, i.e., random disk files, compool-defined.

3.3.3 System IIB

The first operational model on the CDC 3800s under the compool concept, was Model 13, delivered to the AFSCF in October 1969.

The acquisition of the second 32K of core and the CDC 3811 Relocation Module (see Figure 4) required a complete redesign of the dynamic program environment loading module, 'SYSPED. In addition, the multiple V/E capability (see Section 2.5.3) necessitated complete redesign of the Data Handler, 'SDAHA, and the Data Base Generator, 'SDBG. Numerous other less drastic changes were also needed for the executive during this period of over a year before Model 13 could be certified as operational.

3.4 APPLICATIONS PROGRAMS DEVELOPMENT

The basic philosophy for organizing the applications packages was unchanged, i.e., the SST would comprise the executive, utilities, and general-purpose applications programs to support the initialization, orbit determination, ephemeris generation, orbit planning, and tracking-station communication functions. The FST, on the other hand, would continue to be ephemeris-based, and would comprise programs to support flight-peculiar functions, e.g., command and control, telemetry, or post-flight analysis.

At another level, however, there were two identifiable phases in the development of the SST programs:

- Orbit determination by numerical integration of variational equations.
- Orbit determination by analytic partials, closed-form, differential correction.

Originally both these capabilities were specified to be developed as selectable alternatives [29].

3.4.1 Variational Equations

While this approach to the orbit determination task provided greater accuracy from fewer iterations of the integrator, each iteration was appreciably slower than the analytic partials approach.

3.4.2 Analytic Partials

Extensive study convinced the AFSCF that it was desirable to trade off a slight decrease in accuracy, i.e., requiring more frequent differential correction tasks, for an appreciable increase in the speed of acceptable solutions.

This increased solution speed was most attractive to users. What had earlier been a 20-30 minute computer run now became a matter of 5-8 minutes. It had not been possible to perform the orbit determination process for more than one satellite during a computer "job" of manageable time-span [30]. Similarly, it now became feasible to compute and store more than one ephemeris per vehicle during the same job.

The effect of this speedup in computation upon the data base organization is discussed in Section 3.5.

3.4.3 Parallel SST and FST Development

It was pointed out earlier that SST and FST development customarily proceeded asynchronously. This scheduling luxury was not to be so readily available during the COMPOOL implementation, since the interfaces were numerous and complex. As a result, with the exception of the 'SIMPLE model, the Program Offices were compelled to continue to use System I as their basic SST interface. This is reflected in Table 3 in which Models 11.0-11.5D spanned over 2½ years and comprised nine model deliveries.

3.5 DATA BASE DISCIPLINE

The data base was completely redesigned in the transition from System I to System IIA. The redesign resulted from an extensive analysis of the existing System I data base discipline and its deficiencies. At this time, the acquisition of the second 32K core bank and all the related design changes, which resulted from the shift to System IIB, had little impact upon the data base definition and virtually none upon the data base design changes. In fact, the impacts were limited to immediately effecting the V/E disciplines, the redefinition of certain Guaranteed Data Blocks (GBLKs) as Variable Data Blocks (VBLKs), and the definition of VBLKs rather than Index Data Blocks (IBLKs) as subdirectories into a new compool.

The four areas in which the analysis effort was concentrated were:

- System-wide constants/communication facilities.
- Internally-defined program constants, and buffering of I/O and working space.
- External interfaces
- Subroutine call/return sequences.

3.5.1 System-Wide Data

This portion of the analysis concentrated upon the identification and classification of system-wide data-constants that were common to many users and classes of data that were frequently passed between programs. These data comprised both constant and variable data in item, table, or array format. Once the analysis had identified and classified the data, design efforts devised efficient storage structures and access mechanisms.

Constants were stable within the system and were generally used by more than one program; they were, however, independent of any particular satellite's requirements. Examples of system-wide constants included invariant numerical constants (π, e, factorials) and physical (station information) or geophysical (earth radius, atmospheric models) data which might require change from time to time [7, 8].

The types of variable data were, of course, much more numerous, and were usually satellite-specific. Examples included vehicle information (physical or thrust characteristics), orbital ephemeris data, and command repertoires [13].

3.5.2 Internal Data and Buffers

This analysis made an item by item review of data elements and memory management techniques looking specifically for redundancies, obsolescence, and handling inefficiencies. Design effort were directed toward definition of standard names, values, and structures for data elements and the definition of standard sizes, structures and handling of I/O buffers and working spaces.

The most visible example of internally-defined data was KREFPOOL which is described in Appendix A [9]. There were, however, numerous redundancies and some completely obsolete tables unused by any program. Frequently, redundancies were found to exist only because the same item was stored in differing units, e.g., seconds vs minutes, or English vs metric measures [13].

It has earlier been mentioned that numerous constants, e.g., the value of pi, were redundantly declared, even within the same program. In addition, there were whole tables, e.g., atmospheric or geomagnetic models which were internally stored in subroutines and were thus inaccessible for use by any other subroutines [13].

Buffers presented another problem area for analysis. For example, a number (over half a dozen) of programs read-in or wrote-out satellite-tracking data report records. The internally-defined buffers varied from one thousand to ten thousand 48-bit words, and on occasion, two such routines would be in core simultaneously, i.e., one was called as a subroutine by the other, and each would have a different-size buffer for the same data [14].

3.5.3 External Interfaces

This analysis centered on defining the interface requirements of data inputs from, or outputs to, such other agencies as the Space Defense Center, Atlantic Missile Range, NASA, and NATO. Over 50 interfaces were analyzed on a bit-by-bit basis to ensure that the integrity of the interfaces would be preserved. Tables 4 and 5 provide lists of these external interfaces from a recent publication [31].

3.5.4 Subroutine Calling Sequences

This analysis systematically examined all subroutine calling sequences in System I to determine their characteristics and to classify them by type. The design effort selected one calling sequence as standard in converting programs to System II operation.

The System I SST typically comprised over 300 routines, the bulk of which were subroutines called by approximately 50 user-invoked programs; the typical System I FST comprised over 200 routines with subroutines and user-invoked programs in about the same proportion, i.e., approximately five to one.

Calling sequences were quite varied in type and included the following:

a. No parameters.
b. A variable number of parameters which may occur in any order but only one parameter per computer word (unpacked) which will be called only internally.
c. A variable number of parameters which are in a packed format.
d. A fixed number of parameters which occur in a fixed order but are unpacked.
e. A fixed number of parameters which occur in a fixed order but are packed.
f. The use of the console registers as input.
g. Same as item b, above, except the element is called operation request.

Table 4. Summary of AOES Inputs

	PUNCH CARD		MAG TAPE		PAPER TAPE	
	SPEC.	PROG.	SPEC.	PROG.	SPEC.	PROG.
EPHEMERIS						
ASCENT/INJECTION	CSAE	'SAE	MOSAE	'SAE		
LAUNCH TRAJECTORY			MOTWRT	'SAE		
INITIAL CONDITIONS						
SPHERICAL	CPUNCH	'DICE			PPUNCH	'DICE
EQUATORIAL GEOCENTRIC CART	CPUNCH	'DICE			PPUNCH	'DICE
ECLIPTIC GEOCENTRIC CART	CPUNCH	'DICE			PPUNCH	'DICE
KEPLERIAN	CPUNCH	'DICE			PPUNCH	'DICE
EARTH FIXED	CPUNCH	'DICE			PPUNCH	'DICE
MEAN ORBITAL	CPUNCH	'DICE			PPUNCH	'DICE
ATLANTIC MISSILE RANGE	CPUNCH	'DICE			PPUNCH	'DICE
F AND G	CPUNCH	'DICE			PPUNCH	'DICE
SPADATS					PSPADAT	'DICE
IMU & RTN VECTORS	CIMU	'DIME			PIMU	'DIME
J-MATRIX	CJMAT	'DIME				
TRACKING DATA						
ZERO RESIDUAL RADAR	CZERO	'DZERO				
SPACETRACK B-3	CSPATR	'DEFORM				
TRACE	CTRACE	'DEFORM				
ATLANTIC MISSILE RANGE					PAMR	'DEFORM
NASA GOLDSTONE					PNASA	'DEFORM
PRESTORED			MOSPOT	'DEFORM		
STRIP			MOSTRIP	'STRAW		
BIRD BUFFER RECORDING			MOBBR	'DEFORM		
PLANETARY PERTURBATIONS			MULIB	'SDBG		
			MOPERT	'SDLG		
DATA BASE						
GENERATION			MUDB	'SDBG		
MODIFICATION			MULIB	'SDBG		
TELEMETRY AND COMMAND						
TELEMETRY MODES			MOTRANS	'SUPPORT		
SINGLE & BLOCK COMMANDS	CCDC	'SACRED				

Table 5. Summary of AOES Outputs

	PUNCH CARD		MAG TAPE		PAPER TAPE	
	SPEC.	PROG.	SPEC.	PROG.	SPEC.	PROG.
EPHEMERIS						
POLYNOMIAL APPROXIMATION	CEALS	'DEALS	MDEALS	'DEALS	PEALS	'DEALS
STATION AND REV XINGS			MOLEEP	'DUST		
STATE VECTORS						
ECI SPHERICAL	CPUNCH	'DPUNCH			PPUNCH	'DPUNCH
EFR CARTESIAN	CPUNCH	'DPUNCH			PPUNCH	'DPUNCH
ECI CARTESIAN					PPUNCH	'DPUNCH
F AND G	CPUNCH	'DPUNCH				
MEAN ELEMENTS	CRTM	'DRTM				
TRACKING DATA						
SPACE RACK B-3	CSPATR	'DEFORM			PB3T	'DEFORM
STC B-3 RAW					PSPAOBS	'DPUNCH
TRACE	CTRACE	'DEFORM				
EDITED OBSERVATIONS	COBS	'DECOR				
NASA GOLDSTONE					PNASA	'DEFORM
LOCAL LOOK					PLL	'SEND
SCF			MOBBR MODRAW	'SUPPORT 'DECOR	PSCF	'DEFORM
PRESTORED			MOSPOT	'SPDT		
SIMULATED	*	'SIMDATA	*	'SIMDATA	*	'SIMDATA
ANTENNA POINTING						
ANTENNA POINTING (SCF)					PSANT	'SEND
ANTENNA POINTING			MOTRANS	'SEND		
COMMAND ACQUISITION					PCOMAC	'SEND
SPACETRACK					PSPACE	'SEND
REENTRY						
TEAPOT incl. Air Space Reserv. Force Deploy.					PTPOT	'DROP 'DEPLOY
RADIO FREQUENCY						
RFI PREDICTION	CDRFI	'DRFI	MODRFI MOSAT	'DRFI 'DLASTIC		
DATA BASE						
SELECTIVE RETRIEVAL			MUDB	'SDBR		
FULL RETRIEVAL			MUDB	'SDBRTRV		

*'SIMDATA will output all tracking data types/media acceptable as input to AOES programs

For System II, the fixed, unpacked (item d, above) type of calling sequence was selected as standard for all subroutines. Thus, all subroutines except those using no parameters (item a, above) were to be modified. Numerous internal buffers were to be replaced by compool-defined tables/arrays to allow direct access to those buffers by other routines [19].

3.6 JOVIAL COMPILER MODIFICATIONS

The primary attribute, compool-sensitivity, was already present in the J3 JOVIAL compiler available through other Air Force contracts. The changes to produce J4 JOVIAL were relatively modest, and included [19, 32]:

- Enhancement to allow external subroutines to have input and output parameters.
- Enhancement to permit alternate entrances to subroutines.
- Provision of encoding/decoding capabilities.
- Acceptance of SABER code as JOVIAL 3800-compatible "direct code" in compilations.
- Restriction of all I/O processes to system I/O only.
- Provision of a "load-and-go" capability.

4. COMPOOL OBJECTIVES AND RESULTS

The objectives for the implementation of the COMPOOL fall into the following: three main areas:

- Simplication of the tasks of design, development, and maintenance of central computer software.
- Improvement in the throughput and accuracy of existing AFSCF hardware resources.
- Facilitation of more positive configuration management and interface control of the AFSCF data system.

4.1 WORK SIMPLIFICATION

Programming for the AFSCF data system presents some difficulties not inherent in most sofware development tasks. First, it is a complex, high technology system, not easily grasped. Second, new and modified programs must be compatible with this system and integrate into it with minimum perturbation. Third, there are many software suppliers whose products must integrate. Fourth, the volume of system changes is very high, so that the programmer is aiming at an unstable target. These conditions present many possibilities for errors and incompatibilities.

Analyses revealed the following four areas of potential improvement in programming:

- To provide a tool for communication between the elements of the system
- To ease the task of the programmer
- To guarantee the integrity of data
- To facilitate program maintenance

4.1.1 Intra-System Communication

The principal mechanism for communication between subsystems in System I was through the writing and reading of the Reset Tape, an inefficient mechanism. While it was true that this tape provided a "reset" or recovery capability between jobs, i.e., satellite-specific batches, it was constantly being read and written (updated) as functions operated within a batch job. In fact, the tape maintenance program, RESET, was in the environment of every operational function, at least to read, if not also to write the Reset Tape.

The Special Operating Modes (SOMs), e.g., SOM3, were an approach to reducing this extensive tape manipulation which was so costly of computer time. Portions of the inter-function communications needs were accommodated by the compool-like KODMTABL and KREFPOOL (see Figure 14 and Table 6 in Appendix A) so long as functions (identified by names Kx--x) operated successively. However, whenever another (non-Kx--x) program intervened, the RESET function was required to precede it to insure the storage of the data that had been built up in KODMTABL into Data Block 4 of the Reset Tape.

In addition, the confidence level in both hardware and software frequently dictated the writing of a Reset Tape even within a string of SOM3 function operations to guarantee recovery without having to repeat a tedious preceding operation.*

Better communication between system elements was provided by adopting:

- A fixed, standard calling sequence.
- Defined data blocks of varying levels of control.
- Standard data structures.
- Standard, compool-defined I/O buffers.

Using several different calling sequences caused interface incompatibilities. While it did not make a great deal of difference in system efficiency which calling sequence was used, adopting a standard interface simplifed program interactions and reduced coordination requirements.

Data of different permanence and usage frequently occurred in the same data block, resulting in inefficient transfers and accesses. Data blocks and varying levels of protection and were often lost between programs. Data were blocked into three levels of control:

- Constants and control information in read-only storage.

*It was not uncommon to use 30 or more minutes to perform an orbital correction (KODM) function.

- Guaranteed or provisionally guaranteed data that the operating system always saved.
- Temporary working space for communications and interfaces.

To simplify data structures, standard definitions of items, tables and arrays were established.

Without standards, individually defined I/O buffers were of many different sizes, made inefficient use of storage, and caused many interface incompatibilities. Standard buffers were established and defined in the compool, greatly simplifying interface interactions.

4.1.2 Programming Simplifications

Prior to the compool implementation, each individual programmer was responsible for declaring the data elements that his program used, no matter how often those data items appeared elsewhere in the system. Furthermore, data items definitions remained local to the program and were not available for reference by other programs in the system.

The immediately obvious programming advantage of a compool would be to relieve the programmers (of the several hundred applications routines) of their concern with global data location and format. Thus, beyond the initial declaration of any new external data into the compool, the programmer could more heavily concentrate upon the logical and mathematical aspects of his program design, secure in the knowledge that the (JOVIAL) compiler would provide any necessary masks, shifts, or conversions necessary to use the data as he intended.

This is not to say that the programmer was to be completely relieved of the responsibility for data, since he was required to submit Compool Change Requests (CCRs) for the original declaration of data, and these were to reflect efficient data organization for his (and others') intended use of the data [12].

Among the additional simplifying characteristics were:

- Where packing of data items for storage was indicated as appropriate, the compiler could perform this task, as well as generate the logic required to unpack the data for use.
- The self-documenting aspect of COMPOOL would reduce the programmer's documentation task.
- For debugging and checkout, COMPOOL would support dumps of storage in symbolic format, rather than binary or octal.
- To simplify the construction of similar tables, arrays and blocks by implementing a "like" declaration capability.

As a result of these considerations, the programmer's task was simplified by:

- Reducing the number of data declarations required
- Predefining working and communication space
- Automating load and link operations
- Providing a symbolic corrector capability
- Providing automatic units conversions

Since many of the data elements used are defined centrally in the compool, the number of data declarations the programmer must make is reduced. The number of declaration errors committed is also reduced. Standard, compool-defined working and communication space removes some planning and programming effort and prevents some interface incompatibilities.

The dynamic linking and loading capability of the operating system using compool definitions of program and data blocks eliminates loading and linking instructions and provides a fast load and go capability during program development.

The symbolic corrector capability not only eliminates much clerical work in preparing program correctors but provides an efficient alternative to frequent recompilations.

The automatic units conversion permits data to be displayed or entered in any mode desired while preserving standard units internally. Programmer effort in creating and using conversion routines is greatly simplified.

4.1.3 Data Integrity

It was expected that the COMPOOL would resolve several issues bearing upon data integrity. First, since the selection of internal units of measurement was made by the programmer(s), there was little uniformity. Time was expressed in seconds or minutes, angles were handled in degrees or radians, while units of length ran the gamut of English and metric forms. This diversity guaranteed the existence of code for units-conversion in routine after routine, to transform input data into the format required for internal processing.

This absence of definitional control even presented problems in output, where one would expect user specification to prevail. For example, coordinate systems used to express vectors varied, even to ambiguity in such factors as flight-path-angle, longitudinal direction, and internal system reference-time [13].

Finally, the turnover in maintenance programmers over time had resulted in some striking programming peculiarities. For example, in one large ephemeris computation program, the value of pi was declared as a constant 19 times (named PI, PIE, PIVAL, VALPI, PVAL, GPI, KWPI, STPI, etc.) in precision ranging from six to sixteen decimal digits (the computer could handle only 9+digits), and with one value declared as incorrect in the sixth decimal digit.* In the total SST, pi was declared over 200 times, again in varying degrees of precision. This situation obtained in numerous other cases, due no doubt to disinclination on the part of the maintenance programmer to search the program for the existence of the constant he needed at the moment. When the constant was a function of pi, e.g., 2π, $\pi/2$, π^2, $e\pi/2$, the proliferation increased even more.

*It was common practice for programmers to prefix their initials to tags they originated, probably for ease of future reference.

To guarantee the quality and integrity of data, a very important consideration in such a high technology system, the compool control operation provides a centralized source of data and data definitions with closely observed standards. Actions to guarantee the integrity of data include:

- Distribution of library tapes containing constants and global data definitions to all users.
- Distribution of data bases with standard data.
- Guaranteed saving of specified data blocks.
- Central, controlled definition of all global data.
- Standard, control data formats.
- Standard units of measurement.

Central definition and control of all constants guarantees the accuracy and precision of the values of the constants and avoids redundant, individual definitions with attendant opportunity for error. Central control and distribution of standard data prevents insertion of non-standard, inadequately inspected data blocks being used to support development or, more importantly, real operations.

Guaranteed saving of designated data blocks avoids the inadvertent loss of data between programs due to program error or failure. This ensures that data presumed updated by a previous operation will, in fact, reflect the latest maintenance run or operation. Provision for inhibiting the guaranteed updating is made for operational gaming and other tentative results that should not necessarily replace "real" data.

Central control of global data definitions permits stringent quality assurance measures to be applied to ensure efficiency and proper classification and to avoid redundancies and errors. Standard formats simplify programming and quality assurance measures as do standard internal units of measure.

The implementation of COMPOOL was expected to improve and extend this dynamic loading of program environment by providing:

- Expanded, controlled overlay of data items, tables and arrays.
- Reduction in the amount of permanent (absolute) core requirements.
- Relocatability of the entire program environment, including correctors and/or "bit" (binary) program tapes.

The control and conservation of storage, a very important objective in view of the very limited capacity of the FSCs, was improved by:

- Commonality of data structures
- Extensive overlay capabilities
- Compool hierarchies
- Efficient segmentation algorithms
- Policing current core allocations

Insisting upon maximum utilization of common data and eliminating all unnecessary private data keeps the demand for storage down. The overlay capability makes efficient use of storage, keeping the amount of resident, but inactive core at a low level.

Similarly, a hierarchy of compools is used: a system compool, always core resident, contains data common to the system and all users; project compools with user-specific data; and temporary compools used for development only. This hierarchy ensures that only those data definitions necessary to the current task are in core while it operates.

The core configuration and addressing arrangement of the AFSCF computer makes segmentation a critical factor in efficient use of storage. Data blocks are segmented to fit into available core and the dynamic loading and overlaying provides a near 'virtual storage' capability.

4.1.4 Program Maintenance

Programs in the AFSCF were in a continual state of change, much of which required revision of data elements to accommodate new parameters or the adjustment of the value of constants. Much reprogramming had to be done to reflect data changes with little guarantee that all affected routines would be detected and maintained. Programs were apt to pass through the hands of a number of programmers (and contractors) for maintenance over the years. It was expected that the COMPOOL would simplify program maintenance by at least:

- Substituting recompilation for reprogramming in cases of data modification or restructure
- Facilitating the use of symbolic correctors for temporary (or minor semi-permanent) program modifications or corrections.

These expectations have been successfully realized in System IIB. Program maintenance is facilitated by:

- Using a compool-sensitive compiler
- Well-defined interfaces
- Numerous analytic aids

The compool-sensitive compiler permits updating all programs impacted by a data change by redeclaring the data element once in the compool and recompiling the affected programs. Individual program changes, with attendent opportunity for error, are avoided.

The establishment of standard calling sequences and data interfaces tends to reduce the number of programming choices, make program interactions easier to understand and maintenance simpler. The simpler interfaces have reduced the number of errors.

The analytic aids include both symbolic and disassembled binary compool listings, listings of the characteristics of a compiled program, system and program set-use tables, environmental listings down to the table-item level,

and core maps, if desired, for every dynamic load. These aids make it easy to locate data errors, avoid duplicate data and program names, and understand the structure and operation of the programs.

4.2 RESOURCE UTILIZATION

The desire to achieve efficient utilization of computer resources is given special impetus by the very limited amount of core storage available in the CDC 3800's. As an index of the success in attaining this objective, it may be noted how the system has grown in size, sophistication and throughput without a major upgrade in the computer resources available during the period studied (see Section 2.1).

The analysis of the problems in this area revealed four objectives that could potentially be attained:

- Control of allocation of storage
- Eliminate the need for programs to communicate in a fixed area of core
- Eliminate redundant or obsolete data definitions
- Control/reduce data environments of programs

4.2.1 Storage Allocation

Since all programs, and a limited amount of data (see Figure 13 in Appendix A) were relocatable, core allocation was dynamic for those relocable elements. Thus, the addition of a program, or the substitution of a modified version, onto the SST did not require a "link-edit" run. Instead, the executive (SMTC) recomputed the core-map whenever a program was to be operated.

The dynamic loading also enables the regular policy of core loads to ensure the elimination of redundant and obsolete data. Core loads for a task may be fitted to the specific needs of the run and not general loads to fit all options for a task. The dynamic loading makes the core reallocations easy and core maps permit monitoring loads for efficient utilization.

4.2.2 Fixed Intercommunication Locations

Jobs run on the CDC 3800 FSC were typically a mixture of programs operating in SOM and programs operating in normal SMTC mode (see Appendix A). Programs operating under SOM ran in a core environment that was, to a large extent, fixed, whether the programs required all the tables or not. For instance, depending upon the mode, environments over 6K or 9K respectively were dedicated.

The problem presented by the colocation of data tables and control words was described in Reference 14 of Appendix E as follows:

> "Basically, a control word is related to some aggregation of data and contains parameters which govern the use of those data. In this consideration, the simultaneous presence of both control word and data is required. But many control words are set by programs (or options) other than those which will use the data, at least during the particular operational phase. If the control word is embedded in the data an obviously wasteful use of core results during the setting phase.
>
> Examples are legion in the present system: the entire 3140-word Timer File 3 must be brought into core to enter selected events into the 10-word ACMSON Table; the entire 1548-word Data Block 2 must be brought into core to enter control information into the 50-word Station Requirements Table.
>
> The decision, then, is to disassociate the control words from the data; the former will reside more or less permanently in core, while the latter will enter core only for data manipulation."

Communication of data between routines was accomplished by buffers which were defined for compilation in the calling or called program. The sizes of these buffers, for instance, was usually determined unilaterally, i.e., as optimal (at the time) in the opinion of the programmer designing the program-containing the buffer.

This tended to work aginst the economic constraint of using common, i.e., existing, routines in new software development whenever feasible. Frequently the development programmer, attempting to hold down core requirements, would define a new routine which co-opted the math and logic of the "common" routine but changed the size (and perhaps the organization and/or format) of the intercommunicating buffer. The implications of this "cloning" practice for configuration management can easily be imagined.

The elimination of fixed-address communication, made possible by the compool definition of data elements, improves the flexibility of operations by:

- Making all programs and data relocatable
- Permitting dynamic linking and loading
- Allowing symbolic correctors

Although the dynamic reallocation of core each time a task is loaded provides great flexibility and gains efficient utilization of storage, it demands a price in computation time. However, if a task is to remain invariant between successive passes, the system provides a push-pull capability that will save and restore an allocation without dynamic address computation.

The symbolic corrector capability may operate during reallocation to make corrections, adjustments or adaptations to programs to fit current needs.

4.2.3 Redundant and Obsolete Data

The problem of redundantly-defined numeric constants was discussed previously. An even greater problem was presented in the identification of obsolete, i.e., unused data values. The lack of an automated set/use capability meant that only a painstaking, instruction-by-instruction analysis could produce this important program management tool. Even then, of course, it was subject to the human error inherent in the performance of a tedious, repetitive task.

The purging of redundant and obsolete data from the compool is enhanced by the procedures employed to review and control additions and modifications to the compool. These procedures include:

- Procedures for reviewing proposed changes and maintaining the COMPOOL
- Analyses of current content of the compool

Close review of all compool-changes resulting from adding new programs or modifying existing programs keeps data redundancies to a low level, ensures the observance of data standards (including the isolation of constant, guaranteed and transient data) and efficient handling and storage utilization. Considered is the impact of changes on other users, the coordination of changes with users and the restructuring of the system.

Analytic listings are used to detect redundancies and impacts and evaluate segmentation and efficiency. The system-set used listings are specifically used to detect obsolete items; those neither set nor used by the current system.

4.2.4 Program Environments

The lack of control over the environment of programs often resulted not only in inefficient utilization of storage, but incompatibilities among the programs. For instance, the lack of control over intercommunication buffers resulted in larger-than-required program environments. For example, a 1598-word output buffer for matrix display (ATAMAT) was maintained while the program computing the two matrixes to be displayed in upper-lower triangular format (ARAY) was limited to processing 21 parameters, i.e., the maximum output could be only 462 words [15].

As a corollary to the control of storage utilization, the control and reduction of program environments keeps storage requirements at a minimum. The features of the COMPOOL operation aimed at control of program environment are:

- Environment analysis and display capabilities
- Commonality
- COMPOOL change procedures
- Data grouping by use

Each of these features support other objectives. In-depth analysis of the program environment helps detect inefficiencies and, in combination with set-use listings, may suggest trade-offs and comparisons of alternative configurations that might be more efficient. COMPOOL change procedures flag inefficiencies, detect improper use of routines and inefficient segmentation. Separating control from application-specific information enhances integrity and proper operation.

4.3 CONFIGURATION MANAGEMENT

In a system where there are many project-specific subsystems and sometimes more than one model of the master system programs and flight support programs active at any one time, each being maintained for latent errors, tight configuration management is essential to the continued effectiveness of operations. The AFSCF data system is a very dynamic one, always under adjustment to meet new requirements. Delivery of software to meet launch schedules places a premium upon efficient management and the early detection of difficulties. The high load of time-critical developments and the importance of avoiding confusion over the capabilities of multiple models, plus the problems involved in coordinating the activities of several software suppliers and users, were some of the principal factors in the COMPOOL implementation decision.

The AFSCF had the four following major objectives for the configuration management features in the COMPOOL:

- To facilitate the standardization of data structure and data handling
- To simplify interface problems
- To centralize the source of data
- To contribute to overall system management

4.3.1 Standardization

The analyses revealed that since the system had evolved through the years and through the efforts of a number of contractors, there was a diversity of approaches to common problems. For example, not only did the expression of state vectors vary as to coordinate system, e.g., earth-fixed vs inertial or polar vs cartesian, but the units in which they were internally computed varied, as well. Dates were expressed in Gregorian or Julian reference; time was maintained in decimal seconds or decimal minutes, as local or Greenwich and as Universal or Ephemeris time; angles were computed in decimal degrees, degrees/minutes/seconds, or in radians; length was carried in English or metric units or even in earth radii [7, 8, 9].

There was also a need for grouping related data into smaller blocks so that routines need have only those data actually required for their environment. Thus, for instance, while the same integration routine should be used for trajectory computation of ascent, on-orbit, and reentry cases, the perturbations due to the earth's non-homogeneity, i.e., higher-order zonal and tesseral harmonics, were of importance only in the on-orbit case. It was desirable, therefore, that these rather sizeable data tables could be eliminated from core when integrating the other two cases.

Another area of concern comprised large buffers (thousands of words) for the accommodation of voluminous serial data, such as satellite tracking observations or telemetry data streams. There was an evident need to provide smaller fixed-length buffers which could support double or even triple buffering of these long, variable-length files.*

*It should be borne in mind that although the FSC configuration included disks at the time (see Figure 2), they were still being used as pseudo-tape files, with the large buffers that would be consistent with the searching of long, serial tape files.

The standarization of data structures and handling coupled with COMPOOL-procedures simplifies the configuration management task. Some of the ways in which configuration management is supported are:

- Change evaluations
- Configuration control and accounting
- User control of system structure

Standardized data structures simplify evaluation not only of compool changes but of all new programs and modifications. The compool and its procedures provide a channel of communication among all contractors and help notify contractors and users of, or gather from them, the impacts of change. the reviews in turn avoid non-standard treatment of data.

Configuration control is simplified by easy accounting for data elements and all changes to them. Standards help control unnecessary change.

The visibility into data and system structure provided by the regular issuance of compool documentation and compool change reports plus the involvement of user personnel in review procedures gives the user a level of control over system structure and content not realized by most systems. This is a feature upon which the AFSCF places high value in helping it to control the capriciousness and idiosyncracies of individual contractors and users.

4.3.2 Interface Control

There were two basic problems associated with the coordination of interfaces in the AFSCF. First, with many independent software suppliers and users, it was difficult to control all commonly used data. Second, transfer of data to and from other agencies (NASA, Space Defense Center, etc.) were particularly hard to keep current.

Although data intercommunications between programs were provided by the files on the Reset Tape (see Figures 13 and 14, Appendix A) and/or temporarily by SOM techniques such as the KREFPOOL (see Table 6, Appendix A), it was still difficult to control units of measurement or table structure

(serial/parallel, packed/unpacked) interfaces. With the diversity of Program Offices and their software contractors (CPACs), one or the other of the interfacing routines would frequently require the addition of a data-transformation capability, with the concomitant increase in core requirements and operating time.

External interfaces, i.e., the transfer of data in machine-sensible form between the AFSCF data system and other agencies, e.g., Space Defense Center, NASA or Atlantic Missile Range, were particularly hard to document in current, unambiguous form. Specifications for such interfaces varied in format and level of detail, existing (in some cases) only in documentation for the individual programs of the agency's computer(s).*

Therefore, simplifying interfaces not only between programs but between developers and between users was a prime objective of the COMPOOL system. Simplification is attained by:

- Compool change control
- Standardization of calling sequences
- Standardized intercommunication buffers
- Standardized block structures
- Analytic listings

*This was true in the AFSCF as well; many AFSCF machine-sensible output formats were described only in Milestone 7 User's Manuals, documents prepared and intended for use at the CDC 3600 FSC console operators' positions, rather than as Interface Control Documentation [6]. There was one rather amusing result of this problem of interface control. The SST contained a program to convert the Ephemeris Tape from CDC 1604/3600 format to IBM 360 format for use by another agency, i.e., 48-bit words to 36-bit words. After several years, it was determined that the agency had replaced their computer with CDC 3600 equipment, and had written their own program to convert the AFSCF-supplied 36-bit Ephemeris Tape to 48-bit format!

The change procedures and regular analytic reports keep all developers, users and system managers abreast of the current state and composition of the system. The standardized calling sequences, intercommunication buffers and block structures keep program interfaces standard and simple.

4.3.3 Data Sources

Similarly, the control of data values and formats presented a set of difficulties that included problems both internal and external to the AFSCF data system.

Clearly, it was desirable to provide centralized control of interfacing data structures rather than leaving this decision in the hands of individual programmers working for CPACs.

Further, an analysis of existing programs revealed a variety of inefficiencies in both core-usage and operating time. For example, the use of factorials in expansion-series was sometimes seen to be handled by computing a set of factorials into an internally-defined table for referencing by the program as needed; in other cases, the factorials were recomputed for each use. Of course, there was no general purpose table of factorials available in RIPOOL, KODMTABL, KREFPOOL, etc., [7, 8. 9]. Even more subtle was the fact that even though the CDC 3600 floating point DIVIDE instruction was almost twice as slow as the MULTIPLY instruction, and factorials usually appeared in denominators in such expansions, no case was discovered in which the programmer had decided instead to use the reciprocals of the factorial for multiplication.

The external problems related to the building and maintenance of data bases for specific program-office users. Difficulties with numerical constants have been discussed (see Section 4.1.3) as have been those with units of measurements (see Sections 4.1.3 and 4.3.1). In addition, numerous geophysical "constants," were (and are) not , in fact, constant but are periodically being refined by geodetic survey efforts. These include values related to the size, sphericity, or homogeneity of mass of the earth, e.g., earth radius, oblateness, or harmonics [13].

Another, even less stable aggregation of geophysical data was stored in tables contained in the program code of subroutines. Any change to these values required, at the very least recompilation of the subroutine; if the change was one of increased precision, i.e., lengthening tables or changing step-sizes, reprogramming was required. Examples of this class of data included atmospheric models, geomagnetic models, tracking-station obscura, and atmospheric refraction coefficients [13].

Since each Program Office mission generated its own requirements for accuracy, precision, and speed of computation, it was vital to preserve autonomous data base control while maintaining the highest degree of currency and consistency in these data base values. This was a major objective in the COMPOOL implementation.

Hence, control over the source and quality of data quite apart from the data definition in the compool was an important objective of the COMPOOL implementation. The COMPOOL supports such control through:

- Providing library tapes
- Permitting project specific data bases
- Permitting private data bases

Supplying controlled and validated data avoids contretemps over casually supplied and unreliable data and is a service to users and developers alike. Permitting project-specific data bases, with values that may override the generally common information in the COMPOOL, adapts control to the peculiar needs of projects, yet economically supports the needs of most users. Having private data bases satisfies the sensitivity and security requirements inherent in the AFSCF users community.

4.3.4 Overall System Management

Improving the overall manageability of the AFSCF is a goal implicit in most of the above objectives. There was a desire to make operations simpler and more understandable, to keep burgeoning costs down, and to enjoy the benefits of standardization in avoiding errors and incompatibilities. The COMPOOL would not only provide a firm basis upon which the base standardization dicta rest but provides a strong support for the enforcement of standards observance.

An important element of cost control was to ensure that CPACs used common subroutines and global data structure wherever possible, rather than developing or defining new ones in each case. A centralized definition of calling sequences and data structures would facilitate this control.

The maintenance of special-purpose (AMT) routines and data bases, after the CPAC development contract was phased out presented problems because of the diversity of techniques and documentation. The standardization implicit in the COMPOOL approach would reduce the complexity of this task. Similarly, uniformity would facilitate the inclusion of certain AMT programs into the SST, making them available for use by other program offices for support of their satellites.

Clearly the general performance of baseline definition and change control should be improved by the standardization effect of COMPOOL. The Air Force and its Computer Program Integration Contractor (CPIC) would be better able to track and record modifications and corrections of deficiencies in the highly dynamic software system.

Finally, the coding of all SST and AMT programs in JOVIAL would provide a large measure of machine independence in the event of an upgrading of the FSC hardware at some future time.

It is believed that the overall objective of more manageability in the AFSCF data system has been achieved. The standards, the coordination procedures the careful review procedures, and the configuration control procedures practice all contribute to the result:

- Enhanced configuration control
- Improved control over data bases
- Much improved interface control
- Encouragement in the use of common Government-furnished subroutines as well as common data
- Enhanced security of sensitive data
- A measure of independence from machine limitations

In short, original expectations have been met and in many instances exceeded. As high as the development and implementation costs were, the return on the investments has far outweighed the costs.

(Page 180 blank)

Appendix A - PRE-COMPOOL DATA BASE DESIGN

1. INTRODUCTION

This material, excerpted from Reference 5 of Appendix E, is furnished to provide detailed information on the data base discipline as it existed under System I (i.e., pre-COMPOOL). Although direct comparison with the System IIB (COMPOOL) data base discipline is not practical, the types of data required in each case are analogous.

It is important to understand the functions of two tapes, the Reset and Ephemeris Tapes. The Reset Tape was read and subsequently rewritten by most programs, thus performing an intercommunication function. The Ephemeris Tape was used by other programs, e.g., command operation, to obviate their necessity for computing ephemeris predictions.

2. PRE-FLIGHT INITIALIZATION

During the pre-flight period (prior to launch) the CDC 3600 computer computes, generates and displays a satellite's nominal (predicted or desired) orbital path. From these data the 3600 further computes selected tracking information, to be transmitted to remote tracking stations (RTSs), specifying when (the acquisition times, rise and set) and where (vehicle position) a satellite should or will be located with reference to AFSCF RTSs. RTS personnel (and equipment) will subsequently use these data to acquire and track an assigned satellite.

The CDC 3600 eventually processes the resulting in-flight tracking information to produce three classes of prepass message tapes for transmission to an RTS. These comprise:

- Vehicle acquisition and antenna-pointing data.
- Telemetry modes, i.e., decoding/decommutating data.
- Vehicle-commanding data.

2.1 Pre-Flight

Several data system tasks (and their associated computer programs) are essential for accomplishing vehicle orbit prediction and the subsequent generation of prepass data. These data system tasks which compute, process and display nominal tracking information include:

- Design/create the data package (vehicle-specific parameters).
- Initialize and generate the reset tape.
- Generate nominal ascent trajectory.
- Generate nominal orbit and/or reentry ephemeris predictions.
- Generate prepass data (tracking, telemetry, commanding).

2.2 Orbital/Ephemeris Computation

The computational requirements for orbit determination and prediction for each vehicle series may vary based on the vehicle's orbital requirements (e.g., geosynchronous orbit, elliptical orbit, high or low altitude, orbit adjusts, effect of perturbing bodies, etc.). Orbit determination uses tracking data (range, azimuth, elevation, range rate) to determine the epoch (initial) conditions.* Orbit prediction takes an intial set of orbital conditions [Keplerian, Polar, etc.] and predicts where a vehicle will be, and when.

No matter what the unique requirements are for a program, certain basic data about the satellite vehicle, the stations which can or should track it, and vehicle orbital dynamics, must be input initially to the CDC 3600. These data include:

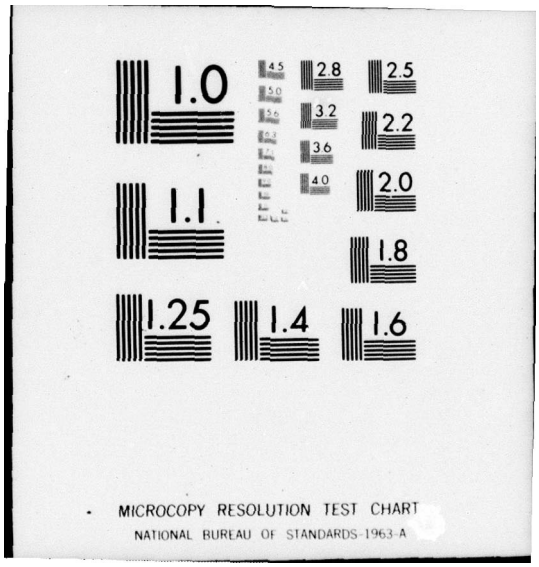
- Vehicle parameters, such as vehicle weight, fuel flow rate, booster ignition and shut-off, thrust magnitude, orbit adjusts, etc.
- Orbital parameters such as desired orbital eccentricity, argument of perigee, orbital inclination, semi-major axis, right ascension, etc.

*Any arbitrary point in time from which all calculations will be made.

AD-A041 678 SYSTEM DEVELOPMENT CORP SANTA MONICA CALIF F/G 9/2
AN INVESTIGATION OF PROGRAMMING PRACTICES IN SELECTED AIR FORCE--ETC(U)
JUN 77 G H PERRY, N E WILLMORTH F30602-76-C-0180
UNCLASSIFIED SDC-TM-(L)-5839/000/00 RADC-TR-77-182 NL

3 OF 3
ADA041678

END
DATE FILMED
8-77


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

- Vehicle ephemerides, such as desired flight path angles at epoch, position and velocity vectors at specified points and times on a flight path.
- Station coordinates and equipment (antennas).
- Perturbing effects bodies influencing orbital mechanics at various altitudes, e.g., sun, moon, atmosphere, etc.

Various types of data are output during data processing which allow tracking data users to assess the progress of the orbital planning task, that is, the determination of a "best-fit" orbital path based on such variables and constraints as station capabilities, conflicts with other satellites, mission objectives, perturbing bodies, etc. There are two types of computational techniques currently used for predicting orbital paths using the data which are input. These are the Breakwell Method Tracking (BMT) and the General Purpose Tracking Programs (GPTP). These programs, when given a set of six basic parameters (commonly known as initial conditions, nominals, or epoch conditions) and an associated time, can establish instantaneous orbits.

2.2.1 Reset Tape(s)

The Reset Tape is the main source of data communication between the various functions in the data system. The Reset Tape contains flight-specific data for both BMT and GPTP and the Fairchild Timer, and in addition has room for other data in unassigned data blocks.

Records with similar, specialized data are organized into data blocks. A Multiple Vehicle Reset Tape has many groups of vehicle-specific data blocks.

A Reset Tape may hold up to 23 separate sets of vehicle-specific data; each set must be for a different vehicle. The vehicle-specific set must contain at least three data blocks. Data Block 0, Data Block 1, and one of the other data blocks 2 through 10; the maximum size is 11 data blocks--Data Block 0 through Data Block 10. The first five data blocks have a strictly defined format and are accessed directly or relatively by reference symbols. The other six data blocks are more flexibly used; depending on the need, they

may or may not be assigned data. They may consist of one to ten records with a maximum size of 2000 words per record [6]*.

2.2.2 Reset Data in Core

For the BMT programs, Data Block 1 contains RIPOOL variables such as:

- Time of events.
- Orbital parameters.
- Other vehicle parameters such as:
  - Drag factors
  - Vehicle
  - Clock time

Data Block 2 contains tracking station information:

- Locations.
- Antenna types.
- Input data format requirements.
- Vehicle/orbit parameters.
- Constraints and events.
- Vehicle acquisition information, including acquisition times and length of passes.

Data Block 2 is located in the Table RESETBL in core. Data Block 3 contains Fairchild Timer information (see Figure 13) [7].

*Refers to references listed in Appendix E.

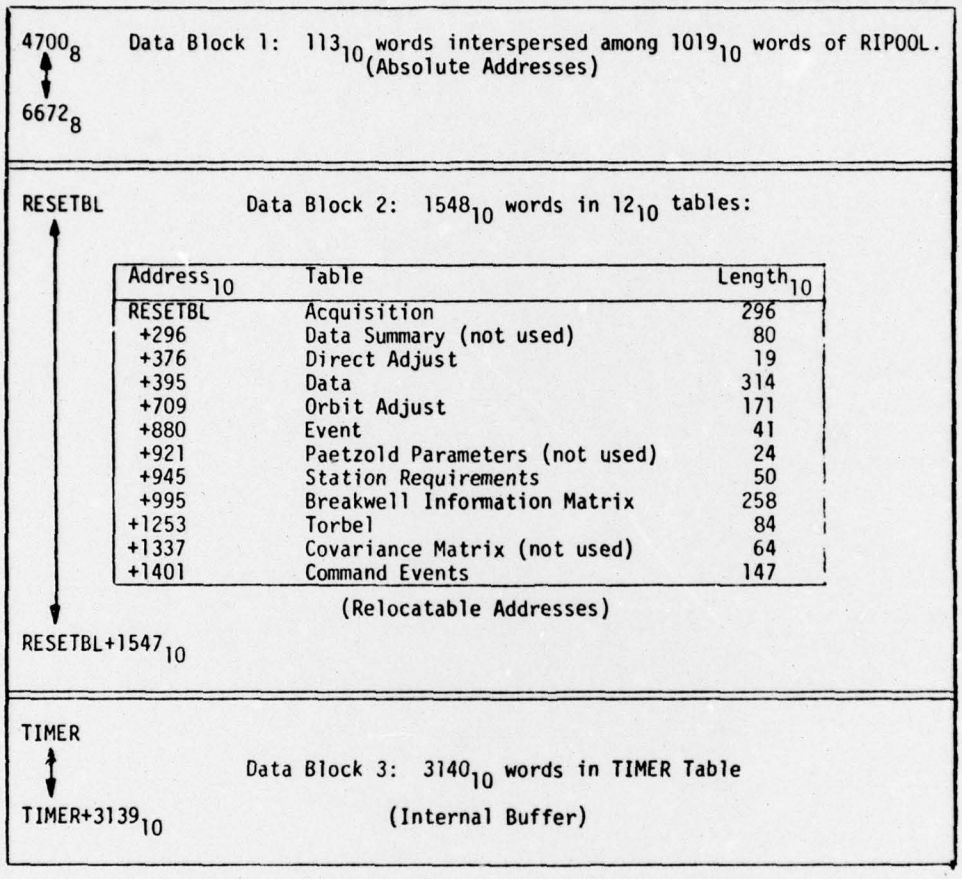


Figure 13. Typical Core Storage Map, BMT Programs

For the GPTP, Data Blocks 1, 2, and 3 contain the same information as the BMT above; Data Block 4 contains initial condition parameters (i.e., epoch, perturbing body parameters and ephemerides, additional station data, i.e., antennas, coordinates, etc. (See Figure 14) [8].

The GPTP programs operate in an SMTC Special Operating Mode (SOM3) which guarantees the absolute core location (as loaded) of Data Blocks 1 through 4, as well as a relocatable communication table of 308 words, KREFPOOL, for all GPTP programs operating during that sequence, i.e., the call for SOM3 (see Table 6) [9].

Additional SOMs perform similar temporary preservation of the core locations of Reset Tape Blocks 5 through 10 for vehicle-specific programs on FSTs.

4700_8 Data Block 1: 113_{10} words interspersed among 1019_{10} words of RIPOOL.

(Absolute Addresses)

6672_8

10000_8 Data Block 4: 3700_{10} words in 13_{10} tables - KODMTABL:

$Address_8$	Table	$Length_{10}$	Logical Record
10000	Bookkeeping (BKI)	10	0
10012	Reference Time (RTI)	10	1
10024	Initial Condition Parameters (ICP)	10	2
10036	Differential Equation Parameters(DEP)	30	3
10074	Dimensions (DIM)	10	4
10106	Perturbing Bodies (PTB)	20	5
10132	Stations (STP)	750	6
11510	Constraints (BND)	100	7
11654	Information Matrix (ATA)	936	8
14031	Ascending Nodes (NOD)	293	8
14753	Zero Residuals (ZER)	901	8
16560	Output Specifications and Miscellaneous (OUT)	101	9
16725	Descriptive Orbital Characteristics (DOC)	130	9
17127	Spares	29	

(Absolute Addresses)

17163_8

17164_8 Data Block 2: 1548_{10} words in 12_{10} tables - RESETBL:

$Address_8$	Table	$Length_{10}$	RESETBL
17164	Acquisition (ACQ)	296	+0
17634	Data Summary (SUM)	80	+296
17754	Direct Adjust (ADJ)	19	+376
17777	Data (DAT)	314	+395
20471	Orbit Adjust (OAJ)	171	+709
20744	Event (EVT)	41	+880
21015	Paetzold Parameters (PAE)	24	+921
21045	Station Requirements (SRT)	50	+945
21127	Breakwell Information Matrix (TM)	268	+995
21531	Torbel (TOR)	84	+1253
21655	Covariance Matrix (MTX)	64	+1337
21755	Command Events (DAQ)	147	+1401

(Absolute Addresses)

22177_8

22200_8

SPETAB3: Three words in length

(Absolute Addresses)

22202_8

22203_8 RCLANK

Figure 14. Typical Core Storage Map, GPTP (SOM3)

Table 6. Contents of KREFPOOL (1 of 6)

WORD NO.	TAG	RELATIVE ADDRESS$_8$	DEFINITION	STORED BY
1	XK	0	Radius Vector $\vec{R}$ Components: x, in e.r. (At time TEPOCH)	*KTRAJ, KDRAGROT
2	YK	1	Radius Vector $\vec{R}$ Components: y, in e.r. (At time TEPOCH)	*KTRAJ, KDRAGROT
3	ZK	2	Radius Vector $\vec{R}$ Components: z, in e.r. (At time TEPOCH)	*KTRAJ, KDRAGROT
4	XDOT	3	Velocity Vector $\vec{V}$ Components: x, in e.r./min. (At time TEPOCH)	*KTRAJ
5	YDOT	4	Velocity Vector $\vec{V}$ Components: y, in e.r./min (At time TEPOCH)	*KTRAJ
6	ZDOT	5	Velocity Vector $\vec{V}$ Components: z, in e.r./min. (At time TEPOCH)	*KTRAJ
7	ALPHAK	6	Right Ascension α: $\tan^{-1}(y,x)$, in e.r.	KARTSPHR
8	BETAK	7	Flight Path Angle β: $\tan^{-1}$ (CAPG/CAPD), in e.r.	KARTSPHR
9	DELTAK	10	Declination δ: $\tan^{-1}[z/(x^2+y^2)^{\frac{1}{2}}]$, in e.r.	KARTSPHR
10	A	11	Azimuth A: $\tan^{-1}[R(x\dot{y}-\dot{x}y)]/(R^2z-\dot{z}$CAPD), in e.r.	KARTSPHR
11	R	12	Vector Magnitude: $\lvert\vec{R}\rvert=(x^2+y^2+z^2)^{\frac{1}{2}}$, in e.r.	KARTSPHR,KDRAGROT
12	V	13	Vector Magnitude: $\lvert\vec{V}\rvert=(\dot{x}^2+\dot{y}^2+\dot{z}^2)^{\frac{1}{2}}$, in e.r./min.^{2}	KARTSPHR,KDRAGROT
13	COMPTIME	14	Current Computation Time, minutes	KTRAJ
14	OUTFLAG	15	Exit Flag, Octal Integer: 0 = Current Computed Altitude < (KRASHALT) 1 = Exit with Requested Time 2 = Exit at Apogee 3 = Exit at Perigee 4 = Exit at Rev Crossing 5 = Exit at Thrust on or off	KTRAJ
15	INTEGOUT	16	Octal Integer: Bits 23--0: Location, in ARAY, of Radius Vector Partials Bits 47--24: Location, in ARAY, of Velocity Vector Partials	KTRAJ
16	STAGARAY	17	Octal Integer: ARAY Location, Bits 23--0	KTRAJ
17	OUTYR	20	Computed Date: Calendar Year, Fixed (B0)	KGMTCOMP
18	OUTMO	21	Computed Date: Calendar Month, Fixed (B0)	KGMTCOMP
19	OUTDAY	22	Computed Date: Calendar Day, Fixed (B0)	KGMTCOMP
20	OUTHR	23	Computed Date: Hour_{24}, Fixed (B0)	KGMTCOMP
21	OUTMIN	24	Computed Date: Minute_{60}, Fixed (B0)	KGMTCOMP
22	OUTSEC	25	Computed Date: Second_{60}, Fixed (B5)	KGMTCOMP
23	DATEJUL	26	Julian Time Value: COMPTIME+JULDATE, in days	KPRINTUM

N.B. All Formats Floating Point Except as Indicated.

*Used by KODM as temporary storage; values guaranteed only for KTRAJ driver.

Table 6. Contents of KREFPOOL (2 of 6)

WORD NO.	TAG	RELATIVE ADDRESS	DEFINITION		STORED BY
24	DGPERTS	27	Name 1st		KTRAJ
25	+1	30	Name 2nd		.
26	.	31	.		.
27	.	32	.		.
28	.	33	.		.
29	.	34	.	Perturbing Body, BCD	.
30	.	35	.		.
31	.	36	.		.
32	.	37	.		.
33	+9	40	Name 10th		KTRAJ
34	MRAT	41	Mass Ratio, 1st		KTRAJ
35	+1	42	Mass Ratio, 2nd		.
36	.	43	.		.
37	.	44	.		.
38	.	45	.	Perturbing Body, to Central Body, Earth	.
39	.	46	.		.
40	.	47	.		.
41	.	50	.		.
42	.	51	.		.
43	+9	52	Mass Ratio, 10th		KTRAJ
44	RXJ	53	x-Component 1st		KPERTBOD
45	+1	54	x-Component 2nd		.
46	.	55	. .		.
47	.	56	. .		.
48	.	57	. .	Earth-to-Perturbing Body Vector, in earth radii	.
49	.	60	. .		.
50	.	61	. .		.
51	.	62	. .		.
52	.	63	. .		.
53	+9	64	x-Component 10th		KPERTBOD

N.B. All Formats Floating Point Except as Indicated.

Table 6. Contents of KREFPOOL (3 of 6)

WORD NO.	TAG	RELATIVE ADDRESS$_8$	DEFINITION		STORED BY
54	RYJ	65	y-Component 1st		KPERTBOD
55	+1	66	y-Component 2nd		.
56	.	67	. .		.
57	.	70	. .		.
58	.	71	. .	Earth-to-Perturbing Body Vector, in e.r.	.
59	.	72	. .		.
60	.	73	. .		.
61	.	74	. .		.
62	.	75	. .		.
63	+9	76	y-Component 10th		KPERTBOD
64	RZJ	77	z-Component 1st		KPERTBOD
65	+1	100	z-Component 2nd		KPERTBOD
66	.	101	. .		.
67	.	102	. .		.
68	.	103	. .	Earth-to-Perturbing Body Vector, in e.r.	.
69	.	104	. .		.
70	.	105	. .		.
71	.	106	. .		.
72	.	107	. .		.
73	+9	110	z-Component 10th		KPERTBOD
74	REVCT	111	Rev Crossings, Fixed (BO)		KTRAJ
75	SMAK	112	Semimajor Axis: Varies with Orbit-Type, in e.r.		KEPLERS
76	ECC	113	Eccentricity: Varies with & defines orbit type		KEPLERS
77	INCLK	114	Inclination: $\cos^{-1}$ NZ, in radians		KEPLERS
78	CAPOMEG	115	Longitude of Node: $\tan^{-1}$ |NX/-NY|, in radians		KEPLERS
79	SMAOMEG	116	Argument of Perigee: $\tan^{-1}$ (CAPB)-$\tan^{-1}$ (CAPH) in e.r.		KHCOMPS
80	MANOM	117	Mean Anomaly: Varies with Orbit-Type		KEPLERS
81	RTEST	120	Eclipse Change: Octal Integer, No = 0, Yes = 1		KECLIPSE
82	NODEFLAG	121	Eclipse Flag: Octal Integer		KECLIPSE

0 = No Eclipse
1 = Eclipse by Earth
2 = Eclipse by Moon
-0 = Indeterminate

N.B. All Formats Floating Point Except as indicated.

Table 6. Contents of KREFPOOL (4 of 6)

WORD NO.	TAG	RELATIVE ADDRESS$_8$	DEFINITION	STORED BY
83	HYPFLG	122	Orbit-Type Flag: Para = 1; Hyper = 2; Circ = 4; Ellip = 8	KEPLERS
84	CAPG	123	Interim Factor: $(SMALLG)^{\frac{1}{2}}$, in e.r.^{2}/min.	KARTSPHR
84	SQRTMU	123	Interim Factor: $(MUK)^{\frac{1}{2}}$	KINJFUN
85	NX	124	Normal Vector x Component: $(y\dot{z}-z\dot{y})/CAPG$	KNCOMPS
86	NY	125	Normal Vector y Component: $(z\dot{x}-x\dot{z})/CAPG$	KNCOMPS
87	NZ	126	Normal Vector z Component: $(x\dot{y}-y\dot{x})/CAPG$	KNCOMPS
88	CAPD	127	Interim Factor: $\vec{R}\cdot\vec{V}=(x\dot{x}+y\dot{y}+z\dot{z})$, in e.r.^{2}/min.	KARTSPHR
89	SMALLG	130	Interim Factor: $\{[R(V)]^2 - CAPD^2\}$, in e.r.^{4}/min^2	KARTSPHR
90	CAPH	131	Interim Factor: $[(CAPG)(CAPD)]/\{(SMALLG) - [R(MUK)]\}$	KHCOMPS
91	S3	132	Interim Factor: $MUK/\left\|(SMAK)(MUK)\right\|^{\frac{1}{2}}$, in e.r./min	KEPLERS
92	PERIOD	133	Orbital Period: $[2\pi(SMAK)]/S3$, in min.	KEPLERS
93	FRACSEC	134	Computed Date: Fractional Seconds	KGMTCOMP
94	OUTMINTO	135	Elapsed Time Value: in minutes	KGMTCOMP
95	ALTPER	136	Altitude at Perigee: $[(1-ECC)(SMAK)] - \frac{ONELESSE}{\{[1+(CAPH)^2][1+(ELLIPTE)^2-2(ELLIPTE)]\}^{\frac{1}{2}}}$ in e.r.	KAPOGPER
96	ALTAPOG	137	Altitude at Apogee: $[(1-ECC)(SMAK)] + \frac{ONELESSE}{\{[1+(CAPH)^2][1+(ELLIPTE)^2-2(ELLIPTE)]\}^{\frac{1}{2}}}$ in e.r.	KAPOGPER
97	VELAPOG	140	Velocity at Apogee: $(S3)[\left\|(1-ECC)/(1+ECC)\right\|]^{\frac{1}{2}}$, in e.r./min.	KAPOGPER
98	VELPER	141	Velocity at Perigee: $(S3)/[\left\|(1-ECC)/(1+ECC)\right\|]^{\frac{1}{2}}$, in e.r./min.	KAPOGPER
99	ITERNO	142	Iteration Number	KODM
100	COSVE	143	$[\left\|\vec{R}\right\|^2+\left\|\vec{S}-\vec{R}\right\|\left\|\vec{S}\right\|^2]/2\left\|\vec{R}\right\|\left\|\vec{S}-\vec{R}\right\|$	KSVE
101	SVE	144	Sun-Vehicle-Earth Angle: $\cos^{-1}$ (COSVE), in e.r.	KSVE
102	PSVE	145	SVE Projection: $\cos^{-1}\frac{[(XK)^2+(YK)^2]^{\frac{1}{2}} + [(XK-RXJ)^2 + (XY-RYJ)^2]^{\frac{1}{2}}}{2\,[(XK-RXJ)^2 + (YK-RYJ)^2]^{\frac{1}{2}}\,[(XK)^2+(YK)^2]^{\frac{1}{2}}}$	KSVE

N.B. All Formats Floating Point Except as Indicated.

Table 6. Contents of KREFPOOL (5 of 6)

WORD NO.	TAG	RELATIVE ADDRESS$_8$	DEFINITION	STORED BY
103	LMDA	146	Longitude, w/r Greenwich, Positive Eastward, Mod 2π in e.r.	KVROT
104	RVX	147	x-Component Atmospheric Drag: [(OMEGAE)(YK)]+XDOT, in e.r./min.	KDRAGROT
105	RBY	150	y-Component Atmospheric Drag: YDOT-[(OMEGAE)(XK)], in e.r./min.	KDRAGROT
106	RVZ	151	z-Component Atmospheric Drag: ZDOT, in e.r./min.	KDRAGROT
107	RVA	152	Vector Magnitude: $[(RVX)^2+(RVT)^2+(RVZ)^2]^{\frac{1}{2}}$ in e.r./min.	KDRAGROT
108	CAPB	153	Interim Factor: ZK/{[(YK)(NX)]-[(XK)(NY)]}	KHCOMPS
109	MESS	154	Atmospheric Drag Factor: varies with ALTITUDE, in 1/e.r.	KDRAGROT
110	RDRAG	155	Atmospheric Drag Factor: varies with ALTITUDE, in 1/e.r.	KDRAGROT
111	ONELESSE	156	Interim Factor for Latitude: 1-ELLIPTE	KLATLONG,KALT
112	SQ1LESSE	157	Interim Factor for Latitude: $(1-ELLIPTE)^2$	KLATLONG,KALT
113	CAPQ	160	$(ZK)^2/[(XK)^2+(YK)^2]$	KQCOMP
114	CAPE	161	Eccentric Anomaly: varies with Orbit-Type	KEPLERS
115	ECOSOMEG	162	Breakwell Parameter: (ECC)(cos SMAOMEG)	KEPLERS
116	SMAOSIN	163	Breakwell Parameter: (SMAOMEG)(sin SMAOMEG)	KEPLERS
117	ALTITUDE	164	Altitude: $R-(ONELESSE/\{[(CAPQ+SQ1LESSE)/(1+CAPQ)]^{\frac{1}{2}}\})$, in e.r.	
118	LONGIT	165	Geocentric Longitude: [(COMPTIME)(OMEGAE)]+ ALFAG, in e.r.	KLATLONG
119	LATITUDE	166	Geocentric Latitude: $\tan^{-1}$ (SMAQ/SQ1LESSE), in e.r.	KLATLONG
120	NODRATE	167	Node Rate: NZ $\{[(JTERM)(S3)]/[1-ECC^2)^2(SMAK)^3]\}$	KRATES
121	PERIGRAT	170	Perigee Rate: [2.5(NODRATE)][(NZ)-(1/NZ)]	KRATES
122	SIDERPD	171	Latest Nodal-Sidereal Period, in minutes	KEVALTRJ
123	APOGTIME	172	Apogee Time: COMPTIME Value at Apogee, in min.	KEVALTRJ
124	PERITIME	173	Perigee Time: COMPTIME Value at Perigee, in min.	KEVALTRJ
125	SMAQ	174	$(CAPQ)^{\frac{1}{2}}$	KQCOMP
126	RMS	175	Total RMS of Radar Residuals	KPARCOM

N.B. All Formats Floating Point Except as Indicated.

Table 6. Contents of KREFPOOL (6 of 6)

WORD NO.	TAG	RELATIVE ADDRESS$_8$	DEFINITION	STORED BY
127	NOBS	176	Sun-Spin Angle, Degrees	KSVE
128	APERIOD	177		Not Used
129	OAJLOC	200	Initial Address of Parameters in OAJ Table (Bits 23--0)	KICK
130	AMAG	201	$\|\bar{PAX}, \bar{PAY}, \bar{PAZ}\|$ during an accelerated adjust, in e.r./min^2	KICK
131	TFROMIC	202	Time from latest Epoch: COMPTIME-TEPOCH, in min.	KPRINTUM
132	NEXTIME	203	Next KTRAJ Computation Time, in minutes	KEVALTRJ
133	NOPBSLVD	204	Housekeeping Item: Number of Perturbing Bodies	KTRAJ
134	PAX	205	x-Component } Thrust Acceleration Vector	KTRAJ
135	PAY	206	y-Component } earth radii/minute	KTRAJ
136	PAZ	207	z-Component }	KTRAJ
137	LOOKANG	210	Breakwell Parameter: Look Angle, in e.r.	KEPLERS
137	LOGRLOGA	210	Drag Factor: $[(\Delta RHO/(RHO)]$ (ΔRVA)	KDRAGROT
137	KANOMPER	210		Not Used
138	KODMPGNO	211		KPARCOM
139	RSV	212	Angle ($\bar{sun}$-vehicle) ($\bar{moon}$-vehicle), in radians	KSVE
140	RMV	213	Magnitude ($\bar{moon}$-vehicle), in e.r.	KSVE
141	SVM	214	Magnitude ($\bar{sun}$-vehicle), in e.r.	KSVE
		215	Spares	
		:		
		307	Spares	

N.B. All Formats Floating Point Except as Indicated.

APPENDIX B - SYSTEM IIB DATA BASE DESIGN

1. INTRODUCTION

This material is excerpted from References 20, 29, and 36 of Appendix E, and is provided to furnish detailed information on the data base discipline as it exists in the current COMPOOL implementation.

Pages 196 through 204* provide a general description of the data blocks their types, and their structures. Pages 205 through 209** describe the ephemeris interface. Figure 15, pages 210 through 212,*** illustrates the compool definition of the ephemeris.

These capabilities replaced the Reset and Ephemeris Tapes of System I (pre-COMPOOL).

2. DESCRIPTION OF DATA BLOCKS, TYPES, AND STRUCTURES

The following nine pages are reprinted from SDC TM-(L)-2760/355/00.

*Extracted from SDC TM-(L)-2760/355/00.
**Extracted from a Data Dynamics, Inc., document (DDI-13-44-01) entitled "An Introduction to the Advanced Orbit/Ephemeris Subsystem."
***Extracted from SDC TM-(L)-5048/813/00.

1. INTRODUCTION

1.1 SCOPE OF EFFORT

A definition of 'compool' is prerequisite, conceptually, to an exposition of the philosophical underpinnings of this effort. Explicitly, the compool "...is merely a dictionary containing definitions of available library programs and procedures and of data common to two or more programs or procedures."[1] Implicitly, however, the compool is much more than that. It is a technical tool, almost inseparable from the data base of data, programs, and procedures which it describes, defines, and (implicitly) controls. The contribution of an operating compool to the system-management's coordination and control of system configuration and content has been described,[2]

> ...the compool is a powerful technical tool for system programming equal in importance to procedure-oriented languages and compilers... not only does it minimize the task of matching programmer data needs with the programs containing [or producing] the needed data, but management can now standardize data structure and data handling procedures and can control storage space allocation. Compool is economical because only one data description is necessary, and it is efficient because it prevents errors and incompatibilities. Compool provides an easy means of maintaining several programs for changes in data structure...permits a single item or table, array, or data block to be referenced symbolically by several programs, at the same time insuring that scaling, location, and other attributes [e.g., format, classification, etc.] are compatible from program to program.

---

[1] TM(L)-2760/351/00 (Reference a), page 13

[2] TM(L)-2222/013/00 (Reference b), page 4f.

1.2 DESIGN PHILOSOPHY

The basic philosophy of this compool design has, therefore, been rather easy to come by:

a. there is an existing system which has evolved into an aggregation of equipment, data, and procedures

b. there are directives which describe capabilities which are to be added to the system

c. there is a wealth of historical information on problems encountered and fixes generated during this evolutionary process.

This design must optimize (or at least, suboptimize) the accommodation of (a), the incorporation of (b), and the avoidance of (c).

2. DESIGN OBJECTIVES

Broadly considered, the over-all design objective has been to facilitate the maintenance of existing program capabilities, the addition of new ones, and the servicing of increasing numbers and types of vehicles by the SCF; meanwhile accomplishing the necessary trade-offs to reduce both computer-operating times and computer core-space needs.

2.1 SPECIFIC OBJECTIVES

Specific compool and data base design objectives, are:

a. To bring into the compool, and thus under the control of the system, a majority of the system's data.

b. To completely distinguish between guaranteed and transitory data, and to guarantee the integrity and preservation of all guaranteed data.[1]

c. To achieve flexibility by organizing the data base into identifiable, manipulable components that are at once smaller and more functionally arranged than are the current RIPOOL, KODMTABL, KREFPOOL, RESETBL, etc.

---

[1] Throughout this document, the following definitions of 'guaranteed data' and 'transitory data' apply:

guaranteed data--those values, either originally supplied or generated by a preceding program, which must be implicitly guaranteed to subsequent user-programs, e.g., constants, epoch-conditions with their related time, orbit-adjusts and their status, etc.

transitory data--values which are generated by a program for imminent use by itself or a procedure within its environment, but which lose all significance when that program is through its current operation, e.g., instantaneous epoch-conditions and time, input-output time and positions, etc.

d. To recognize program and contractor interfaces and to contribute to the simplification and accommodation of these interfaces.

e. To provide a meaningful and consistent set of data descriptions and dimensional units within the purview of the compool and the data base.

2.2 OVER-ALL OBJECTIVE

The eventual establishment of a system of programming standards and controls (including, for instance, a standards and conventions document), to facilitate the integration of individual programs produced by diverse associate contractors into a usable, workable whole, is certain; the compool and data base must contribute to and fit into that system. This has been a continuing and pervasive design objective--the one that has often finally "tipped the balance" toward a particular design feature.

3. DESIGN CHARACTERISTICS

Some characteristics of the System II compool and data base design coincide with the current system practices; other characteristics disclose important conceptual departures which are described in this Section.

3.1 RELOCATABLE ENVIRONMENTAL LOADING

All Master Tape(SST or AUX)[1] data and procedures, all guaranteed data, and buffer-areas for certain other data, e.g., Ephemeris Tape, or Variable Block Reset (VBLKR) data, will be loaded or reserved as part of the environment of any program about to be operated.

3.2 DATA ELEMENTS

The compool is organized into the standard JOVIAL data elements of simple items, tables, and arrays. An additional element, the data block, has been incorporated, and may comprise any number or assortment of simple items, tables, and arrays. There are six different types of data blocks identified: CBLK, TBLK, MBLK, GBLK, VBLK, and IBLK.[2] Data blocks will be identified in the compool by name, e.g., NUMCCON, PHYSCON, STDATA, etc., and by type, e.g., CBLK, MBLK, etc. Three types, GBLK, VBLK, and IBLK, are further subdivided by a suffix. viz., GBLKR, VBLKR, and IBLKR are data blocks that reside on disc in the Reset Area (RA), and are written onto the Reset Tape (Reset) at the proper times.

---

[1] TM(L)-2760/351/00 (Reference a) Glossary, page 151 ff.

[2] TM(L)-2210/000/00 (Reference c)

3.2.1 The CBLK

This is a "read only" data block, i.e., it will be read into core by the monitor, SYMON, and will be completely protected against being changed, on disc, by any program. This is not to say that CBLK values cannot be changed, in core, during the execution of a particular program; such changes, however, will be transitory, since the CBLK will be read into core, from disc, as part of the environment of any subsequent program needing it.

3.2.2 The TBLK

This is purely a structured, defined buffer-area, which may be used either for intra-program communication (e.g., as KREFPOOL has been) or for read-in or write-out of data on other tapes (e.g., Ephemeris, Tracking Data, etc.). The monitor, SYMON, will allocate these buffer-areas into core, for the whole TBLK, as defined in the compool, but will transfer no data at load-time.

3.2.3 The MBLK

Items, tables, or arrays in an MBLK are likewise a matter for space-allocation, only, by SYMON--no data will be read into these areas by the executive at load-time. However, unlike the TBLK, the MBLK will not be allocated as an entity. SYMON will allocate core-space for only those items, tables, or arrays used by the program whose environment SYMON is setting up.

3.2.4 The GBLKR

All GBLKR data are guaranteed-value data. (GBLKR's reside on the disc in the RA, and when it is written, on the Reset Tape.) All GBLKR's and GBLKR-values referenced by a program are read into its core-environment before the program operates; all GBLKR's and GBLKR-values that were referenced by any "setting" instruction, will be rewritten back onto disc, unless inhibited, when the program exits and before the succeeding program is loaded. Thus, the disc will always contain the very latest guaranteed data.

3.2.4.1 This updating procedure will be automatic, unless suppressed by the operating programs. For the circumstances where human choice may be exercised, e.g., in the acceptance of a KODM-vector, or in the case of error returns, it is expected the program would delay, or negate, this updating of GBLKR's on the disc.

3.2.4.2 All GBLKR's are fixed-length, although they may include tables and arrays which are defined as "variable"--this variability is as defined in JOVIAL, and is a variability for usage, rather than referring to the core-space reserved.[1]

3.2.4.3 While the CBLK is designed for "constants" data, it is recognized that 'constant' is a relative term and that many values thought of in this way are empirical values, 'constant' only in a particular context; such values will be better accommodated as a GBLKR. An example is the set of zonal and tesseral harmonics (and associated longitudes). It is expected that there may be two, or more, sets of such values available for use, with the selection varying from project to project. Each set, then, will be on cards (or prestored). In the preparation of the initial Reset or RA by the Data Base Generator program, SDBG, the particular set required will be written as the GBLKR which will be available, thereafter, for all operational programs needing harmonics data.

3.2.4.4 There is another convenience attaching to this definition of certain "constants" into GBLKR's. Assume, for instance, that technical direction or project decision might indicate that a temporary (e.g., for one pass only) modification should be made to particular values in the GBLKR HARMON set of harmonics coefficients. These modifications can be accomplished with symbolic correctors,[2] input via cards or typewriter. Such newly-modified values, then, would remain part of the GBLKR until modified by some later program or until the end of the job.[3]

---

[1] TM(L)-2210/000/00 (Reference c)
[2] TM(L)-2760/351/00 (Reference a) page 61.
[3] op. cit., page 53.

The next job, of course, would get the value as defined in the GBLKR on its own Reset Tape.

3.2.4.5 What is now defined as a CBLK, can later be redefined, in the compool, into a GBLKR. For example, CBLK VSHGT, furnishes a table 'VSHGT, comprising items 'VH and 'VS, containing values of velocity of sound vs. height. An SDBG operation with its card (or tape) inputs would determine what values were actually to be loaded into GBLKR VSHGT. If this were redefined into a GBLKR VSHGT, the programs using 'VH and 'VS would require neither modification or recompilation.

3.2.5 The VBLKR and the IBLKR

These are defined in pairs, in a one-to-one correspondence. A VBLKR is a data block of "guaranteed-type" data, of length ≤ m records. Each record is n entries in a (JOVIAL) variable-length table of m entries (each entry may be j items) whose length, m, defines the maximum number of records the associated VBLKR may contain, and whose NENT defines the actual number of records of data the associated VBLKR does contain, at that point in time.

Thus, VBLKR STDATA, has a maximum length of ten records. Each record is a (JOVIAL) variable-length table of thirteen entries. Each entry is 26 words which contain 55 items, some of which are packed and some of which are overlaid. IBLKR ISTDATA comprises a (JOVIAL) variable-length table of ten entries. Each entry is two items, 'ISNO and 'ISREVS, which reproduce the similar two items from the last entry in each record in the associated VBLKR STDATA, which are 'SNO and 'SREVS.

3.2.5.1 IBLKR's will be read in by the monitor, SYMON, at load-time. VBLKR's may be allocated a one-record-length buffer (339 words, in the case of VBLKR STDATA) at load-time, but the user-program will fill that buffer, via SYMON routines, with whichever record he determines he wants by examining the associated IBLKR. If the program-need is for more than one record of the VBLKR, simultaneously, program-defined buffers will be set up by the programmer.

The contents of VBLKR's and IBLKR's have been described as "guaranteed-type" data, which means that the data are of a type that must be guaranteed-values, but that the updating of such data will be the responsibility of the program generating or modifying the values. This, of course, leaves the updating, by any program, to be decided by whatever logical criterion is suitable, but if the updating is not exercised, the next user of these data will get the same values as did the program at issue.

3.2.5.2 This technique offers great flexibility. The program which needs one record, or less, of VBLKR data may use the SYMON-provided buffer; the program which needs more of the VBLKR at one time may use a LIKE-table buffer of whatever length it needs, and may, optionally, use a SYMON-provided buffer. If operating experience dictates that VBLKR's should be lengthened or shortened, changes will be minimal: redefinition of the VBLKR, the IBLKR, and the program-internal buffer-sizes, plus recompilation.

3.2.5.3 A modification of this treatment exists in the case of IBLK IPBDMTX and VBLK PBDMTX (which, it will be noted, do not have the suffix "R" in their designation) since these data remain part of the system data base, and are not part of the RA or the Reset Tape. This IBLK-VBLK pair are specified and filled by SDBG in the initialization process in a manner similar to GBLKR's, IBLKR's, and VBLKR's.

3. EPHEMERIS INTERFACE DESCRIPTION

The following five pages are reprinted from DDI-13-44-01, a Data Dynamics, Inc., documented entitled "An Introduction to the Advanced Orbit/Ephemeris Subsystem.

3.3 EPHEMERIS INTERFACE

The orbit determination function, by fitting to tracking data, determines a set of initial conditions (position, velocity, drag, etc., at some epoch time) for use by the equations of motion of the ephemeris generation function. The orbit determination routines are the link between the "real" world of the space vehicle (as represented by tracking data) and the "mathematical model" world of the computer programs.

When the initial conditions have been determined, they are used to generate ephemerides predicting the future orbit of the vehicle. These ephemerides may take many forms: inertial ephemeris, local station

tracks, displays, etc. If each user were to generate his own ephemeris with similar but different equations of motion, a considerable expenditure of computer time would be required, duplicate information would be generated, and comparisons would become difficult to evaluate.

Instead of passing a vector of initial conditons to user programs and agencies, a basic inertial ephemeris is generated and stored by the AOES for use by these programs and agencies to compute and output commands, payload information, etc. The interface between the AOES and user programs is well-defined and accurate.

A knowledge of the ephemeris interface (as described in this section) and its associated data structure (as described in Appendix A) is fundamental to an understanding of the design and use of the AOES.

3.3.1 DEFINITION OF THE VEHICLE/ EPHEMERIS

The stored ephemeris is defined as IBLKR, VBLKR's. A significant departure from the existing system (OES), is the provision for multiple orbits per vehicle and for multiple vehicles within one data base. The single vehicle, single ephemeris capability is, of course, retained as a special case in the AOES. Each ephemeris has a unique identification (tagged by system time from the real-time clock). A listable directory, 'VEDIR block, enables the user to determine the name, vehicle/ ephemeris number, and the offset time for each of the stored vehicle/ ephemerides.

The stored ephemeris is defined, generated, and manipulated as follows:

- On the disc there is room for 48 vehicle/ephemerides. Each vehicle/ ephemeris is identified by a vehicle number, ephemeris number (30-bit identification computed from the system time when the ephemeris was generated), and an internal/ external flag. An ephemeris generated

by a function card request, 'DGEN, is termed "external", while an ephemeris generated, for example, by an orbit determination iteration is termed "internal". The vehicle number, ephemeris number, and the internal/external flag together make up the vehicle/ephemeris number.

- To facilitate easy identification of vehicle/ephemerides, the user may assign an eight-character BCD label to each vehicle/ephemeris.

- A COMPOOL-defined IBLK, 'VEDIR (vehicle/ephemeris directory), contains a table, 'VETBL, of those ephemerides currently allocated disc storage space. 'VETBL consists of 48 entries containing the vehicle/ephemeris number, a BCD identifier for each vehicle/ephemeris, and the offset time (i.e., number of days from 1 January 1950) for the vehicle/ephemeris.

- There are many vehicle/ephemeris-specific and vehicle-specific blocks in the AOES. An example would be observations which are vehicle-specific, and ephemeris data points which are vehicle/ephemeris-specific. To minimize the disc storage requirements, the concept of "dummy" and "working" vehicle/ephemerides was introduced. There must be at least one "dummy" vehicle/ephemeris for each vehicle and up to 47 "working" vehicle/ephemerides associated with the "dummy".

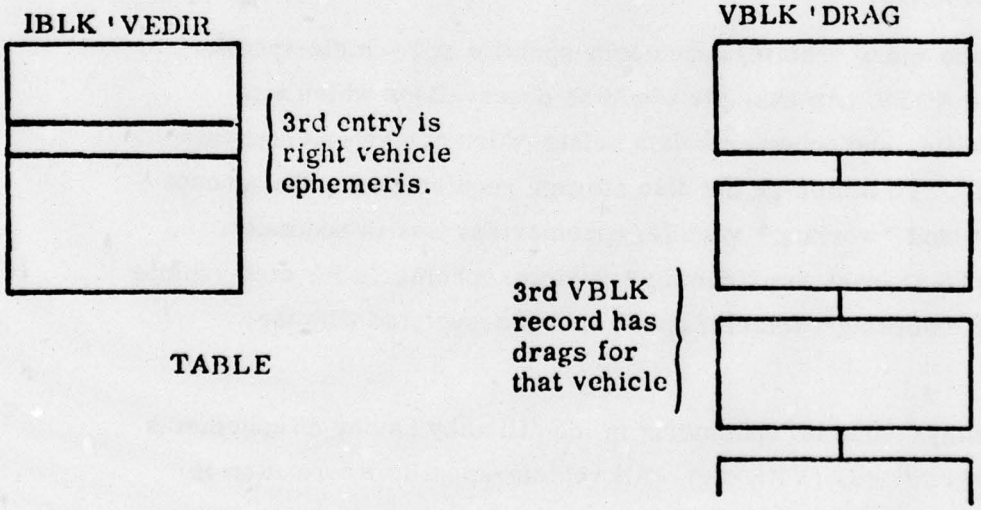
- A "dummy" vehicle/ephemeris is identified by having an ephemeris number of zero (i.e., 'VEE = 0). All vehicle-specific information is associated with the "dummy". Using the "dummy" as the base, numerous "working" vehicle/ephemerides may be created.

- A "working" vehicle/ephemeris contains all vehicle/ephemeris-specific data such as drags, OAJ's, initial conditions, etc. The concept of a "working" vehicle/ephemeris is especially useful for planning

purposes. For example, many "working" vehicle/ephemerides may be generated using different orbit adjusts to study the behavior of the orbit for the different cases.

- Initial conditions, drags, orbit adjusts, and all other data that affect an ephemeris reside on disc. For example, there may be 48 drag tables on the disc at one time. Such data is organized into VBLK's. User programs are responsible for getting VBLK information into core: core is allocated but no data is transferred. User programs must use the data handler ('SDAHA) to read VBLK information into core. The programs index through 'VEDIR, determine the entry number of the vehicle/ephemeris they are using, and request the corresponding record number from 'SDAHA.

Example:

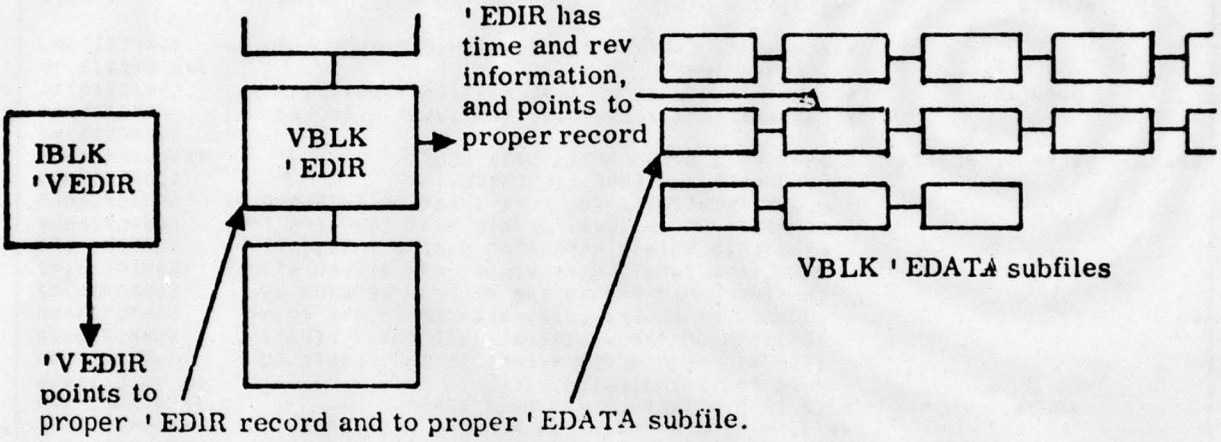


3.3.2 STORING AND USING THE V/E

The situation is complicated somewhat when attempting to store the ephemerides themselves. Here, each ephemeris contains a large, variable number of ephemeris data records and requires a directory

to facilitate finding a desired time point in the file.

Example:



SYMON and COMPOOL capability allows VBLK's to be defined with subfiles. One COMPOOL definition in effect provides for multiple files of information. To read a record from such a VBLK, the data handler ('SDAHA) is given the VBLK name, the subfile name or number, and the number of the desired record. The above diagram shows how such information is obtained. A search of the table entries ('VETBL) in 'VEDIR provides an entry number which is used to point to the proper 'EDIR (Ephemeris Data Directory). The 'EDIR defines a subfile of 'EDATA records.

Entries in 'EDIR contain time and rev information and a "SYMON number" for each ephemeris data record. The user may, therefore, search the directory to locate the record of interest, then pass the "SYMON number" to 'SDAHA to request the proper 'EDATA record.

Figures 1, 3, and 5 of Appendix A contain detailed illustrations on the structure of 'VEDIR, 'EDIR, and the overall AOES data base structure.

4. COMPOOL DEFINITION OF THE EPHEMERIS

The following three pages are reprinted from SDC TM-(L)-5048/813/00.

```
                                                           SYSTEM DEVELOPMENT CORPORATION
        29 OCTOBER 1976                     91                     TM-(L)-5048/813/00


        #EDATA                  COMSYS XQ - SYSTEM II COMPOOL FOR MODEL 15.18          #EDATA
        ↓          ↓            ↓         ↓         ↓         ↓         ↓          ↓
                                ##SUN-VEHICLE-EARTH AND SUN-VEHICLE-MOON ANGULAR   S08XC215650
                                  INFORMATION                                    ##S08XC215700
         OVERLAY                #ECARAY=#ECSVE,#ECALFA,#ECBETA,#ECSVED,#ECADOT,    S08XC215750
                                #ECSVM,#ECGAMA,#ECSVMD,#ECGDOT,#ECMVE,#ECLAZ  $    S08XC215800
                                                                                   S08XC215850
EDATA   VBLKR      #EDATA       R 2622 RV $ ##EPHEMERIS DATA BLOCK               ##S08XK215900
                                ##DISC NAME, NO CORE ALLOCATED.                    S08XC215950
                                  BLOCK #EPHIOB IS THE CORE IMAGE OF THIS BLOCK.   S08XC216000
                                  CONSISTS OF AN ARRAY #EDATX THAT CONTAINS THE    S08XC216050
                                  EPHEMERIS POINTS GENERATED DURING INTEGRATION.   S08XC216100
                                  THERE ARE TWENTY-NINE POINTS PER RECORD STORED   S08XC216150
                                  IN TIME ORDER WITHIN THE RECORD. RECORDS ARE     S08XC216200
                                  INDEXED BY BLOCK #EDIR. EACH EPHEMERIS POINT     S08XC216250
                                  CONSISTS OF TWELVE ITEMS. (SEE ARRAY #EDATX).    S08XC216300
                                  ITEM #XEPH IN BLOCK #VEBKI IS THE EXISTENCE      S08XC216350
                                  FLAG FOR THE RELATED FILE.                     ##S08XC216400
         ARRAY     #EDATX       1 12 29 F $ ##EPHEMERIS DATA ARRAY               ##S08XC216450
                                ##(0) = TIME OF POINT IN MINUTES FROM OFFSET       S08XC216500
                                        UNITS(MIN)                                 S08XC216550
                                  (1) = X-COMPONENT OF POSITION                    S08XC216600
                                        UNITS(ER)                                  S08XC216650
                                  (2) = Y-COMPONENT OF POSITION                    S08XC216700
                                        UNITS(ER)                                  S08XC216750
                                  (3) = Z-COMPONENT OF POSITION                    S08XC216800
                                        UNITS(ER)                                  S08XC216850
                                  (4) = X-COMPONENT OF VELOCITY                    S08XC216900
                                        UNITS(ER/MIN)                              S08XC216950
                                  (5) = Y-COMPONENT OF VELOCITY                    S08XC217000
                                        UNITS(ER/MIN)                              S08XC217050
                                  (6) = Z-COMPONENT OF VELOCITY                    S08XC217100
                                        UNITS(ER/MIN)                              S08XC217150
                                  (7) = X-COMPONENT OF ACCELERATION                S08XC217200
                                        UNITS(ER/MIN**2)                           S08XC217250
                                  (8) = Y-COMPONENT OF ACCELERATION                S08XC217300
                                        UNITS(ER/MIN**2)                           S08XC217350
                                  (9) = Z-COMPONENT OF ACCELERATION                S08XC217400
                                        UNITS(ER/MIN**2)                           S08XC217450
                                  (10)= FIRST ATMOSPHERIC PARTIAL                  S08XC217500
                                        UNITS(ER)                                  S08XC217550
                                  (11)= SECOND ATMOSPHERIC PARTIAL                 S08XC217600
                                        OR INTEGER REV NUMBER OF THE POINT.        S08XC217650
                                DOUBLE POINTS ARE STORED FOR DISCONTINUITIES,      S08XC217700
                                E.G., DRAG CHANGE OCCURRED AT TIME X (X = POINT    S08XC217750
                                OF DISCONTINUITY). TWO POINTS WITH IDENTICAL       S08XC217800
                                TIMES WILL BE STORED. THE FIRST POINT WILL         S08XC217850
                                CONTAIN THE OLD DRAG ACCELERATION AND THE SECOND   S08XC217900
                                POINT WILL CONTAIN THE NEW DRAG ACCELERATION.ALL   S08XC217950
                                REV CROSSING POINTS ARE ALSO STORED. INCOMPLETE    S08XC218000
                                DATA RECORDS (I.E., RECORDS CONTAINING LESS        S08XC218050
```

Figure 15. Compool Definition of the Ephemeris (1 of 3)

```
      #EDIR                 COMSYS XQ - SYSTEM II COMPOOL FOR MODEL 15.1B                      #EDIR
      +          +          +      +      +      +      +                                      +
                               THAN TWENTY-NINE EPHEMERIS POINTS) ARE PADDED                S08XC218100
                               WITH ALL UNUSED WORDS SET TO THE VALUE                       S08XC218150
                               CONTAINED IN ITEM #LFN (SEE BLOCK #NUMCON).                  S08XC218200
                               BY MODIFYING ITEM #NTHPT IN BLOCK #VEBK1, THE                S08XC218250
                               NUMBER OF POINTS STORED CAN BE VARIED.                       S08XC218300
                               SEE ALSO DESCRIPTION FOR BLOCK #EPHIOB.                    ##S08XC218350
                                                                                            S08XC218400
EDIR      VBLKR      #EDIR     R 48 RV S ##MASTER EPHEMERIS DIRECTORY BLOCK.              ##S08XK218450
                             ##THIS INDEXES A FILE (#ENTNUM) OF RECORDS OF                  S08XK218500
                               EPHEMERIS SUB-DIRECTORIES IN THE BLOCK #DEDIR.               S08XK218550
                               EACH ENTRY IN #DETABL POINTS TO ONE EPHEMERIS                S08XK218600
                               SUB-DIRECTORY FILE.  THESE ENTRIES ARE STORED IN             S08XK218650
                               TIME SEQUENCE AND ARE USED FOR QUICK ACCESS TO               S08XK218700
                               THE FILE WHICH CONTAINS THE EPHEMERIS DATA                   S08XK218750
                               DIRECTORY OF INTEREST.  ONE #EDIR FILE PER                   S08XK218800
                               VEHICLE EPHEMERIS.                                         ##S08XK218850
          ITEM       #ESTIME   F S ##START TIME OF EPHEMERIS GENERATION                   ##S08XC218900
                             ##I.E., EPOCH TIME. THIS DOES NOT NECESSARILY                  S08XC218950
                               EQUAL THE EARLIEST EPHEMERIS TIME (#ETIME1)                  S08XC219000
                               SINCE EPHEMERIS DATA CAN BE GENERATED PRIOR TO               S08XC219050
                               EPOCH.                                                       S08XC219100
                               UNITS(MIN)                                                 ##S08XC219150
          ITEM       #EREV1    I 48 S S   ##EARLIEST REV IN FILE                          ##S08XC219200
                             ##FOR WHICH EPHEMERIS DATA HAS BEEN                            S08XC219250
                               GENERATED FOR THIS V/E                                     ##S08XC219300
          ITEM       #ETIME1   F S ##EARLIEST TIME IN FILE                                ##S08XC219350
                             ##IN MINUTES FROM OFFSET. FOR WHICH EPHEMERIS                  S08XC219400
                               DATA HAS BEEN GENERATED FOR THIS V/E                         S08XC219450
                               UNITS(MIN)                                                 ##S08XC219500
          ITEM       #UCOTR    I 48 S S ##COUNT OF TOTAL RECORDS##                          S08XK219550
          ITEM       #EPHEND   B S ##EPHEMERIS IS/IS NOT FULL INDICATOR                   ##S08XC219600
                             ##1 = VBLK #EDATA IS FULL                                    ##S08XC219650
          ITEM       #D1PERH   F S  ##1/SCALE HEIGHT AT EPOCH FOR #DANA##                   S08XC219700
                             ##NEGATIVE RECIPROCAL OF SCALE HEIGHT. THIS ITEM               S08XC219750
                               IS EVALUATED AT EPOCH BY #DAD FOR USE (BY                    S08XC219800
                               #DANA) IN PARTIALS CALCULATIONS. SCALE HEIGHT                S08XC219850
                               IS DEFINED AS THE DIFFERENCE IN HEIGHT OVER                  S08XC219900
                               WHICH THE ATMOSPHERIC DENSITY DECREASES BY A                 S08XC219950
                               FACTOR OF E.                                                 S08XC220000
                               SEE ITEM #LOGRPA IN BLOCK #CDTBLK                            S08XC220050
                               UNITS(ER)                                                  ##S08XC220100
          TABLE      #ETABL    V 57 S 2 S ##NENT = NUMBER OF EPHEMERIS                      S08XK220150
                               SUB-DIRECTORIES.  THERE IS ONE ENTRY FOR EACH                S08XK220200
                               FILE OF THE EPHEMERIS SUB-DIRECTORY ON DISC.                 S08XK220250
                               ENTRIES ARE SORTED IN ASCENDING TIME ORDER                 ##S08XK220300
           BEGIN                                                                            S08XC220350
            ITEM     #ETIME    F 0 0 S ##LAST TIME IN THE SUB-DIRECTORY FILE              ##S08XK220400
                             ##IN MINUTES FROM OFFSET. FOR WHICH EPHEMERIS                  S08XC220450
                               DATA HAS BEEN GENERATED FOR THIS V/E                         S08XC220500
```

Figure 15. Compool Definition of the Ephemeris (2 of 3)

```
#EPHIOB          COMSYS XQ - SYSTEM II COMPOOL FOR MODEL 15.18          #EPHIOBZ
  BEGIN                                                                 S08XC227900
    ITEM  #ET     F 0 0 $ ##TIME OF POINT, IN MINUTES FROM OFFSET       ##S08XC227950
                  ##UNITS(MIN)                                         ##S08XC228000
    ITEM  #EX     F 1 0 $ ##X-COMPONENT OF POSITION                     ##S08XC228050
                  ##UNITS(ER)                                          ##S08XC228100
    ITEM  #EY     F 2 0 $ ##Y-COMPONENT OF POSITION                     ##S08XC228150
                  ##UNITS(ER)                                          ##S08XC228200
    ITEM  #EZ     F 3 0 $ ##Z-COMPONENT OF POSITION                     ##S08XC228250
                  ##UNITS(ER)                                          ##S08XC228300
    ITEM  #EXD    F 4 0 $ ##X-COMPONENT OF VELOCITY                     ##S08XC228350
                  ##UNITS(ER/MIN)                                      ##S08XC228400
    ITEM  #EYD    F 5 0 $ ##Y-COMPONENT OF VELOCITY                     ##S08XC228450
                  ##UNITS(ER/MIN)                                      ##S08XC228500
    ITEM  #EZD    F 6 0 $ ##Z-COMPONENT OF VELOCITY                     ##S08XC228550
                  ##UNITS(ER/MIN)                                      ##S08XC228600
    ITEM  #EXDD   F 7 0 $ ##X-COMPONENT OF ACCELERATION                 ##S08XC228650
                  ##UNITS(ER/MIN**2)                                   ##S08XC228700
    ITEM  #EYDD   F 8 0 $ ##Y-COMPONENT OF ACCELERATION                 ##S08XC228750
                  ##UNITS(ER/MIN**2)                                   ##S08XC228800
    ITEM  #EZDD   F 9 0 $ ##Z-COMPONENT OF ACCELERATION                 ##S08XC228850
                  ##UNITS(ER/MIN**2)                                   ##S08XC228900
    ITEM  #EPARTA F 10 0 $ ##FIRST ATMOSPHERIC PARTIAL - OR #LFN        ##S08XC228950
                  ##UNITS(ER)                                          ##S08XC229000
    ITEM  #EPARTE F 11 0 $ ##SECOND ATMOSPHERIC PARTIAL                 ##S08XC229050
    ITEM  #ER     I 48 S 11 0 $ ##REV NUMBER OF THE POINT               ##S08XC229100
                  ##IF #EPARTA = #LFN                                  ##S08XC229150
  END                                                                   S08XC229200
  ITEM    #EBLK   I 48 U $                                            S08XC229250
                  ##THE DISC RECORD NUMBER OF THE EPHEMERIS             S08XC229300
                    DATA BLOCK IN CORE                                 ##S08XC229350
  ARRAY   #EIOA   360 F $##EPHEMERIS BLOCK INPUT/OUTPUT AREA           ##S08XC229400
  ITEM    #DATSNO I 48 U $ ##SYMON NUMBER OF THE CURRECT EPHEMERIS      S08XK229450
                    DATA RECORD THAT IS IN CORE                        ##S08XK229500
  ITEM    #DIRSNO I 48 U $ ##SYMON NUMBER OF THE CURRENT                S08XK229550
                    SUB-DIRECTORY RECORD THAT IS IN CORE               ##S08XK229600
  ITEM    #DIXNO  I 48 U $ ##INDEX OF THE CURRENT EPHEMERIS POINT IN    S08XK229650
                    #EPHIOB                                            ##S08XK229700
  ITEM    #DMENT  I 48 U $ ##MASTER EPHEMERIS DIRECTORY ENTRY NUMBER    S08XK229750
                    INTO #ETABL                                        ##S08XK229800
  ITEM    #DESENT I 48 U $ ##EPHEMERIS SUB-DIRECTORY ENTRY NUMBER       S08XK229850
                    INTO #DETABL                                       ##S08XK229900
  ARRAY   #EIOH   12 30 F $ ##EPHEMERIS DATA WORK ARRAY##               S08XK229950
  OVERLAY         #ETBL=#EBLK,#EIOA=#EBLK,#EIOB $                       S08XK230000
                                                                        S08XC230050
EPHIOBX TBLK      #EPHIOBX L $ ##LIKE BLOCK TO #EPHIOB                  ##S08XC230100
                  ##USED BY #DIFFER WHEN COMPARING EPHEMERIDES,         S08XC230150
                    AND USED BY #DEEDX.                                ##S08XC230200
                                                                        S08XC230250
EPHIOBZ TBLK      #EPHIOBZ L $ ##LIKE BLOCK TO #EPHIOB                  ##S08XK230300
```

Figure 15. Compool Definition of the Ephemeris (3 of 3)

APPENDIX C - GLOSSARY

Chief Programmer. An exceptionally proficient programmer charged with the technical and administrative performance of a Chief Programmer Team. In addition to being the chief designer and quality engineer of the programs assigned to the team, the Chief Programmer is charged with programming the top-most module or main routine in the program and the direction, training, guidance and administration of the members of the team.

Chief Programmer Team. A team of programmers organized around and in support of an exceptionally proficient programmer, consisting of a Chief Programmer, a Backup Programmer, and a Program Librarian plus as many Assistant Programmers or other specialists as is deemed necessary.

Compool (Communications Tag Pool). An array of information describing the structure, location and other pertinent information about stored data or the elements of a data base and program system, to be used by programs in symbolically referencing the data elements of a software system.

Compool Sensitive. Characteristic of a program capable of referencing a compool which indicates that the program refers to data only by their compool-defined symbology and never by machine address-related coding.

Computer Program Configuration Item. A computer program entity or element identified for configuration control, ranging from a single program to a complete software system and including a set of independent routines procured as an aggregate.

Configuration. The functional and/or physical characteristics of hardware/computer programs as set forth in technical documentation and achieved in a product. (AFR 65-3).

Configuration Control. The systematic evaluation, coordination, approval or disapproval, and implementation of all approved changes in the configuration item after formal establishment of its configuration identification. (AFR 65-3).

Configuration Control Board (CCB). A board composed of representatives of various functional organizations used to (1) serve as a body to review, verify classification of, and approve/disapprove proposed changes and deviations, and (2) to perform total impact evaluation of proposed engineering changes. (MIL-STD-XXX).

Configuration Status. The current content and structure and state of development or maintenance of a configuration, including all approved changes and verified deficiencies to the configuration.

Conventions. A customary or agreed upon manner of proceeding; a convention is established as a result of common practice or agreement.

Data Base. A file or files of information that exists in permanent or semi-permanent storage, excluding transitory or impermanent information, either to be operated upon or contributing to the operation of an information processing system.

Embedded Software. A computer program subsystem that is an integral part of a weapons system or Command, Control and Communication system and that is procured as one of the elements of the overall system and not as a separate entity.

Environment Analysis. The evaluation of the operating environment and interactions of a program system, including such information as all programs and data called or interacted with, the size and constitution of system elements, the mapping of storage and operating time allocations, and similar analyses.

Environmental Data. All data processed by an information processing system or influencing its processing, including program constants, parameters, variables, inputs, outputs, internal data stores, reference tables, and switches and switch settings.

Modern Programming Practices. Those programming tools, techniques or methods that are of proven worth in assuring Programming productivity or program quality of either a technical or a managerial nature.

Modularization. The practice of forming small units of code according to rules for singularity of function, minimization of external interfaces, simple control structures, and singularity of exits and entrances.

Naming Conventions. Rules for the formation of symbolic identifiers for program and data elements, often including coding to show element type, program or data module membership, and relative location as well as mnemonic content.

Paragraphing and Indentation. The practice of formatting code to show the structure of the program, including breaking the code into logical blocks with spacing and dividing symbols and the indentation of code to show subordination.

Program Commentary. The practice of including explanatory remarks with source code to enhance its understanding. Comments include prologue information describing a module or unit of code with interface definitions, environmental comment, and identifying author and composition date and annotations of the code to define data and operations.

Program Environment. All those factors or elements influencing the performance of a program including all data elements manipulated by the program and all interfacing hardware and software.

Program Librarian. A member of a programming team charged with the responsibility of mediating interactions with the Program Library or, in an interactive system, of monitoring Program Library activity, and maintaining logs of Program Library activity, files of program listings and of program documentation, and issuing periodic reports of library activity and contents.

Program Standards. An accepted criteria or established measure regarding a specified characteristic of a computer program, including rules for modularization, commentary, element naming, and paragraphing and formatting of source code.

Programming Style, Elements of. Rules for writing readable and understandable programs.

Project Environment. All those elements or factors establishing the conditions under which a project is performed including working facilities, customer relations, personnel organization and experience, computing facility, programming tools, and conditions of stress resulting from shortages of time, personnel or machine capabilities, managerial or customer pressure, etc.

Quality Assurance. Actions taken to ensure that all functional, structural and performance requirements specified for a system are met; commonly restricted to those reviews, audits, inspections and tests aimed at verifying and validating performance and designs but more generally any technique for ensuring quality.

Reviews and Audits. The periodic assessment of the relative excellence and state of a system and the evaluation of the degree of completeness and consistency of the system products. Military procurement practices identify System Design Reviews, System Requirements Reviews, Preliminary Design Reviews, Critical Design Reviews, and Functional and Physical Configuration Audits as the recognized reviews and audits for a system procurement.

Software. A set of computer programs, procedures and associated documentation concerned with the operation of a data processing system.

Software Development. Those activities undertaken to analyze, design, code and test software.

Software Development Life Cycle. All activities and events existing in the life of a computer program from its initial conception to its replacement or discontinuance of use, normally divided into conceptual, definition, design, development, test, and operations and maintenance phases.

Software Quality. Those attributes of a computer program that enhance its effectiveness and efficiency of use and maintenance such as reliability, efficiency, understandability, modifiability, maintainability, etc.

Standard. An accepted or established measure or criterion of performance or other characteristic of an object or activity by which its quality is judged.

Standard Practice. A standard or convention adopted for general use by a project.

System Data Control. The exercise of control over the content and organization of the data structures in an information processing system, including the processing flow, allocation of storage space and media and processing time, and control of changes to the data base definition.

System Librarian. A person assigned responsibility for the creation and maintenance of a Program Library for a project, controlling access to the library, ensuring observance of the programming standards and quality criteria for library members, ensuring the integrity of stored program configurations, and maintaining activity logs and library indexes and issuing regular reports.

ACRONYMS AND ABBREVIATIONS

AFSCF	Air Force Satellite Control Facility
COMPOOL	Communication Tag Pool
CPAC	Computer Program Associate Contractor
CPDC	Computer Program Development Center
CPIC	Computer Program Integration Contractor
JOVIAL J3	Jules Own Version of the International Algebraic Language, Model J3.
NASA	National Aeronautics and Space Administration
Reset Tape	A recording of the flight specific initialization data required to adapt the AF SCF support system to a particular operation.
RTS	Remote Tracking Station
SCOPE	Satellite Control for Operational Programs and Environment
SDF	Software Development Facility
SMTC	SCOPE Master Tape Control program
STAGE	Station Ground Environment simulation system
STC	Satellite Test Center
SYMON	System Monitor
SYSTEM I	A pre-COMPOOL Operating System for the CDC 3800
SYSTEM IIB	COMPOOL-sensitive Operating System for the CDC 3800

(Page 218 blank)

APPENDIX D - REFERENCES FOR COBRA DANE STUDY

1. Kernigham, B. W. and Plariger, P. S., "The Elements of Programming Style," McGraw-Hill, 1974.

2. Van Tassel, D., "Program Style, Design, Efficiency, Debugging and Testing," Prentice-Hall, 1974.

3. Meyers, G. J., "Reliable Software Through Composite Design," Mason/Charter, London, 1975.

4. Boher, F. T., "System Quality Through Structured Programming," Proceedings of the FJCC, 1972.

5. Baker, F. T., "Chief Programmer Team Management of Production Programming," IBM Systems Journal, No. 1, 1972.

6. Searle, L. V., "Computer Program Configuration Management," TM-5327, SDC, June 1974.

7. Brooks, F. P., "The Mythical Man Month," Addison Wesley, 1975.

8. TM-(L)-356/300, "COBRA DANE Programming Practices Manual," SDC, Oct. 12, 1973.

9. Yourdon, E., "How to Manage Structured Programming," Yourdon, Inc., 1976.

APPENDIX E - REFERENCES FOR COMPOOL-SENSITIVE SYSTEM STUDY

This appendix provides a list of source documents which are, with one exception, all SDC material. Most of these documents are available to authorized recipients through the Air Force Computer Program Development Library (AFCPDL), Santa Monica, California. No reference in this list is classified. However, a few are not available for general distribution.

1. TM-(L)-2318/000/00, "3600 Channel Configuration," SDC, 30 June 1966.
2. TM-(L)-2319/002/00, "Manual and Program Requirements for Passive Tracking Evaluation," SDC, 29 March 1966.
3. TM-(L)-2318/003/01, "Computer Program Development Center 3800 Computer Acceptance Test Package," SDC, 2 October 1967.
4. TM-(L)-2318/003/02, "Computer Program Development Center 3800 Computer Acceptance Test Package," SDC, 5 February 1968.
5. TM-(L)-1146/001/01, "Satellite Control Facility Information System Description: Tracking Information Flow," SDC, 9 May 1966.
6. TM-(L)-2735/704/01, "3600 Computer Operating Instructions for Model 9/1, BESST-9.1A (MS-7)," SDC, 17 October 1966.
7. TM-(L)-1071/029/00B, "Reset Tape Description," SDC, 25 January 1965.
8. TM-(L)-1736/036/02, "KODMTABL Description," SDC, 25 January 1965.
9. TM-(L)-1736/049/01, "KREFPOOL Description," SDC, 5 February 1965.
10. TM-(L)-2760/473/00, "System II Monitor (SYMON)," SDC, 1 August 1966.
11. TM-(L)-1835/000/01, "Interface Specification for Program 638," SDC, 1 June 1964.
12. TM-(L)-3583/000/00A, "Procedure for Producing and Maintaining Compools," SDC, 3 January 1968.
13* N-(L)-22553/000/00, "Listing of Data Being Considered for the System 2 Data Base," SDC, 24 May 1965.
14* N-(L)-22553/001/00, "The System 2 Data Base: Design Considerations, Criteria, and Objectives," SDC, 17 August 1965.
15* N-(L)-22553/002/00, "Two Model 9 Data Base Modification Plans," SDC, 11 October 1965.

*Not available for internal distribution.

16. TM-(L)-2222/013/00, "System Data Control;" Chapter XIII, Program System Design, SDC, 22 July 1965.

17. TM-(L)-2215/000/00, "System I and II Implementation Plan," SDC, 22 January 1965.

18. TM-(L)-2639/000/00, "Standards and Conventions for the SCF Software System II," SDC, 13 September 1965.

19. TM-(L)-2988/000/01, "Implementation of System I Programs in System II," SDC, 20 June 1966.

20. TM-(L)-2760/355/00, "A Proposal for Compool Contents for System II, Milestone 3," SDC, 28 January 1966.

21. TM-(L)-4199/000/00, "Compool," SDC, 11 January 1965.

22. TM-(L)-2760/351/01, "System II Monitor Program Design (SYMON), Milestone 3," SDC, 15 March 1966.

23. TM-(L)-2233/001/01, "References Relating to System I, Phases I and II," SDC, 27 April 1965.

24. TM-(L)-5527/000/01, "AFSCF Computer Program Segment Interface Control Documentation (ICD)," SDC, 29 November 1976.

25* SDC Model Turnover Letter L-SV-361 to AFSCF, 29 October 1969.

26* SDC Model Turnover Letter L-SV-401 to AFSCF, 10 December 1969.

27* SDC Model Turnover Letter LSCD-908 to AFSCF, 25 September 1970.

28* SDC Model Turnover Letter LSCD-1030 to AFSCF, 17 December 1970.

29. DDI-13-44-01, An Introduction to the Advanced Orbit/Ephemeris Subsystem, 1 June 1969.

30. TM-(L)-4163/001/00, "System Support Tape, SST-13A, Timing Measurements and Analysis," SDC, 31 March 1970.

31. TM-(L)-5550/000/00, "AFSCF Computer Program System Users Guide," SDC, 2 July 1975.

32. TM-(L)-4723, "Requirements for Conversion from SABER TO JOVIAL Direct Code," SDC, 11 September 1965.

33* System Development Corporation Bulletin, 18:1, "SDC Supports AFSCF for 17th Year," 10 January 1977.

*Not available for external distribution.

34. TM-(L)-5048/711/00, "System IIB Users Manual, Model 15.1," SDC, 6 August 1975.

35. TM-(L)-5048/713/00, "AOES Users Manual, Model 15.1B," SDC, 6 August 1975.

36. TM-(L)-5048/813/00, "System II 3800 Computer Program Model 15.1B Comsys XQ," SDC, 29 October 1976.

MISSION
of
Rome Air Development Center

RADC plans and conducts research, exploratory and advanced development programs in command, control, and communications (C^3) activities, and in the C^3 areas of information sciences and intelligence. The principal technical mission areas are communications, electromagnetic guidance and control, surveillance of ground and aerospace objects, intelligence data collection and handling, information system technology, ionospheric propagation, solid state sciences, microwave physics and electronic reliability, maintainability and compatibility.