JPL PUBLICATION 78-53

Standard Practices for the Implementation of Computer Software

A. P. Irvine
Editor

September 1, 1978

National Aeronautics and
Space Administration

Jet Propulsion Laboratory
California Institute of Technology
Pasadena, California 91103

JPL PUBLICATION 78-53

Standard Practices for the Implementation of Computer Software

A. P. Irvine
Editor

September 1, 1978

National Aeronautics and
Space Administration

Jet Propulsion Laboratory
California Institute of Technology
Pasadena, California 91103

The research described in this publication was carried out by the Jet Propulsion Laboratory, California Institute of Technology, under NASA Contract No. NAS7-100.

FOREWORD

The JPL Tracking and Data Acquisition Staff, which designs, implements, and manages the Deep Space Network for NASA's Office of Space Tracking and Data Systems, realized more than 8 years ago that the tracking stations were beginning to look more like computer facilities. A major difficulty in reaping the full benefits of computers for providing tracking services was the inability to make software development an understood branch of engineering. Under the leadership of Walter K. Victor, then Deputy Assistant Laboratory Director for Tracking and Data Acquisition at JPL, formal efforts were begun in software methodology research. In addition, within the Tracking and Data Acquisition Office itself, a software seminar was started for the management staff. The goal was to educate ourselves so that we could manage software better as an extension of the engineering management techniques already being used for hardware. One result of this long effort is the issuance of the practices presented here.

ACKNOWLEDGMENT

This work is a revision of six Standard Practices that were originally developed by the engineering staff at the Jet Propulsion Laboratory involved in the implementation of various computers into the Deep Space Network. Writing and production of the original six practices was coordinated by Daniel C. Preska. The review process of those practices was managed by Edward C. Posner. Mahlon F. Easterling, then Manager of the Tracking and Data Acquisition Planning Office at JPL, oversaw the entire process. Technical documentation support was provided by Richard C. Chandlee. Special acknowledgment should be given to R. C. Tausworthe, P. I. Westmoreland, and A. F. Ellman for their contributions.

ABSTRACT

Standard Practices for the Implementation of Computer Software provides a standard approach to the development of computer programs. This approach covers the life cycle of software development from the Planning and Requirements phase through the Software Acceptance Testing phase. All documents necessary to provide the required visibility into the software life cycle process are discussed in detail.

CONTENTS

CONTENTS (cont'd)

CONTENTS (cont'd)

CONTENTS (cont'd)

CONTENTS (cont'd)

CONTENTS (cont'd)

CONTENTS (cont'd)

CONTENTS (cont'd)

CONTENTS (cont'd)

Figures

Figures

CONTENTS (cont'd)

Figures

Tables

CHAPTER 1
GUIDELINES AND POLICIES

SECTION I
INTRODUCTION

A. PURPOSE

The purpose of this Standard Practice is to provide a standard approach for implementing operational computer programs. Specific aims of the standardized implementation approach are to:

(1) Produce operational software which is correct, on schedule, and within cost limitations.
(2) Provide a methodology for software implementation management comparable to that which exists for hardware implementation.
(3) Assure compatibility with existing and planned capabilities and equipment.
(4) Satisfy post-delivery needs (ease of operating, maintaining, and sustaining).
(5) Reduce program costs over the life cycle by balancing implementation and operational costs with needs.
(6) Provide a means for effective communications and efficient coordination of personnel and disciplines throughout the implementation.

This Standard Practice therefore prescribes an orderly and effective way of implementing correct and maintainable software.

B. SCOPE

The scope of this Standard Practice extends from program justification activities through transfer and delivery of the implemented program.

C. APPLICABILITY

These Standard Practices were originally designed to govern the development of software for the Deep Space Network (DSN) of the Jet Propulsion Laboratory. Consequently, there are occasional references to institutions unique to the DSN. However, the principles described here may be generalized to apply to the development of all software where there is a desire for reliability, maintainability, and usability.

SECTION II
METHODOLOGY AND POLICY

A. METHODOLOGY FOR IMPLEMENTING SOFTWARE

These software implementation methods and practices are based on the application of engineering practices proven to be effective in the implementation of hardware, coupled with the concepts and theorems of structured programming and its consequent enabling of top-down construction, which, in turn, is conducive to documentation of the implementation activities as they occur.

These methods enable effective team operations for large efforts and effective application and management of the practices and concepts. The methods are also complementary and compatible with other methods currently used (and, for that matter, with new technology under development) for the more iterative, parallel thought processes involved in arriving at a computer program definition starting from a problem, performing an analysis, and evaluating proposed solutions. Moreover, having obtained this program definition, the techniques described here provide for an efficient, orderly, and complete program construction. Such program construction is characteristically a linear or serial process, whereas much of the iterative process ideally should occur in advance of the actual implementation.

The methodology has the following three basic elements:

(1) Principles of structured programming
(2) Top-down construction
(3) Concurrent design and documentation, coding, and testing

These three elements are more fully detailed in Reference 1.

1. Principles of Structured Programming

Structured programming is a program organizing and implementing activity that involves the concept of representing the control logic of arbitrary computer programs with iterations of a limited set of basic prescribed program (or flowchart) structures. Application of this concept of iterating and nesting a small number of standard structures results in a highly organized and structured representation of the design and code that is relatively easy to read, implement, test, and understand. Coding of the design by using prescribed code structures is referred to as "structured coding." Also, structured coding typically makes use of indentation and line spacing to display the program flow and modularity clearly to persons reading the code, so that the reader can locate the code corresponding to flowcharts or to other module descriptions.

A counterpart of structured programming in hardware implementation can be found in the design of complex control logic circuits. There, it is a widespread standard engineering practice to use only a specified small set of basic logic operations such as AND, OR, and NOT in the design of any arbitrarily complex logic functions, so as to provide a structured design that is relatively easy to understand, implement, and test. This practice is theoretically sound, and is based on a theorem in Boolean algebra that states that arbitrarily complex logic functions can be expressed in terms of basic AND, OR, and NOT operations.

Analogously, the Structured Programming Theorem (which applies to any so-called "proper program" — any program with only one entry and exit and no unenterable subprograms) states that any (nonreal-time) program, arbitrarily large and complex, can be expressed by basic structures that need include only the operations for performing (1) functional sequencing, (2) conditional branching, and (3) conditional iteration (looping). The set of primary, prescribed program structures that satisfies the theorem is presented in References 1 and 2. This prescribed set of structures is shown in Figure 1-1 and consists of:

(a) FUNCTION structure, consisting of any computational element, including the possibility of not performing any operation.

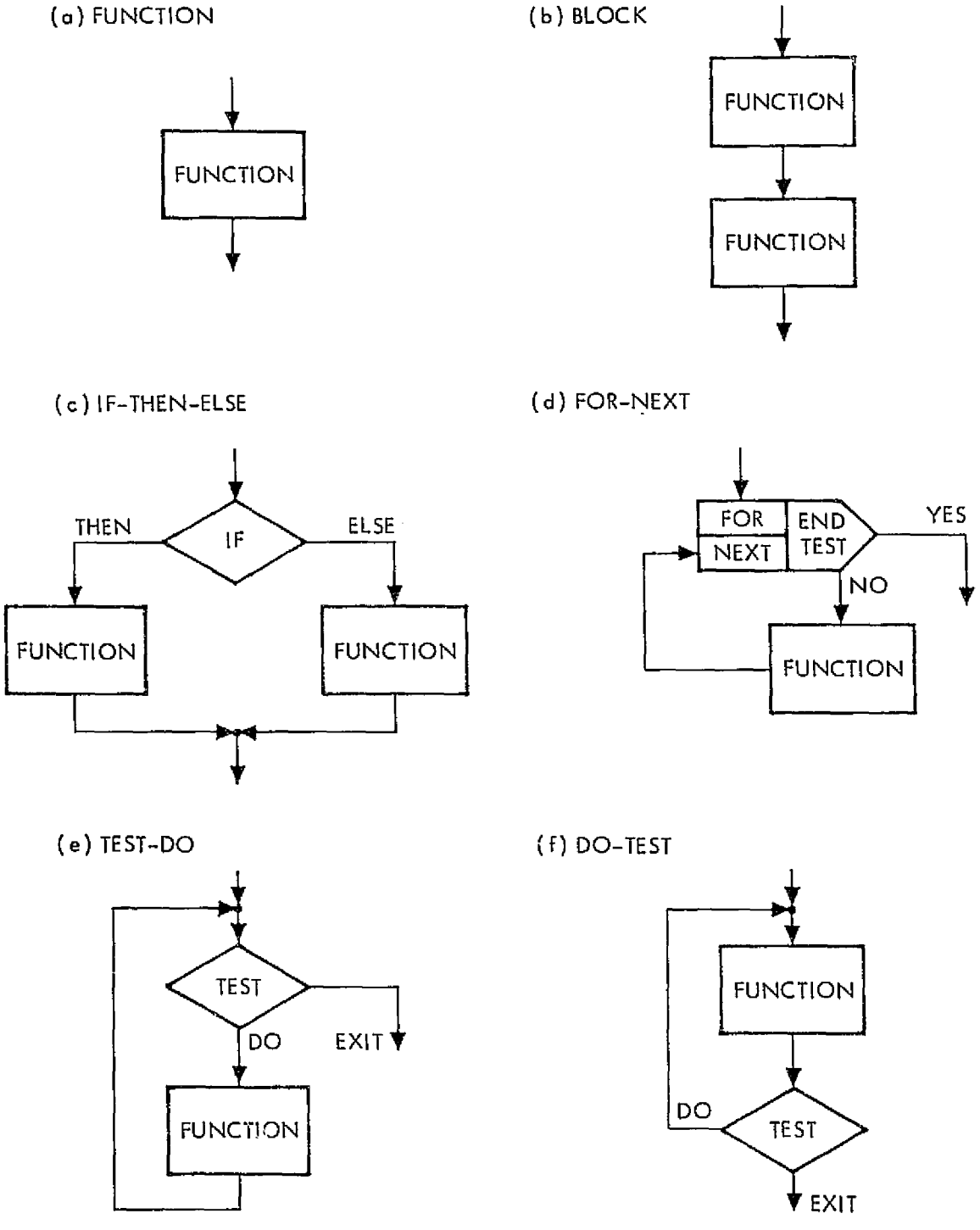


Figure 1-1. Primary Program Structures

(b) BLOCK, consisting of performing two or more FUNCTIONS in sequence.

(c) IF-THEN-ELSE structure, consisting of a conditional branch element (multiple - more than two - possible alternatives are permitted in a slightly generalized form of IF-THEN-ELSE; see Reference 1).

(d) FOR-NEXT structure, which provides a conditional loop. It is an indexed special case of the TEST-DO structure described below. The FOR part of FOR-NEXT is actually an initialization function before the TEST-DO; the function of the TEST-DO includes an increment of the loop index.

(e) TEST-DO structure, which tests for condition, then does the function.

(f) DO-TEST structure, which does the function, then tests for the condition before repeating.

The ease with which the primary structures can be translated into code depends upon the characteristics of the programming language. To facilitate this translation, the DSN intends to adopt the language MBASIC* as its standard for nonreal-time programs. The main advantage, however, is the portability of DSN application programs from one machine to another.

A sample flowchart that demonstrates the use of these structures is presented in Figure 1-2, where Modules 1, 3, 4, 5, and 7 represent functional sequencing operations, Module 2 represents a conditional iteration (looping) operation, and Module 6 represents a conditional branching operation. This flowchart was taken from the WAD Report Writer Pilot Software Project, which employed structured programming and top-down, concurrent principles. It should be noted that each complete structure shown in Figure 1-2 has only one entry and one exit. This characteristic of structured programming allows the modules to be expanded (or consolidated) to construct flowcharts of any length or complexity. For example, Modules 2 through 7, when considered together, represent a function module. The amount of detail provided on any one chart, therefore, can be chosen by the designer. A striped module

*A trademark of the California Institute of Technology.

ORIGINAL PAGE IS
OF POOR QUALITY

Chart 1.5
WADRPT
3 of 3
5-12-74

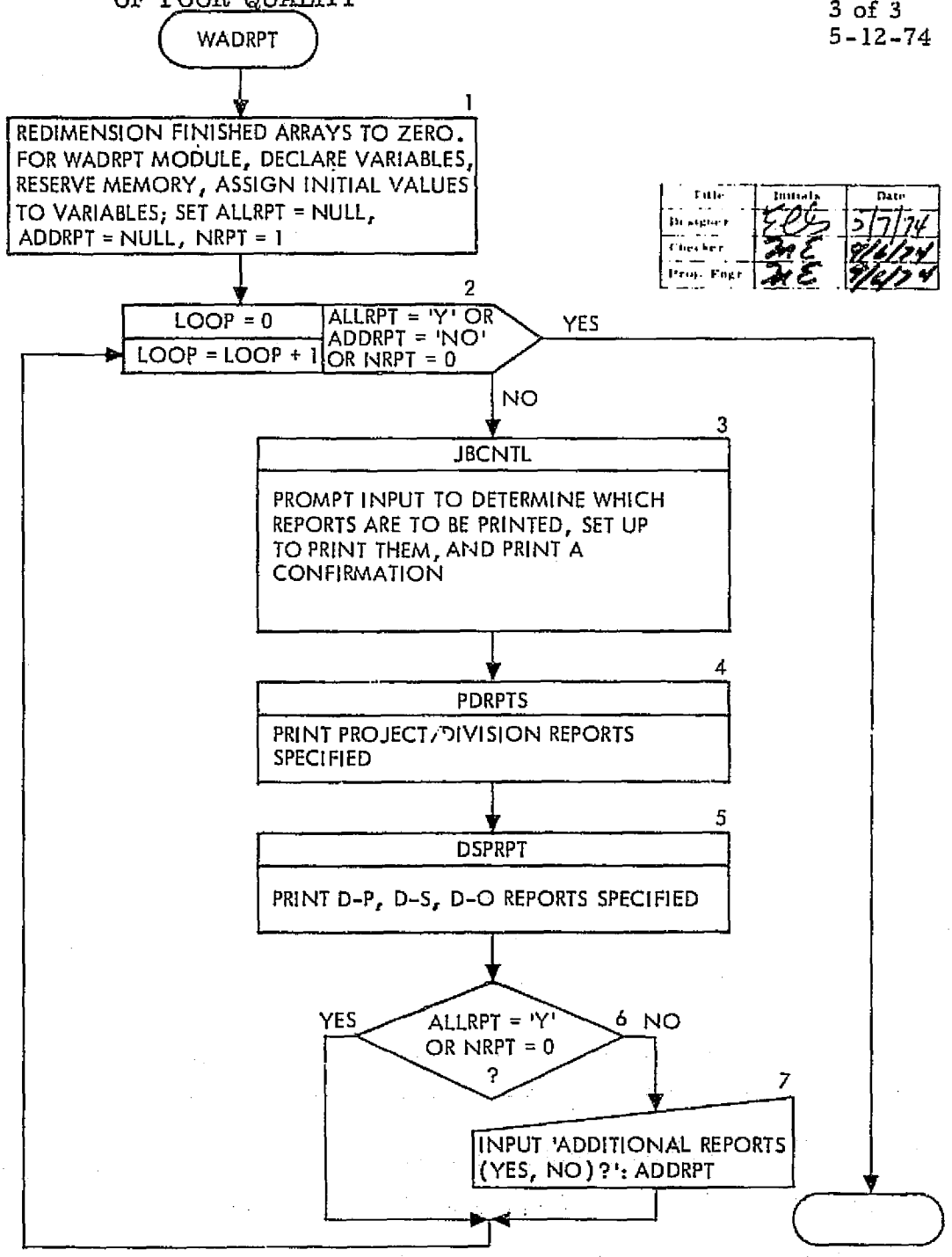


Figure 1-2. Example of Structured Flowchart

(blocks 3, 4, and 5) is used to indicate that further expansion is provided on separate flowcharts. Limiting the amount of detail on a given flowchart allows the total process and flow under immediate consideration to be presented and read literally from top to bottom.

Other items to note in Figure 1-2 are the provision and use of:

(1) Chart Identification - placed in upper right corner, containing chart number, module name, paging, and date. This allows easy access and reference by users and quick and easy updating by page releases.
(2) Approval Block - for Cognizant Development Engineer's control (or PE).
(3) Program Structure - uses prescribed primary set of program structures (see Figure 1-1).
(4) Standard Symbols - uses symbols and module numbering system* as presented in Reference 1 (derived from ANSI and extended for structured programming).
(5) Update Bars - placed in outside margin for flagging the most recent changes.

2. Top-Down Construction

The top-down approach involves the concept of stating the total problem in its operational environment and progressing in a systematic manner to greater levels of detail to fully understand the problem and to define a solution (a proposed program) based on this understanding. The early stages of this program-defining process are characterized by iteration and nearly parallel thought processes, of which most, ideally, should have been completed in advance of the implementation. Nevertheless, some time must be allowed for this activity, as estimated by the CDE. The iterations converge on those basic characteristics from which the program can be defined. An orderly, more serial process of program production or construction follows and includes:

*Each module is numbered uniquely for identification; required explanatory narrative is keyed to this number, and supplied with the flowchart. More detail on this procedure will be found in Chapter 4.

final detail design, translation of this design to code, auditing and testing the code, and documentation of the design, code, and test processes. The top-down procedure for this construction process, as described later, is referred to as top-down construction.

Specific procedures for performing the earlier thought processes have not yet been formalized or prescribed, since development is still underway and also, ideally, many of the earlier processes should have occurred in advance of the implementation. It is of interest to note that the top-down approach applies to the earlier activities, as well as to the construction activities, although it is not necessarily applied to specific program modules but more to the related aspects of data, function, and procedure. The data aspect deals with information and data in the problem and in the program, including input, intermediate, and output data objects and structures. The function aspect deals with the transformations acting on the data, i.e., what is done to transform one data object into another in the passage through the program. The procedure aspect deals with the logic to select paths through the program, and the sequence of operations (functions and their component instructions) on these paths, i.e., how the program performs its task.

Also, it should be noted that while the top-down approach for the highest (least detailed) levels of the program leads to a hierarchical tree expansion, the lower, more detailed levels may lead to very similar or identical module functions in several independent expansion paths. To avoid unnecessary duplication of implementation effort, and in the interests of promoting commonality and economy in subsequent design and maintenance, it is prudent that the program definition identify and accommodate these situations. These instances can be recognized by forming a candidate high-level definition or structure at least once, although typically, several iterations may be needed depending on size and complexity. Emerging from ongoing development are suggestions on special tools and aids for handling this commonality aspect.

The top-down approach is also a valuable management tool, in that the resulting end-to-end overall definition of the proposed program and its component parts and structure provides program design guides and data needed for

estimating the scope of the total job, for determining the nature of the work and needed resources, for planning and scheduling the work through the various phases, and for managing and conducting design reviews.

Top-down construction is especially adapted to Structured Programming because the Structure Theorem allows the top levels to be constructed first and then the lower levels to be detailed (unstriped) and constructed in an orderly and rigorous fashion. Specifically, in terms of detailing and flowcharting in preparation for construction, one starts with a single striped module (level zero). That module is analyzed and expanded into a flowchart with two or more modules. The structure of the flowchart must be either a prescribed structure (e.g., from Figure 1-1) or a permitted combination of prescribed structures, where any or all of the modules may be striped. Each striped module at this point (level 1) is expanded into a flowchart in the same way. The modules which are not striped need not be expanded, and this means that they can be directly translated into code without further design. This process is repeated at the next level and continued until there are no striped modules which have not been expanded. However, as discussed under Paragraph B, Item 6.a.3, several levels may be combined into builds for purposes of concurrent coding and testing, so that the correct amount of complexity is tested, balanced against the cost of testing.

External interfaces are defined, negotiated, and implemented early, and then used in the subsequent development, thus minimizing the occurrence of possible serious program integration problems after the internal development has been completed. Also, the need for developing program "drivers" is reduced or eliminated. This progression of the implementation from the interfaces into the detailed program computations is fundamentally sound and is consistent with the theory of computable functions, which requires that at any point of computation all elements needed to compute the next value have already been computed.

3. Concurrent Design and Documentation, Coding, and Testing

In terms of coding and testing, any unstriped module can be coded as soon as the flowchart on which it appears has been completed and signed off. Moreover, the program can be run provided any striped modules are properly represented by modules of temporary code called dummy stubs (code which produces data values needed to run the rest of the program). The dummy stubs are intended to work for one or more special test cases. There is a theorem in Structured Programming which says that if the part already coded is proven to be correct, then it will still be correct after the rest of the program is coded, and thus need not be checked again. The test cases verify, build by build, the correctness of the module when imbedded in higher builds of code by testing every module-connecting path and performing other tests as needed to uncover, for example, errors in logic, computation, formatting, timing, recovery, and documentation. Correcting errors as they are introduced minimizes present rework and avoids later serious impacts due to compounding of errors. It can therefore be expected that the total amount of testing (correctness and acceptance) will be greatly reduced, since, at completion of coding, there is no need to repeat extensive internal program (correctness) testing. The program is correct. Needed are only those tests (acceptance) that demonstrate to the user the program's responsiveness to requirements in its full operational environment. The modifications to this paragraph needed for real-time programs are under development.

Figure 1-3 represents the process followed in doing concurrent design and documentation, coding, and testing from the top down. Note that Figure 1-3 is a process flowchart (sequence of manual activities in the implementation process), not a computer program flowchart (sequence of machine operations in the program), but follows the applicable structured flowcharting conventions. The figure shows that the module design as documented must be approved before the module coding and correctness test design are begun, both of which must, in turn, be completed and checked before the correctness testing is performed. However, after design approval, the test is designed at the same time as the coding is being done. Testing and approval of the test and code audit results formally complete the build at this level.

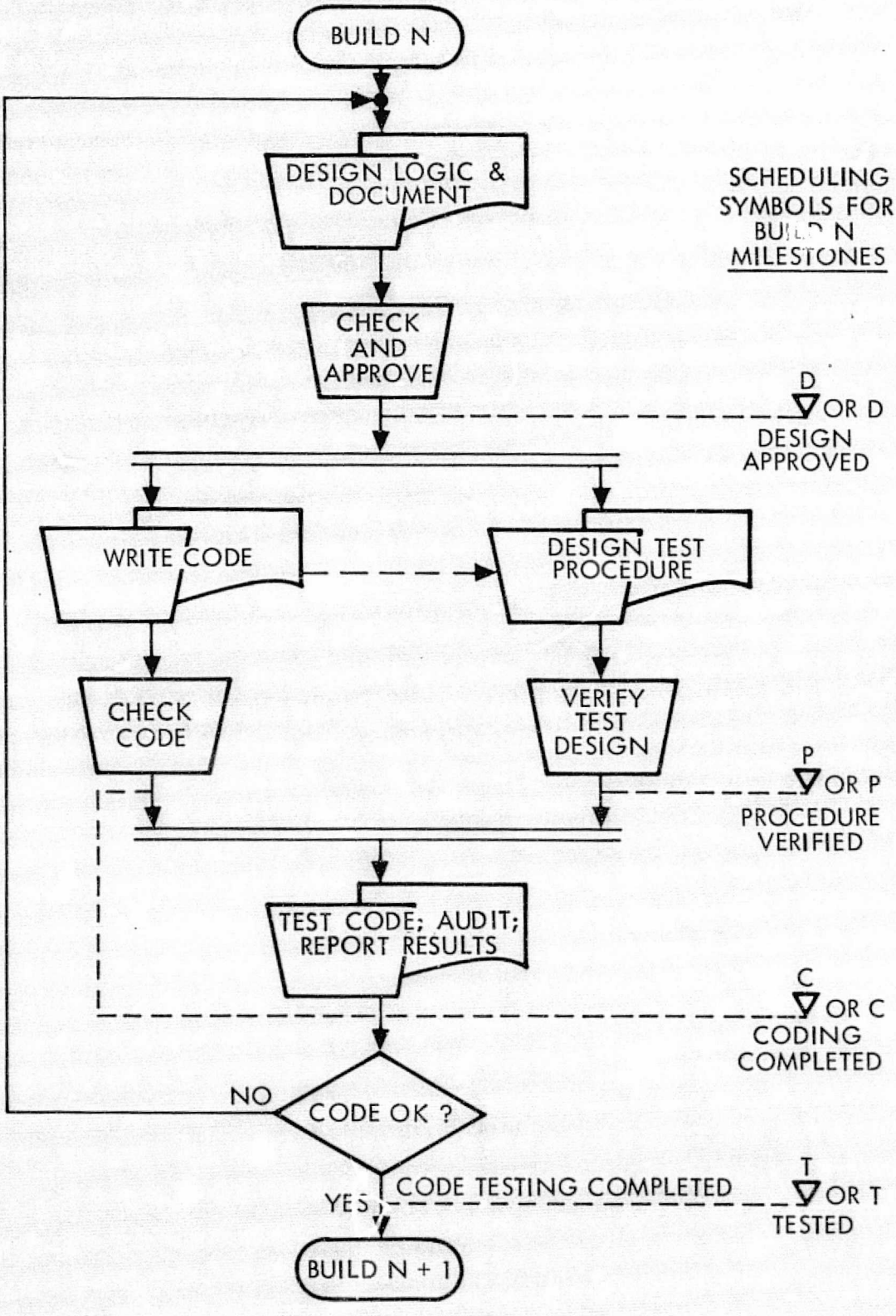


Figure 1-3. Top-Down Concurrent Construction Process for Build N

This process, then, in more detail, is first to design and document the expansion, making use of information from the previous builds to produce a description of any resulting unstriped modules adequate for translating directly into code. It is also required that a description of any resulting new striped modules be produced that, together with material from higher levels, will be adequate to subsequently design and test them. In this framework, the design does not differ from its documentation. The design and documentation must therefore be adequate to code and design correctness tests at the given level and to design the next level module, that is, to design (expand) all the striped modules that appear in a given level. Of equal importance, the design and documentation must be adequate for maintaining the program after it has been transferred to operations.

After an independent check and approval of the design, the unstriped modules are coded and the correctness test is designed, including any necessary test code or dummy stubs. In line with good engineering practice, the program code is checked and the test design is verified prior to conducting the actual test runs.

The code is then tested, and the results, including an audit of the code against the design, are provided to the CDE (or PE) for a decision to either proceed to the next level of implementation or to cycle back and reconsider the present level design for required changes and subsequent recoding and retesting activities.

Following the satisfactory completion of this testing, the cycle of design and documentation, code, test design, and test/audit is repeated for the next level. Each level is formally completed before formally proceeding to the next to allow the CDE to effectively manage the overall implementation. Informal "look-ahead" work on both design and test procedures in lower levels can proceed, even though the design for build N + 1 is not signed off until the code on the current build N is satisfactorily tested and audited: however, the amount of resources to be spent in this way is at the discretion of the CDE.

Progress within each level of the design can be tracked by the use of program build milestones, as shown in Figure 1-3 and described and listed in Table 1-1. At completion of the last build, the program is complete, correct, and ready for final acceptance testing.

Acceptance test milestones for DSN subsystem software are shown in Table 1-2. The planning and preparation for acceptance testing is performed in parallel with the program implementation, beginning with preparation of SRD inputs. Software implementation is complete at transfer (milestone X); the remaining milestones pertain, if applicable, to the total subsystem acceptance, in which the software is an active part.

B. POLICIES FOR IMPLEMENTING SOFTWARE

This section summarizes the software implementation policies that are applicable to operational programs. These policies form the baseline criteria for software implementation practices and result from a combination of (1) experience, (2) ongoing research efforts, and (3) the continual needs as stated in Section I.

1. Adherence to Software Standard Practices

Standard Practices governing software implementation and documentation are to be followed for all computer programs to be transferred to operations.

a. Comments

(1) Programs subject to this policy are those used in the day-to-day operation of the DSN, including management and administration.

(2) Anticipated exceptions, if any, to the application of the DSN Software Standard Practices will be identified during the planning phase and included in the SRD.

Table 1-1. Program Build Milestones

Symbol	Description
D	Design Approved The program architectural design definition presented in the Software Definition Document is detailed in terms of control logic in preparation for coding, and checked independently. Approval of the logic design authorizes the start of coding and test designing, which may, however, at the discretion of the CDE, have started before formal design approval.
P	Test Procedure Design Verified The correctness test procedure and necessary test code for testing the new build module against the approved design are designed, assembled, and verified. Test Procedure Design Verified indicates that the test procedure and associated test code are complete and available. Test design should formally start at the same time that module coding starts, that is, after design approval.
C	Code Completed The code for the approved design is generated and checked. Code Completed indicates that the code is complete, has been checked and is ready for correctness testing. Also, it is available for an independent audit against the design.
T	Tested The correctness test is executed. Results are evaluated and provided, along with code audit results, to the CDE for acceptance, which permits this cycle, starting with Design Approved, to formally begin at build N + 1 (this does not preclude informal look-ahead work on design and test procedures). Correctness testing is conducted only after the code is complete (C) and the test procedure design is verified (P).

Table 1-2. Acceptance Test Milestones for DSN Subsystem Software

Symbol	Description
V	Subsystem First Demonstration Test Completed The subsystem (hardware and its software) interfaces and operation may be demonstrated and validated, at the discretion of the CDE, with the subsystem in a tracking station (or control center) environment. Limited interface-type data flow tests can then be supported.
X	First-Station Test Accepted; Software Transferred The subsystem is tested, typically at the Compatibility Test Area, JPL, Pasadena (CTA 21), for acceptance and transfer to operations. For software for an existing computer, this milestone is the same as V, above. The results of the software acceptance tests and the program documentation are reviewed. Acceptance of the program, documentation, and test results signifies the end of the software implementation (project milestone 6, see Section III). The software products are transferred from the implementing organization (CDE) to the operations organization (software COE and the DSN Program Library).
Y	Subsystem Implementation Completed The subsystem hardware and/or software assemblies are installed, tested, and delivered to the Station Directors by the subsystem COE. Completion signifies that all subsystem requirements have been implemented and accepted.
Z	System Performance Test Completed System-level engineering tests are conducted by station personnel using standard procedures, station-by-station. All interfaces and system requirements are verified. Upon completion, the station can support project tests and begin specific mission-dependent test activities.

b. Examples

(1) A computer program is requested for doing a tradeoff study and making plots for procuring an item. The program will not be used after a decision is obtained. Standard Practices may or may not be used, at the discretion of the implementer.

(2) A general-use computer program is requested for doing monthly tallies of current costs and dollar obligations. The results will be used for setting future budgets and commitments. Implementation will be in accordance with Standard Practices, and the program will be transferred to operations.

2. Adherence to Referenced Practices

Referenced practices, standards, and requirements are considered to be part of the Standard Practices and are to be followed as an extension of Paragraph B.

a. Comments

(1) Acceptable methods or techniques already established and available are referenced rather than repeated or restated.

(2) As new techniques and practices become known and available, their adoption, if approved, is facilitated by this method.

b. Examples

(1) System Standard Practices apply for software implementation, and are referenced as needed.

(2) Also, certain appropriate practices and approaches as selected from key sources are identified and adopted in the Standard Practices. An example is the adoption of the flowcharting symbols, terminology, and usage, as prescribed in Reference 1,

as the standard. These are derived from wide and accepted usage (References 3 and 4), with extensions and refinements made to conform to the needs of structured programming and shown in Section IV of Chapter 3.

3. Use of Standard High-Level, Stable Programming Languages

Coding of the software design will be performed using standard high-level languages provided for real-time and nonreal-time programs. The design and implementation of these languages shall be under change control. Alternates to the standard languages can be used where justified by a life-cycle cost analysis, taking into account the initial implementation cost plus the cost of maintaining and sustaining the program over a 10-year life (or the known life span, if different).

a. Comments

(1) A stable, high-level standard language promotes communications among the various people involved with the software over its life-cycle and from project to project. This reduces the initial implementation and continuing maintenance and sustaining costs. High-quality implementation user documentation of the language is needed to maximize this benefit from language standardization.

(2) A standard language can provide a degree of isolation from computer and operating system changes and modifications. With adequate control, the language can buffer the users from the changes. Only the compiler need be changed; therefore, each user is not required to compensate for the external changes. This further reduces sustaining and change engineering costs.

(3) For nonreal-time DSN computer programs, the MBASIC language has been designated as the standard language. Use of the MBASIC language results in relatively short programs of simple structure which are relatively easy to write and understand, and, more importantly, are independent of machine hardware and operating

system. Presently only an MBASIC interpreter exists, so that for programs which consume a great deal of time in an interpretive version, life-cycle cost criteria may indicate that some other language be used until the MBASIC compiler is implemented.

(4) Future revisions to the Standard Practices will incorporate language standards for real-time software and data base manipulation. Presently, the situation for real-time programs, where operational speed, core utilization, or input/output efficiency are key parameters, forces the decision to use machine or assembly language, or a nonstandard medium-level language such as FORTRAN.

4. Top-Down Concurrent Implementation

Software will be implemented in a top-down manner to allow the design and documentation as well as coding of the program to progress concurrently with the generation of correctness test procedures and the correctness testing itself.

a. Comments

(1) A running version of the skeleton program is available early in the project by the implementation of the external interfaces and the use of internal dummy stubs.

(2) Also, the necessary program documentation tends to be generated more as a natural consequence of the process rather than by treating documentation as an extra required task that interrupts the implementation process.

b. Example

Suppose that the implementation of a new processing system, consisting of many complex subprograms, is initiated and an individual is assigned responsibility for one of the subprograms. An early running version of the subprogram can be made available to determine the fit into the overall implementation. Any refinements needed can be

iterated and fed back long before the detailed and relatively nonchangeable part of the subprogram design is completed. This is done by providing dummy stubs for the striped modules in the subprogram and providing the subprogram to the main program, with the other subprograms being dummy stubs. Also, to maintain a coordinated effort, the individual subprogram status and any changes made throughout the system are documented and communicated as the implementation progresses.

5. Use of Structured Programming Principles

Only prescribed programming structures will be used for flowcharting the design and translating the design into computer code.

a. Comments

(1) The prescribed structures allow the normal flow to progress through the program from top to bottom, avoiding the need for the kind of GOTO statements which cause erratic flow. Also, their single-point entry and exit characteristic avoids sneak paths (and their associated surprises) within the design.

(2) An exception to the single-exit characteristic for non-real-time programs is the abnormal termination, of which there are two types. One type terminates the execution of the program in the event of a predetermined condition without regard to program structuring. The other type also terminates execution without regard to program structuring but returns the operation to a specified point (or points) earlier in the program. Either type may be incorporated at any point in a program, if appropriate. This is treated in more detail in Reference 1.

(3) Structuring of charts allows easy identification and tie-in of required explanatory narrative and supporting material.

(4) The theory of Structured Programming is not yet completely developed for real-time programs, but it is expected that few if any exceptions will be necessary. Subsequent revisions of this document will incorporate advances in the methodology as developed and demonstrated through practical experience and research.

b. Example

(1) Figure 1-1 is an illustration of a set of prescribed structures.
(2) Figure 1-2 is an illustration of the use of prescribed structures in flowcharting.

6. Modular Implementation

By applying Policies 4 and 5, modules will be implemented in hierarchical subordinating levels of detail. Each module will be limited in length and complexity to a single page-size (8-1/2 x 11-inch) flowchart by using the Striped Module technique of hierarchical expansion.

a. Comments

(1) This allows a manageable implementation to be planned, along with the identification and division of work responsibilities and actual work assignments.
(2) User maintenance is facilitated by easily understood modularized programs.
(3) It may be that some striped modules, when expanded, require several more levels to conform to the readability requirement of fitting on a single page. If the total complexity of the expansion is still low enough to be readily comprehended in one individual's mind, the CDE may decide to save resources at low risk by consolidating these builds for the purpose of design, code, and correctness test approval (retaining the separate single-page flowcharts). One exception so far identified where

this appears reasonable is where the expansion consists merely of a long string of sequential functional operations with no looping or control decisions.

(4) Use of the permitted multiple-decision structure may require an exception to the 8-1/2 x 11-inch page size, where many branches are involved. Use of sequentially applied statements with fewer output branches would hinder understanding in such instances.

(5) Builds can take place concurrently; i. e., different individuals or subteams can construct their own builds and test their part of the program, supplying stubs for the parts they are not responsible for implementing. This facilitates division of labor.

7. Implementation Team

For sufficiently large projects, the Cognizant Development Engineer will establish and direct an implementation team that is supported by the following main functions:

(a) Programming Secretariat
(b) Quality Assurance
(c) Design, coding, test design, and testing specialties, as needed

Comments

(1) For a typical software project:

(a) The CDE oversees the implementation process.
(b) The Programming Secretariat function maintains the project's records and files, and serves as the center of team communications under the direction of the CDE.
(c) Specialist support is functionally independent and basically performs module-by-module development under the direction of the CDE.

(2) For small projects (6 person-months or less) these functions, except for QA, might all be assumed by the CDE.

8. Standard Project Milestones

Standard project milestones will be used for technical and management planning and control of the software implementation.

Comments

(1) Established project milestones serve to aid in the overall planning, directing, and reporting of all ongoing work throughout a project.
(2) Internal program build milestones are based on work completion on a build-by-build basis.
(3) The completion of various builds, as signalled by a build passing the correctness test and QA requirements for it, properly serves as a milestone for an optional and perhaps informal design review. The decision as to which combination of builds on which branches shall be used for these milestones occurs early in the design definition phase, and priorities should therefore be specified in the SDD.
(4) Guidelines, and special tools and aids for performing, managing, and the orderly scheduling of the iterative parallel thought process involved in early program definition are currently under development.
(5) Anticipated exceptions, if any, to the key activities defined in Paragraph B of Section III will be identified during the planning phase and incorporated into the SRD.

9. Project Scheduling

Schedules for meeting the project milestones will be established and periodically assessed by the Cognizant Development Engineer and controlled by the cognizant manager.

Comments

(1) Schedules should reflect the requested delivery date, the detailed work involved, and the resources available based on allocated priorities.

(2) Internal scheduling of the Program Build Milestones (Table 1-1) provides a day-to-day work plan that the CDE can use to assess the implementation progress. These internal schedules are based on both completion of builds in the various subprograms, and also on completion of the various steps required for approval of a given build, indicated in Figure 1-3.

(3) Scheduling of iterative, parallel thought processes such as problem analyses and solution evaluations that occur early in, or even prior to, the implementation are not, at present, formalized; however, a time estimate should be allocated by the CDE. Ideally, most of the iterative process should occur in advance of the actual implementation.

10. Project Reviews

During the planning phase, dates will be set for all formal design reviews; the Cognizant Development Engineer will coordinate and conduct the reviews.

Comments

(1) The reviews will be in general accord with the intent of Paragraph C of Section III. Additional information on review content and conduct is provided in the Standard Practices covering the particular phase of implementation. Optional reviews should be conducted at the completion of key levels in various subprograms as determined by the CDE, who typically conducts the reviews.

(2) Cognizant management can serve as the CDE alternate for conducting design reviews.

(3) The formal reviews provide the means for independent outside checks of the overall implementation process and progress.

(4) If a design review is conducted prior to SRD approval, the person preparing the SRD, or his management, conducts the review.

(5) All software reviews, including the required formal reviews, should be low-cost, use existing documentation when available, and be combined with subsystem reviews, when practicable.

11. Project Documentation

Documentation will be compiled concurrently with the implementation progress and will be available on a continuous basis for use and review.

Comments

(1) Narrative will accompany each chart (program flowchart, data structure chart, etc.) and will be produced concurrently with the chart development.
(2) Document content and format, as applicable, will be in accordance with specific guidelines contained in the appropriate Standard Practice.
(3) Anticipated exceptions or substitutions to the key documents defined in Paragraph A of Section III will be identified during the planning phase and included in the SRD.
(4) During the implementation, the change control process is project-internal; changes will be controlled by the CDE or the cognizant manager, prior to SSD approval, but changes require concurrence of the COE and CSE during the acceptance testing phase.
(5) Following transfer to operations, changes to documents and the program will be controlled by the Change Control Board.

12. Quality Assurance

Quality assurance will provide an independent check on software quality.

Comments

(1) Quality assurance activities are to be an integral part of the implementation process.
(2) Audits are conducted by independent (not project-controlled) auditors, whose activities will be directed toward satisfying the general intent of Paragraph E of Section III.

(3) Specifically, QA will audit each code module prior to certifying its status and will certify, prior to program transfer to operations, the reported code documentation status.

SECTION III
SOFTWARE IMPLEMENTATION PROCESS AND GUIDELINES

A. IMPLEMENTATION PHASES

The software implementation process extends from an initial program justification activity through program delivery and involves a chronological progression of activities which can be grouped or identified as specific phases of effort. The phases of software implementation involve (1) planning and specifying requirements, (2) design definition, (3) design and production, (4) acceptance, and (5) operations and maintenance. A convenient measure of the project's progress is available by noting the completion of the phases; therefore major milestones are defined mainly at the end points of the implementation phases. Figure 1-4 presents an overview of a typical software implementation sequence, and Figure 1-5 summarizes the DSN software management and implementation plan, as derived from Figure 1-4. Table 1-3 outlines the support responsibilities involved throughout the implementation. These are briefly discussed in the following paragraphs.

1. Planning and Requirements

a. Initiation. Initially, the nature of a given problem, and the need for and benefits of a possible software solution are determined. For DSN subsystem software, the need is identified by the Subsystem Functional Design Document, which assigns the functional requirements on the subsystem to either hardware or software. The functional requirements and performance parameters are listed in the Subsystem Functional Requirements Document. Further planning and analysis are then accomplished to identify the operational environment (computer and equipment), overall project schedules, and total costs; also included are resource commitments and more refined cost estimates for the next phase of activities, accurate to within a ten-percent goal. This information is contained in the Software Requirements Document (see Chapter 2 for a typical content outline).

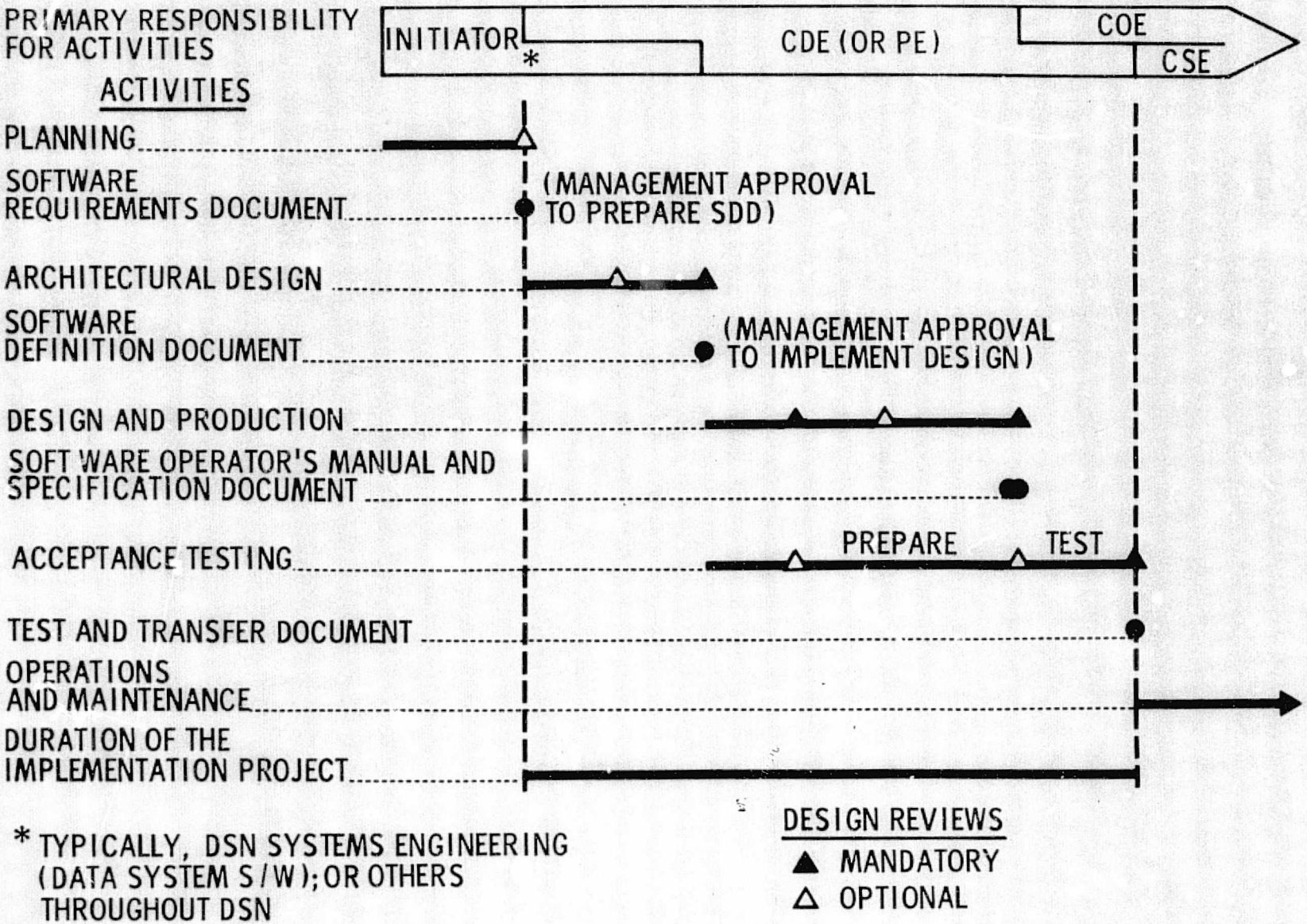


Figure 1-4. Sequence of Activities in a Typical Software Implementation Project

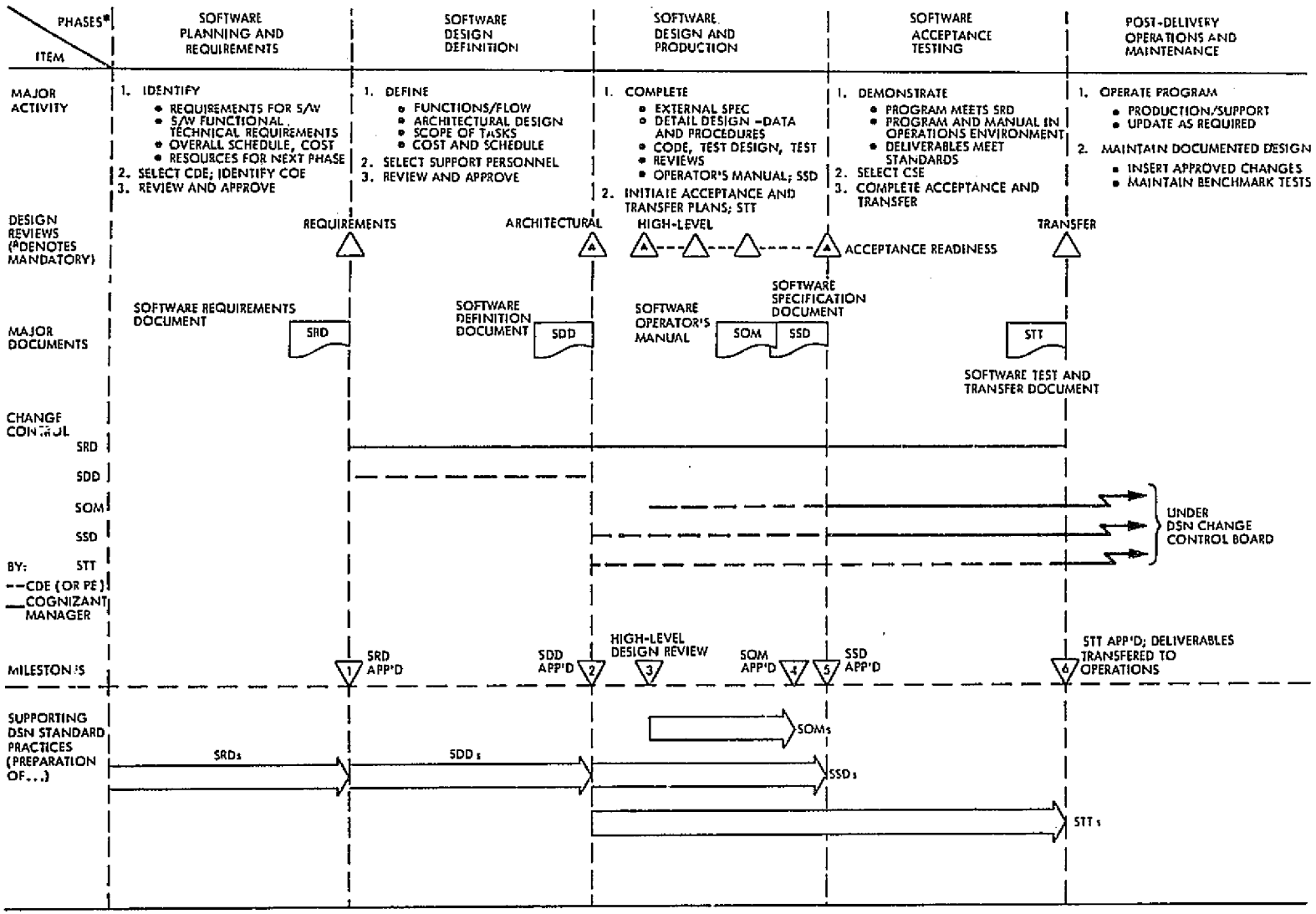


Figure 1-5. DSN Software Management and Implementation Plan

Table 1-3. DSN Software Implementation Process Support Responsibilities

Support Activity	Implementation Phase**				Operations & Maintenance
	Planning & Requirements	Design Definition	Design & Production	Acceptance Test and Transfer	
Coordination of Activities	Subsystem Engineer (SSE) or Initiator	Cognizant Development Engineer (CDE)*	CDE*	SSE and CDE*	COE
Design	N/A	CDE	CDE	N/A	Cognizant Sustaining Engineer (CSE)
Acceptance Test Input and Support	Cognizant Operations Engineer (COE) (if assigned)	COE	COE	COE	COE
Documentation of Results	SSE or Initiator	CDE	CDE	CDE*	CSE (for sustaining changes)
Plan and Conduct Review	SSE or Initiator	CDE*	CDE*	CDE*	SSE, if applicable
Approval	Cognizant Management (CM)	CM (Implementer)	CSE, COE	COE, Quality Assurance (QA) and CM	QA/CM
Milestones	Software Requirements Document (SRD)	Software Definition Document (SDD)	Software Specification Document (SSD) Software Operator's Manual (SOM)	Software Test and Transfer Document (STT)	Engineering Change • Requests (ECRs) • Orders (ECOs)

*Or Project Engineer, for large implementations.
**For DSN subsystem software, subsystem functional design precedes the formal software implementation.

b. SRD Review and Approval. The SRD forms the basis for program justification; management approval of the SRD authorizes the program Design Definition Phase to begin. The SRD identifies the overall requirements for the program (functional, management, and technical, as needed), the overall implementation schedule, an initial estimate of the resources needed for the entire implementation, and an accurate (ten-percent goal) estimate of the resources needed, for the Design Definition Phase only. It denotes detailed requirements of the program only as necessary to understand the functional requirements of a program. In cases where substantial additional technical material is needed, this should be contained in another document or documents to avoid diluting the primary purpose of the SRD, and should be referenced or appended.

SRD approval signifies authorization to proceed to the next phase and is based on a review of the planning information. If a formal review is warranted for probing the reasoning that went into the SRD, it is performed by the design review process. As such, the review process is one of critiquing the planning activity. The review is not limited to a document review, and it is not necessarily one of concurrence. Furthermore, these software design reviews can be combined with subsystem reviews to conserve resources.

After approval, changes within the original scope of the SRD are under internal change control and are approved by the CDE. Other changes require the same signatures as the original. Following transfer to operations, the SRD is kept on file in the Program Library, but does not survive the project as a formal document and is not maintained with updates beyond program transfer.

c. Responsibilities. As shown in Table 1-3, the coordination of all planning activity, the documentation of results, and the conducting of the review leading to SRD approval are the responsibility of the Subsystem Engineer for the applicable subsystem, or the initiator for other software. Acceptance test input during the planning is provided by the Cognizant Operations Engineer, who is assigned by the Operations Organization or the User Organization.

Approval of the SRD is made by the initiator's cognizant management and others involved such as the Funding, Systems Engineering, Implementing, and Operations organizations. The management approval level required is determined by the regular procurement management structure based on the estimated total program cost, but does not go above the Assistant Laboratory Director level. Prior to approval, cognizant management selects a CDE to assume responsibility for the subsequent implementation of the program.

2. Design Definition

a. Architectural Design. The requirements contained in the SRD are translated into a program definition or architectural design which forms the design base at the highest levels and interfaces of the program. The architectural design summarizes the complete program, placing emphasis on the highest levels of the program (e.g., high-level striped module identification and connectivity, major program modes, data structures, and information flow). This identifies and scopes the remaining work and can be used for refining the program completion cost estimate and schedule (to within ten percent, as a goal), and for work planning and coordination activities. This architectural design is contained in the Software Definition Document (see Chapter 3 for a content outline). Along with this program definition information, the SDD includes some generalized and/or high-level module correctness and product acceptance test design goals.

b. SDD Review and Approval. A review of the proposed architectural design will be held at the start of the Design and Production phase. The review provides an overall assessment of the project costs, resources needed, schedules, functional design, and interfaces. Also reviewed are relevant hardware, corresponding functional requirements, and any other background information or other related documents needed for assuring the successful implementation of the architectural design. The detailed design and program production formally begin after SDD approval. The SDD is not updated, since its purpose — providing the baseline architectural design — has been achieved.

The SRD however, is revised and updated to maintain an accurate record of any changes in requirements that may occur during the implementation phase. Changes to the SRD require the same approvals as the original document. Implementation changes that do not affect requirements remain under implementation project control and require only CDE approval. Upon transfer to operations, the SDD and the SRD are delivered to, and retained by, the DSN Program Library. The documents remain "as is" and are not revised to reflect any future program changes.

c. Responsibilities. The CDE (or PE on large implementations) is responsible for the coordination of the design definition activities. Responsibility for the planning and conduct of the SDD design review also lies with the CDE, although assumable by the cognizant manager of the implementing organization.

Acceptance test information in the SRD is further detailed by the COE and provided to the CDE, who is responsible for the documentation of the program and test information in the SDD.

The SDD is approved by the implementing organization line management prior to proceeding to the detailed design and production activities. Other approvals that may be required for certain projects will be identified in the SRD.

3. Design and Production

a. Top-Down, Concurrent Implementation. The architectural design as presented in the SDD is expanded using the top-down methods and concurrently coded, tested, documented, and approved in successive stages. Expansions include both program procedure and specific data representations.

As the program construction progresses, the information from the as-built configuration is compiled; it comprises the main part of the Software Specification Document (see Chapter 4 for a content outline), which is used mainly for program maintenance after transfer to operations. Operating instructions

and procedures as verified from the unit correctness testing are compiled to build the Software Operator's Manual (see Chapter 5 for a content outline). Planning for the acceptance and transfer to operations of the completed program is initiated early in this phase.

b. Design Reviews. Two major design reviews are typically held during this phase for all but the smallest implementations. A program high-level design review is held early in the design and production phase to identify and correct any problems in the technical, resource, or schedule areas. An Acceptance Readiness Review of the program and its documentation is held after the program construction is completed. This review provides approval of the SOM and SSD, concurrence with the preliminary STT, and authorization to begin formal acceptance testing. Informal status reviews and working sessions as well as additional formal design reviews may be held throughout the Design and Production Phase at the discretion of the CDE and prior to the mandatory Acceptance Readiness Review.

c. SSD and SOM Documents. The SSD contains the complete, as-built detailed design and code listings that describe the program as delivered. The SSD is approved by the CSE, which authorizes its use in acceptance testing. Final approval and acceptance of the SSD is obtained during the mandatory Acceptance Readiness Review. The approved SSD is delivered along with the program for acceptance testing and transfer to operations. The SSD remains with the program throughout its operational life, serving as the technical design shop manual for program maintenance and any subsequent program changes. Engineering change requests and change orders are written against the SSD, SOM, and STT, as required. The basic criterion for the SSD is that it be usable for program maintenance without assistance from the CDE.

The SOM contains the operating instructions necessary for effective program use in the operational environment by an operator not necessarily familiar or associated with the program design or implementation. On large implementations, the SOM is written in parallel with the program design and production, and is adequate for executing the present program build. As program (dummy) stubs are replaced by active module code, the appropriate

operating information is concurrently incorporated into the SOM. The completed SOM is approved by the COE and delivered with the computer program for acceptance testing and transfer to operations.

d. Responsibilities. The overall responsibilities as assigned in the program definition phase continue, but the specific duties involved vary in relation to the specific program. For large efforts, additional support personnel may be phased into the work plan to form an implementation team; team structure and interaction must be effective to handle the needed coordination and communication activity throughout the implementation. Team structuring for large implementation efforts is discussed in more detail in Reference 1. The Cognizant Sustaining Engineer is assigned prior to the Acceptance Readiness Review.

4. Acceptance Testing and Transfer to Operations

a. Software Test and Transfer Document (STT). The STT identifies all necessary acceptance tests and conditions for formal transfer of the software from implementation to operations. It is prepared in parallel with the program implementation (see Chapter 6 for a content outline). Following acceptance testing, the STT is approved by the COE and cognizant line management and delivered with the computer program; the acceptance tests serve as benchmark* tests for program maintenance during the operational life of the program.

b. Acceptance Testing. The program is tested in accordance with the STT, as concurred with at the Acceptance Readiness Review. The final acceptance tests are directed mainly toward validating the full-up, end-to-end throughput characteristics of the program and verifying the program documentation. The program internal structure has already undergone extensive testing during design and production, and its correctness has been established. Acceptance test milestones for software are described in Table 1-2. If the program or documentation require changes during acceptance testing, the CDE, COE,

*Verification of original capability, as transferred, using the original tests as the control reference.

and CSE must mutually agree to the change prior to approval. During testing, the STT is revised as required to incorporate the as-tested configuration and conditions.

c. Transfer Review. An optional status review may be held prior to transfer to operations to ensure that all conditions contained in the Transfer Agreement have been met. The transfer process is formally completed when the Transfer Agreement is signed by the CDE, QA, COE, user organization manager, and cognizant implementation manager; this completes the implementation.

d. Transfer to Operations. Upon completion of the Transfer Agreement, the computer program and its documentation are formally transferred to operations and delivered to the Program Library. In addition, the project provides all the project notebooks and other informal documentation, including correctness testing and its results, to the COE for use as troubleshooting aids and for maintenance.

e. Responsibilities. Generally, the responsibilities as assigned in the design and production phases continue, with emphasis shifting to acceptance testing and final approval prior to transferring the program to operations.

Acceptance test criteria are coordinated by the program initiator, in conjunction with the COE. Procedures are then planned by the CDE, in accordance with the test criteria, and the procedures are documented and executed by the CDE. The CSE will typically be available to assist the CDE. Results are evaluated by the initiators, users and/or COE, and Quality Assurance and documented by the CDE.

In addition to supporting the test evaluation, the QA function is responsible throughout the project for code auditing, just as it is responsible for hardware inspection in a hardware project. The module-by-module auditing is augmented at the time of transfer to operations by a final quality certification of the code, SSD, SOM, and STT.

5. Operations and Maintenance

Following transfer of the program, the project team is disbanded. Responsibilities are then shared between the COE and CSE.

The COE is responsible for the initial integration of the program into operations and the subsequent installation of approved modifications. As such, the COE provides training and troubleshooting support as needed. Also, the COE serves as the single point of control to review and recommend changes to the program, as required. The COE's responsibility also includes efficient data base utilization and coordination of data base maintenance, when relevant.

The CSE is responsible for providing program sustaining engineering necessary to maintain the operational integrity and support capability of the program. When a new capability for an operational program is identified (by ECR) and approved (by ECO), the CSE assists in the definition, develops a modification, and supports its installation, if needed.

The software change process after transfer to operations, which also applies to hardware, is accomplished through the Change Control Board. Therefore, changes require CCB review and approval before implementation. Unauthorized changes to transferred software are prohibited. The CSE, in concert with the COE, implements all authorized ECOs. Changes for cosmetic or explanatory reasons may possibly be exempted from the change procedures if no executable elements are involved, but only upon the review and recommendation of the CCB.

For extensive new capabilities, an SRD may be needed, as determined by the CCB. The SRD is prepared by the requester with support from the COE and CSE. SRD approval authorizes the development and test of the modification. Following the design and implementation, a formal transfer of the program modification and updated documentation completes the effort.

B. MILESTONES

Based on the activities discussed in Paragraph A of this section, six major milestones are established which allow the overall software implementation project to be planned and its in-progress development to be monitored. These are shown in perspective to the total activity involved in the Management and Implementation Plan of Figure 1-5. The milestones in order of occurrence are as follows:

(1) Software Requirements Document Approved
(2) Software Definition Document Approved
(3) High-Level Design Review Completed
(4) Software Operator's Manual Approved
(5) Software Specification Document Approved
(6) Software Test and Transfer Document Approved; Transfer Agreement Signed; Deliverables Transferred to Operations

C. REVIEWS

Working sessions of the implementation project team and informal status reviews are, of course, held throughout the software project, as called for by the CDE or by policies of the implementing organization.

At designated periods throughout the implementation (see Figure 1-5), formal design reviews are held involving both the project team and others not directly involved in the detailed day-to-day implementation. Specific details of these reviews vary, depending on the applicable phase of implementation covered and on the size of the implementation. These details, along with suggested guidelines, are presented in some depth in the following Chapters. Mandatory reviews have been indicated previously in Paragraphs A and B of this section.

Basically, the purpose of each formal review is to assure that the implementation to the present point is acceptable in terms of (1) meeting costs and

schedule, (2) technical requirements, and (3) readiness to proceed to the next phase of implementation. The last two phases of implementation are acceptance testing and post-transfer operations and maintenance.

Depending on the results of the review, either approval to proceed is given or necessary adjustments are identified and corrective actions taken before proceeding.

D. DOCUMENTATION

Documentation is for communicating and storing information. To be effective, it must be timely, clear and understandable, applicable (the right information in enough detail), and reasonable in cost. The documentation process is highly dependent on the activity involved. For this reason, specific detail and guidelines for the various software implementation documents are presented in Chapters 2 through 6; each chapter covers a major implementation phase of activity. However, certain general information and guidelines apply to the total process and are discussed below.

1. General Document Information

Both the SRD and SDD are internal project documents containing information used throughout the implementation. They should be informal, low-cost working documents, clear and complete for their purposes. The SRD will normally not be maintained beyond program delivery and the SDD not beyond its approval. Program functional and technical detail developed during the implementation can be retained by subsequent attachment (referenced or appended) to the SSD, or rewritten into the SSD.

The SSD, SOM, and STT are separate documents delivered along with the completed program and maintained throughout the operational life of the program. These are compiled concurrently with the program and test implementation, and therefore are timely and closely represent the actual state of development at any instant. As survivable and useful documents throughout the program life, some degree of formality is in order, but the prime emphasis

must be on information content and completeness. The costs of these documents might typically be comparable and in line with those of their counterpart documents associated with deliverable hardware, if there is hardware delivered along with the software.

2. The Program Library

In addition to receiving, storing, and distributing software deliverable items, the Program Library controls the program numbering system and assigns identifying numbers to new programs.

Following is a typical number assigned by the Program Library to a computer program:

DOI-5123-OP-R

The letter prefix (DOI) identifies the program as a DSN program.

The four-digit number is assigned sequentially.

The two-letter suffix denotes the program use; OP for operations program (used in a tracking station while a spacecraft is visible), SP for support program, TP for test program, etc.

The final letter is the program revision letter, which progresses through the alphabet as the program is revised. A first release has no revision letter.

The kinds of documents that can be required for a program are:

Functional Requirements Document	Sometimes, where applicable
Technical Requirements Document	
Software Requirements Document	
Software Definition Document	Always
Software Specification Document	
Software Operator's Manual	
Software Test and Transfer Document	

The document number is formed by attaching the document abbreviation (SRD, SDD, SSD, etc.) ahead of the assigned program number. For example, a Software Requirements Document would be:

SRD-DOI-5123-OP

An initiator of a program requirement obtains a program number from the Prcgram Library. The family of documents generated in support of the program could be as large or as small as necessary, according to the complexity of the program.

3. General Documentation Guidelines

The following general guidelines apply:

(a) DSN abbreviations and designators, and flowchart symbols, terminology, and usage, adopted from Reference 1, will be used throughout.
(b) Top-down hierarchical flowcharting and module identification (Reference 1) will be used, along with text and narrative keyed to their corresponding modules.
(c) All information should generally progress from a top-level understandable base through increasing levels of detail in small, understandable increments that allow traceability from any level of detail back to its top-level base point.
(d) The outlines in each chapter guide the preparer through this process.
(e) All pages should be the standard size of 8-1/2 by 11 inches (including figures, flowcharts, listings, etc.) to facilitate handling and storage and, more importantly, to limit the complexity of a single page.
(f) Typed manuscripts should be formatted in accordance with their respective Standard Practices.
(g) The SSD, SOM, and STT should be delivered to the Program Library in reproducible form (black on white preferred).

E. QUALITY ASSURANCE

Quality assurance efforts are always prudent in all phases of the software implementation process. They are intended to help reduce testing and maintenance costs by improving the clarity, accuracy, and consistency of the finished software product. Independent auditing of the code is therefore required before a given module can be certified as having met these goals.

The code audit includes an auditing of conformance to Standard Practices, to design, and to project-unique guidelines and conventions. The code audit may be performed independently of the module correctness testing; the audit results should be timely and formally reported to the CDE. An audit prior to correctness testing may be preferred in order to minimize the possible need for retesting based on the results of the audit. Classical methods, as well as new techniques and tools shown to be effective, will be used by QA.

Before completion of the transfer of the software from implementation to operations, QA shall also certify the status of the SSD (including the code), the SOM, and the STT.

SECTION IV
DSN SOFTWARE IMPLEMENTATION FUNCTIONAL TASKS

A. GENERAL

The software implementation process is accomplished through the systematic performance of basic functional tasks, in combinations suitable for meeting the needs of a specific project.

These functions are administered, as appropriate, by the Cognizant Development Engineering function (see Figure 1-6 and Paragraph C). A Programming Secretariat assists the CDE, maintains the project files, documents, and working papers, and serves as a central source and deposit of project data. As such, the project communication and task operational interaction tend to be to, from, and through the Programming Secretariat (see Figure 1-7). Direction of the effort is provided by the Initiator/User Representative and the cognizant management through the CDE.

B. INITIATOR/USER REPRESENTATION (Systems Engineering, Cognizant Operations Engineering, Other)

The Initiator/User Representation function supports the Cognizant Development Engineering function by supporting the initiation of a software project, monitoring the implementation and acceptance, and evaluating the deliverable software items. Specific responsibilities include:

(1) Preparing or supporting the preparation of Software Requirements Documents and conducting the SRD review.
(2) Participating in all design reviews.
(3) Coordinating acceptance test and evaluation activities.
(4) Participating in Transfer Agreement activities.

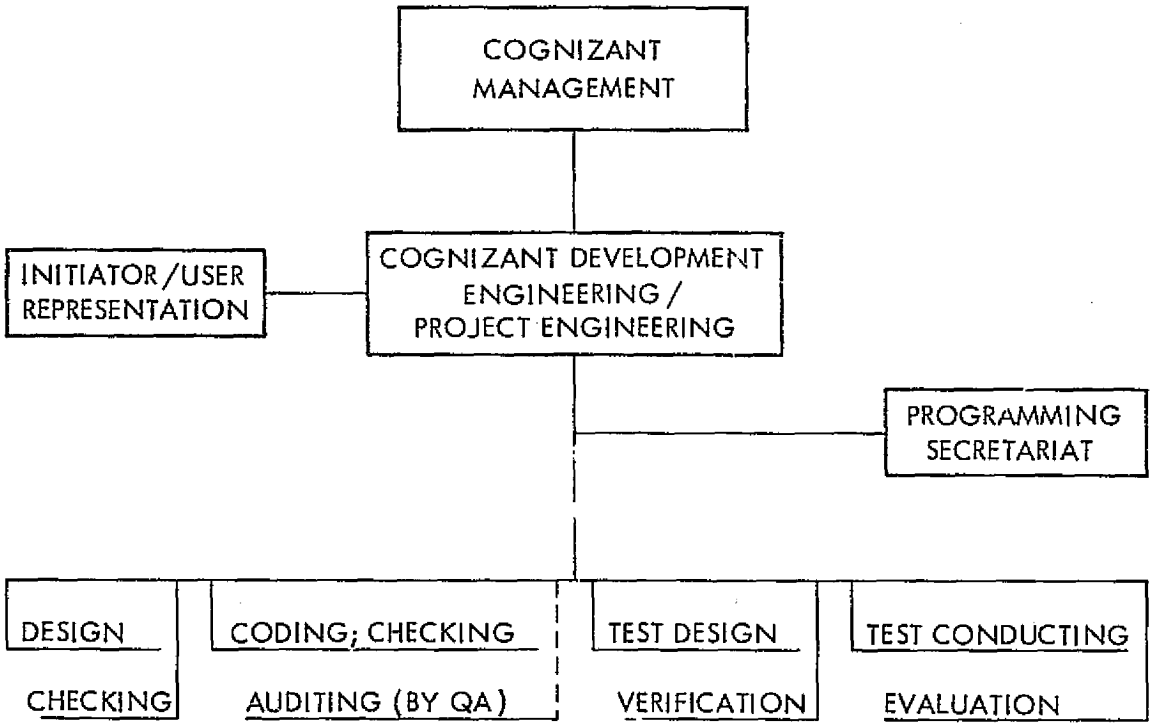


Figure 1-6. Software Implementation Functional Tasks

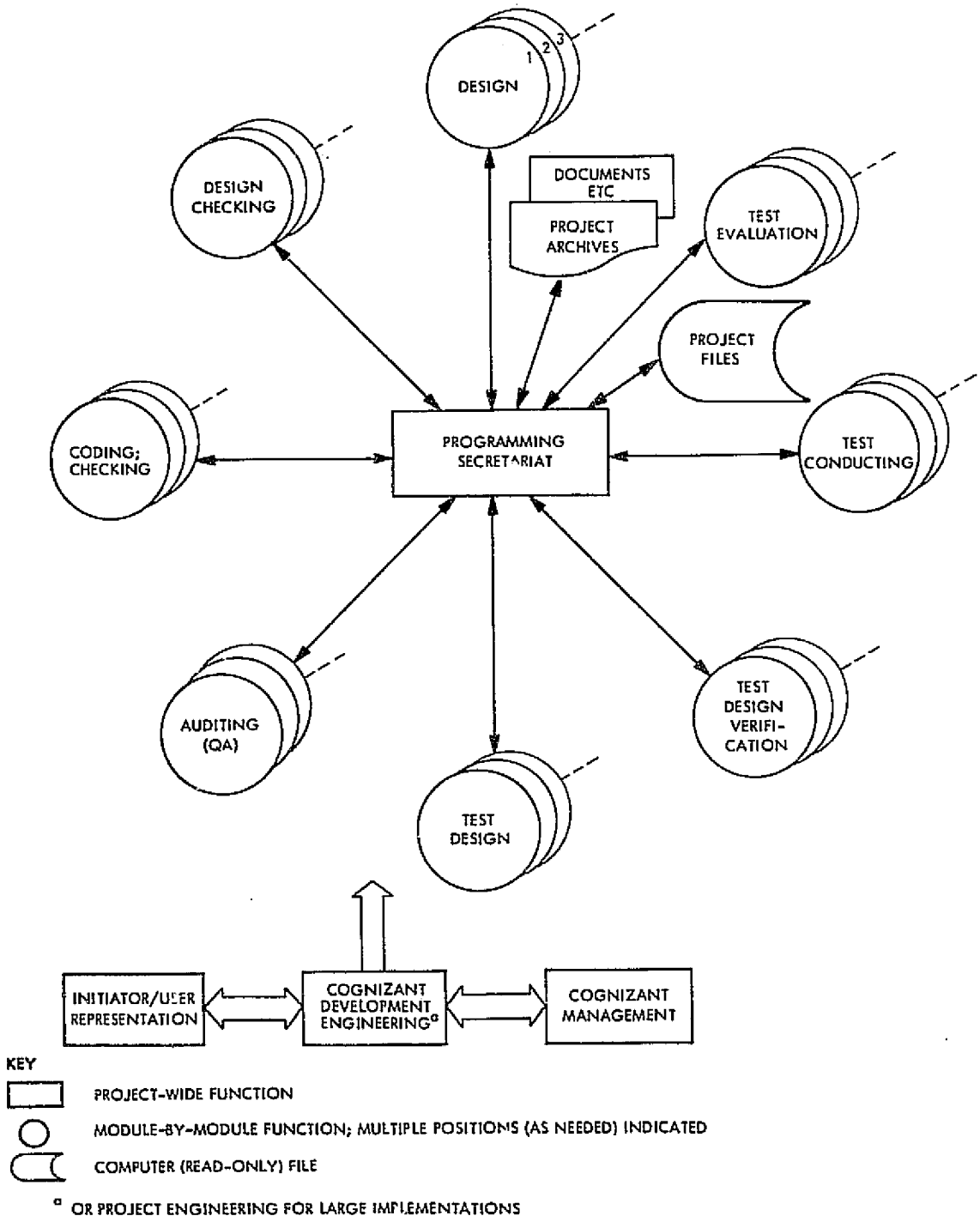


Figure 1-7. Software Implementation Tasks and Operational Interactions

C. COGNIZANT DEVELOPMENT ENGINEERING OR PROJECT ENGINEERING*

Cognizant Development Engineering (which is Project Engineering on very large implementations) within an implementing organization has the overall responsibility for the software implementation, reports to the cognizant manager, and is specifically responsible for:

(1) Evaluating Initiator/User input and requests during the program planning phase.
(2) In some cases, providing assistance to the Initiator in the preparation of the SRD.
(3) Generating the SDD in response to the SRD.
(4) Overseeing the program implementation process through program delivery, including design and documentation, coding, testing, conduct of design reviews, and preparation of the SSD, SOM, and STT.
(5) Leading and coordinating the efforts of the following support functions:

   (a) Programming Secretariat
   (b) Design; design checking
   (c) Coding; checking
   (d) Test design; test design verification
   (e) Test conducting; test evaluation

(6) Negotiating with the QA Organization for code auditing and other QA services needed during the program implementation.
(7) Generating an implementation schedule extending from SRD approval through Transfer Agreement.
(8) Continuously assessing implementation progress, and, as necessary, arranging for the reallocation of resources to meet schedules and commitments, with the concurrence of the cognizant manager if a budgeting change is required.

*Typically, there is always an individual assigned to serve as the CDE; however, on very large implementations, a Project Engineer is assigned for the administration and coordination of the effort of many CDEs. The PE's role is similar to that of the CDE, except at a higher level.

(9) Overseeing the conduct of the acceptance test and Transfer Agreement.

D. PROGRAMMING SECRETARIAT

The Programming Secretariat supports Cognizant Development Engineering in a wide range of activities during the software implementation. This support involves administrative tasks such as filing and distributing design charts, module and test codes, test results, and change requests. The Programming Secretariat can also be responsible for certain implementation tasks such as assignment of program labels and names for each striped module, maintaining a current list of variables or program glossary, recording progress against the project schedules, and accumulating the as-built SSD. Because of this heavy involvement in project communication and data handling, the Programming Secretariat is a key function during the implementation.

The Programming Secretariat reports to Cognizant Development Engineering and is specifically responsible for:

(1) Accumulating project-internal documents.
(2) Filing code in a program file, which is read-only for all others.
(3) Filing and distributing test procedures and associated test code.
(4) Accumulating test results and reports and certifying completion of modules to the project members after approval of the test results.
(5) Maintaining the program variable list or program glossary, which includes all data structure names and alphanumeric program labels.
(6) Assigning and keeping track of program labels and names in consultation with coders.
(7) Administering the project-internal change control process.
(8) Accumulating and disseminating project status information.
(9) Assembling the SSD package, including design flowcharts and narrative, code listings, files, and tapes, as modules are completed.
(10) Performing other duties as assigned by Cognizant Development Engineering according to the policies of the implementing organization.

E. DESIGN

The Design function supports Cognizant Development Engineering and is responsible for:

(1) Performing detailed design of modules, and documenting (concurrently) with flowcharts, detailed supporting narrative, and tables, especially data structure definitions and resource access relationships.
(2) Supporting the overall program architectural design and providing inputs to the SDD.
(3) Providing program design status to the Programming Secretariat.
(4) Supporting the Programming Secretariat in assembling the SSD.
(5) Performing other duties as assigned.

F. DESIGN CHECKING

The Design Checking function supports Cognizant Development Engineering and is responsible for:

(1) Providing an independent design check of the flowchart and narrative for correctness, clarity, adherence to established practices, and for generally good engineering quality.
(2) Checking that the build N design is adequate for build N coding and test design, for build N+1 design, and for program maintenance after transfer to operations.
(3) Providing the status of the design check to the Programming Secretariat.
(4) Performing other duties as assigned.

G. CODING; CHECKING

The Coding; Checking function supports Cognizant Development Engineering and is responsible for:

(1) Generating module code for the build N design.
(2) Providing checked (running) code to the Programming Secretariat for subsequent correctness testing.
(3) Performing other duties as assigned.

H. CODE AUDITING

The Code Auditing by Quality Assurance is responsible for:

(1) Providing an independent audit of the code vs. design (on a module-by-module basis) for:

    (a) True translation of design into code
    (b) Adherence of code to established practices

(2) Providing the code audit status to Cognizant Development Engineering and to the Programming Secretariat.
(3) Providing code and documentation status audits before transfer to operations is completed.

I. TEST DESIGN

The Test Design function supports Cognizant Development Engineering and is responsible for:

(1) Generating test plans for concurrent correctness verification of build N code design against the requirements in the SRD and the architectural design of the SDD; also test plans for verifying the SOM.
(2) Designing dummy stubs and providing the corresponding code to be inserted to allow the entire program to be run.

(3) Generating test rationale and required test data, test code, test assembly code, and detailed test procedures.
(4) Continually providing to the Programming Secretariat test design detail and status.
(5) Consulting with the COE to help prepare user acceptance test plans for the STT.
(6) Performing other duties as assigned.

J. TEST DESIGN VERIFICATION

The Test Design Verification function supports Cognizant Development Engineering and is responsible for:

(1) Providing an independent verification that the test procedures and test code test all connecting paths and interfaces of the current build N module.
(2) Confirming that the test can be conducted and will, if successful, form a convincing test of build N correctness.
(3) Providing the test design status to Cognizant Development Engineering and to the Programming Secretariat.
(4) Providing testability considerations to the design activity.
(5) Performing other duties as assigned.

K. TEST CONDUCTING

The Test Conducting function supports Cognizant Development Engineering and is responsible for:

(1) Performing the correctness tests on a build-by-build basis and gathering all results.
(2) Providing testing detail and all results to the Test Evaluation function.
(3) Providing support to the Design function, as needed, if retesting is necessary.
(4) Performing other duties as assigned.

L. TEST EVALUATION

The Test Evaluation function supports Cognizant Development Engineering and is responsible for:

(1) Evaluating test results.

(2) Submitting test reports to Cognizant Development Engineering and to the Programming Secretariat for distribution.

(3) Performing other duties as assigned.

CHAPTER 2
THE STANDARD PRACTICE FOR THE SOFTWARE REQUIREMENTS DOCUMENT

SECTION I
INTRODUCTION

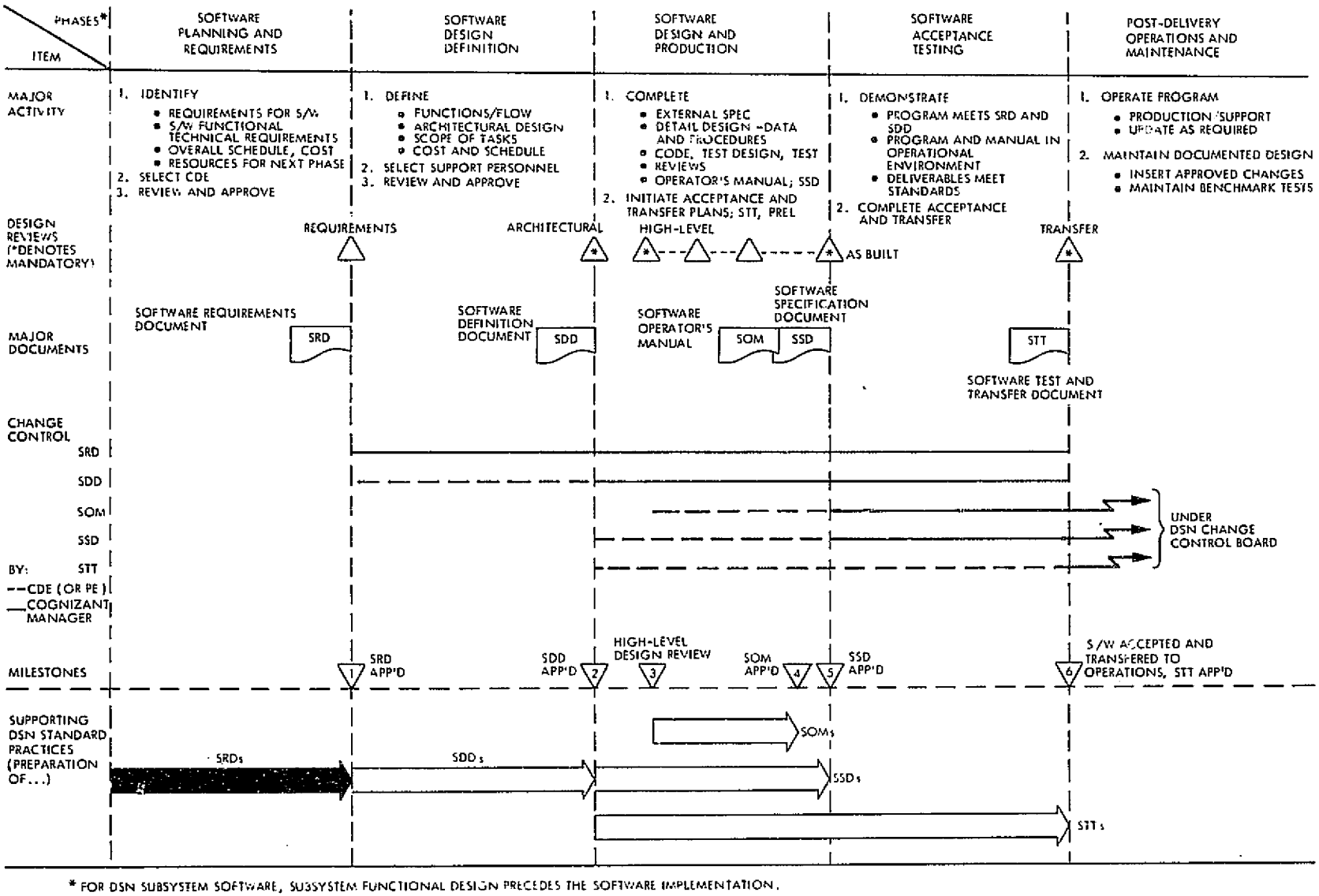
A. PURPOSE OF THIS STANDARD PRACTICE

This Standard Practice provides information and guidelines to assist an initiator in preparing and identifying the requirements for software and in obtaining management approval to proceed with the implementation. The Software Planning and Requirements Phase and its relationship to the overall software implementation process are shown in Figure 2-1. As shown in Figure 2-1, the results of the activity in this initial phase are documented in a Software Requirements Document. The SRD is reviewed and approved before proceeding to the next phase, the Software Design Definition, where the program's architectural design (high-level functional framework) is defined.

The guidelines and practices herein apply to SRDs written for proposed computer programs and capabilities (new programs or major extensions* to existing programs) which are intended to be transferred to operations.

Computer programs discussed in this book comprise both support (stand-alone capability for analysis, test, management, administration, and sustaining activities) and subsystem (on-line operations) software. For support software, the SRD is typically the top document, which describes and justifies the task to be undertaken in sufficient detail to justify the effort and to obtain approval for funds (or person-hours) to continue or complete the design. For subsystem software, the system and subsystem functional requirements documents and the subsystem functional design documents are the top documents, with the latter describing the capabilities to be provided by a combination of hardware,

*As determined by the Change Control Board, but typically major extensions require at least one person-year of effort.



* FOR DSN SUBSYSTEM SOFTWARE, SUBSYSTEM FUNCTIONAL DESIGN PRECEDES THE SOFTWARE IMPLEMENTATION.

Figure 2-1. DSN Software Management and Implementation Plan (Software Planning and Requirements)

software, and human functions. The subsystem functional design document is, therefore, the governing document which describes the software functions to be provided and thus justifies the effort (see note on Figure 2-1). The SRD is this case is not the top document, but it does reference and summarize the technical information from the functional design document while concentrating on other management information such as cost and schedule.

SRDs prepared in accord with this Standard Practice will provide to management the kind of information that it needs in both coverage and depth for making an informed decision to proceed. The following chapters provide guidelines and practices for preparing the subsequent documents needed as the project progresses.

B. SCOPE OF SOFTWARE REQUIREMENTS DOCUMENTS

The SRD provides to management a general understanding of what is needed, why, when, and how much the effort might cost. The SRD information, therefore, provides the main program justification upon which the management approval decision is made. For subsystem software, reference is made to the functional design document, and management information is added. Specifically, the scope of the SRD information will be to a level of detail to provide only:

(1) Sufficient functional and technical information and requirements to identify the program adequately to establish its need, and also to permit the next activity, the architectural design, to be completed. If the information is not extensive, it may be written into the SRD; otherwise it should be provided by reference to other documents. Where the information is not available in existing documents or where it is very complex or extensive, it may be appropriate to prepare (or reference) a separate Functional Requirements Document, and include it as an appendix to the SRD. The form can be that of the SRD, expanded to the detail needed; this allows correlation of technical detail with the controlling high-level requirements.

(2) An overall implementation schedule (with project need dates).

(3) A total-cost estimate for the complete implementation.

(4) An accurate estimate (10-percent goal) of the resources needed to complete the next phase of effort - generating the architectural design.

Any two of the first three items above should be firmly specified, while the remaining item may be bounded, e.g., "...costs shall not exceed...". This permits early renegotiation of resources and requirements.

Any applicable project-special guidelines or conventions, not explicitly prescribed by the Standard Practices, should be identified in the SRD for review and approval by cognizant management. The management approval level required for the SRD is the same as that required by normal organizational procurement policies for procurements in the amount of the estimated total cost of the program implementation, but does not go above the Assistant Laboratory Director level.

The SRD is maintained in the implementing organization and serves as a key working document throughout the implementation. During the implementation, there may be changes generated that must be reflected back into the SRD. Minor changes (for correctness) can be effected through a timely, low-cost internal (Cognizant Development Engineer approval) project change control process, which operates by memorandum. Major changes are submitted for initiator/management approval if they change the scope of what was already approved. Following the implementation, the SRD is filed in the Program Library, but it is not formally maintained, that is, it is not updated once the program is transferred to operations.

SECTION II

SRD CONTENTS

A. CONTENT OUTLINE

A typical SRD content outline is shown in Figure 2-2 along with an identification of personnel involved in preparing, concurring with, and approving the SRD. A brief introductory statement of the problem, the proposed program solution, and the program justification and rationale are presented first, followed by requirements and information which can be broadly categorized as (1) Management, (2) System, and (3) Program Requirements. Supplemental technical information, if extensive, can be included either in appendices, in a pre-existing Functional Requirements Document, or in other companion documents that may be generated and referenced.

The outline covers the typical items needed by management to assess the need and significance of the proposed program. For a given program, the actual contents may vary and will depend on the complexity of the program and the program interfaces. Items identified by an asterisk (*) in Figure 2-2 are considered always necessary, and there could be items in the table of contents other than those listed.

As a rule of thumb, SRDs typically might be expected to be about five or fewer pages for small (not exceeding six person-months) efforts, and seldom more than 15 or 20 pages for the very largest effort. In all cases, emphasis should be on brevity and effective exchange of information and not on formality, since the SRD is an implementation "working" document and does not formally survive beyond the transfer of the program to operations.

For SRD reviews, refer to Figure 2-3, Typical SRD Review Items and Guidelines.

*1. INTRODUCTION

*1.1 Problem Statement

*1.2 Program Description

*1.3 Justification for Program

1.3.1 Rationale and Relation to Other Programs

*2. MANAGEMENT REQUIREMENTS

*2.1 Standard Practices

2.2 Special Procedures

*2.3 Implementation Schedule, Resource Estimate, and Plan

*2.4 Resource Estimate for Architectural Design Phase

*3. SYSTEM REQUIREMENTS

*3.1 Hardware

*3.1.1 Computer, Peripherals, and Subsystems

3.2 Environmental

3.2.1 User, Operational, and Support

*4. PROGRAM REQUIREMENTS

*4.1 Overall

4.1.1 Structure, External Interfaces, Operating Modes and Options

4.1.2 Programming Language Priority

4.1.3 Utility (User Forgiving, Recovery, Interactive, etc.)

*4.1.4 Competing Characteristics

4.2 Processing

4.2.1 Inputs/Outputs (Conditions)

4.2.2 Algorithm and Accuracy

4.2.3 Flow Interfaces (Special)

4.2.4 External Data Bases and Files

*4.3 Testing

4.3.1 Correctness Testing

4.3.2 Acceptance Testing

SIGNATURES	
Prepared by:	Initiator(s)
Concurred by:	CDE (if different from Initiator)
Approved by:	Cognizant Management (Note)

Note: Approval of both the technical approach and the commitment of resources, to the regular management level as determined by the total resources required.

5. APPENDICES

(such as Functional Requirements Document, complex algorithm descriptions, Progress Report articles, manuals, other references, glossary, etc.)

*Item considered always necessary.

Figure 2-2. Typical Outline for a Software Requirements Document

Item	Guidelines
1. Breadth of Coverage	Requirements should be general, provide the essence of the program (the what, why, when, and cost), and bound the capabilities without performing the subsequent design.
2. Depth of Detail	Technical detail needed for credibility and management understanding and for completing the architectural design is provided either by reference or appendix (if so extensive that it breaks flow).
3. Testing	Overall test criteria for understanding and bounding the subsequent correctness and acceptance test efforts are indicated. In particular, any special tests or diagnostic programs required must be completely identified.
4. Special Conditions	Special practices, conventions, or requested deviations from the Standard Practices, if any, as listed in the SRD, must be reconciled before proceeding to the next phase.
5. Competing Characteristics	The reviewers (or review board) should be in basic agreement (or arrive at such agreement) with the priorities as assigned to the list of competing characteristics presented in the SRD.
6. Level of Need	The program need and justification are reviewed. The estimated costs are weighed against the level of capabilities proposed. The level of need that the available funding will support is reviewed; if downward adjustments in capability are needed, the list of item 5 above is used to retain the highest priority capabilities.
7. Schedules	Interactions with existing schedules are examined for assessing demands on available resources. Also the firmness of the schedules is estimated based on the availability of firm inputs. Contingency plans, if any, are reviewed (or recommended).
8. Resources	The availability of the requested resources is weighed against management's willingness to commit these resources to the effort in this time frame.
9. Approval	Approval to proceed is based on the aggregate results and outcome of the SRD review, taking into consideration the recommendations of the reviewers.

Figure 2-3. Typical SRD Review Items and Guidelines

Item	Guidelines
1. Breadth of Coverage	Requirements should be general, provide the essence of the program (the what, why, when, and cost), and bound the capabilities without performing the subsequent design.
2. Depth of Detail	Technical detail needed for credibility and management understanding and for completing the architectural design is provided either by reference or appendix (if so extensive that it breaks flow).
3. Testing	Overall test criteria for understanding and bounding the subsequent correctness and acceptance test efforts are indicated. In particular, any special tests or diagnostic programs required must be completely identified.
4. Special Conditions	Special practices, conventions, or requested deviations from the Standard Practices, if any, as listed in the SRD, must be reconciled before proceeding to the next phase.
5. Competing Characteristics	The reviewers (or review board) should be in basic agreement (or arrive at such agreement) with the priorities as assigned to the list of competing characteristics presented in the SRD.
6. Level of Need	The program need and justification are reviewed. The estimated costs are weighed against the level of capabilities proposed. The level of need that the available funding will support is reviewed; if downward adjustments in capability are needed, the list of item 5 above is used to retain the highest priority capabilities.
7. Schedules	Interactions with existing schedules are examined for assessing demands on available resources. Also the firmness of the schedules is estimated based on the availability of firm inputs. Contingency plans, if any, are reviewed (or recommended).
8. Resources	The availability of the requested resources is weighed against management's willingness to commit these resources to the effort in this time frame.
9. Approval	Approval to proceed is based on the aggregate results and outcome of the SRD review, taking into consideration the recommendations of the reviewers.

Figure 2-3. Typical SRD Review Items and Guidelines

B. OUTLINE DISCUSSION

1. Introduction

a. Item 1.1, Problem Statement. The nature of the problem is described and background information needed for management understanding of its significance and programmatic need is provided. Also, the environment in which the problem is embedded is discussed, together with identification of input sources. Data output needs are identified, along with any peculiar constraints or circumstances associated with the problem which may affect schedule, cost, staffing, programmatic justification, or anything else which management may need to know before granting its approval to proceed to the next phase. For subsystem software, existing functional requirements documents are referenced.

b. Item 1.2, Program Description. A functional description of the needed program capability is presented, including the nature and dominant characteristics of the problem solution and the type of data to be generated. The type of program is identified, such as (1) real-time, batch, or permanently interactive, (2) computational or data manipulation. Also, the general system environment in which the program is to operate is identified, such as, for example, whether the program is on a general-purpose machine or on a dedicated machine.

c. Item 1.3, Justification for Program. Additional information that may be needed to support the management decision to implement the proposed program capability is presented to the level of detail judged necessary based on the projected performance and estimated cost. The information may include such items as results of assessments of alternative methods considered, assessments of the consequences of doing without the proposed program, time criticality of the program, and relation to other programs.

2. Management Requirements

a. Item 2.1, Standard Practices. Applicable software implementation Standard Practices are described in Chapter 1 and are also summarized in

the Management and Implementation Plan, depicted earlier in Figure 2-1 of this document. Individual efforts may require minor deviations or waivers from the Standard Practices. These requested modifications, if any, are identified and presented along with supporting rationale for subsequent management review and concurrence (which may or may not be granted).

b. Item 2.2, Special Procedures. Specific software implementations may require additional project-peculiar guidelines and conventions or certain special procedures. Any additional guidelines, conventions, and special procedures are identified and presented along with supporting rationale for management review and concurrence (or rejection).

c. Item 2.3, Implementation Schedule, Resource Estimate, and Plan. The overall implementation schedule and cost estimates are presented, along with any assumptions and clarifications that may be needed by management to gauge their accuracy or limitations. Preparation detail is provided in Section IV. Figure 2-4 indicates the kind of information needed for developing the overall projected implementation schedule and its corresponding rough cost estimate. The intent is to sketch out an overall work plan to the completion of the implementation and to estimate the total cost to within, say, a factor of two. Any special funding arrangements, limitations, or Budget Change Requests should be identified. For comparison, budgeted resources can typically be identified; however, such resources should not be identified for SRDs included in a procurement package.

Plans for the use of available support facilities, production libraries, and automatic aids should be included, as well as overall plans to do the work in-house or to contract it out.

d. Item 2.4, Resource Estimate for Architectural Design Phase. The near-term implementation schedule and estimate of resources needed for the next phase, covering the program architectural design, are presented. Figure 2-5 indicates the kind of information needed for developing the near-term schedule of activities along with an identification and estimate of the amount of resources needed to complete the program design definition. The accuracy goal of this cost estimate is 10 percent.

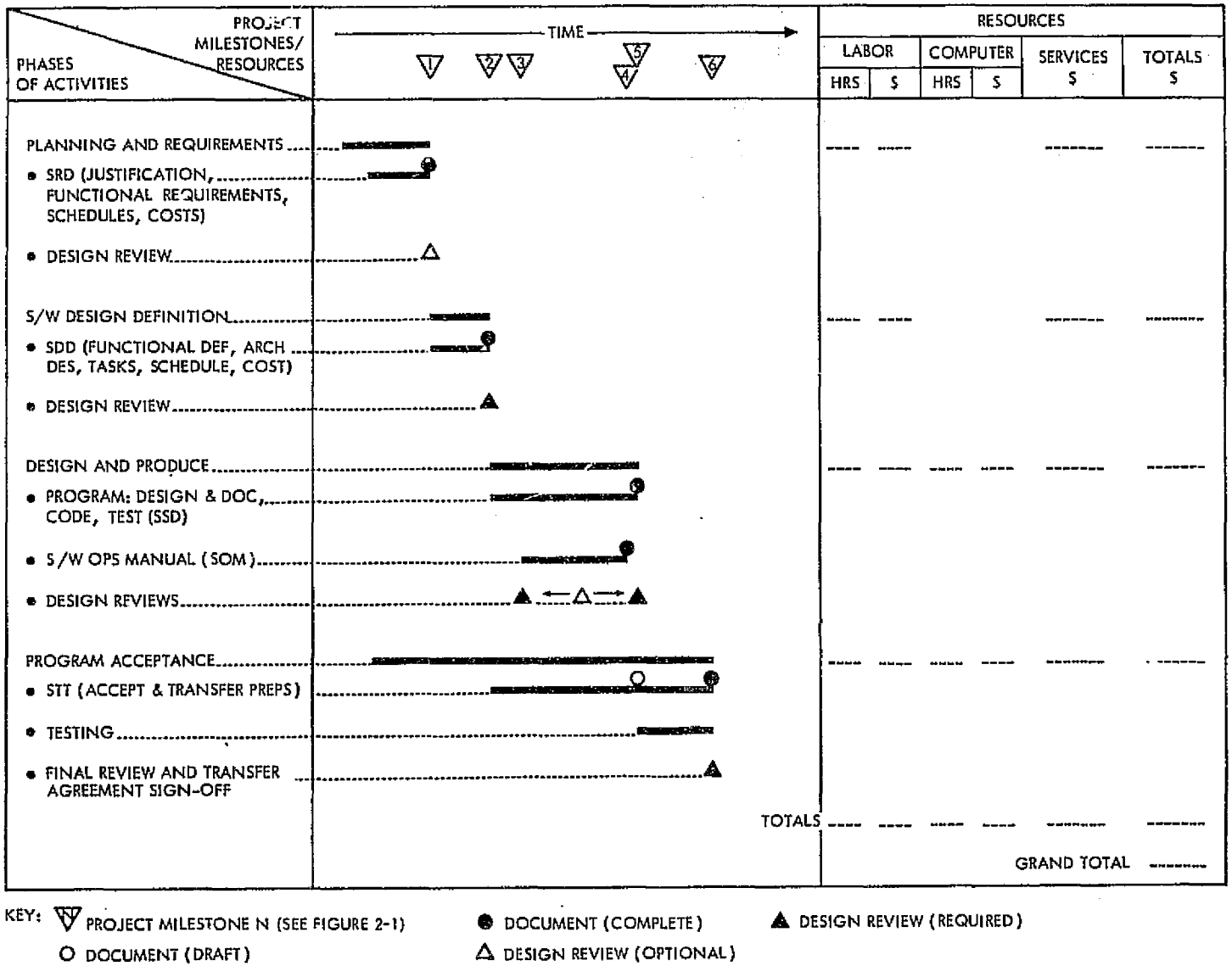


Figure 2-4. Overall Implementation – Activity Schedule and Cost Estimate

PHASES OF ACTIVITIES / PROJECT MILESTONE / RESOURCES	TIME → (1) SRD APP'D (2) SDD APP'D	RESOURCES					
		LABOR		COMPUTER		SERVICES	TOTALS
		HRS	$	HRS	$	$	$
SDD PREPARATION							
• MANAGEMENT INFORMATION		----	----			------	------
• SCHEDULES							
• COST ESTIMATES							
• TEAM SETUP							
• PROGRAM ANALYSIS, DEFINITION, AND STRUCTURE		----	----			------	------
• INPUT/OUTPUT							
• FORMATS							
• DATA STRUCTURES							
• VARIABLE TABLE							
• STORAGE							
• PROCESSING							
• DESIGN CHARTS							
• NARRATIVE							
• SUPPORTING ANALYSIS							
• TEST DESIGN OUTLINE		----	----			------	------
• UNIT (CORRECTNESS)							
• SYSTEM (ACCEPTANCE)							
• TRANSFER (PLANS)							
DESIGN REVIEW		----	----			------	------
• PREPARATION AND PRESENTATION							
	TOTALS	----	----			------	------
						GRAND TOTAL	------

Figure 2-5. Software Design Definition Phase — Typical Items of Activity, Information, and Cost Estimate

3. System Requirements

a. Item 3.1, Hardware. The computer system (computer hardware and its operating software) characteristics are identified (by reference, if extensive) as well as requirements for any special peripheral equipment or demands made by the program on the computer system (memory size, storage, speed). Reference can be made to Automatic Data Processing plans if the computer has not yet been procured.

The related DSN subsystems, or any other subsystems, that are being supported or have cost or schedule influence on the software being implemented are identified and referenced to coordinate hardware changes and the possible effects on the software.

b. Item 3.2, Environmental. Requirements and characteristics of program-external items, such as special equipment, test programs, or resources needed for the implementation, test, and operation of the program, are identified; the CDE can subsequently plan and negotiate for their availability when needed. These include the availability of capabilities and facilities of the user, operations and support areas, such as special test equipment and programs. Also identified are any special characteristics of new or existing programs intended to be inserted into existing assemblages of programs.

4. Program Requirements

a. Item 4.1, Overall and Competing Characteristics. Information and requirements that provide to management an overall understanding of the program and its use are presented. Requirements should be presented (or referenced) only to a level of detail sufficient to allow the architectural design to proceed with assurance to management that what it approved is what is done. Typical coverage might identify:

(1) Structure
(2) Functional interfaces

(3) Major operating modes and options
(4) Programming language priority
(5) Utility
(6) Competing characteristics

Items 1 through 5 are included only if and as they clarify what is needed and when they help establish the necessity and authority for the program. Further descriptions and guidelines for preparing these, when needed, can be found in Section IV.

Item 6, Competing Characteristics, is always included and addresses the total set of requirements that characterize the specific implementation. The relative importance of meeting the principal management and technical requirements for the program, as identified throughout the SRD, is assessed. The principal requirements are then listed in decreasing order of priority, along with a reference to the SRD paragraph where the requirement is specified.

A sample list of program characteristics and requirements, along with the paragraph number in which they are specified and discussed in the SRD (see SRD outline, Figure 2-2), is shown below in decreasing order of priority.

(1) Maximum Program Size (Section 4.1)
(2) Throughput Time (Section 4.2)
(3) Ease of Use and Forgiving (Section 4.1.3)
(4) Numerical Accuracy (Section 4.2.2)
(5) Schedule (Section 2.2)
(6) Ease of Maintenance (Section 2.1)
(7) Extensibility (Section 4.1.1)
(8) Cost to Develop (Section 2.2)

In the event of unforeseen problems, delays, or budget cuts, this list will be referenced in arriving at decisions that will allow resolution of conflicts while preserving the highest priority requirements. The list can also be used to negotiate changes to the SRD before approval, if the resources required for

the original are regarded by management as too large. Changes in scope caused by invoking the list of competing characteristics must be recycled through management approval.

b. Item 4.2, Processing. Requirements affecting the processing of inputs to obtain the required outputs can be stated but only to the extent that realism of resources and schedule can be assessed by management. Typical coverage might identify:

(1) Inputs/outputs
(2) Algorithms and accuracy
(3) Flow interfaces (special)
(4) External data bases and files

These items are included only if and as they clarify what is needed, and when they help establish the necessity and authority for the program. Further descriptions and guidelines for preparing these, when needed, can be found in Section IV.

c. Item 4.3, Testing. Special requirements are defined that will govern the test approach and form the basis for assuring the correctness and acceptability of the completed program. If no special test requirements are given here, it is assumed that the appropriate Standard Practice is sufficient and is referenced; that is, for correctness testing, the SP for the Software Specification Document and for acceptance testing, the SP for the Software Test and Transfer Document.

The Standard Practice for correctness testing addresses top-down module-by-module logic testing; the testing of every individual module connecting link; the testing of the ability to correctly handle conditions such as minimum value inputs, maximum value inputs, and randomized inputs; and whatever else is necessary to convince one of correctness at a given level of the top-down hierarchical expansion. Test practice might typically consist of desk and computer

checking of each module, followed by similar checks of top-down assemblages (builds) of modules and dummy code for incomplete modules. This test sequence continues until the program construction is complete (and correct in the narrower software sense) and ready for acceptance testing.

For real-time correctness, the theory and practice for sufficient testing are still being developed.

The Standard Practice for the acceptance testing of support and subsystem software applies to the completed total program and pertains to the end-to-end validation of the program in its operational environment. The user or user representative, typically the software Cognizant Operations Engineer (if assigned at this point in the process), is involved in the planning of the acceptance tests. As such, the user and/or COE typically provides any special acceptance test planning inputs to the SRD preparer. For subsystem software, system testing is performed by the software COE after the program has successfully completed subsystem testing.

SECTION III
SRD PREPARATION, REVIEW, AND APPROVAL

A. PREPARATION ACTIVITIES

1. Initiation and Requirements Identification

The persons who may initiate an SRD are determined by the management structures of the organizations involved. The initiator formulates (or restates, in reference to a Subsystem Functional Design Document) a statement of the problem from which overall costs, schedules, and functional requirements of the system and program can be identified and readily understood by management, so that the programmatic need for the implementation can be determined. The SRD content (type of information and its organization and format) is outlined and described in Section II, where the outline guides the initiator directly to the main items and points of information needed. The initiator is directed to Section IV and Reference 1 for further helpful information on requirement identification and documenting this information in the SRD.

Throughout the planning, information gathering, and requirement identification process, informal communications involving the initiator, the initiator's management, the using organization, the implementing organization, and other concerned personnel are strongly encouraged, particularly on relatively large programs. The emphasis of the SRD is not directed so much to the computer program as such, but more to the plan for producing it, that is, to the what, why, when, and how much (or expected resource demands and costs). Therefore, the personnel involved in supporting the preparation of the SRD can be managment oriented: there should typically be no need for them to be software experts.

During the planning and requirements phase, an individual is assigned by the implementing organization to assume the Cognizant Development Engineering duties, which include broad implementation responsibility in both the management and technical sense. For larger efforts, the Cognizant Development

Engineering role can be assumed by a Project Engineer. If assigned early in the planning phase, the CDE can also assist the initiator in preparing the SRD, especially for items involving activity breakdown and scheduling. In any event, Cognizant Development Engineering should concur with the SRD before authorization is granted to proceed to the architectural design.

2. Arbitration of Conflicting Requirements

Because of the many requirements specified in the different areas of the total project (i.e., management, system, program, and resource requirements and limitations), it is almost inevitable that some requirements will be incompatible with others to varying degrees. An attempt is therefore made in the planning phase to assess the principal requirements as to their priority in relation to the overall program implementation. Results of this requirements analysis are listed in prioritized order in Section 4.1.4 (Competing Characteristics) of the SRD.

Resolution of unforeseen problems requiring possible future management negotiation is enhanced by the availability of this prioritized list. Also, the subsequent architectural design effort can consider these priorities in selecting a compatible approach for meeting the ordered set of requirements.

The actual resolution of conflicts, if they arise, is made by the CDE and communicated to the cognizant management; the prioritized competing characteristics table is a heavily weighted factor in the final resolution. It is expected that the CDE will be able to resolve the majority of conflicts that fall within the established scope of the original implementation effort. For major conflicts requiring a change in scope modifying the substance or intent of the SRD, negotiation with the organization or function generating the requirements embodied in the SRD will be necessary. These changes in scope will be cycled through normal management approval, the same approval chain as for the original SRD. A new design review may be called for before approving the new or revised SRD.

B. REVIEW AND APPROVAL

The main objective of the SRD review is to assure that the management, system, and program requirements are stated in sufficient detail and coverage to provide an understanding of the implementation schedules, costs, and performance (see Section I, Paragraph B; Scope of SRD).

The review of the SRD can be achieved either by document circulation, with a request for return of redlined comments, or by a formal SRD review - at the option of the cognizant management. The formal review could be held as part of a more general review of subsystem requirements, and several related SRDs can be reviewed during one review. For further information, if a formal review is requested, the preparer may refer to the "Review" information of Chapter 3 on conducting the formal review and guidelines for agenda items and presentation levels of detail. The preparation and distribution of the SRD and other review materials, regardless of the type of review, is the responsibility of the initiator. The personnel participating in the review should represent, at a minimum, the initiator, the using organization, the funding organization (if different from the using), and the implementing organization. Typical SRD Review items and guidelines are shown in Figure 2-3 and can be used as an aid to assure completeness of detail and coverage.

To summarize, approval is sought for commitment of resources to the proposed general technical scope. Approval signatures are based on the estimated total program cost as required by the normal organizational procurement policy; however, approval above the ALD level is not required. Approval by management authorizes the architectural design to begin.

SECTION IV
OVERALL SUPPORT AND PREPARATION AIDS

A. SUPPORT

Overall identification of the support needed and the responsibilities involved in the software implementation process are summarized in Table 2-1 and discussed in Chapter 1. The initiator's main activities of coordination and documentation are shown for the planning and Requirements Phase, with prime support from Cognizant Operating Engineering on acceptance testing. These activities lead to review and approval of the proposed plan, contained in the SRD, by cognizant management. It should be noted that, for subsystem software, the results of the prior functional design activity are referenced and form a starting point for preparing the SRD.

B. PREPARATION AIDS

Aids for formatting the SRD and a brief discussion of the kind and characteristics of information needed for the SRD follow.

1. SRD Format Conventions

The SRD is a project working document which does not formally survive the implementation, so formality of style and formatting are not emphasized, but consistency from document to document is encouraged. Therefore, for convenience and consistency, the preparer is directed to Chapter 4 and is encouraged to use the same formatting conventions as adopted for the formal, surviving documents -- i.e., the Software Specification Document, Software Operator's Manual, and Software Test and Transfer Document.

Basically, these documents are "block-formatted" and "sectionalized," where each section is identified by consecutive arabic numbers. The pages, figures, and page-size tables are numbered consecutively within each section (e.g., 1-1, 1-2, . . . 1-n; 2-1, 2-2, . . . 2-n; etc.). Small tables can be inserted

Table 2-1. DSN Software Implementation Process Support Responsibilities (Planning and Requirements)

Support Activity	Implementation Phase**				Operations & Maintenance
	Planning & Requirements	Design Definition	Design & Production	Acceptance Test and Transfer	
Coordination of Activities	Subsystem Engineer (SSE) or Initiator	Cognizant Development Engineer (CDE)*	CDE*	SSE and CDE*	COE
Design	N/A	CDE	CDE	N/A	Cognizant Sustaining Engineer (CSE)
Acceptance Test Input and Support	Cognizant Operations Engineer (COE) (if assigned)	COE	COE	COE	COE
Documentation of Results	SSE or Initiator	CDE	CDE	CDE*	CSE (for sustaining changes)
Plan and Conduct Review	SSE or Initiator	CDE*	CDE*	CDE*	SSE, if applicable
Approval	Cognizant Management (CM)	CM	CM	Quality Assurance (QA) and CM	QA/CM
Milestones	Software Requirements Document (SRD)	Software Definition Document (SDD)	Software Specification Document (SSD) Software Operator's Manual (SOM)	Software Test and Transfer Document (STT)	Engineering Change • Requests (ECRs) • Orders (ECOs)

*Or Project Engineer, for large implementations.
**For subsystem software, subsystem functional design precedes the formal software implementation.

within the running text and are not necessarily numbered. However, if other parts of the report refer to the table, the table should be numbered and placed at the top or bottom of its page.

Subsections use decimal identification. Appendices are identified alphanumerically, need no blank or separate introductory page, and formatting within the appendices can vary to allow ease of incorporating existing material without modification.

For further format detail and specific information on typewriter tab settings, line spacing, capitalization, underlining, etc., the preparer (and typist) is referred to Chapter 4.

Graphic aids for producing special figures, such as HIPO (Hierarchy plus Input -- Process -- Output) charts, information flow diagrams, etc., along with other conventions, are presented in Chapter 3; these can be used in the SRD. Procedural flowcharts (sequential control logic) are typically not included in SRDs; however, if appropriate for inclusion, see Reference 4, which also presents the adopted standard symbols and usage.

2. Content Characteristics

The SRD must justify the need for the proposed program (for DSN subsystem software, reference is made to the Subsystem Functional Design Document). The following comments are included to aid in preparing the kind of information required to establish this need. These comments parallel the SRD presentation and fall into the broad categories of (1) Management, (2) System, and (3) Program.

a. Management. Chapter 1 presents overall guidelines and practices to apply to software implementation projects. The SRD provides additional information applicable to the specific proposed software on the management structure under which the project is to be carried out, on the identification of the organizations involved in the implementation, on the particular support needed,

and on the organization to which the program will be transferred. With this underlying background information, management can then evaluate the significance and implications of the proposed overall and near-term (architectural design) activity levels, schedules, and cost estimates.

1) Overall Implementation Schedule and Total Cost Estimate. Knowing the need date or desired delivery date and using information from Paragraphs B2 and B3 of this section concerning the relative complexity and sophistication of the software effort and the characteristics of the associated system and hardware, an initial overall schedule of activities can be generated. The overall schedule will be used primarily to bound the time span of proposed activities. Possible conflicts between the projected activities and other ongoing work can be assessed and priority assignments can be made, if needed. The initial schedule is not intended to be a day-to-day work schedule; however, it does form a baseline, which is refined in the next phase based on more detailed inputs. The overall schedule addresses the major activities performed in the various phases of the implementation in meeting the project milestones. Since the results of the major activities are documented and reviewed before moving on to the next phase, an estimate of the activity involved in terms of the document and the review can be made for each phase, and associated costs for each can be estimated. Figure 2-4 displays this concept and is included for visual illustration of these points. The purpose of Figure 2-4 is not to show how to perform the activity analysis or cost estimation, but only to indicate what is needed. For presenting this information in the SRD, a typewritten tabular form is adequate; there is no need for preprinted forms in the SRD itself. It should be noted in Figure 2-4 that the cost for the planning phase will closely reflect the actuals for the preparation of the SRD. Further, the estimate for the design definition (architectural design) is made as discussed below; and it can be used in compiling the total cost estimate. This is a near-term estimate, and an accuracy to within 10 percent is expected for architectural design. The remaining two phases (Design/Production and Program Acceptance) will probably be the items with the highest degree of uncertainty, because they are further into the future and involve complex activities whose costs are sometimes difficult to estimate. Further details as to what is needed in these phases are presented in subsequent chapters.

2) Architectural Design Activity and Cost Estimate. The major areas of activity following SRD approval (Milestone 1) through SDD approval (Milestone 2) are identified and estimated in terms of personnel levels and dollars. Activities involve problem and solution analyses, program high-level definition, and the review as discussed in Chapter 3. Figure 2-5 illustrates typical types of information that are characteristic of the design definition phase. Estimates of the type and degree of skills involved, along with estimates of the amount or level of support needed in each, allow a composite estimate of costs to be made for the design definition phase. The accuracy goal of this composite estimate is 10 percent. This activity plan and cost estimate provide a means of measuring overall performance and progress through the design definition phase, allow the early identification of any major unexpected problems, and permit corrective action to be applied in time to avoid major impacts.

b. System. System information provided should tell management which computer system and equipment is, or may be, involved, whether any new capabilities are required which could involve new automatic data processing equipment procurements, and any other special demands placed on the computer system that might affect costs and schedules or involve long lead time procurements. In addition, associated subsystem hardware should be identified to assure that compatibility can be retained throughout the implementation in the event of changes to the subsystem hardware characteristics or configuration. Also, special environmental requirements must be identified, if imposed by the user, operations, or support areas.

c. Program. Program information and requirements provided are functional in character and provide to management an understanding of the significance and complexity of the proposed program. Emphasis is placed on factors which tend to drive costs up if they are not identified and frozen early; factors involving special complexity, state of the art, or requiring "breakthroughs" that tend to jeopardize schedules and place stringent demands on specific (and possibly scarce or unavailable) personnel; and factors that govern and bound the testing activities needed for program acceptance and delivery. Emphasis at this stage is typically not placed on highly detailed items unless

they would have a major influence on resources, schedule, or programmatic requirements (i. e., specific input and output formats, details on program external and internal data structures, computer internal operations, etc.), since these normally would not be relevant to management approval.

For certain implementations, further identification of program requirements may be needed, other than as outlined in Section II of this chapter. Additional guidelines are provided below for the SRD items, 4.1 (Overall Requirements) and 4.2 (Processing Requirements) of Figure 2-2, covering:

(1) Structure
(2) Functional interfaces
(3) Major operating modes and options
(4) Programming language priority
(5) Utility
(6) Inputs/outputs
(7) Algorithms and accuracy
(8) Flow interfaces
(9) External data bases and files

1) Structure. Introductory information contained in Paragraph 1.2 of Figure 2-2, Program Description, can be expanded to briefly identify the program functional requirements and overall features and options. If appropriate, program modes and external controls, as well as data flow, can be described functionally. Functional block diagrams (not control logic flowcharts but key logic identification) may be included but are not mandatory.

2) Functional Interfaces. This information can identify and clarify, but only functionally, the program inputs and outputs, treating the program as a whole as a "black box" entity. Detailed formats and data structures are not necessarily known or specified at this time. The general data content, sources, availability, storage medium, and perhaps other characteristics for the receipt of inputs and the distribution of outputs may be identified. External interfaces with other programs and all non-program sources can be identified and summarily described.

3) Major Operating Modes and Options. Further detail on program major operating modes, options, and conditions of Paragraph 1.2 of Figure 2-2 can be given, along with associated requirements, so that user organization inputs concerning operation of the program can be factored into the program implementation approval decision early in the cycle.

4) Programming Language Priority. The computer language can be specified, if relevant to approval. Requirements for computer languages other than the standard language(s) (when existing) will be accompanied by a brief explanation of the rationale and reasons behind the decision. If the machine on which the program is to be run has not been chosen, open options should be stated.

5) Utility. Additional user-oriented requirements can be provided to assure ease of use of the program in its operational environment. Specified requirements might relate to items such as (a) recovery from operator input errors, (b) operator prompting, (c) ease of program modification and extension, (d) ease of accommodating data base changes, (e) format flexibility and selection.

6) Inputs/Outputs. General requirements for initialization, checking, and preprocessing of inputs can be stated. Also identified might be requirements for general or specific prompting messages and other certain major output capabilities such as printed forms (tables, plots), files (public, restricted), storage and save capabilities (punched cards, magnetic tape, paper tape), and ease of accommodating flexible inputs.

7) Algorithms and Accuracy. Special existing capabilities, algorithms, or transformations that are available to meet specific requirements can be identified for possible use in the subsequent design definition.

Required accuracy and precision of critical inputs and outputs can be specified for governing the subsequent design definition process and for forming a basis for the subsequent testing activities.

8) Flow Interfaces. Special internal requirements, if any, can be identified, such as program-peculiar sequencing, timing requirements, internal checks, module stand-alone operational capability or transferability to other implementations, and provision for future added capability planned but not presently being implemented. This allows the difficulty of the implementation to be assessed by management, and permits common elements in several ongoing implementations to be identified for further cost reduction.

9) External Data Bases and Files. Requirements can be specified that affect the basic program design and which concern such general items as external data/file availability, accessibility, changeability, multi-user update capability, periodic maintainability, and controllability. Detailed data structures and file format definitions are not normally addressed at this stage, since they would not typically have a major impact on schedule, resources, or programmatic requirements.

CHAPTER 3
THE STANDARD PRACTICE FOR THE SOFTWARE DEFINITION DOCUMENT

SECTION I
INTRODUCTION

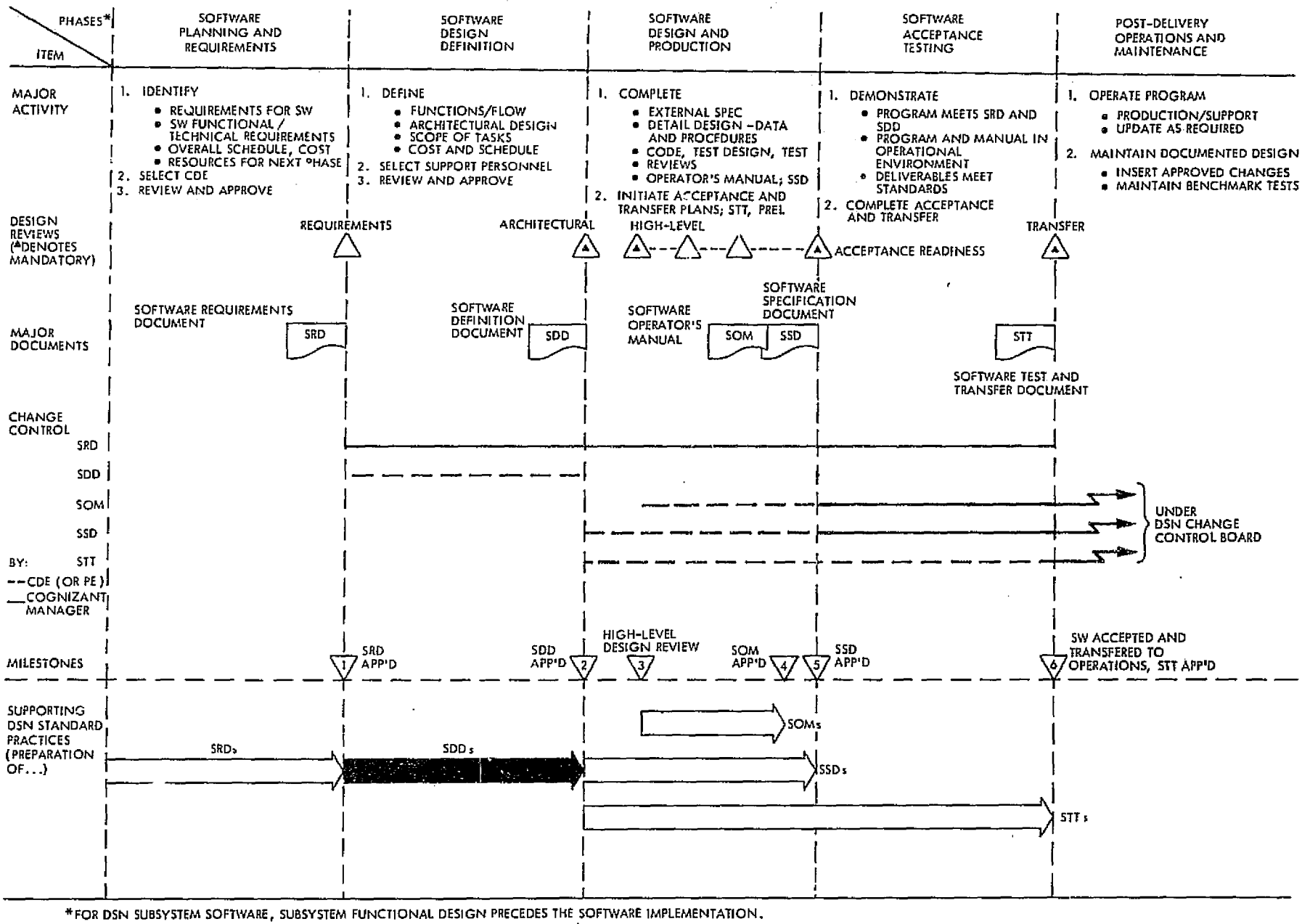
A. PURPOSE OF THIS STANDARD PRACTICE

The Design Definition Phase and its relationship to the overall software implementation process are shown in Figure 3-1. As shown in the figure, the results of the activity in this phase are documented in a Software Definition Document. These guidelines and practices apply to SDDs written for computer programs to be transferred to operations. Such programs are authorized to proceed to the architectural design because the SRD was approved. The SDDs prepared in accordance with this Standard Practice are to provide management with the technical and resource information that it needs in both coverage and depth for making the decision whether to proceed as proposed. This information must allow management to assess:

(1) The CDE's interpretation of the job required by the SRD.
(2) The architectural design overview.
(3) The work breakdown, schedules, staffing, and costs.
(4) The implementing management's monitor plan.

B. SCOPE OF SOFTWARE DEFINITION DOCUMENTS

The SDD contains the implementer's preliminary design and management proposal in response to the requirements presented in the SRD. Therefore, the scope of the SDD information can include:



*FOR DSN SUBSYSTEM SOFTWARE, SUBSYSTEM FUNCTIONAL DESIGN PRECEDES THE SOFTWARE IMPLEMENTATION.

Figure 3-1. DSN Software Management and Implementation Plan (Software Design Definition)

(1) A definition of functional, technical, and verification information, including:

   (a) The preliminary design of the total program, which may include (but is not constrained to) state diagrams, data structures, external data flow, and key algorithms to the extent necessary to demonstrate an understanding of the overall job and its complexity and for assessing whether the proposed solution is a good one (i.e., practicable and responsive to the requirements).
   (b) An overview of the program architecture, including a procedural design down to the level needed for task definition and sizing; 90 percent accuracy in module count can be expected, but not a detailed correctness assessment at this point. The architectural design is done only to the extent necessary to size the job.
   (c) Program performance criteria and quality criteria for acceptance, such as core size, speed, accuracy, and timing.

(2) A high-level Work Breakdown Structure that can include a module breakdown but emphasizes tasks rather than modules (along with the planned personnel/team level of support (not names)). The task descriptions that support the WBS should be available to verify their existence, but these are typically not included in the SDD.
(3) A schedule of all remaining activities, along with a proposed status reporting plan.
(4) A refined cost-to-completion estimate, accurate to within 10 percent, that is based on a module tree and Work Breakdown Structure derived from the preliminary design.

The SDD is reviewed and approved as discussed in Section III, Paragraph B, Review and Approval. After approval, the SDD is kept on file by the Programming Secretariat for information purposes and is not formally maintained or updated as a surviving document. However, the WBS and the schedules are maintained throughout the implementation. The SDD may contain much information that can be redlined or upgraded for subsequent inclusion in the Software

Specification Document when and as the formal design is implemented top-down with a detailed correctness assessment. This redlining process allows retention of vital SDD information, reduces the cost of preparing the SDD, and avoids duplication of effort.

SECTION II
SDD CONTENTS

A. CONTENT OUTLINE

A typical SDD content outline is shown in Figure 3-2, along with an identification of personnel functions involved in preparing and approving the SDD. An introductory description of the proposed program and its overall architecture is presented based on the requirements identified in the SRD. This is followed by more detailed information which describes and defines the (1) Management of the Implementation, (2) System Environment, (3) Program Architecture, and (4) Coding and Test Design Criteria.

The outline covers the major items needed by management and other reviewers to assess the overall problem definition and credibility of the cost and schedule estimates extending through transfer to operations. For a given program, the actual contents will depend on the complexity of the program and the program interfaces. However, items starred (*) in Figure 3-2 are considered always necessary. Conversely, there could well be items in the table of contents other than those listed. The Software Requirements Document is used as a guide to determine exactly what items of an optional nature should appear in the SDD. The SRD might also provide some specific guidelines as to the level of detail, or even a maximum page count, that would be reasonable for the following SDD to be responsive, yet not overresponsive. In all cases, emphasis should be placed on brevity and effective exchange of information and not on formality. This is especially true because the SDD does not formally survive after approval and is retained by the Programming Secretariat for information purposes only.

*1. INTRODUCTION

- *1.1 Purpose and Scope
- 1.2 General Program Description
- *1.3 Architecture Overview

*2. MANAGEMENT INFORMATION

- *2.1 Project Schedules and Reporting Plan
- *2.2 Work Breakdown Structure
- 2.3 Personnel/Team Level of Support
- *2.4 Cost-to-Completion Estimate
- 2.5 Competing Characteristics and Problem Areas
- *2.6 Design Reviews

*3. SYSTEM ENVIRONMENT

- 3.1 Computer, Peripheral Equipment, and Subsystems
- *3.2 Program Operation in System
- 3.3 Software Interfaces and Services

*4. PROGRAM ARCHITECTURE

- *4.1 Preliminary Functional Analysis
- *4.2 Data Characteristics
- *4.3 Preliminary Program Structure and Definition

*5. CODING AND TEST DESIGN CRITERIA

- *5.1 Coding Criteria and Constraints
- *5.2 Correctness Testing Criteria
- *5.3 Acceptance Testing Criteria

6. APPENDIXES
(Glossary, Reference, Technical Information, etc.)

SIGNATURES	
Prepared by:	CDE (Helpers)
Approved by:	Implementation Organization Line Management and others (Note)

Note: Implementer concurs with costs, schedules, and expenditure of resources, before proceeding to next phase. Others (DSN Program Office SE for DSN Subsystem software) concur during the architectural design review.

*Item considered always necessary.

Figure 3-2. Typical Outline for a Software Definition Document

B. OUTLINE DISCUSSION

1. Introduction

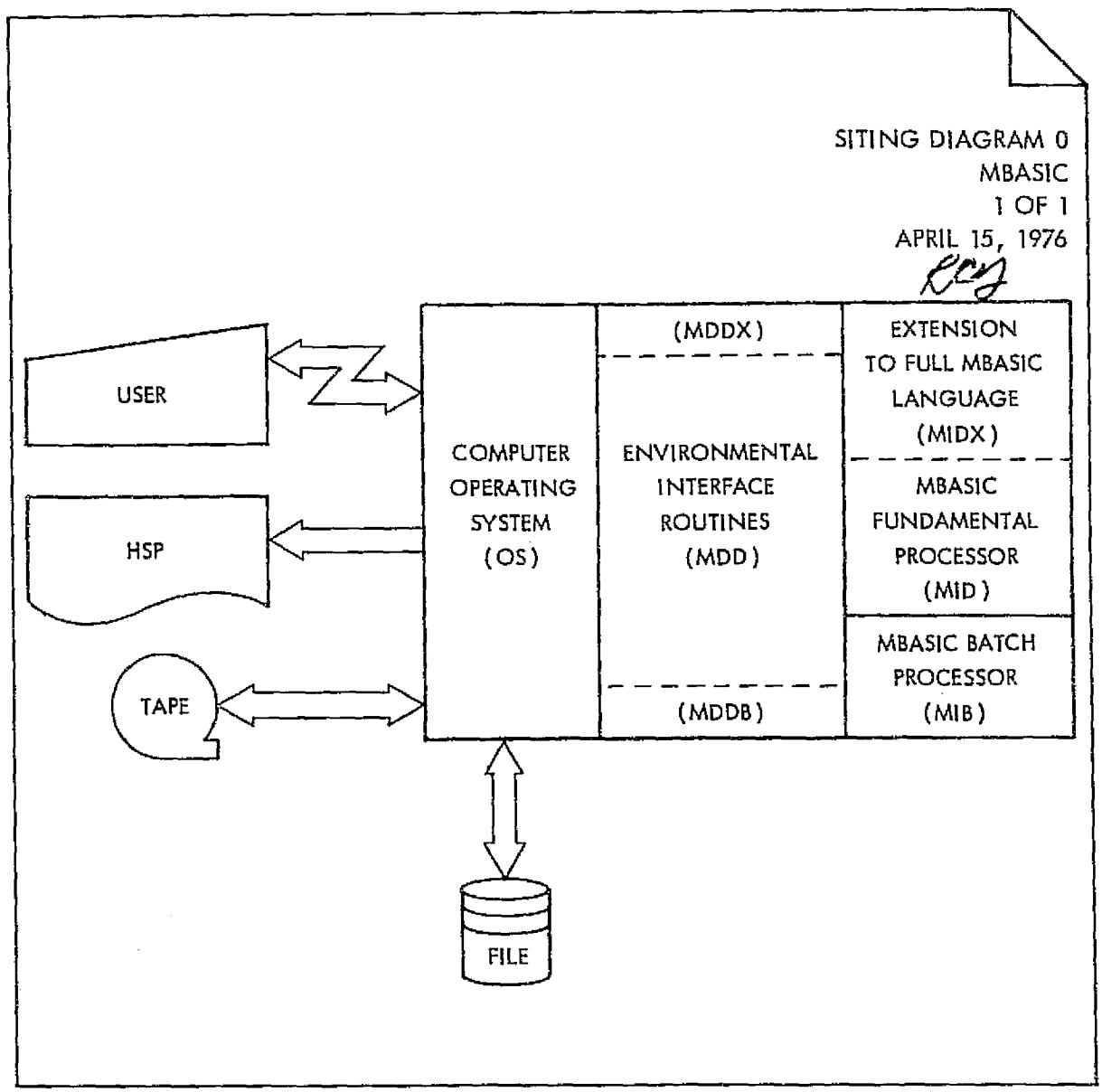
a. Item 1.1, Purpose and Scope. The purpose of the SDD is to provide a proposal that is responsive to the requirements and problem statement announced in the approved SRD, and that provides credible evidence of a 10 percent accuracy in resource estimation. Its scope should be detailed enough to provide an understanding of the complexity of the proposed design, to provide confidence that the costs and schedules are credible, and to assess the staff planning and availability to complete the total job.

This section should introduce and orient the reader to the overall proposal and its highlights by providing brief statements about each main part of the SDD.

b. Item 1.2, General Program Description. The description contained in the SRD (Section 1.2 of the SRD) is referenced and expanded in functional and operational definition based on the information from the architectural design activities. A system-level "siting" diagram (Figure 3-3) can be included which locates the total program in the system. This type of diagram readily displays interactions with the user, the operating system, required peripherals, and other software. An operational state diagram (Figure 3-4) can be included to display the general overall operational capability and states available.

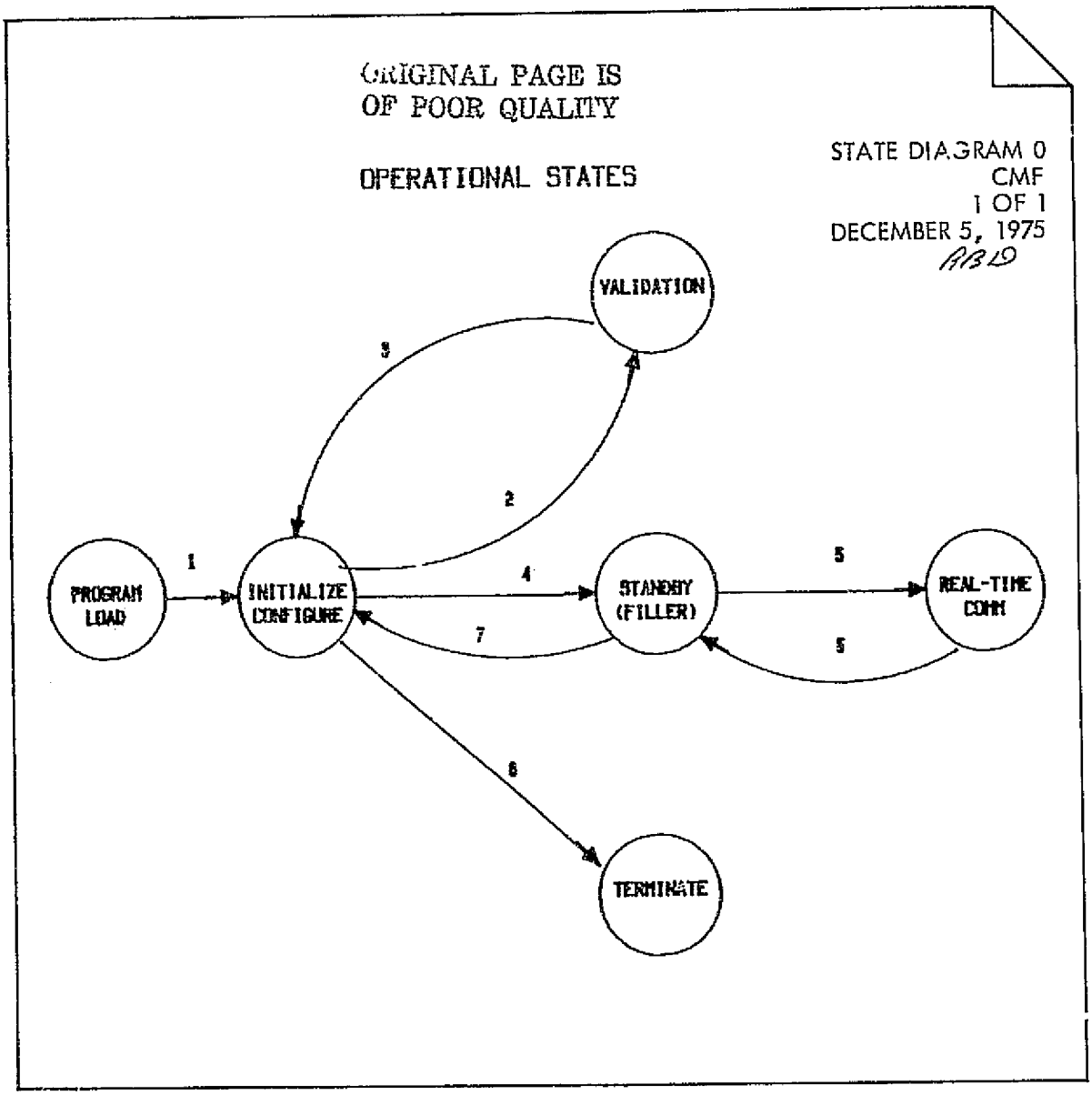
Referring to these diagrams, brief statements about the program's major functions, flow of information, and overall operation are provided to cover the overall program characteristics for preparing the reader for the later, more detailed sections of the SDD. Reference to these later sections can be made at this point.

c. Item 1.3, Architecture Overview. In conjunction with the system-level "siting" diagram, a high-level module count provides an overview of the proposed program. The description and documentation of the architectural design, as discussed in Sections III and IV, should be adequate to show that the SRD requirements are being met. These should be available for inspection.



(CAN BE HAND-DRAWN FOR SDD; CAN BE HAND-PRINTED IF CLEAR AND LEGIBLE)

Figure 3-3. Example of System-Level "Siting" Diagram



(ACCOMPANYING NARRATIVE KEYED TO THE NUMBERED TRANSITIONS DEFINES EVENTS, CONDITIONS, AND ACTIONS WHICH CAUSE CHANGES IN THE PROGRAM OPERATIONAL STATE)

Figure 3-4. Typical Operational State Diagram

Program details are discussed later (in Item 4, Program Architecture) but key characteristics of the program architecture and high-level module tree are identified here.

2. Management Information

a. Item 2.1, Project Schedules and Reporting Plan. Project schedules are set up and defined as discussed in Section III. They should be responsive and compatible with the need dates of the SRD and detailed enough to allow meaningful reporting. If the need dates are not being met, they are addressed in the architectural design review for possible renegotiation. The reporting plan for monitoring the schedules should provide satisfactory progress status monitors for the follow-on activities involved in the detailed design and program production.

b. Item 2.2, Work Breakdown Structure. Based on the architectural design and a high-level module tree, a preliminary high-level Work Breakdown Structure of tasks is defined for the entire implementation, including any special supporting software to be produced and the writing of the operator's manual. The defined tasks should be small enough to facilitate supervision and to allow review of progress relative to plans. The sample format contained in Section V (see Figure 3-11) is suitable for SDD presentation (backed by the Task Description sheets (see Figure 3-12), which are not necessarily presented in the SDD).

c. Item 2.3, Personnel/Team Level of Support. Based on the above schedules and tasks identified through program completion, arrangements for needed support personnel are completed, and the planned work phasing and corresponding implementing team structure are included in the SDD (without specifying names). A format for budgeting project time and resources, which results in an indication of levels of support needed to meet the established schedules, is presented in Section V (see Figure 3-13).

d. Item 2.4, Cost-to-Completion Estimate. The cost-to-completion estimate is presented, based on the architectural design and on the preliminary work breakdown and schedules, and has a goal of accuracy to within 10 percent.

The cost estimates in the SRD for the completion of the architectural design and also for the total implementation are updated for use in the SDD.

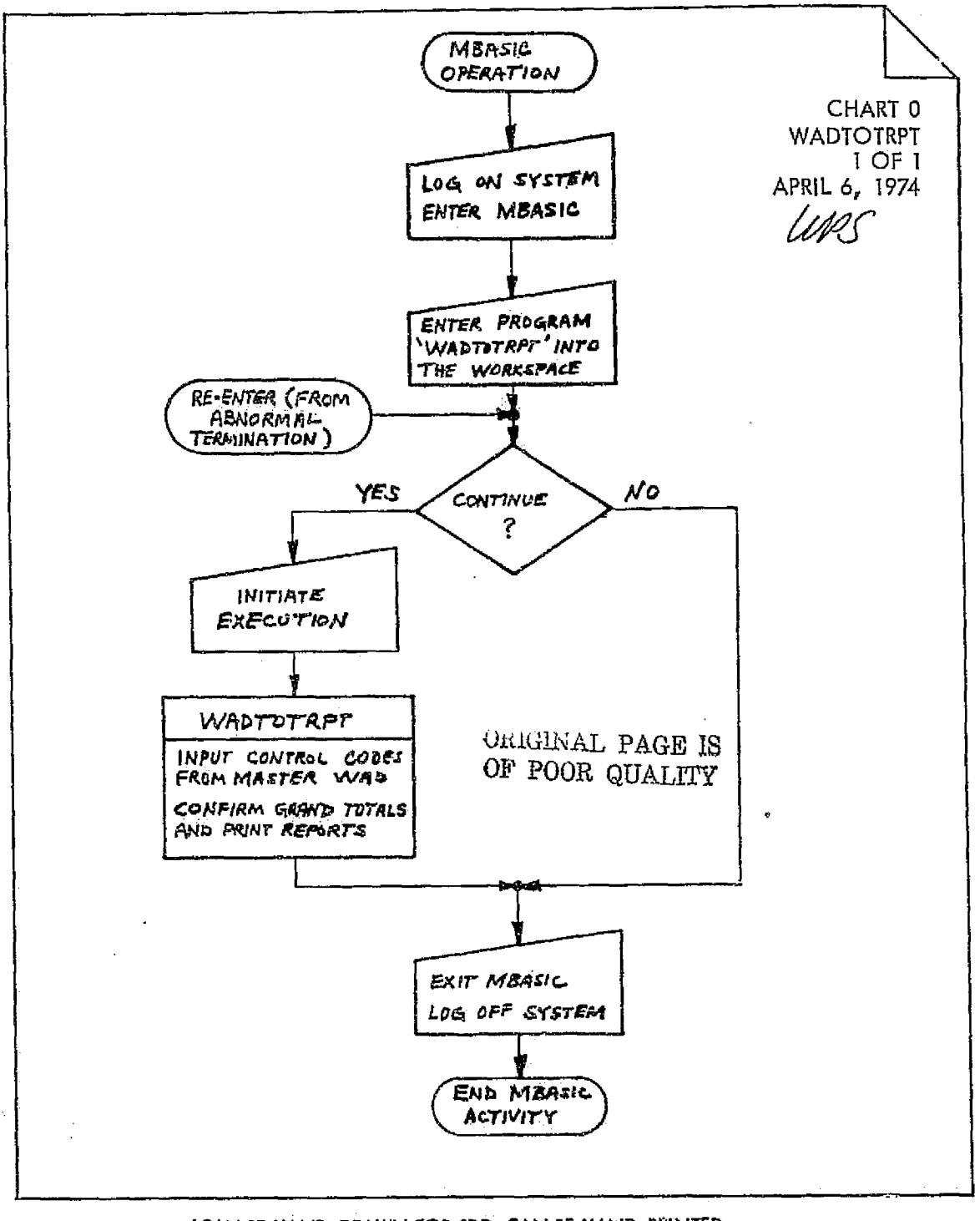
e. Item 2.5, Competing Characteristics and Problem Areas. As a result of the architectural design activity, any concern for meeting the competing characteristics identified in the SRD is discussed. Any requirements not being met must be identified, along with recommendations for their resolution. Any new competing characteristics or complexity which may affect the implementation costs and/or schedules should also be identified. Also, ways of coping with this complexity should be discussed.

f. Item 2.6, Design Reviews. Required design reviews are identified in Chapter 1. Calendar dates for these reviews and any additional required or expected reviews are scheduled and included in the SDD. SDD review and approval are discussed in Section III. Criteria for the software architectural design review are also provided in Section III.

3. System Environment

a. Item 3.1, Computer, Peripheral Equipment, and Subsystems. The SRD descriptions serve as a basis for adding new information available from the architectural design activities. Hardware interfaces, if known or if needing subsequent design, are identified for the computer and peripherals as well as for any associated subsystems. The high-level system diagram discussed in Item 1.3 (Figure 3-3) can be used to illustrate the system environment. Data storage characteristics are defined based on the SRD requirements, and, if known, actual hardware units can be identified early (e.g., for disk, drum, mag tape, etc.) to establish external interfacing equipment. Deviations from DSN standard peripherals, interfaces, etc., should be identified and justified, perhaps by reference.

b. Item 3.2, Program Operation in System. The overall operating aspects and general description of the program can be understood and displayed conveniently with a system-level "Operating" chart. This shows the total program as a "striped" module (Figure 3-5) embedded in its operating system environment.



(CAN BE HAND-DRAWN FOR SDD; CAN BE HAND-PRINTED IF CLEAR AND LEGIBLE)

Figure 3-5. Example of System-Level "Operating" Chart

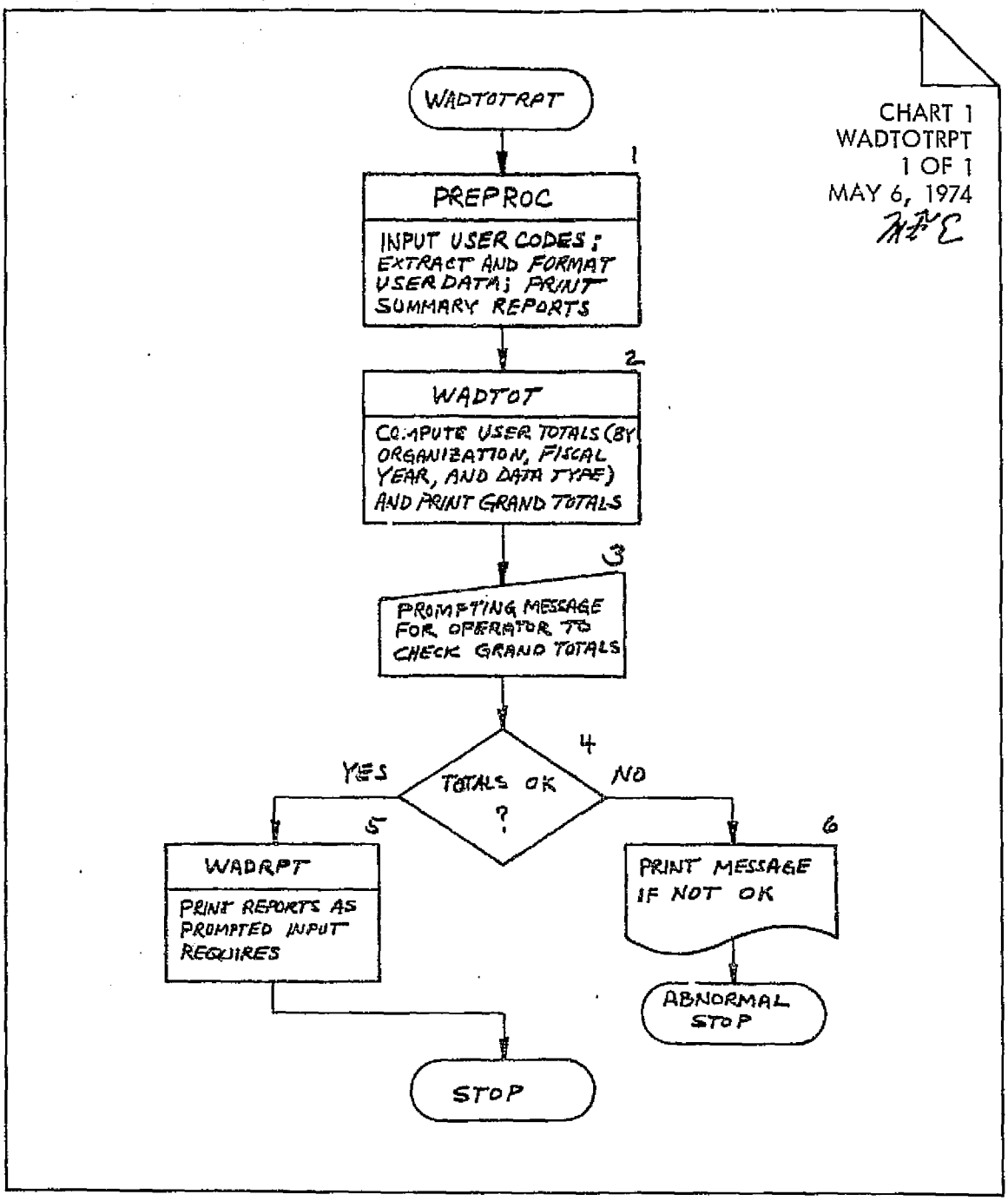
The main operating characteristics can thereby be identified and defined in sufficient detail to understand the nominal system operations and, when appropriate, the nonstandard configuration operations. Any cost or schedule concerns can be identified.

c. Item 3.3, Software Interfaces and Services. The software external interfaces are identified and defined. The SDD should indicate the degree to which these interfaces are constraints or can be design prerogatives. Common services (utility routines, library functions) and common program functions known from the architectural analysis but yet to be implemented are identified and defined in high-level detail. "Common" in this context refers to functions and services used by many programs, and does not refer to subroutines occurring within a single program. This section also identifies and defines any special software needed for program implementation support, since these costs must be included in the SDD estimates.

4. Program Architecture

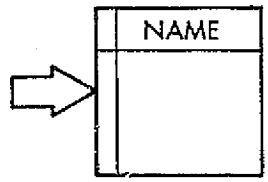
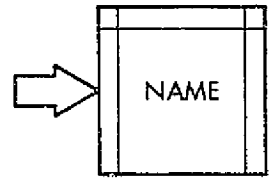
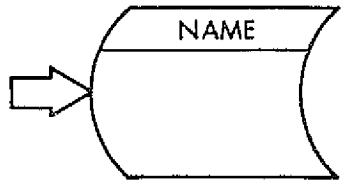
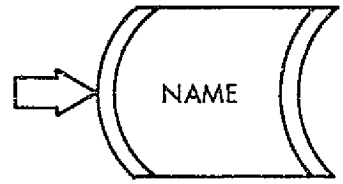
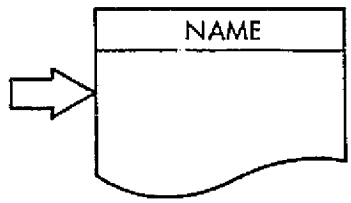
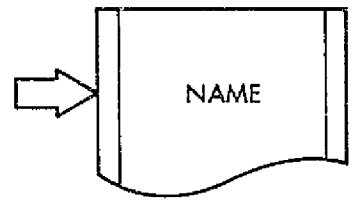
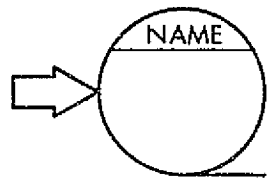
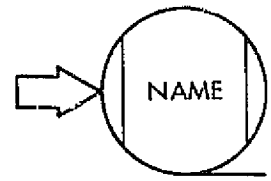
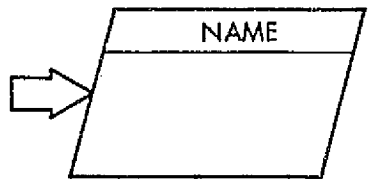
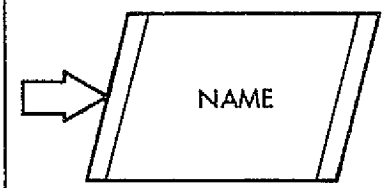
a. Item 4.1, Preliminary Functional Analysis. The analysis and design process is briefly described in Sections III and IV to provide background on this activity. For purposes of the SDD, the thrust of this activity centers on generating a preliminary functional analysis and program architecture that, by inspection, is responsive to (will or will not satisfy) the SRD requirements so as to present a credible estimate of implementation costs and schedules. The results of this activity are described in the SDD, where they form a preliminary set of definitions and descriptions that can be assessed relative to the SRD requirements. Modification of the SRD requirements, by renegotiation, may be necessary. Section IV provides helpful information for effective presentation of results. Various diagrams and charts are described as well as "Striping Conventions," illustrated in Figure 3-6 and described in Figure 3-7, which are to be used for presenting hierarchical nested levels of detail.

b. Item 4.2, Data Characteristics. External data characteristics such as source, availability, accuracy, medium, amount or flowrate, destination, etc., are identified and defined for cost estimation and for scheduling the later



(CAN BE HAND-DRAWN FOR SDD; CAN BE HAND-PRINTED IF CLEAR AND LEGIBLE)

Figure 3-6. Sample Flowchart for Level 1 (Overall Program Flow)

STRUCTURE TYPE	EXPANDED ELSEWARE IN THIS DOCUMENT	NOT EXPANDED HERE, BUT DOCUMENTED ELSEWHERE (AND REFERENCED)
IN CORE		
ONLINE STORAGE		
DOCUMENT		
MAGNETIC TAPE		
ARBITRARY MEDIUM		

(SYMBOLS WITHOUT STRIPING CONVENTIONS ARE NOT EXPANDED ELSEWHERE, EXCEPT IN CODE)

Figure 3-7. Striping Conventions for Information, Data, and Storage Structures

detailed design of data items, input and output formats, and interfaces. Emphasis is on the architectural characteristics of the data and flow interfaces, especially as seen by Operations, and not on coding details. Detail provided should allow the end user to assess the user interactions and concerns on input and output formats, etc. Figures 3-8 and 3-9 show a typical data flowchart and a data flow diagram that can be used for presenting this kind of information in the SDD.

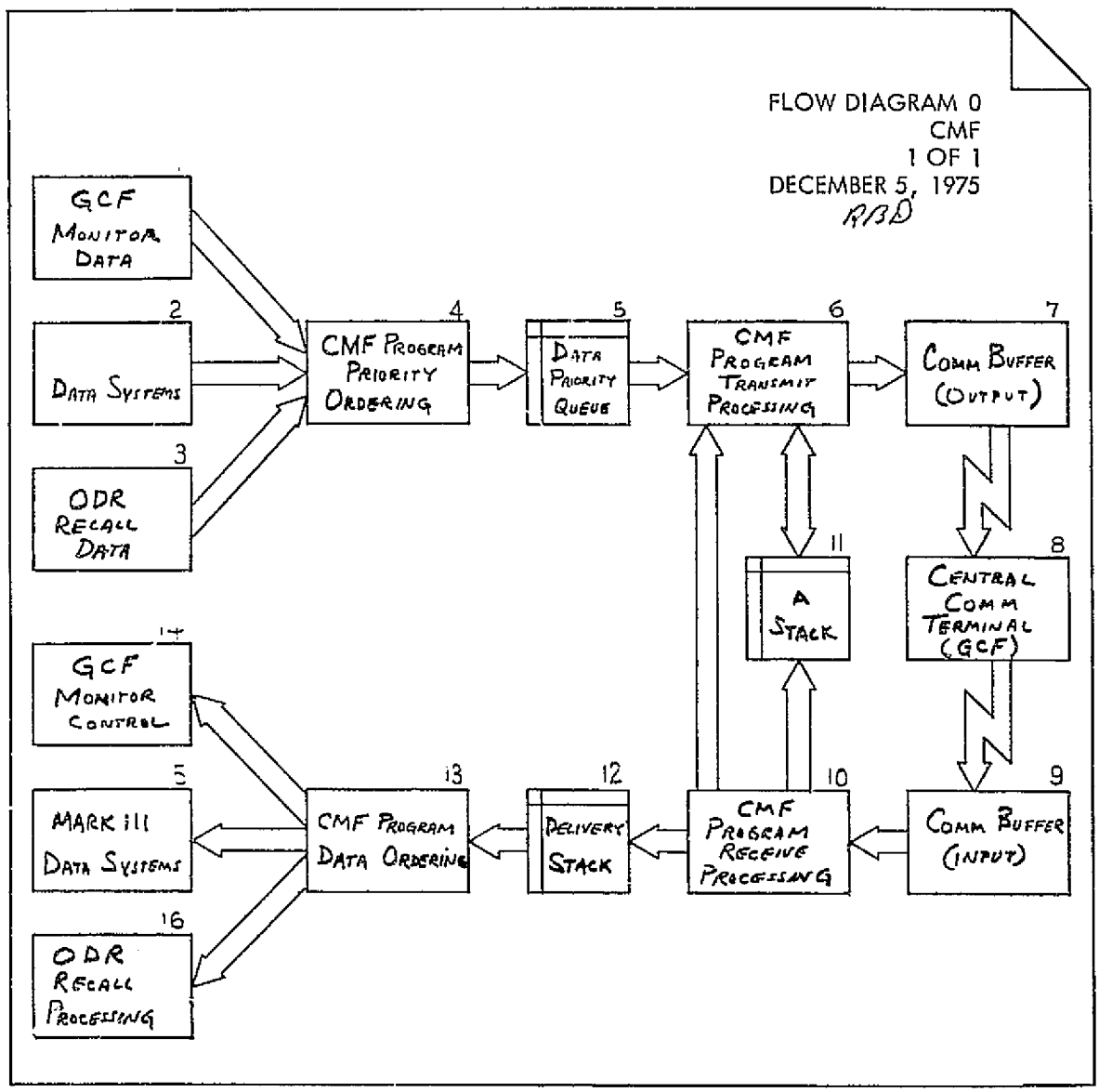
c. Item 4.3, Preliminary Program Structure and Definition. An overview of processing and data structuring is presented to define the main processing algorithms, together with the upper-level procedural designs in the program hierarchy. This part of the SDD is a summary of the complete program design as determined by the architectural design study. The projected figures for number of flowcharts (or equivalent program design mediums), pages of narrative, lines of code, core (or other) storage occupancy, and all other deliverables should be presented. This architecture forms the basis for the resource estimation and scheduling of the later detailed design and for identifying needed status monitors during that later design phase.

5. Coding and Test Design Criteria

a. Item 5.1, Coding Criteria and Constraints. Criteria for coding during the detailed design phase are presented. This section should present criteria or guidelines for aggregating modules into builds as the design units are being completed, module by module. Also, if several coders are being used, this section should identify the criteria by which the modules coded by each are merged together into a "master copy" for code audit and configuration management purposes. Coding constraints such as programming language, standards for submodule linkage, etc., should be identified if pertinent or if they affect the architecture or costs and are visible at this high level of program description.

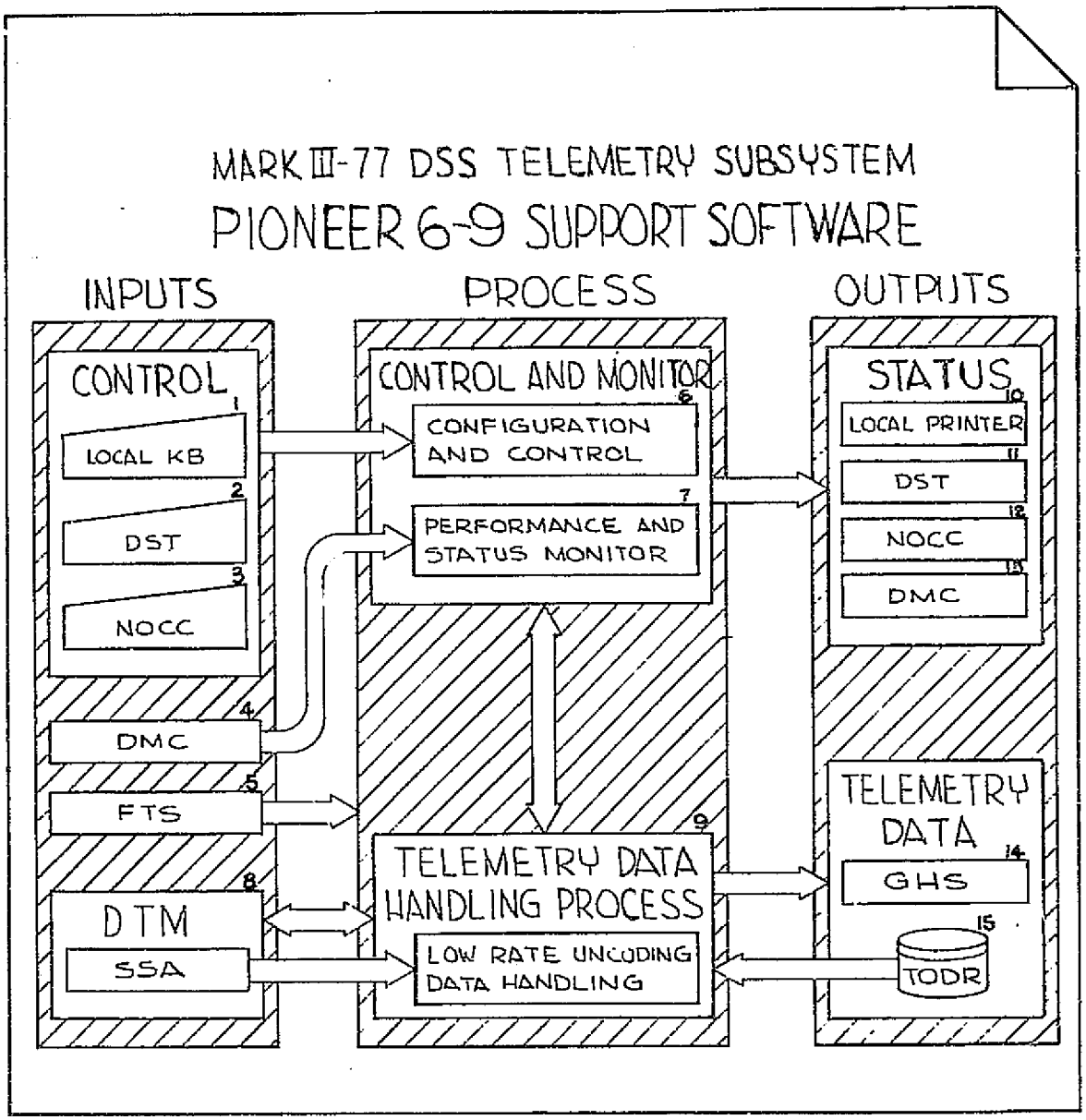
b. Item 5.2, Correctness Testing Criteria. Criteria for verifying the correctness during the program construction process are presented. This section should also present criteria or guidelines for aggregating modules into builds for correctness test purposes. For example, one criterion could be that

ORIGINAL PAGE IS
OF POOR QUALITY



(ACCOMPANYING NARRATIVE PROVIDES DETAIL ON THE TYPE OF DATA FLOWING AND THE INTERACTIONS AMONG PROCESSES. CAN BE HAND-DRAWN FOR SDD; CAN BE HAND-PRINTED IF CLEAR AND LEGIBLE. NARRATIVE IS KEYED TO THE NUMBERS)

Figure 3-8. Typical DSN Data Flow Diagram



(WHEN USED, NARRATIVE FOR THESE DIAGRAMS IS REQUIRED; DATA RATES AND ACCURACIES ARE PROVIDED FOR FLOW)

Figure 3-9. Typical DSN HIPO Diagram

the builds should be as large as possible consistent with the test designer being able to design a correctness test for the build without automatic aids. More specific criteria which depend on program architecture could also be presented if they will influence design or test costs. The requirements and characteristics of any special software needed for correctness testing and criteria or guidelines for its use should be included also.

c. Item 5.2.1, Verification Testing. On large system programs, criteria for pre-acceptance verification testing are also needed to establish and validate acceptance test plans and procedures since they can have a significant effect on the final testing costs and schedules.

d. Item 5.3, Acceptance Testing Criteria. Criteria for acceptance of the completed program are presented for estimating the costs and scheduling of acceptance activities. Also, criteria for verifying the acceptance test procedures (such as normal operations, error seeding, calibration, etc.) are presented. These criteria form the basis of the acceptance plan to be developed later and presented in the Software Test and Transfer Document. Basically, the acceptance centers about an end-to-end demonstration of the program's capabilities to meet the subsystem and system requirements identified in the SRD. This includes an expansion of the program performance requirements, identified in the SRD, into an ordered set of performance criteria for assessment and acceptance. Also, criteria for assessing the program's responsiveness to user inputs in the operating system environment and responsiveness to the needs of operating personnel and end users of the program output should be established.

SECTION III
SDD PREPARATION, REVIEW, AND APPROVAL

A. PREPARATION ACTIVITIES

The CDE, as assigned by the implementing organization, prepares the SSD in response to the approved SRD. Management approval to continue the software effort (proceed to the detailed design and production) is then based on a review of the cost and schedule information contained in the SDD. The SDD content (type of information and its organization and format) is outlined and described in Section II, where the outline guides the preparer to the specific items and points of information upon which the management approval decision is based. Further helpful information for preparing the SDD is summarized below and covers the activities involved in preparing the architectural design and the program production plan. Section IV provides additional aids for preparing and documenting SDDs. References 1 and 5 provide additional detailed information that can be helpful in making resource estimates on the basic design process involved in the resolution of the requirements (the problem) into an implemented solution.

1. Architectural Design

Based on the top-level requirements identified in the SRD (what is needed, why, when, and estimated cost), the problem is defined in terms of major characteristics and functions of a proposed solution (computer program). This effort, therefore, is directed toward defining the problem and determining what the program should do functionally, that is, as expressed in terms of program functions and high-level program control and information flow. The problem definition must be detailed enough to evaluate the environmental constraints of the SRD (for cost and schedule implications) relative to the problem's needs, i.e., adequacy of core storage, ability to meet time constraints, and ability to perform all major functions. Major functions are identified only to understand what must be done, and they may be connected by information flow paths or control, but they are not yet designed – that is, they are not detailed to the level

that they could be implemented. Emphasis is on providing credible costs and schedules by achieving a high level of understanding of the problem at hand and the proposed architectural or skeletal solution, and not, at this point, on a detailed design solution or a detailed correctness assessment and certainly not yet on coding considerations. Forming the architectural solution involves informal but hierarchical thought processes and expansions among the problem, prospective solutions, and related tradeoffs. The hierarchical expansions converge toward choices of the program high-level definition and then toward the architectural design itself.

The results of the above functional analysis form the basis upon which following work is cost estimated, scheduled, and may be expected to conform. Modular functional units, corresponding to the main functions, are defined, together with their connections which control the flow of data from the input (external) interfaces through the functions to the output (external) interfaces. High-level program testing criteria and test goals (for correctness and acceptance) for meeting the SRD requirements are defined. The next-lower-level functions are then defined and the cycle repeats. The combined functional program framework, high-level test criteria, and hierarchic expansions are referred to as the architectural design. Modular units are used to identify and size the task, not to produce an immediately codable design for which correctness can be assessed. This indicates that hand-drawn control and data flowcharts (or equivalent) with little or no corresponding narrative, data structure tables, etc., are a proper medium for the architectural design. Full details on program logic, procedures, sequences, and detailed data structure and characteristics are typically not addressed unless required to size the program and work tasks within 10-percent accuracy. The architectural design, therefore, facilitates the understanding of exactly what is involved, and provides the architecture upon which the remaining program final design and construction can be based. No coding is performed during the architectural design to assess detailed correctness of the program. The architectural design encompasses the follow-on SSD activities that begin after SDD approval, that is, top-down program implementation and construction using concurrent design and documentation, coding, and testing. The architectural design can perhaps be updated and redlined to produce the input for the final finished graphics of the

SSD. Since the major modules or main groups of modules are typically identified by the completion of the architectural design, an estimate of the work involved can be projected for each based on their relative complexity. Because each flowchart is on one 8-1/2 x 11-in. (22 x 28-cm) page, this estimate should not be especially difficult to make.

2. Program Production Plan

Based on the program and test design definitions resulting from the architectural design activities, the remaining tasks can be identified and outlined for purposes of scheduling, determining personnel/team support levels, and estimating the cost-to-completion.

a. Remaining Tasks. The work remaining encompasses the broad activities of program construction through completion, including the as-built Software Specification Document; preparation of the Software Operator's Manual; program acceptance, including the Software Test and Transfer Document; and providing support for all required design reviews (high-level, acceptance readiness, and transfer) and other optional reviews, as determined by cognizant management.

Following SDD approval, program construction comprises the tasks of top-down formal expansion: design and documentation, coding, testing, and test verification.

Preparation of the operator's manual for the program occurs concurrently with the other program construction and acceptance activities. A preliminary operator's manual, that is sufficient to operate the initial builds of the program, is produced after SDD approval.

Program acceptance activities parallel the program implementation and include detailed acceptance test planning, acceptance test procedures generation, demonstration of adequacy of acceptance test plans and procedures, and specification of detailed test data and required test results for acceptance.

Design reviews and required support are discussed in Paragraph B of this section.

b. Scheduling. The remaining tasks, as outlined above, must be allocated sufficient time (including nonproductive queuing delays, iterations, etc.) and coordinated with the milestones and project need dates as presented in the SRD. Schedules for the major milestones and the near-term work are produced by phasing the work breakdown tasks for top-down development of activities. The emphasis is on accomplishing two items:

(1) Refining the overall schedule to completion, based on the preliminary detail that is available from the architectural design and work breakdown activity.

(2) Detailed scheduling of the completion of an initial portion of material identified by the WBS which is to begin the construction phase. After the SDD is approved, the CDE develops and maintains detailed later work breakdown structures and schedules, concurrently with the program construction.

c. Personnel/Team Support Level. Using the information available that outlines and identifies the remaining tasks and the constraining schedules within which they must be completed, the CDE and cognizant management can make personnel/team assignments to match the expertise available to that needed for the tasks and to meet the work loads created by the schedules.

Chapter 1 provides information on the overall support responsibilities throughout the implementation as well as a brief discussion of the separate functions that comprise the software implementation team and the team's operational interactions in performing the software implementation tasks.

d. Cost-to-Completion. The planned schedules for completing the remaining tasks with staffing by personnel matched to the tasks in both number and expertise results in a personnel loading and phasing profile. This profile, along with estimates of supplemental resources such as computer time and

support services, can be used to estimate the cost to complete the implementation. Uncertainties should be identified for determining possible variations and confidence of the estimates. An accuracy to within 10 percent is the goal.

B. REVIEW AND APPROVAL

1. SDD Approval

Approval of the SDD signifies concurrence with its cost estimates and schedules and concurrence with the expenditure of resources to continue the implementation. The SDD is first approved by the implementing organization's line management prior to proceeding to the next phase of activities; this begins the approval process that includes a design review. Implementation activity continues on through this approval period which typically may span a week or two and normally never more than one month.

SDD approval by the implementing organization is always required; other approvals or any special guidelines or procedures for proceeding should be identified in Section 2.2 (Special Procedures) of the SRD and appropriately addressed in the SDD design review.

2. SDD Review

The main objective of the SDD review (also called the architectural design review) is to assure that the overall management, subsystem, and program definitions are stated in sufficient detail and coverage to provide credible evidence of a 10 percent accuracy in resource estimates, including the definition and refinement of the implementation schedules, costs, and performance. (See Section I, Paragraph B, Scope of Software Definition Documents.)

Emphasis is placed on having low-cost reviews. Use of existing project information is encouraged whenever possible, as opposed to producing formal

"Review Presentation" materials, which tend to be high-cost, low-use items. Whenever practical, software reviews should be combined with system, subsystem, or hardware reviews. Also, several software reviews can be scheduled into one review. These review practices tend to consolidate and conserve resources. The concurrent documentation approach of the DSN software methodology makes this relatively easy.

The CDE (or the CDE's management) sets the time and place of the review and issues the review meeting notices and review material (at least one week prior to the review). The review of the SDD can be held as part of a more general review of subsystem requirements.

The CDE (or the CDE's management) conducts the review. The personnel participating in the review should represent, at a minimum, the initiator, the using organization, the funding (if different from the using) organization, and the implementing organization. The CDE (or the CDE's management) appoints the board.

a. Agenda. The CDE also prepares the review agenda, which typically includes:

(1) A brief summary of the SRD requirements to understand the problem being addressed.
(2) A brief description of the program functions and architectural characteristics.
(3) Identification of program functions as related to SRD requirements.
(4) Schedules versus need dates.
(5) Costs versus budget.
(6) Identification of concerns. For example, any inability to meet requirements and/or competing characteristics, the unavailability of needed resources, unrealistic schedules, inadequate funding for projected costs, etc., should be made visible at this time.

b. Criteria. The following criteria for the software architectural design review can be used as aids for assuring completeness of detail and coverage:

(1) Does the program architecture satisfactorily address all SRD requirements? Is requirement modification by renegotiation needed?

(2) Is the architectural design detailed enough to identify major tasks, including the number of modules involved and a detailed work breakdown for each task?

(3) Is the architectural description and documentation adequate and available for inspection?

(4) Have satisfactory progress status monitors during the final detailed design phase been provided as part of the architectural task?

(5) Did the architectural design activity necessitate backup coding? If so, was the minimum level exceeded?

(6) Is the architectural design and documentation adequate for the later detailed design and implementation?

(7) Is the work breakdown sufficiently detailed to verify cost and schedule estimates within the goal of ±10%?

(8) Does the work breakdown structure (and schedule) have tasks which are small enough to facilitate supervision and review by management to determine progress relative to plan?

(9) Have the implementation testing criteria, plans, and procedures been adequately defined as part of the architectural task and included in the SDD?

(10) Does the architectural definition of the program provide sufficient information for the end user to assess the appropriateness of user interactions and input-output formats?

(11) How close was the SRD cost estimate and schedule for the architectural phase to the actual architecture cost and schedule? Similarly, how close was the initial SRD estimate of the cost and schedule of the final design and construction to the refined estimation in the SDD?

SECTION IV
PREPARATION AND DOCUMENTATION AIDS

A. PREPARATION

1. Functional Analysis

The SDD analysis and definition activities provide the basis and justification for the resource estimates and involve problem definition and program structuring which generally are based on:

(1) Dividing an inclusive but relatively nondetailed description of the problem into a manageably small number of pieces with relatively simple and well-defined interfaces.
(2) Expanding the degree of detail of description of each piece by repeating the division step described above until each piece or subpiece of the original is described in the level of detail desired.
(3) Displaying the pieces of this step-by-step (stepwise) expansion of detail to provide a complete and functional description of the problem.

Specific procedures for performing these thought processes have not yet been formalized. However, the top-down approach should be applied to these activities, not only for defining specific modules but also for defining, concurrently, the related aspects of data, function, and procedure. The data aspect deals with the information and data in the problem and in the program, including input, intermediate, and output data in the problem structures as seen by the world external to the program. The function aspect deals with the processes or transformations acting on the data, that is, what is done to transform one data object into another in the passage through the program. The procedure aspect deals with the control logic to select paths through the program, and the sequence of operations (functions and their component instructions) on these paths, that is, how the program performs its task.

The top-down approach leads to a hierarchical, tree-structured expansion of program detail. At lower, more detailed levels, however, several expansion paths may lead to very similar or identical functional elements. To avoid unnecessary duplication of implementation effort, and in the interests of promoting commonality and economy in subsequent design and maintenance, it is prudent that the program definition encourage, identify, and accommodate such common software. These instances are recognized by forming a candidate high-level definition, and proceeding through the architectural design. Some iteration will usually be necessary to maximize this commonality in the architectural design. For major implementations involving multiple SRDs and SDDs, this iteration typically takes place outside any one implementation team.

2. Preliminary Functional Descriptions

Functional descriptions and displays of the results of the above functional analysis are produced that are generally adequate, even in preliminary form, for understanding and assessing the complexity of the problem and the proposed architectural solution for estimating the costs and schedules. Following SDD approval, these descriptions can be expanded and updated concurrently with the program construction to form the as-built program specifications and may be included in the SSD.

As an overview, the program's location or site within the system can be effectively displayed using a system-level "siting" diagram as shown in Figure 3-3. The data and function aspects described above can readily be displayed and described.

Operational state diagrams can also be useful in displaying and describing the results of the functional analysis activity. These are simple "bubble" charts indicating various essential states or operating modes (to be discussed below) that are required in any acceptable proposed program. Figure 3-4 presents a typical operational state diagram using computer-based graphics. Hand drawings are acceptable in an SDD. Descriptive narrative keyed to the numbered paths, emphasizes operator actions and other program input needed to effect a change in state (to move from one bubble to another).

Still addressing the operational aspects, a top-level "operating" chart can be used to display the total program as a single striped module imbedded in its operating system environment, as shown in the hand-drawn sample of Figure 3-5 (taken from a pilot project – Work Authorization Document Report Writer Program). This locates the program in an operating perspective to the system and provides a good beginning point for the top-down definition and description of the program on a modular basis.

Major program modules can then be identified and defined by providing detail from the functional analysis to expand the top-level charts. These form the highest levels of the program architecture, which the remaining architectural design expands. (See Figure 3-6 for a sample expansion.) The expansion of functional, data, and procedure aspects of the design proceed concurrently as discussed in detail in Reference 1. It is noted that these expanded designs are prepared in rough-and-ready hand-drawn working sheets that are readable by others to provide a basis for sizing the total job, for determining schedules and needed resources, and for estimating the remaining costs. Detailed design refinements and correctness assurance are addressed later, and final graphics are provided for the SSD.

The above system-level "siting" diagrams, operational state diagrams, and architectural design flowcharts aid in the communication and preliminary description of the ongoing, evolving design.

B. DOCUMENTATION AND GRAPHIC AIDS

Good, readable documentation consisting of both narrative and graphics is essential for a high level of communication and interaction among involved personnel (e.g., initiators, implementers, users, reviewers). The documentation, consisting of words, symbols, diagrams, and charts, must be clear, unambiguous, and timely – yet cost effective. Documentation is written only once, but read many times; therefore, emphasis is to be placed on readability.

Formatting of the overall SDD is discussed below to aid in the documentation preparation. This is followed by a brief discussion of techniques and formats that can be applied to supporting elements of the SDD such as tables and figures.

Standardized flowcharting symbols and terminology have been adopted from Reference 1. These are applied as relevant for the more informal hand-drawn architecture flowcharts which may appear in the SDD.

1. SDD Format Conventions

The SDD is a nonsurviving informal information document that provides credible evidence of a 10-percent accuracy in resource estimation, so emphasis is placed on its information content rather than on its formality of style and formatting. However, consistency from document to document aids readability. Therefore, the CDE is directed to Chapter 4 and is encouraged to use the same formatting conventions as adopted for the surviving formal documents, i.e., the SSD, SOM, and STT.

Basically, these documents are "block-formatted" and "sectionalized," where each section is identified by consecutive arabic numbers. To facilitate change releases, the pages, figures, and page-size tables should be numbered consecutively within each section (e.g., 1-1, 1-2, . . . 1-n; 2-1, 2-2, . . . 2-n; etc.). If existing automatic equipment is not programmed for this pagination scheme, any other scheme that allows easy change updating is acceptable for SDDs. Small tables can be inserted within the running text and are not necessarily numbered. However, if other parts of the report refer to it, a table should be numbered and placed at the top or bottom of its page.

Subsections use decimal identification. Appendices are typically identified alphanumerically and need no blank or separate introductory page. Formatting within the appendices can vary to allow ease of incorporating existing material without modification.

For further format detail and specific information on typewriter tab settings, line spacing, capitalization, and underlining, the preparer (and typist) is referred to Chapter 4.

2. Information Flow Graphics (from Reference 1)

a. Symbol Striping Conventions. Information and data flow graphics utilizing charts and diagrams can be useful in identifying the source of required input, the required routing through processes, and the destination of required output. For example, inputs may emanate from existing data base files, or from information to be supplied by a group of users or operators. Outputs may be destined to files, or perhaps, to users by mail. Conventions which standardize the horizontal and vertical striping of th diagram symbols to indicate that additional definitions are detailed elsewhere are shown in Figure 3-7.

b. Sample of a Typical Data Flowchart. Figure 3-8 presents a typical application of information flow graphics to diagram the data flow through a DSN subsystem software program. When used, an accompanying narrative would explain the routing and rationale.

c. HIPO Diagrams (see Reference 6). Hierarchy Plus Input-Process-Output (HIPO) diagrams are a specific form of the general information flow diagram applied to data flow and functions or processes of a system. Each diagram contains three major sections:

(1) Input – the data items used by the processing steps and connected to them by arrows.
(2) Process – a series of numbered descriptions of a given function. These are connected by arrows to the input needed to perform the function and the output created by the function.
(3) Output – the data items created or modified by the processing steps and connected to them by arrows.

An initial overview high-level diagram might provide a general description of main functions, along with major input and output data items. However,

processing detail may typically be better described in hierarchical, architectural, procedurally-oriented diagrams, so HIPO may not be as useful in those lower levels, and typically only one or two HIPO charts might appear in most SDDs. Figure 3-9 is a useful HIPO-style diagram that gives the data flow overview in an SDD. Again, this may be hand-drawn in the SDD.

Structured programming facilitates information flow diagrams because the functions can be considered as single entities. The functions are designed in "segments" or "blocks," each with a single control entry and single control exit. These segments can then be detailed functionally using HIPO-type diagrams for each. The Standard Practice limits individual information and control flowcharts and any other diagrammatic documentation for software to one page-length. This limits the level of detail that can (and should) be presented in an SDD, since only a few of these diagrams would typically appear in an SDD. Each form of diagrammatic documentation should be included in the SDD when it best expresses the architectural overview of the program. As other team members are added to construct modules after SDD approval, all diagrams which aid their work in various paths of the detailed design hierarchy would then, of course, be used and included in the SSD as it is built up.

3. Mode Diagrams

A mode is defined (Reference 1) as a way of operating a program to perform a certain subset of the processing requirements that normally are associated together in the program function. Program operating modes can be displayed in diagrams showing the inputs, processing, and outputs (similar to the HIPO display of functional flow), where the processing is selected to be only a partial set of the full capabilities of the program. Figure 3-10 presents an illustrated general mode diagram at a level of detail suitable for inclusion in an SDD.

Further discussion concerning mode diagrams and associated decision table construction and use can be found in Reference 1. This material may be very relevant in preparing SDDs for some types of programs.

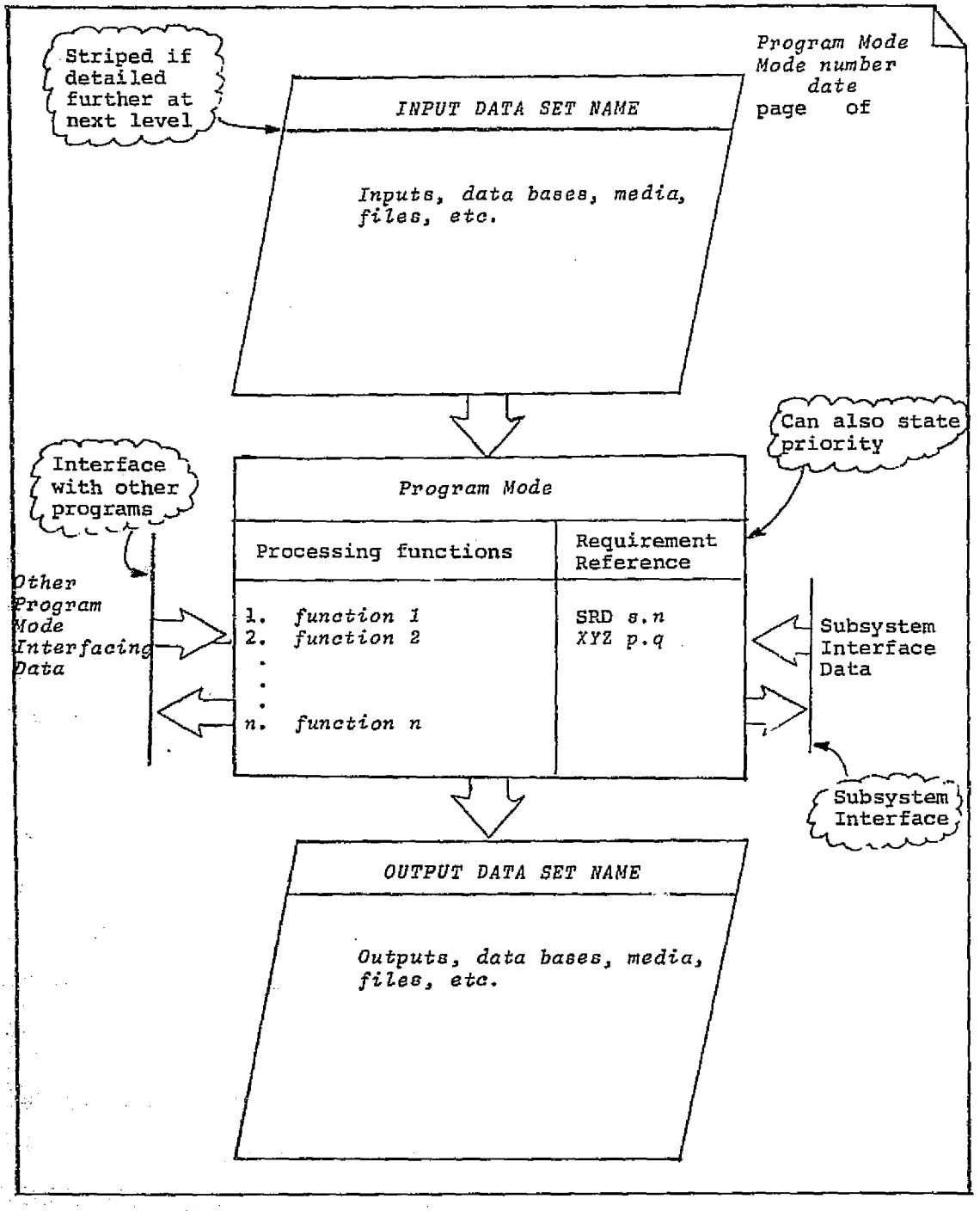


Figure 3-10. An Illustrated Mode Diagram

SECTION V
IMPLEMENTATION MANAGEMENT AIDS

Several aids are identified for use in managing software implementations, along with brief comments on their application. The aids include work breakdown, task description, task budgeting, and project status reporting techniques. These aids come into use mainly after SDD approval, and the SDD must contain not only the architecture of the program but also the management plan that allows for their use in the follow-on design, implementation, and status monitoring.

A. WORK BREAKDOWN, DESCRIPTION, AND BUDGETING

Based on the defined program architecture and module tree, a Work Breakdown Structure is generated, which defines work packages (elements) on a family tree basis. These are finite, in fact short, manageable tasks with quantifiable inputs, outputs, schedules and assigned responsibilities. The WBS tree is further supported by Detailed Task Descriptions, which succinctly describe the work to be accomplished, identify the responsible task leader, specify inputs required by date, and schedule outputs. This information is then available for project budgeting of time and resources down to the individual task level. A first-cut WBS would typically be prepared to aid in production of the cost and schedule estimate of the SDD. This rough cut typically need not be part of the SDD, however.

1. Work Breakdown Structure Elements

The CDE uses the WBS to arrange work into finite, manageable tasks which are components or elements of higher-level tasks or projects. For generating the WBS, the CDE uses the following criteria as a guide:

(a) Each subtask should be significant in terms of level of effort and time required to be accomplished, but not so long as to deprive management of visibility.

(b) It should have a clearly defined start and finish, with a finite end product (software routine, document, demonstration, etc.).

(c) Duration of all subtasks should be approximately the same order of magnitude.

(d) Each task should be characterized in terms of work elements, resources, and schedule.

(e) Each software task should be relatable (either in interface terms or interchangeability) to other tasks at the same level, and be explained relative to the module tree produced in the architectural design.

An example of a partial WBS is shown in Figure 3-11 for the DSS Telemetry Subsystem Software, used to produce the program after SDD approval.

2. Detailed Task Descriptions

Each major task identified in the WBS can be described in more detail. During the SDD phase, the detail should be sufficient to estimate cost and schedule within 10 percent. A detailed task description should be filled out for each task requiring the duration cited in item (c) of the above listing. Having the architectural detail (probably flowcharts) facilitates the sizing of tasks related to detailed module design. The sample format shown in Figure 3-12 facilitates the recording of this detail, which would include the following during the SSD phase:

(a) Date – the date on which the description form is completed.

(b) Rev. – the revision number of the task description, starting with 0 for the initial issue and progressing sequentially.

(c) WBS Task Title – using the top level WBS as a reference, include second level expansion, and narrative title.

(d) Module Tree Reference – refer to module tree for tasks involving module implementation.

Software

Detailed Task Description

Date_____ Rev_____

WBS/Task Title_____

Task Mgr_____ Duration__/__/____ to__/__/____

Task Description:

Task Schedule:

Task Budget:

Task Scope: Code_____ Flowcharts_____ Text_____

Inputs Required:

Outputs:

Task Interfaces:

Figure 3-11. Sample Work Breakdown Structure

```
1.          DTM Software
1.1         DTM Software Management
 1.1.1       DTM Software Cognizant Manager
 1.1.2       DTM Software CDE
1.2         DTM Software Planning and Requirements
 1.2.1       DTM SRD
 1.2.2       DTM Software Requirements Design Review
1.3         DTM Software Design Definition
 1.3.1       DTM SDD
 1.3.2       DTM Software Design Definition Review
1.4         DTM Software Design and Production
1.4.1        DTM SSD
 1.4.2       DTM SOM
 1.4.3       DTM Software Detailed Design Reviews
 1.4.4       DTM Software Modules Design and Production
  1.4.4.1     TLMOS  (Telemetry Operating System)
  1.4.4.2     SYMEQU  (DTM Software System Equates)
  1.4.4.3     USEMAC  (DTM Users Macros)
  1.4.4.4     TLMCOM  (Telemetry Common Data Structure)
  1.4.4.5      INT  (Telemetry Initialization Task)
  1.4.4.6      FTS  (FTS Interface Task)
  1.4.4.7      SSR  (Star Switch Router Task)
  1.4.4.8      MIO  (Message Input/Output Task)
  1.4.4.9      NOC  (Network Operations Center Input Task)
  1.4.4.10     PCT  (Program Control Task)
  1.4.4.11     SSA  (SSA Control Task)
  1.4.4.12     SDT  (Sequential Decode Task)
  1.4.4.13     SBD  (Software Block Decode)
  1.4.4.14     MCD  (Maximum-Likelihood Convolutional Decoder)
  1.4.4.15     BDA  (Block Decoder Assembly)
  1.4.4.16     DSD  (Dual Sequential Decoder)
  1.4.4.17     F69  (Pioneer 6-9 Formatter)
  1.4.4.18     FIO  (Pioneer 10/11 Formatter)
  1.4.4.19     FLS  (Helios Formatter)
  1.4.4.20     FVK  (Viking Formatter)
  1.4.4.21     FMF  (MJS Formatter)
  1.4.4.22     FPV  (Pioneer Venus Formatter)
  1.4.4.23     HSD  (High Speed Data Output Task)
  1.4.4.24     WBD  (Wideband Data Output Task)
  1.4.4.25     HSD  (High Density Recorder Output Task)
  1.4.4.26     DIS  (Digital Instrumentation Subsystem)
  1.4.4.27     HRP  (High Rate Playback)
  1.4.4.28     SDP  (Sequential Decode Playback)
1.5         DTM Software Test and Transfer
 1.5.1       ATP  (DTM Software Acceptance Test Procedures)
 1.5.2       AT   (DTM Software Acceptance Test)
 1.5.3       STT  (DTM Software Test and Transfer)
```

(THIS BREAKDOWN IDENTIFIES TASKS IN A HIERARCHIC STRUCTURING OF DETAIL. THE "STUBS" OF THE STRUCTURE ARE DETAILED AS IN FIGURE 3-12.

Figure 3-12. Sample Detailed Task Description Format

(e) Task Manager* — name (or initials) of the individual responsible for task accomplishment (the CDE or assistants).

(f) Duration — from date task is planned to begin to date task is planned to be completed.

(g) Task Description — brief narrative description of task elements; what the task is intended to accomplish, and what this task is part of.

(h) Task Schedule — simplified bar chart indicating start, finish, and appropriate milestones for task elements (once schedule has been derived).

(i) Task Budget — time-phased manpower, by skills categories necessary to accomplish task. This should include supervision, design, coding, testing, documentation, and any required supporting services (e.g., typing, Programming Secretariat, QA representation, etc.)

(j) Task Scope — estimate of the number of pages of flowcharts, the number of lines of code, and pages of narrative text. Should be correlated to Task Schedule.

(k) Inputs Required — documentation, other task outputs, hardware, or other resources which are necessary for accomplishment of this task, including required dates if known.

(l) Outputs — interim and final outputs of tasks, such as design documentation, tests accomplished, reports, supporting documentation, program listings, etc. Also output dates for each, after schedule has been derived.

(m) Task Interfaces — identify directly interfacing tasks, common software candidates, interchangeable modules, or anything that will help characterize relationship of this task to other tasks.

3. Task Budgeting

The WBS and Detailed Task Descriptions can be used by the CDE to plan and budget time and resources during the architectural design and again later as necessary. A sample technique for Task Budgeting is presented in Figure 3-13.

*Since this breakdown is not part of the SSD but is for Project use only, it is reasonable to mention names here.

ORIGINAL PAGE IS
OF POOR QUALITY

DSS TELEMETRY SUBSYSTEM SOFTWARE DEVELOPMENT SCHEDULE AND MANNING

	WBS TASK NAME	1976 JAN	FEB	MAR	APR	MAY	JUN	JLY	AUG
1									
2	1 MANAGEMENT								
3	1.1 COGNIZANT MANAGER	6	6	6	6	6	6	6	6
4	1.2 CDE	12	15	8	6	6	6	6	6
5									
6	2 PLANNING + REQUIREMENTS								
7	2.1 SRD (COMPLETED) ▲								
8	2.2 DESIGN REVIEW ▲								
9									
10	3 DESIGN DEFINITION								
11	3.1 SDD		1	3	5	8	4		
12	3.2 DESIGN REVIEW ▲								
13									
14	4 DESIGN + PRODUCTION								
15	4.1 SSD	- -	- -	4	4	5	5	5	6
16	4.2 SOM						4	8	8 ▲
17	4.3 DESIGN REVIEWS	- -	- -	- -	- -	- -	- -	- -	- -
18	4.4 MODULE PRODUCTION								
19	4.4.1 TLMOS ▲	- -	- -	- -	- -	- -	- -	- -	10 ▲
20	4.4.2 SYMEQU ▲								
21	4.4.3 USEMAC ▲								
22									
23									

NOTE: NUMBERS DENOTE MAN-MONTHS OF EFFORT

Figure 3-13. Sample Task Budgeting Format

This format or equivalent can also be used to record actual resource expenditures for implementing change requests (during implementation) or ECOs (after transfer to Operations) to be subsequently appended or attached to the SDD, to maintain a current cost and design history of the program.

B. STATUS REPORTING BEFORE SDD APPROVAL

1. Architectural Design and Build Status

Figure 3-14 represents a format and method that can be used for recording and reporting progress of the overall architectural design. The Programming Secretariat records status information that is extracted from records and project-approved items that are received for filing. No additional effort is made by the CDE. Module numbers and titles are preliminary, as detail added later may affect these; however, some traceability from the architecture to the detailed design will be a natural result of module names, functionally chosen.

When using this method, blank areas clearly indicate potential trouble areas. Once identified, these can be appropriately addressed. Information recorded along each horizontal line gives a clear indication of the progress in the specific module.

The utility of Figure 3-14 can be extended by having two lines for each module entry. One line would provide the a priori or predicted scheduling information (from the CDE) and the second line would provide actual performance (recorded by the Programming Secretariat).

2. Software Technical Program Progress Report

A narrative report submitted on, say, a monthly basis can be used to keep management and the Project Office abreast of the implementation. Since this type of report would typically be used only during the SSD phase, details as to content and format are included in Chapter 4 and are not duplicated here.

WRW PROJECT ARCHITECTURAL DESIGN AND IMPLEMENTATION

CHART	NAME	DIAGRAM	NARRATIVE	DESIGNED	CODED	T.PROCEDURE	TESTED
1	WADTDTRPT	5/6/74					
1.1	PREPROC	5/7/74					
1.1.9	TALLY	4/10/74					
1.1.10	SORTDSP	4/10/74					
1.1.10.4	FORMSTR						
1.2	WADTDT	5/22/74					
1.2.7	COMPUT	4/18/74					
1.2.7.4	PPDS						
1.2.9	SUMS	9/24/73					
1.2.10	CHKSUM	9/17/73					
1.2.10.4/7	GTPRNT						
1.5	WADRPT	5/12/74					
1.5.3	JBCNTL	2/14/74					
1.5.3.6	QUERY	9/24/73					
1.5.4	PDRPTS	2/14/74					
1.5.4.4	DIVSUM	9/24/73					
1.5.4.6	PRJSUM	9/24/73					
1.5.5	DSPRPT	2/14/74					
1.5.5.5	RPTPRT	9/24/73					
1.5.5.5.13	TYPRPT	9/24/73					

Figure 3-14. Sample Implementation and Build Status Information Format

CHAPTER 4

THE STANDARD PRACTICE FOR THE SOFTWARE SPECIFICATION DOCUMENT

SECTION I
INTRODUCTION

A. PURPOSE OF THIS STANDARD PRACTICE

The relationship of the Software Design and Production Phase to the overall software implementation process is shown in Figure 4-1. As shown in the figure, the SSD is initiated at the start of the Design and Production Phase and is completed and approved prior to or during the Acceptance Readiness Review.

The approved SSD is used throughout the Software Acceptance Testing Phase, where changes are under internal project control. Changes are authorized by the CDE after mutual agreement with the COE that a change is required. Following transfer to operations, changes are implemented only by approved Engineering Change Orders, with possible exceptions for cosmetic or explanatory changes as recommended by the COE or as directed by the Change Control Board.

B. SCOPE OF SOFTWARE SPECIFICATION DOCUMENTS

The SSD contains the program descriptive and as-built design information needed to maintain the program throughout its operational life. To accomplish this, the scope of coverage of the SSD includes:

(1) A complete set of design information, including the design approach, functional specifications, program as-built specifications, and criteria used for correctness assessment.
(2) Sufficient supporting information for program maintenance, such as program listings, detailed sample formats, memory maps when appropriate, etc.

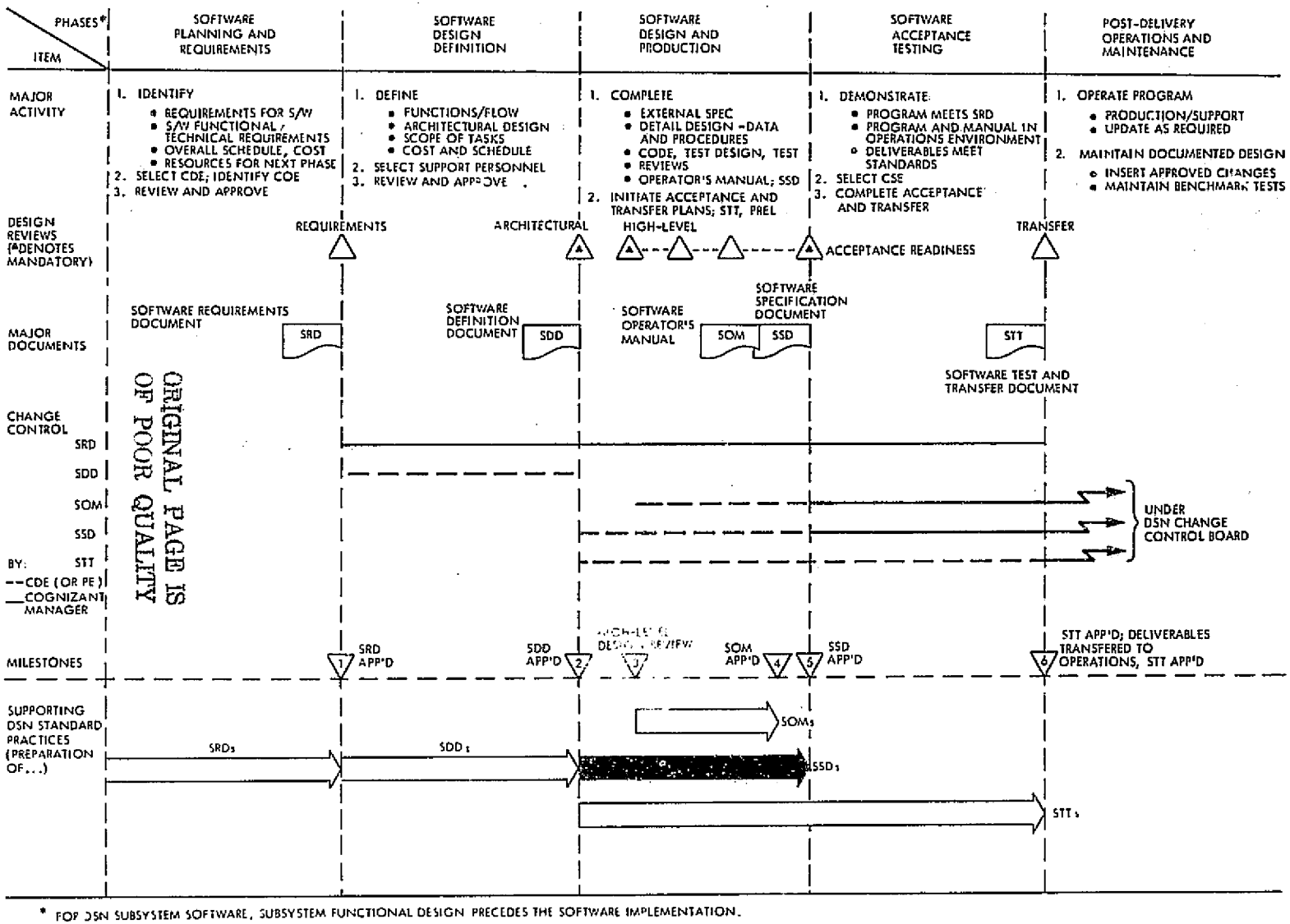


Figure 4-1. DSN Software Management and Implementation Plan (SSD Software Design and Production)

(3) A complete set of flowcharts (or equivalent), with explanatory narrative provided on a module-by-module basis.

(4) The above items documented in a survivable, maintainable form, since they will be used throughout the operational life of the program. The level of detail and degree of format formality will be specified in the program Software Requirements Document. Standard classes of detail and format are defined in Reference 1.

SECTION II
SSD CONTENTS

A. CONTENT OUTLINE

A typical SSD content outline is shown in Figure 4-2, along with an identification of personnel responsible for preparing, concurring in, and approving the SSD. Introductory information, which includes the program description, is followed by program conventions and standards, environment and interfaces, functional specifications, program specifications, and verification and test information. Detailed supporting materials for program maintenance are appended, such as sample input and output formats, memory maps, tables, figures, and program listings.

The outline lists the major information items needed for specifying the as-built program and for maintenance. For a given program, the actual contents may vary and will depend on the complexity of the program and its interfaces. However, the items in Figure 4-2 are considered to be basic and, in most cases, always necessary. There may be items in the table of contents other than those listed. The SRD may, for example, identify required additions or exceptions. The information contained in the SSD should be sufficient to allow the CSE to maintain the program in its operations environment.

For review information and criteria, refer to Section III, Paragraph B, Review and Approval.

B. OUTLINE DISCUSSION

1. Introduction

a. Item 1.1, Purpose and Scope of SSD. The purpose of the SSD is stated, along with the scope of coverage and a brief summary of major sections of the document.

1. INTRODUCTION
   - 1.1 Purpose and Scope of SSD
   - 1.2 General Program Description
   - 1.3 Applicable Documents
2. STANDARDS AND CONVENTIONS
3. ENVIRONMENT AND INTERFACES
   - 3.1 Hardware Configuration and Interfaces
   - 3.2 Software Environment and Interfaces
4. FUNCTIONAL SPECIFICATIONS
   - 4.1 Functional Overview
   - 4.2 Software Configuration and Modes of Operation
   - 4.3 Detailed Functional Specifications
   - 4.4 Data Base Specifications
5. PROGRAM SPECIFICATIONS
   - 5.1 Design Philosophy, Rationale, Approach, and Organization
   - 5.2 Main Program Detailed Design
   - 5.3 Subroutine Detailed Designs
   - 5.4 External Subroutine Interfaces
   - 5.5 Data Structure Definitions
   - 5.6 Resource Allocation and Access
6. VERIFICATION AND TEST INFORMATION
   - 6.1 Correctness Test Criteria
   - 6.2 Summary Correctness Test Results
7. APPENDIXES
   - 7.1 Glossary
   - 7.2 Formats
   - 7.3 Memory Maps
   - 7.4 Decision Log
   - 7.5 Other Tables and Figures
   - 7.6 Source Code Listings

SIGNATURES

Prepared by: CDE

Concurred and Approved by: CSE (if different from CDE)

Note: Approval indicates readiness for use in acceptance testing.

Figure 4-2. Typical Outline for a Software Specification Document

b. Item 1.2, General Program Description. A general description of the program is provided, including its purpose, the nature of the problem, the type of data generated, processed, or transmitted, and the type of program, such as real-time, interactive or batch, computational or data manipulation, developmental or operational, etc. For perspective, the overall system, applicable subsystem, and environment (hardware and software) in which the program operates is identified. A data flow block diagram with narrative may be useful here. Also, the general system/subsystem/program operating mode can be identified (e.g., "interrupt-driven, real-time, dedicated operations with 16 K-words core and 5m-byte disk"). Main system and program constraints imposed by requirements which influenced the design, such as core size, timing constraints, etc., should be identified.

c. Item 1.3, Applicable Documents. All documents required to maintain the program are listed along with the location of the document and/or how it is obtained. Other appropriate reference documents may also be listed.

2. Standards and Conventions

Standards and conventions used in the SSD and in programming are provided. The source, any exceptions, and any project-unique standards that apply are identified. If these are contained in standards documents, this information may be referenced; standards identified in the SRD may be extracted and inserted in the SSD. The categories of standards can include:

(a) Policy, practices, procedures
(b) Coding, including programming language, argument-passing, etc.
(c) Documentation conventions, nomenclature
(d) Correctness testing
(e) QA

3. Environment and Interfaces

This section provides a description of the system in which the program operates and is an expansion of Section 1.2. If information pertinent to this

section is contained in another document (e. g., the SDD), the pertinent material may be extracted and inserted or appended.

a. Item 3. 1, Hardware Configuration and Interfaces. A description of the pertinent aspects of the hardware configuration in which the program operates is provided, including functional diagrams. All subsystem hardware, such as computers, standard peripherals, and specially designed interface equipment, should be identified and functionally described. All subsystem-unique, nonstandard, or specialized input/output devices are identified, as well as the pertinent functional characteristics of these devices, such as (1) control parameters, (2) response parameters, (3) interrupt generators, (4) interrupt priority, (5) timing parameters, etc. Special characteristics of the devices can be identified, such as (1) use of non-ASCII codes, (2) line rates, (3) half duplex or full duplex, (4) automatic control codes or other control characters. If detailed descriptions of the hardware are needed, reference should be made to the appropriate hardware documentation.

b. Item 3. 2, Software Environment and Interfaces. The operating system is described in terms of the facilities provided and any constraints which it may have imposed on the program design. Key elements considered to be part of the system software are identified. Program interaction with the operating system and how it utilizes the operating system facilities in meeting the program interfaces and interfaces with other programs are described or referenced. All programs which provide essential data to, or receive essential data from the program are identified, including any special data transfer characteristics or special software that may be involved. Constraints such as buffering, queueing, timing, or protocol requirements which are imposed by these interfaces are stated; the parameters for data messages and the characteristics of data control parameters associated with the data are defined. Typical data parameters which may be defined are (1) input and output formats, (2) syntax, (3) message lengths and frequency, (4) device assignment, (5) special considerations such as on-line vs. off-line printing/display, (6) data units, and (7) data ranges.

4. Functional Specifications

This section specifies the functional characteristics of the software. Functional specifications which have been appropriately described in another document (e. g., the SDD) may be incorporated or referenced if under Engineering Change Control. The content of the Functional Specifications Section need not be restricted to the major topics outlined below. Additional paragraphs should be included if required to adequately state a subsystem's functional specifications. This section may become lengthy, particularly with respect to paragraph 4. 3, which requires the program functional behavior to be specified in detail. If lengthy, the section may be published as a separate volume of the SSD.

a. Item 4. 1, Functional Overview. A discussion of the overall functional behavior of the program, the principal modes of operation, the different software configurations (if any), and the major data flows are described. The intent is to prepare the reader for the detailed functional specifications to follow.

b. Item 4. 2, Software Configuration and Modes of Operation. For each hardware configuration described in Paragraph 3. 1, the various program operating modes (where the processing is selected to be only a certain partial set of the full capabilities of the configuration) are identified. If the modes have functional submodes, these should be described hierarchically; that is, by expanding the detail of key parts of the main mode. The modes and submodes are named and numbered to allow convenient reference to them as may be needed throughout the SSD. The events, conditions, and computations which control transitions between modes are described. State diagrams (see Section IV of Chapter 3) and decision tables (described in Reference 1) can be used to illustrate and express mode-transition logic and to identify major functions within modes.

c. Item 4. 3, Detailed Functional Specifications. The as-built program functional behavior is specified using hierarchically refined input. processing, and output specifications beginning with the broad program end-to-end functional characteristics and proceeding to functional details. Inputs, processing

functions, and outputs can be illustrated using one-page data flow diagrams and standard documentation techniques. (Refer to Section IV of Chapter 3.) Specifications should cross-reference requirements identified in appropriate requirements documents.

d. Item 4.4, Data Base Specifications. Overall characteristics of the data base, data files, and external data structures are specified, with emphasis on program-unique access and utilization factors. Reference to applicable data base documents is preferred for data base specifications and detail, including descriptions of structure, how the data base or files are created and maintained, or specifications of privacy, security, validation, and control parameters.

5. Program Specifications

A description and specification of the as-built program design is presented to provide information needed for program maintenance. The philosophy, rationale, and design approach provide the basis for understanding the program organization. Program procedural detail is then provided by specifying the design in a top-down hierarchical manner.

a. Item 5.1, Design Philosophy, Rationale, Approach, and Organization. A brief description of the design philosophy, discipline, and approach taken in the program implementation is presented, along with the rationale. The general order of priorities adopted for completing the design is described only if important to the maintenance of the program. Key factors which had major influence on the design are listed, such as (1) program size and execution speed constraints, (2) vulnerability to operator error, (3) vulnerability to system errors, etc. This can be accompanied by a processing overview in terms of the major algorithms and data structures. A description of the relationships between functional specifications and individual program components is presented. Data flow between the program processing modules may be described to illustrate how the program architecture accommodates functional requirements. Finally, the resulting program organization can be illustrated by a tier chart or summary table of hierarchic modules.

b. Item 5.2, Main Program Detail Design. A detailed algorithmic description of the top-level main program design is provided, including a control flowchart (Chart 1), or equivalent. The hierarchic detailed design of the main program is presented in nestings of one-page flowcharts (or equivalent) accompanied by narrative keyed to the separate design elements.

c. Item 5.3, Subroutine Detailed Designs. A detailed algorithmic description of the top-level design of each subroutine is provided, including a control flowchart (Chart Si), or equivalent. The hierarchic detailed design of each subroutine procedure is presented in nestings of one-page flowcharts (or equivalent), with accompanying narrative keyed to the separate design elements.

d. Item 5.4, External Subroutine Interfaces. A description of the interfaces and characteristics of each external (Xi) module or subroutine called by the program is provided or referenced. The description typically can include (1) the purpose and function of the subroutine, (2) the calling sequence, (3) all external programs and subroutines called, (4) common data areas, (5) operating system interface data, (6) mathematical equations, if appropriate, (7) execution speed and core usage, if relevant, (8) input/output, (9) restrictions on use, and (10) error messages. A high-level control flowchart (Chart Xi), or equivalent, may also be appropriate, but detailed internal information is typically only referenced.

e. Item 5.5, Data Structure Definitions. Internal data structures are described, including usage, formats, memory allocations, links, etc. Data structures may be identified with those parts of the program (levels of access) which operate on them.

f. Item 5.6, Resource Allocation and Access. Resource allocations made by the program are described, such as input/output devices, internal core storage management, memory maps, buffers, etc. In addition to the services provided by the resources, their availability to the program components is described, as well as the management of these resources by the program.

6. Verification and Test Information

This section presents the test guidelines that were followed during the production testing of the as-built design.

a. Item 6.1, Correctness Test Criteria. The criteria used to assess program correctness are described. Typically included are criteria for the paths exercised, the required interfaces and stubs and their characteristics, the data ranges used, the errors or overloads applied, any special test code used, such as path monitors or trace printers, and whether test code will be removed or retained in the delivered program module. Criteria for test data selection and for determining the validity of observed program responses may be included. Existing test procedures used for this correctness testing may be referenced.

b. Item 6.2, Summary Correctness Test Results. Summaries of the results of the correctness testing are presented to aid in correctness testing during program maintenance.

7. Appendices

a. Item 7.1, Glossary. Abbreviations, mnemonics, acronyms, and other uniquely used terms contained in the SSD are listed alphanumerically, along with their definitions or explanation of their use.

b. Item 7.2, Formats. Graphic layouts or actual samples of program formats are appended, including formats for inputs, outputs, data structures, files, syntax rules, error messages, etc.

c. Item 7.3, Memory Maps. Graphic descriptive layouts that show core and disk storage allocation (i.e., memory maps) are appended as applicable. Also, descriptive information can be provided for program dynamic reallocation, overlays, swapping, etc.

d. Item 7.4, Decision Log. All major design decisions which may affect later maintenance activities are listed and described.

e. Item 7.5, Other Tables and Figures. Other data of value for program maintenance can be appended, such as Data Structure Definition Tables and Resource Access Requirements Tables for presenting the data and resource specifications, respectively (as described in Reference 1).

f. Item 7.6, Source Code Listings. Unaltered copies of the computer program listings are appended to provide ready access to the as-built source code in support of Section 5 of the SSD. The listings should contain appropriate identification and should be suitable for use by Quality Assurance when auditing the design vs code. If lengthy, this appendix may be a separate volume of the SSD.

SECTION III
SSD PREPARATION, REVIEW, AND APPROVAL

A. PREPARATION GUIDELINES

1. Top-Down, Concurrent Generation

Detailed program implementation is governed by the requirements stated in the SRD and based on the program architectural design presented in the SDD. Standards and conventions identified in the SRD establish quality and workmanship guidelines for the project. The preliminary functional analysis presented in the SDD is expanded in compliance with these standards and conventions to form the functional and procedural specifications for the program. The program is then implemented in steps and concurrently verified against the specifications and requirements.

The SSD contains the organized collection of key implementation notes and records, produced concurrently with the above activities. Timely and useful working documentation therefore is made available for project-wide use during the implementation as well as throughout the operational life of the program. When practical, the working documentation should be written in the format and to the level of detail suitable for incorporation in the SSD, thus minimizing documentation costs.

a. Standards and Conventions. The standards and conventions used during the implementation are provided for later maintenance activities. "Standards" is defined as criteria for measuring the quality of products and processes on a broad basis (i.e., industry-wide, company-wide, organization-wide). Example: This Standard Practice. "Conventions" is defined as criteria governing products and activities of hierarchically smaller units (projects, systems, etc.) more oriented to specific production tasks, and they specify local policies for implementing the broad-based standards. Common to both, however, is the property that through their application, a consistent

and acceptable end item will be produced. When standards and conventions are documented elsewhere, they are referenced in the SSD, as applicable. When applicable standards and conventions are not documented elsewhere, they are defined in the SSD as needed for maintenance.

b. Functional Specifications. The functional descriptions from the architectural design are refined during program design. The preliminary functional description and graphics contained in the SDD may be used as a starting point. Employing top-down design methods, the external details are typically defined first, followed by specification of the program input, processing, and output characteristics, with progressing levels of detail as the design proceeds. Hierarchy plus Input-Process-Output diagrams with accompanying narrative, as discussed in Chapter 3, can be used as a documentation tool.

c. Program Specifications (Charts and Narrative). The program algorithmic design is presented in the SSD by the use of flowcharts or an equivalent format that allows explanatory narrative to be keyed to the separate design elements. Guidelines for flowcharting and preparing accompanying flowchart narrative are provided in Section IV of this document. If a different but equivalent format is used, then the applicable set of conventions or guidelines used must be included or referenced in the SSD. When a different format is to be used, the format will normally be identified in the SRD. The basic requirement is that the documentation must be sufficient for coding and for comprehensive use by the CSE.

d. Correctness Tests. The following guidelines can be used to achieve a high degree of confidence in the correctness of a build aggregated for testing:

(1) Test the module or group of modules in a build under test using the previously written and tested part of the program, incorporating stubs of dummy code for incomplete submodules. Path monitors can be included to trace the flow.

(2) Traverse each flowline at least once.

(3) Run the program, with stubs, using variations and modifications to the input to cover typical and alternative cases and perhaps some random ones.
(4) Judge for correctness from path monitoring information.
(5) Work from the Software Operator's Manual being developed concurrently.

Actual test records are typically annotated with handwritten notes and kept with other project notes until transfer. This provides diagnostic and status information throughout the implementation while not requiring extensive correctness test descriptions or reports.

2. Quality Assurance

Independent code auditing as an ongoing process throughout the implementation is recognized as essential for assuring that acceptable software products are submitted for timely transfer and delivery to operations. Therefore, module-by-module or build-by-build inspections are performed throughout the implementation, followed by a final audit and certification of the deliverables prior to transfer to operations.

The CDE provides Quality Assurance with all information needed, such as application standards, conventions, documentation, and listings that are produced concurrently with the implementation. QA then compares the source code with flowcharts, narrative descriptions, standards, and conventions in order to uncover discrepancies between the different software design descriptions. The auditing process is generally performed as follows: First preliminary audits or inspections are performed by QA on a module-by-module or build basis, immediately as code and documentation become available. Prior to transfer, QA performs a final audit of the SSD and other software deliverables as requested, to certify overall conformance with the applicable standards or to identify problems of nonconformity. A QA signature is required on the Software Transfer Agreement (see Figure 6-7 of Chapter 6); therefore, early detection of problems by QA and timely correction by the CDE are necessary to avoid delivery delays and/or critical rework.

B. REVIEW AND APPROVAL

The SSD outline is submitted, along with other program and test items, for management review and concurrence at the program High-Level Design Review held early in the software design and production phase. At the end of the production phase, the completed SSD is approved by the CSE and then submitted for general review at the Acceptance Readiness Review. The following paragraphs describe typical SSD-related subjects covered during these reviews.

1. High-Level Design Review

a. Agenda. SSD-related items for the High-Level Design Review typically include:

(1) Any carry-over items from the Architectural Design Review affecting program design and construction.
(2) SSD Content Outline (draft).
(3) Present SSD production plans and coding status (if different from the published, approved plans, work breakdown structure, and schedules contained in the SDD).
(4) Identification of any real or anticipated differences between the implemented capability and the SRD requirements, with specific reference to the list of Competing Characteristics contained in the SRD.
(5) Other pertinent SSD items, problems, and concerns.

b. Evaluation Criteria. The following criteria can be used during the High-Level Design Review as aids for assessing the SSD production progress, the completeness of the proposed SSD, and the soundness of the production plan. There should be evidence that:

(1) The completed high-level design provides sufficient information to assess its correctness and to continue the design and construction of lower-level modules.

(2) The SSE, COE, QA Representative, and others have sufficient visibility and access to in-progress implementation information and concurrent documentation to effectively participate in their inspection, audit, and use as the program evolves.
(3) The near-term schedules and executable builds, identified for correctness testing, reflect present progress and experience.
(4) The QA plan accommodates code audits on a module-by-module basis.
(5) The module code reflects the exact design as presented on the approved module flowchart (or equivalent).
(6) The flowchart (or equivalent) and narrative are adequate for the later sustaining activity.
(7) Standards and conventions are being adhered to.

c. Concurrence. Concurrence to proceed with the implementation of the program and production of the deliverable documentation is based on the evaluation of the program status and implementation plan made at the High-Level Design Review. In some cases, the concurrence may be "conditional," depending upon the solution to significant problems.

2. Acceptance Readiness Review

a. Agenda. SSD-related items for the Acceptance Readiness Review typically include:

(1) Any carryover items concerning the code and SSD from the High-Level Design Review.
(2) Implementation status and deficiencies affecting program construction and documentation.
(3) Deficiency workarounds and completion dates.
(4) QA inspection status.
(5) Code and SSD capabilities and concerns related to support of acceptance testing.
(6) Other pertinent concerns.

b. Acceptance Criteria. The following SSD-related criteria can be used as aids in determining readiness for acceptance testing. There should be evidence that:

(1) The SSD is sufficiently complete to allow effective authorized program modifications during acceptance testing operations.
(2) The SSD will adequately support the program in its operational environment.
(3) The SSD can support acceptance testing (with authorized work-arounds, patches, etc).
(4) The SSD has been reviewed and approved by the CSE.

c. Approval. The SSD is approved prior to the Acceptance Readiness Review by the CSE. Other approvals that may be required (line managements, etc.) are specified in the SRD. Approval indicates that the SSD can support acceptance testing.

SECTION IV
PREPARATION AIDS

A. SSD IDENTIFICATION AND FORMAT

1. SSD Identification

The SSD is given a document identification number by prefixing "SSD" to the program number assigned by the Program Libary. The document number appears at the top center of each page of the document.

2. Format

The SSD is block-formatted and sectionalized, with each section identified by consecutive arabic numerals. The pages, figures, and page-size tables are numbered consecutively and prefixed by the section number within each section. Small tables may be inserted within the running text and are not necessarily numbered. However, if other parts of the SSD refer to the table, the table should be numbered and placed at the top or bottom of the page. Paragraphs are numbered within sections using decimalized numbers. Appendices are identified alphabetically. Formatting within the appendixes can vary to allow existing material to be incorporated without modification. Appendices do not require an introductory separator page.

A Decimal Format as shown in Figure 4-3 can serve as a guideline for formatting the SSD. The following exceptions are noted:

(a) Illustrations or figures are included in the running text or on the immediate next page to their first reference.
(b) An appendix should begin on its first page (i.e., an introductory separator page is not needed).
(c) Flowcharts (or their equivalent) and accompanying narrative should conform to the example formats contained in this appendix.

The GCF high-speed data system shall be used for JPL/DSCC communications. High-speed data transmissions shall normally be limited to exception reporting. All communications shall be at scheduled time periods of not greater than 1-hour duration, at least on a weekly basis.

2.2 PERFORMANCE PARAMETERS AND DESIGN CONSTRAINTS

Implementation of the CCA Assemblies shall meet the performance requirements and design constraints specified below:

(1) Data base organization shall minimize redundancy of data and eliminate duplication of data records (file integration).

(2) Data base organization shall allow efficient data access by the use of the MBASIC user programming language.

(3) Data base structure shall allow efficient data record access from more than one type of storage medium (e.g., disk or magnetic tape storage).

(4) Data base file content shall support, as a minimum, the following managerial activities:

(a) Life cycle studies of DSN systems (e.g., DSN Tracking System).

(b) DSN systems configuration control to a subassembly/replaceable-module level.

(c) DSN schedule analysis and review.

(d) DSN budget data analysis and review.

(e) DSN documentation status reviews.

(f) Local DSCC/JPL record keeping and analysis.

(g) Quality assurance record keeping and analysis.

(h) Facilities record keeping and analysis.

(i) Logistics record keeping and analysis.

(j) Mission-readiness analysis.

(5) Data entry device locations shall be convenient to the normal working stations of users supplying transaction data (local data entry).

Figure 4-3. Sample of Standard Decimal Format

Figures 4-3 through 4-5 are examples of typical page formatting. Figure 4-6 is an example of narrative that accompanies a flowchart (or equivalent). A suitable SSD flowchart using standard symbols with unique symbol identification is illustrated in Figure 4-7.

B. MODULE FLOWCHARTING AND DOCUMENTING GUIDELINES

1. Guidelines for Flowcharting

Flowchart material included in the SSD should conform to the following:

(a) The use of flowchart symbols is limited to those illustrated in Paragraph C of this section. The symbols are based on ANSI Standard X3.5 but have been expanded where necessary to conform to the needs of structured programming. Also included are conventions for extra-normal terminations and real-time interrupt configurations. No other configurations should be used.

(b) Symbols should not be varied in proportion. Symbol text should be brief but exact; narrative material accompanying each flowchart should be used to extend, explain, and expand upon the symbol function.

(c) Identification information should be placed in the upper right corner of each flowchart. The following configuration is suggested:

Chart Number	(decimal number)
Module Name	_____
Date	(date prepared)
Page ____ of ____	(identifies the number of narrative and flowchart pages for the above module)

(d) A control block should be included on each flowchart. The configuration below is suggested:

Title	Initials	Date
Designer		
Checker		
Proj Engr		

821-9

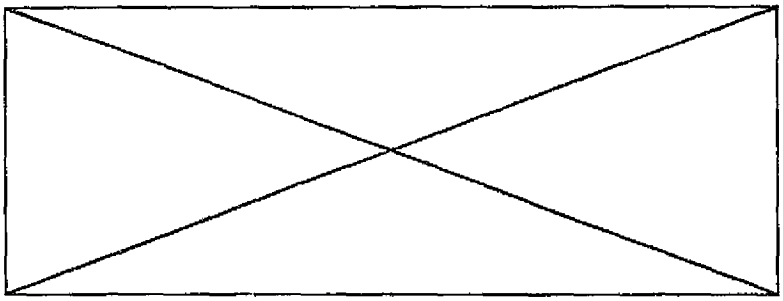


Figure 1-1. Data Acquisition Functional Operation of Configuration Control and Audit Assembly (Continuation 1)

Information transfers between CCA Assemblies are accomplished by utilizing the GCF High-Speed Data Subsystem and are effected at scheduled time periods on a noninterference basis with normal HSD traffic during DSN tracking operations.

NOTE

Information received by a CCA Assembly via the HSD communications link consists of exception reports (e.g., exceptions to Deep Space Station (DSS) 61 planned configuration for Viking support), selected contents of a data base, or file/file record updates to designated data base files.

All HSD block transfers are error-checked (including a block outage check) by the receiving CCA Assembly, and retransmission is requested for any data blocks found to be missing, out of sequence, or in error. The data base information content of the error-free HSD blocks is extracted, identified, and then stored according to data type (exceptions, updates, requested contents) for later inclusion into the data base.

Data base information acquired from local sources consists of transaction inputs from local CCA users. Transactions conform to an established format, determined by the identification (ID) of the file and/or user ID. As an example, a stockroom clerk has issued a number of units of expendable stock and desires to record his transaction for data base entry. The stockroom clerk activates a CCA Assembly data terminal and types in a standard message that identifies the user, the data base file (logistics), and the operating mode (transaction entry). The appropriate software module then interacts with the stock-

1-3

Figure 4-4. Sample of Standard for Placement of In-Text Figure and Note

821-9

APPENDIX A

PROGRAM BUILD AND ACCEPTANCE TEST SCHEDULING
FOR DSN SUBSYSTEM SOFTWARE

Program build and acceptance test milestones for DSN subsystem software are given in Table A-1; software development guidelines and practices are described in Reference A-1; and a sample of a demonstration program size and development-time listing is given in Table A-2. Progress within each level of the design can be tracked by the use of the program build milestones. At the completion of the last build, the program is complete, correct, and ready for final acceptance testing. The planning and preparation for acceptance testing is performed in parallel with the program implementation. Software implementation is complete at transfer; the remaining milestones pertain, if applicable, to the total subsystem acceptance, of which the software is an active part.

Table A-1. Program Build and Acceptance Test Milestones for DSN Subsystem Software

Symbol	Name	Description
D	Design Approved	The program architectural design definition presented in the Software Definition Document (SDD) is detailed in terms of control logic in preparation for coding and is checked independently. Approval of the logic design authorizes the start of coding and test designing, which may, however, at the discretion of the CDE, have started before formal design approval.
P	Test Procedure Design Verified	The correctness test procedure and necessary test code for testing the new build module against the approved design are designed, assembled, and verified. Test Procedure Design Verified indicates that the test procedure and associated test code are complete and available. Test design should formally start at the same time that module coding starts (that is, after design approval).

A-1

Figure 4-5. Sample of Initial Page of an Appendix

5.3(4.4.1) FINDNO Procedure

On entry, an input line has been received into an input buffer via EMSGIN (U14). The LINe PoinTeR, LNPTR, is positioned so as to extract the first character of the line.

This procedure discards leading blanks, if any, and looks for an MBASIC statement number. If a statement number is present, the digits are converted to an integer and placed in the Value variable V; the Class variable C is set to 11 to indicate that the first symbol on the line is an integer. If a statement number is not present, C is set to 0, indicating the input line does not have a statement number; V retains its entry value. See Table 7.2.2.5 for further symbol class and value definitions.

On exit, C and V contain the symbol Class and Value values above, the Current CHaRacter variable, CCHR, holds the character which stopped the number scan, and LNPTR points to the next input character to be fetched. A value of V>999,999 indicates the statement number is too large.

.1/U17 GET the first character from the current input line (see EMSGIN/U14) into CCHR.

.2-.4 CATegorize CCHR. If blank, scan to and fetch the first non-blank. Set Character Class, CC, and Character Value, CV, as specified in Table 7.2.2.5. CC will be 2 if CCHR is a digit, and CV will contain its integer value.

.3-.4 If the first non-blank character on a line is not a digit, set C to record that no statement number is present, and exit.

.3-.5 If the first non-blank character is a digit, set C to record that a statement number is present. Initialize V to the first statement-number digit as an integer, and set the loop structure flag FiND SWitch, FNDSW, so as to iterate, bringing in digits.

.6-.14 Bring in remaining digits and convert them into V as follows. After GETting each character:

.7-.9 If the character is blank, set FNDSW to terminate the scan, as the statement number has now been extracted, and exit.

.10/P4 Otherwise, CATegorize the character seen, as in step 2 above.

.11-.12 If not a digit, terminate the iteration, and exit.

.13/E143 If CCHR is a digit, ACCUMulate its value into V. Set FNDSW=1 to terminate the iteration if V exceeds 999,999.

.14 Iteration continues until all digits have been processed or until overflow was detected.

Figure 4-6. Sample Flowchart Narrative Page

ORIGINAL PAGE IS
OF POOR QUALITY

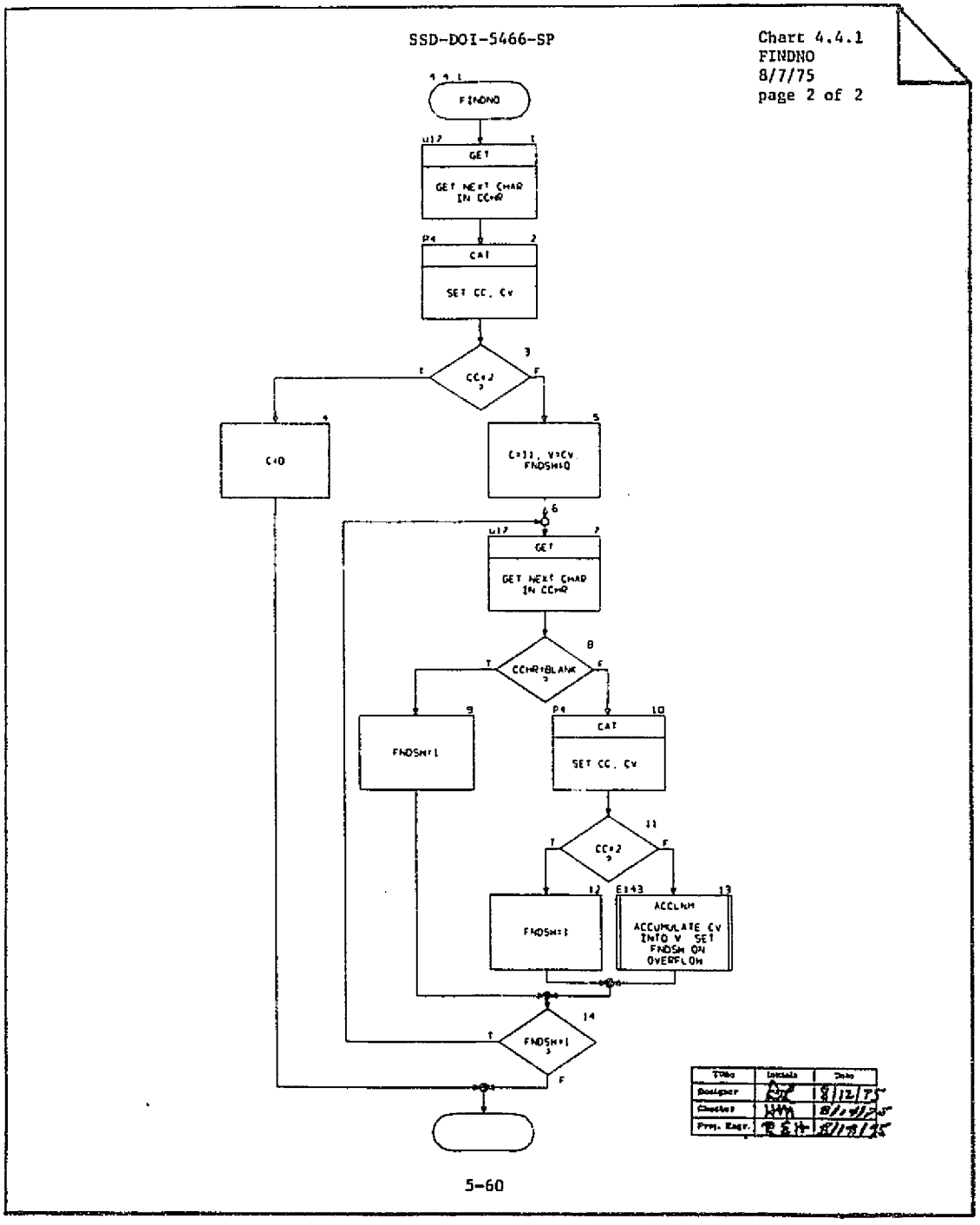


Figure 4-7. Sample Flowchart Page

(e) A flowchart (Figure 4-7) should not exceed one 8-1/2" x 11" page except in unusual circumstances, e.g., when a single decision results in multiple branches that will not fit on one page, and when the symbol-striping convention only adds to confusion.

(f) Each chart is given a decimalized number that identifies its location in the design hierarchy. All symbols on the chart are numbered consecutively. A possible exception is the numbering of collecting nodes. The numbering system recommended is the "preorder traverse" (top to bottom while observing left to right order; see Reference 1). The symbol number is placed at the upper right of the symbol.

(g) Flowchart symbols that fall into the following two categories should always be cross-referenced. The cross-reference identifier should be located at the upper left of the symbol.

(1) Subroutines: A striped box representing a subroutine code segment which is given an alphanumeric level-1 identifier.

(2) Major program segments: A striped box representing a major code segment which is given a level-1 integer program identifier.

(h) When alpha characters are used in a subroutine chart identifier, the submodule symbols of that chart are numbered by concatenating the subroutine chart and symbol numbers. (Example: A striped symbol given the number 5 on the level-1 flowchart, CAT/P4, would be P4.5.)

(i) In striped modules, the mnemonic label is placed above the stripe with the functional description below.

(j) Unstriped process symbols should contain precise, functionally unambiguous, and explicit specifications for coding as space permits. Decision symbols are never striped.

(k) For decision symbols, branching to the left should always denote <u>true</u> and to the right <u>false</u>.

(l) Decision collecting nodes should be placed directly under the vertex of the decision symbol.

(m) Only one arrowhead is used per flowline, at termination.

2. Guidelines for Module Documentation

In the guidelines which follow, the term "module" refers to the flowchart (or equivalent) and its accompanying narrative. The flowchart narrative should only extend, explain, and expand upon the flowcharted algorithm. In general, documentation of each module should be sufficient to assess the correctness of the module without referring to deeper levels of the design hierarchy (see Figures 4-6 and 4-7).

(a) Identify each module page with the chart number, module name, date, and page number.

(b) Prior to the description of the algorithm of the current module, state all assumptions on inputs, common data, etc., used by this module. Describe the module function and all outputs, actions, and constraints.

(c) Number and key the narrative continuity to the flowchart as shown in Figure 4-7. Subroutines or major subprograms invoked may be distinguished, if desired, by attaching their cross-referencing level-1 chart numbers to the step number, as .n/Sm.

(d) The narrative need not include a description of every numbered symbol on the chart, so long as the overall narrative is comprehensive. Descriptions of symbol groupings or combinations of numbered steps can be used when more meaningful.

(e) Do not include irrelevant or extraneous information. Be brief; don't repeat information that exists on the flowchart. When required, use references rather than repeating information appearing in parent modules or in auxiliary tables. Functional detail given in a parent module may be repeated when needed to

(1) Identify which subfunctions belong to which symbols on the current flowchart, or

(2) Detail a given function into subfunctions, or

(3) Clarify the current execution state (for example, by giving an explanation of the meaning of a certain condition which has caused entry to the current module, etc.).

(f) Include statements which provide rationale, assumptions, or other clarifying explanations of the algorithm as needed to lend meaning and readability to the text. Describing the intended significance of an action (e.g., setting or testing a flag) can save a reader much time in understanding what that algorithm is supposed to do. It is important to provide information for every loop, stating what assumptions are valid during each iteration (i.e., the loop invariant assertion).

(g) Specify control logic for a given module flowchart and narrative, so that module control can be assessed for correctness with no aid other than references to preceding levels of the design.

(1) All decisions are to be explicit and determinable within each individual module; there should be no need to refer to deeper levels of design or code.

(2) Unstriped symbols must have explicit values, conditions, and reasons stated for all control flag settings.

(3) For striped symbols which alter one or more control flags for the current module or a parent module, state the explicit values, conditions, and reasons for all control flags altered.

(h) Document the functional characteristics of a module (flowchart and narrative combined) to that point which permits an audit of the algorithm of the current module against its stated function and an assessment of functional correctness. Specifically,

(1) Unstriped symbols are to be described explicitly enough to permit coding without functional ambiguity and without reference to deeper levels of design or previously completed code.

(2) Interfaces of external subprograms that are not part of the current design (vertically striped symbols) are to be described explicitly, in sufficient detail that any subprogram satisfying the stated interface characteristics will operate correctly in the current design. References to documents that provide additional information may be included, but explicit interface characteristics must always be described in the SSD.

(3) Striped symbols of the current module which receive control flags from the current and/or parent modules must have accompanying narrative explaining all conditions, values, and actions for each setting of the flags.

(4) Functional descriptions of striped symbols may be quite broad, but must be complete, and tell what functions that submodule performs. No function may appear in a submodule breakdown at level n+1 that is not a component function of that module as described at level n.

(5) Module functions should be described using assertions defining the computer state upon entry, the specific action that takes place, and the relevant state of the computer upon exit. The computer state is defined as the condition of all computer resources accessible and pertinent to the program (memory, files, interrupt, I/O channels, etc.).

C. FLOWCHARTING SYMBOLS AND USAGE (from Reference 1)

Flowchart symbols represent the functions of an information processing system. These functions are input/output, processing, flow path and direction, and annotation. A basic symbol is established for each function and can always be used to represent that function. Specialized symbols are established for use in place of a basic symbol to give additional information.

The symbols tabulated and described below are derived from Reference 4, "American National Standard Flowchart Symbols and Their Usage in Information Processing," ANSI X3.5-1970. In some cases, the ANSI symbols and usage have been modified to accommodate the needs of structured programming. These flowchart symbols are presented in the following four tables.

Table	Name
4-1	Symbol Usage in Flowcharting
4-2	Basic Flowchart Symbols
4-3	Specialized I/O Flowchart Symbols
4-4	Specialized Process Symbols

Table 4-1. Symbol Usage in Flowcharting (sheet 1)

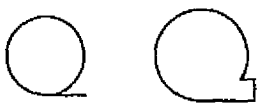
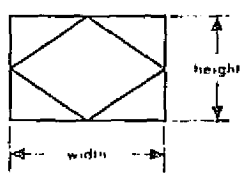
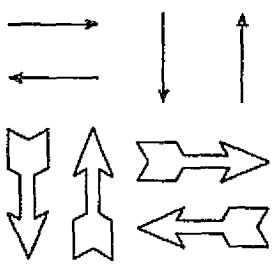
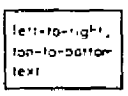
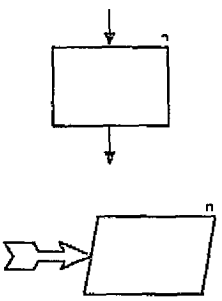
Illustration	Usage
	Symbol Shape. The shapes of the symbols should conform closely enough to those shown in Tables 4-2, 4-3, and 4-4 to preserve the identifiable characteristics of the symbol. The curvature of the lines and the angles formed by the lines may vary slightly from those shown in this standard, so long as the symbol is easily recognized.
	Symbol Size. Flowchart symbols are distinguished on the basis of shape, proportion, and size in relation to other symbols. Proportion of a given symbol is defined by the rectangle in which that symbol can be inscribed. Dimension and relative size of the rectangles are given with each symbol by a pair of numbers (width: height). The size of each symbol may vary, but the dimensional ratio of each symbol shall be maintained.
	Symbol Orientation. The orientation of each symbol on a flowchart should be the same as shown in the tables of this section. Flowline symbols (either control, information, or data flow) may be drawn left-to-right, top-to-bottom, right-to-left, or bottom-to-top. The principal flow of control is top-to-bottom. The principal flow of information or data should be depicted as left-to-right. Avoid diagonal flowlines except in special circumstances.
	Flowchart Text. Descriptive information within each symbol shall be presented so as to be readable from left-to-right, top-to-bottom, regardless of the direction of flow outside the box.
	Symbol Identification. The identifying number n assigned to a symbol on the current flowchart shall be placed above and to the right of its vertical bisector. This number concatenated with the chart identifier c , forms the unique symbol Dewey-decimal identifier, c.n . If the symbol is striped and if there is no explicit symbol cross-reference, then the number c.n becomes the (implicit) cross-referencing chart number at the next hierarchic level.

Table 4-1. Symbol Usage in Flowcharting (sheet 2)

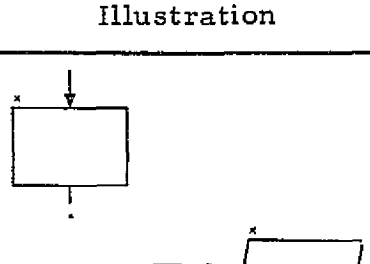
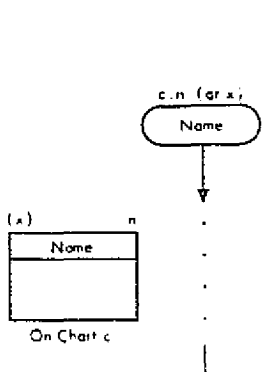
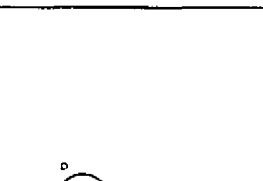
Illustration	Usage
	Symbol Cross-Reference. The identifying chart number or other cross-referencing element shall be placed above the symbol and to the left of its vertical bisector. When such notation appears, it takes precedence over the symbol identifier as the cross-referencing scheme.
	Symbol Striping. A horizontal line is drawn within, completely across, and near the top of the symbol, and a reference to the detailed representation is placed between that line and the top of the symbol. The terminal symbol shall be used as the first and last symbols of the detailed representation. The first terminal symbol contains the name reference which also appears in the striped symbol. The location of the detailed representation chart is placed above and to the left (x) of the vertical bisector of the striped symbol if the same striped symbol appears more than once in the set of flowcharts. Otherwise, the detailed representation chart number is formed by prefixing the chart number containing the striped symbol to the symbol number on that chart, c.n . In either case, the chart number is placed above and to the left of the vertical bisector of the entry terminal symbol.
	Connector Identification. A common identifier, such as an alphabetic character, number, or mnemonic label (A) is placed within the connector as shown. Additional cross-referencing for off-page connectors shall be the page number, p , placed above and to the left of the vertical bisector of the symbol. NOTE: The use of such connectors to off-page continuations is discouraged except for decision structures with too many branches to fit on one page.

Table 4-1. Symbol Usage in Flowcharting (sheet 3)

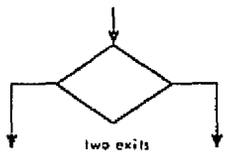
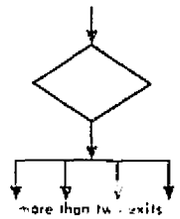
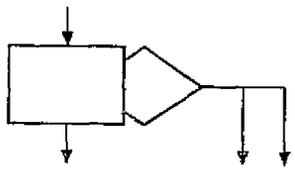
Illustration	Usage
  	Multiple Control Flow Branches. Multiple branches from a symbol are restricted to the decision symbol. The text within the symbol shall state the explicit predicate or event which causes the branch. For each conditioned branch, each exiting flowline is to be labeled by text which identifies the predicate outcome. Normally, true exits to the left, false to the right in binary decisions; multiple branches exit in case-order from the left (if there is an explicit case order). Event-actuated branches need only annotate exiting flowlines when there are multiple events depicted by a single decision symbol.

Table 4-2. Basic Flowchart Symbols (sheet 1)

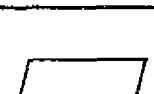
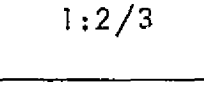
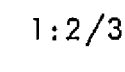
Symbol and Dimensions (Width: Height)	Meaning
 1:2/3	Input/Output Symbol. Represents an I/O function or medium, such as making available information for processing (input), or recording processed information (output).
 1:2/3	Process Symbol. Represents any kind of processing; for example, the process of executing a defined operation or group of operations resulting in a change in value, form, or location of information.
1:2/3	Decision Symbol. Represents a specific decision or switch operation that determines which of a number of alternate paths is to be followed. This symbol may not be striped.
 1:2/3	Comment, Annotation. Used to enclose descriptive comments or explanatory notes as clarification. The broken line is connected to any symbol where the annotation is meaningful.
	Sequence, Control Flow. Program sequence is indicated by single-line arrows connecting symbols. Arrowheads are necessary to show direction. Use only one arrowhead per flowline, at its end.
	Information, Data Flow. Flow of information or data is indicated by double-line arrows connecting symbols. Arrowheads are necessary to show direction. Tails are optional, but recommended.

Table 4-2. Basic Flowchart Symbols (sheet 2)

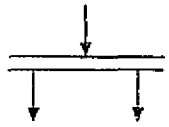
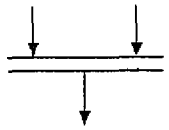
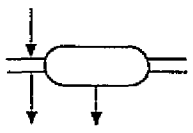
Symbol and Dimension (Width: Height)	Meaning
○ 1:1	Loop Collecting Node. The small open circle represents the iteration point in a looping operation.
⊛ 1:1	Decision Collecting Node. The small asterisked circle represents the merging of alternative flow paths in a program.
	Begin Concurrent Mode, or Fork. Represents the beginning of two or more concurrent (parallel or interleaved) processes.
	End Concurrent Mode, or Join. Marks the end of two or more concurrent (parallel or interleaved) processes.
 1:3/8	Terminal Symbol. Represents the entry or exit point of a flowchart.
 1:3/8	Interrupt Symbol. Represents the enabling (or arming) of an interrupt which may initiate a concurrent (preemptive interleaved) process. The process re-executes each time the event occurs until both processes reach their join. The event identifier (name) is placed in the terminal symbol.

Table 4-2. Basic Flowchart Symbols (sheet 3)

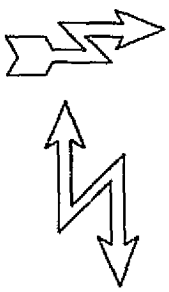
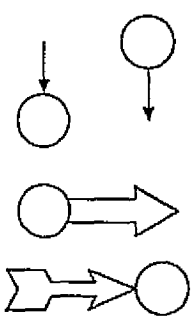
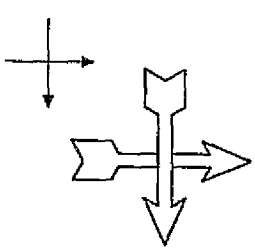
Symbol and Dimensions (Width: Height)	Meaning
	Communication Link. Represents a function in which information is transmitted by a telecommunication link. Arrowheads are necessary to show direction. Tails are optional, but recommended.
	Connectors. Out-connectors and in-connectors for control and data flow. A set of two such connectors represents a continued flow direction when the flow is broken by any limitation of the flowchart. The use of such connectors to off-page continuations is discouraged except for multiple decision structures with too many branches to fit on one page.
	Crossing of Flow Paths. Flowlines may cross; this means they have no logical interrelation. Crossing of flowlines is discouraged except when absolutely necessary.

Table 4-3. Specialized I/O Flowchart Symbols (sheet 1)

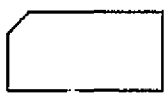
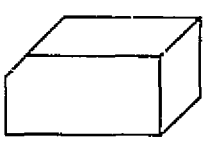
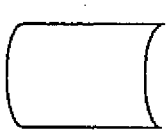
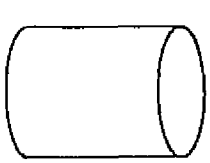
Symbol and Dimensions (Width: Height)	Meaning
 1:1/2	Punched Card. Represents an I/O operation in which the medium is punched cards.
 5/4:2/3	Deck or File of Cards.
 1:2/3	On-Line Storage. Represents an I/O function utilizing any type of on-line storage, such as magnetic tape, drum, or disk.
 1:1	Magnetic Tape. Represents an I/O function in which the medium is magnetic tape.
 1:1/2	Punched Tape. Represents an I/O function in which the medium is paper tape.
 5/4:2/3	Magnetic Drum. Represents an I/O function in which the medium is a magnetic drum.

Table 4-3. Specialized I/O Flowchart Symbols (sheet 2)

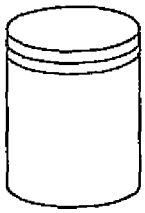
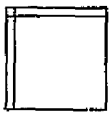
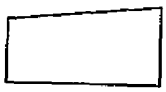
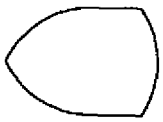
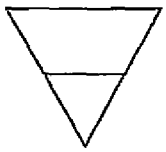
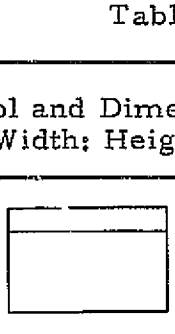
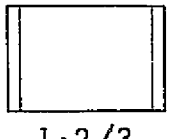
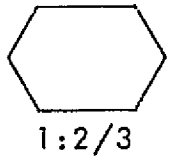
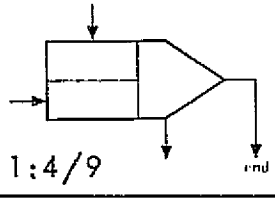
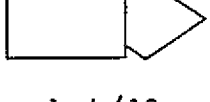
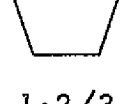
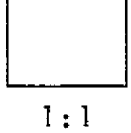
Symbol and Dimensions (Width: Height)	Meaning
 2/3:5/4	Magnetic Disk. Represents an I/O function in which the medium is a magnetic disk.
 1:1	Core. Represents an I/O function in which the medium is core storage.
 1:2/3	Document. Represents an I/O function in which the medium is a document. It is used often to denote output of hard-copy material on either line printers or typewriter terminals.
 1:1/2	Manual Input. Represents an input function in which information is entered manually during processing, such as by on-line keyboards, switches, or pushbuttons.
 1:2/3	Display. Represents an I/O function in which the information is displayed for human use at the time of processing by means of online indicators, video devices, console printer, plotters, etc.
 1: $\sqrt{3}/2$	Off-Line Storage. Represents the function of storing information off-line, regardless of the medium on which the information is recorded.

Table 4-4. Specialized Process Symbols (sheet 1)

Symbol and Dimensions (Width: Height)	Meaning
 1:2/3	Striped Module. Represents a named module (subprogram or subroutine) which has a more detailed representation elsewhere in the same set of flowcharts. Similar horizontal striping conventions apply to other symbols as well, when they are detailed in this way.
 1:2/3	External Module. Represents a named module (subroutine) or logical unit which is not detailed in this same set of flowcharts. Similar vertical striping conventions apply to other symbols as well, when they are detailed elsewhere.
 1:2/3	Preparation. Represents the preparation of a medium for processing, such as obtaining core storage, declaring data structures, or initializing variables.
 1:4/9	Indexed Looping. Represents loop initialization, predicate testing, and update functions. Testing always follows every initialization and update.
 1:4/10	Non-normal Exit. Represents the exit from a process due to abnormal or paranormal events.
 1:2/3	Manual Operation. Represents any off-line process geared to the speed of the human being without using mechanical aid.
 1:1	Auxiliary Operation. Represents an off-line operation on equipment not under direct control of the central processor.

CHAPTER 5

THE STANDARD PRACTICE FOR THE SOFTWARE OPERATOR'S MANUAL

SECTION I
INTRODUCTION

A. PURPOSE OF THIS STANDARD PRACTICE

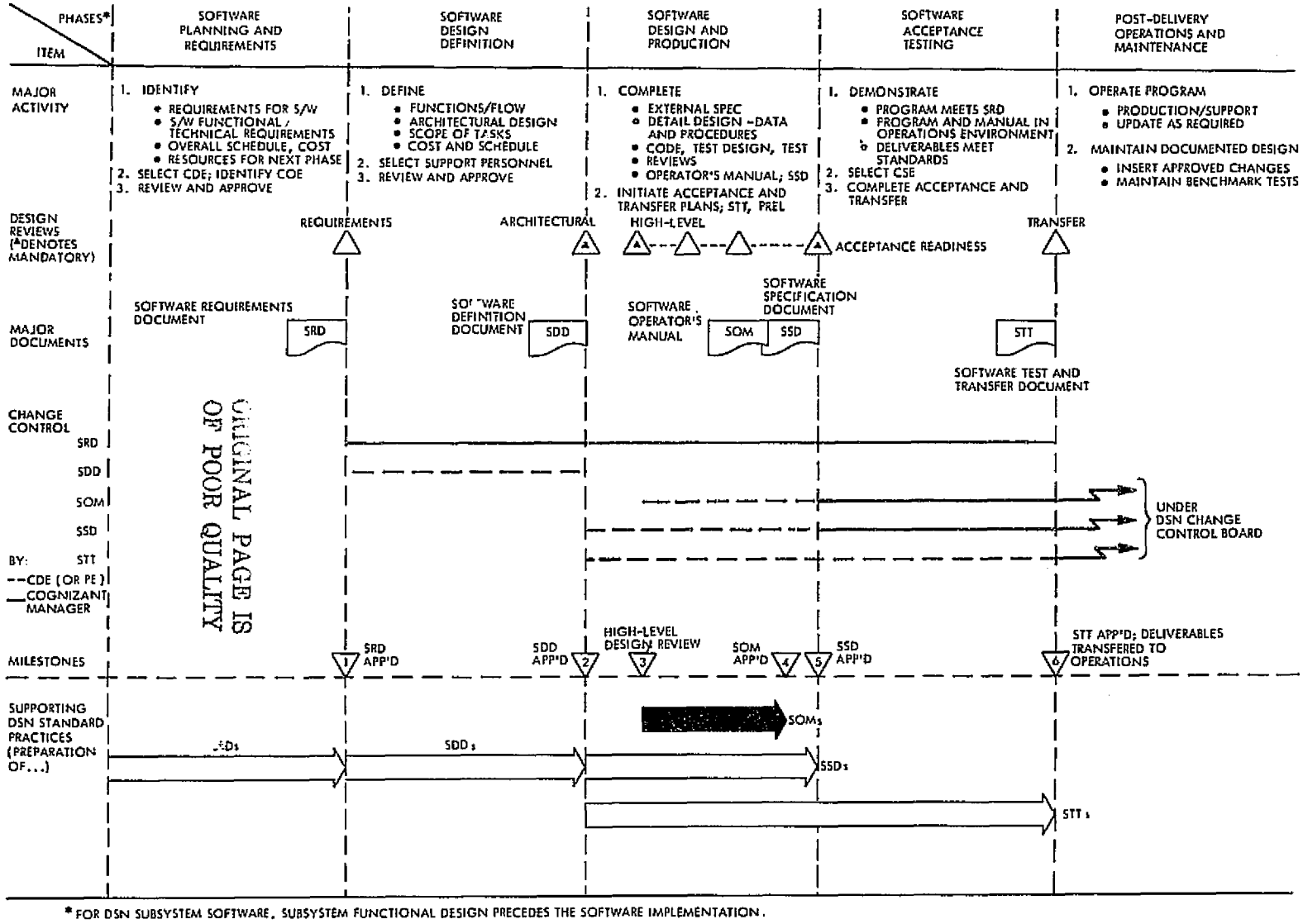
The Software Design and Production Phase and its relationship to the overall software implementation process are shown in Figure 5-1. As shown in the figure, the Software Operator's Manual is formally initiated at the High-Level Design Review and is completed and approved prior to the Acceptance Readiness Review.

The SOM is used throughout the Software Acceptance Testing phase, where changes are under internal project control and authorized jointly by the CDE and COE. Following transfer to operations, changes are implemented only by approved Engineering Change Orders (ECO), with possible exceptions for cosmetic or explanatory changes as recommended by the COE or as directed by the DSN Change Control Board.

B. SCOPE OF SOFTWARE OPERATOR'S MANUAL

The SOM contains the information and instructions needed to execute the program by an otherwise experienced operator without recourse to the implementer for assistance. When required for specific programs, the SOM also contains user information for program applications. The scope of coverage of the SOM includes

(1) A complete set of operating instructions, procedures, and input directives to execute the program in its operational environment to obtain its full output capability.
(2) When required, user instructions to fully utilize and understand the program operation for broad application of its output.



*FOR DSN SUBSYSTEM SOFTWARE, SUBSYSTEM FUNCTIONAL DESIGN PRECEDES THE SOFTWARE IMPLEMENTATION.

Figure 5-1. DSN Software Management and Implementation Plan (SOM Software Design and Production)

(3) Operating examples and operations information in living appendices to allow easy updating, as required from implementing approved Engineering Change Orders.

(4) Self-teaching techniques for ease of use and training, such as use of sample runs, self-explanatory error messages and diagnostics, narratives, etc.

(5) The above items, documented in a survivable, maintainable form, since they ultimately become the most used and most valuable in terms of program utility throughout the operational life of the program. The level of detail and degree of format formality will be specified in the program Software Requirements Document. Standard classes of detail and categories of format medium are defined in Reference 1.

SECTION II
SOM CONTENTS

A. CONTENT OUTLINE

A typical SOM content outline is shown in Figure 5-2, along with an identification of personnel responsible for preparing, concurring in, and approving the SOM. Introductory information is followed by environmental considerations and constraints, detailed operating instructions, and key sample runs. Other helpful information or user instructions, as identified in the SRD, are also included. Detailed supporting materials for program operations and use are provided as appendices.

The outline lists the minimum content for program operations. For a given program, the actual contents may vary and will depend on the complexity of the program operations and interfaces. However, the sections and paragraphs listed in Figure 5-2 are considered to be basic and, in almost all cases, necessary. The software requirements document is used for establishing exceptions or additions to the SOM content. At a minimum, the information contained in the SOM should be sufficient to allow an experienced operator to execute the program in its operational environment.

For review information and criteria, refer to Section III, Paragraph B, Review and Approval.

B. OUTLINE DISCUSSION

1. Introduction

a. Item 1.1, Purpose and Scope of SOM. The purpose of the SOM is stated, along with the scope of coverage and a brief summary of the document. Information in the SOM is presented for ease of use by operations personnel.

1. INTRODUCTION

   1.1 Purpose and Scope of SOM

   1.2 Overview of Program Operation

   1.3 Applicable Documents

2. OPERATIONS ENVIRONMENT

   2.1 Hardware Configuration

   2.2 Software Configuration

   2.3 Operating Constraints

3. OPERATING INSTRUCTIONS

   3.1 Setup and Initialization

   3.2 Control Instructions and Sequences

   3.2.1 Starting Run

   3.2.2 Mode Control and Selection

   3.2.3 Normal Termination

   3.2.4 Aborting and Recovering Run

   3.3 Emergency or Precautionary Procedures

4. SAMPLE OPERATIONS AND PROCEDURES

   4.1 Baseline Sample Runs (selected input, instruction set, and results)

   4.2 Sample Variations (Brief case narratives with all detail appended)

5. OTHER (User Information, Helpful Observations, and Other SRD-Specified Items)

6. APPENDIXES

   6.1 Operator Inputs, Error Message, and Diagnostic Tables (if required)

   6.2 Other Tables and Figures

   6.3 Detailed Operating Examples (supports item 4.2 above)

SIGNATURES	
Prepared by:	CDE COE (Assists)
Concurred and Approved by:	COE (and SE for DSN Subsystem Software)

Note: Approval indicates readiness for use in acceptance testing.

Figure 5-2. Typical Outline for a Software Operator's Manual

Therefore, it is appropriate to identify the operations organization, or possibly even define the operator background or experience that was assumed for preparation of the SOM.

b. Item 1.2, Overview of Program Operation. Overall program operating characteristics are briefly described for operator orientation to detailed material covered in the SOM. Key operator actions and responses can be identified, where appropriate. Typical applications are described in general terms, with emphasis on any special operational features or characteristics. Inputs and outputs for the major operating modes can be identified. The use of a top-level "Operating" chart (Chart 0) and the use of "Operational State" diagrams can facilitate descriptions. These charts and diagrams are described in Reference 1.

c. Item 1.3, Applicable Documents. All documents required to operate the program are listed, along with the location of the document and/or how it is obtained. Other appropriate reference documents may also be listed.

2. Operations Environment

This section of the SOM provides system information that is oriented toward the operating characteristics and special features involved and also identifies any operating constraints imposed on the program by its operational environment.

a. Item 2.1, Hardware Configuration. The system in which the program will be installed is identified. Required peripheral equipment that is accessible to the operator is identified in terms of quantity, arrangement, unusual features, and special conditions. If applicable, variations in hardware configurations for different program operational modes are described. Special characteristics of devices used for storage, input, output, transmission, reproduction, etc., should be included as appropriate, as well as any special checkout or calibration procedures.

b. Item 2.2, Software Configuration. The general description of the overall program operation contained in Section 1.2 of the SOM is expanded to provide operating detail on interfacing programs, routines, libraries, and any special supporting software needed to operate the program. Also, any operator interactions with special data base software or data management and information systems software are described. Detail might include, for each, such items as the identification of the storage medium, access protocol, the listing of external files, registers, or buffers, or any other software operating detail and/or constraints for interfacing with peripheral equipment such as disks, drums, magnetic tapes, consoles, etc.

c. Item 2.3, Operating Constraints. All operating constraints placed on the equipment, software design, and operational environment are identified. Any precautionary measures for operating the program are emphasized. Precautionary measures or fail-safe features of the equipment or software design are identified for operator awareness, such as the use of default options (to avoid operator type-ins for standard operations), the effect of illegal inputs, the extent of diagnostics, etc.

3. Operating Instructions

Complete instructions are provided to set up and operate the program effectively and to handle any emergency conditions that may arise during the program operation. Emergency procedures are noted throughout, as appropriate. Also, important error diagnostics are discussed as they arise throughout these operating sequences. If required, a composite list of all operator inputs, error messages, and diagnostics, with an explanation of each item, are appended.

a. Item 3.1, Setup and Initialization. The program setup, loading, and initialization procedures are described. The operations environment information contained in Section 2 may be referenced as appropriate. If different system, equipment, or program configurations are needed for different types of

runs (such as pre-run checkout or calibration runs), then setup and initialization information for each is provided. Critical error messages and operator responses for setting up and initializing the program are provided.

b. Item 3.2, Control Instructions and Sequences. Control instructions and input directives are identified as to function, format, and operational characteristics. Sufficient detail is provided to enable computer operating personnel, and possibly others, to make effective use of the program, including run start, run control, and termination of operations.

1) Run Start. All information needed by an operator for starting a run is provided in step-by-step detail. Reference to Section 3.1 of the SOM can be made. Typical information that an operator needs for starting a run includes

(a) The run identification and all required inputs.
(b) The starting sequence performed by the operator.
(c) Appropriate operator responses to error diagnostics during start-up.
(d) Helpful operating characteristics, such as the estimated run time, any expected processing delays, etc.
(e) Operator start-up precautionary or emergency procedures.

2) Run Control. Run control during processing is defined. When applicable, control options are identified, along with their purpose and operational sequence needed for their execution. Selection criteria and detailed operating instructions required for each option or mode are provided. Decision Tables, as described in Reference 1, may be used to present this detail. Appropriate operator responses to error diagnostics during the runs are described.

3) Normal Termination. Detailed operator information is provided for normal termination of a run and includes returning the program and all files referenced, updated, or created to a secure state. Also, system sign-off and equipment shutdown procedures are described.

4) Premature Termination, Restart, Recovery. Procedures, sequences, or individual commands for terminating a run prematurely at selected operating points that provide restarting capability are identified; the corresponding restart sequence is described in step-by-step detail. Also, recovery procedures are described for aborted runs due to operator errors, bad data, etc.

c. Item 3.3, Emergency or Precautionary Procedures. This paragraph contains a complete listing of all emergency or precautionary procedures, with detailed information on when and how to invoke them. This listing serves as a readily accessible single source of all emergency procedures applicable to operating the program.

4. Sample Operations and Procedures

a. Item 4.1, Baseline Sample Runs. For each type of run or operational mode, a sample run is provided that consists of a selected set of input, a step-by-step sequence of operating instructions, and a sample of the output. The intent is to provide sufficient information to allow a new (or occasional) operator to become familiar with operating the program by performing or viewing practice runs.

b. Item 4.2, Sample Variations. Variations to the baseline sample runs of Paragraph 4.1 can be described to convey special features or unusual capabilities of the software. Also, any required modification or new operating characteristics that might arise later due to ECOs can then be included here. Information should be brief and in narrative form, with all supporting detail appended and referenced.

5. Other Information

The Software Requirements Document may require additional operator or user information to be provided in the SOM. For example, information for the broad use and understanding of the program and the application of its output

may be required, along with SOM operator information on the effective execution of the program. This other information, as identified in the SRD, is provided in this paragraph; see Chapter 4 for guidelines on preparation of user-oriented information. Any other operating information that can be helpful to the program operator or user in executing or utilizing the program is provided here.

SECTION III
SOM PREPARATION, REVIEW, AND APPROVAL

A. PREPARATION GUIDELINES

1. Top-Down, Concurrent SOM Generation

The CDE prepares the SOM in parallel and concurrently with the program construction and correctness test activities. Emphasis is placed on delivering complete and effective information for executing the program through all its options and capabilities. Generally, as the program construction proceeds in a top-down manner, operational information in greater and greater detail is added. Therefore, the SOM is prepared by the CDE as a natural consequence of this information becoming available (without requiring a significant "added on" documentation effort). The SOM information should be adequate for running the current program build and also for inspections, reviews, and audits. Evolving program operating information is discussed below.

a. Environment. Effective program operation, whether during implementation or operations activities, depends heavily on adequate identification and description of the environment in which the program is embedded. Emphasis in the SOM is placed on the post-transfer operations environment. Any temporary or permanent operating constraints or limitations that the SOM must accommodate are identified as they become known, and they are later verified during acceptance testing.

b. Operator Inputs, Error Messages, and Diagnostics. Operator inputs, error messages, and diagnostic information are collected and verified as the program implementation progresses on a build-by-build basis.

c. Program Constraints and Limitations. This information is similarly collected, explained, and made available across the project throughout the implementation. The operating constraints that apply to the final program are

consolidated in Section 2 (Operations Environment) of the SOM, but they should also appear throughout the SOM, as appropriate, in the form of notes, precautions, or emergency procedures.

d. Sample Operations. As major builds or capabilities are completed, sample runs can be generated. Selected samples are included in Section 4 (Sample Operations and Procedures) of the SOM to provide for familiarization and training in program operation.

e. User Information. Where user instructions are needed for a user to fully and effectively utilize the program, these user instructions are to be provided in Section 5 of the SOM (see Section IV, Paragraph B).

2. Quality and Maintainability

As major builds are implemented top-down, complete and detailed operating instructions are prepared and provided for build-by-build correctness testing. Thus test execution provides a verification of the operating instructions, as well as the build performance. The effects of any identified deficiencies or workarounds (temporary or permanent) are noted and, if feasible, corrected as soon as they become known.

B. REVIEW AND APPROVAL

The outlined SOM is reviewed, along with other program and test items, for management concurrence at the program High-Level Design Review held early in the software design and production phase. At the end of the production phase, the completed SOM is approved by the COE and then submitted for general review at the Acceptance Readiness Review. The following paragraphs describe typical SOM-related subjects covered during these reviews.

1. High-Level Design Review

a. Agenda. SOM-related items for the High-Level Design Review typically include

(1) Any carry-over items from the Architectural Design Review affecting program operability.
(2) SOM Content Outline (draft).
(3) Present SOM production plans (if different from the published, approved plans, work breakdown structure, and schedules contained in the SDD).
(4) Identification of differences between the "implementation" and "post-transfer operational" environments and interfaces that the SOM generation and verification process must accommodate.
(5) Other pertinent SOM items, problems, and concerns.

b. Evaluation Criteria. The following criteria can be used during the High-Level Design Review to assess the adequacy of the proposed SOM and the soundness of its production plan. There should be evidence that

(1) The SSE, COE, QA Representative, and others have sufficient visibility into the SOM development.
(2) Near-term schedules for the production of the SOM reflect present progress and experience.
(3) The SOM plan incorporates the SOM-related requirements contained in the SRD, if any.
(4) The standard DSN documentation preparation and formatting conventions are being used as described in Chapter 4.

c. Concurrence. Concurrence to proceed with the implementation of the program and production of the deliverable documentation is based on the evaluation of the program status and implementation plan made at the High-Level Design Review. In some cases, the concurrence may be "conditional," depending upon the solution to significant problems.

2. Acceptance Readiness Review

a. Agenda. SOM-related items for the Acceptance Readiness Review typically include

(1) Any carry-over items concerning the SOM from the High-Level Design Review.
(2) SOM status and present deficiencies affecting program operations.
(3) SOM deficiency workarounds and completion dates.
(4) Anticipated SOM-related limitations of a permanent nature.
(5) SOM QA inspection status, if applicable.
(6) Other pertinent concerns.

b. Acceptance Criteria. The following SOM-related criteria can be used as aids in determining readiness for acceptance testing. There should be evidence that

(1) The SOM is complete with a full set of instructions, directives, and procedures, along with needed operating information.
(2) Using only the SOM, an operator can execute the program in its operational environment; operator workarounds, as needed, appear to be adequate.
(3) All nonself-explanatory diagnostics and error messages are explained adequately in the SOM.
(4) All known constraints and limitations are covered in the SOM.
(5) All emergency conditions and procedures are adequately noted, flagged, or otherwise emphasized.
(6) The SOM is self-teaching and easy to use.
(7) The SOM can be acceptance tested, as the program itself is undergoing acceptance testing.
(8) The SOM has been reviewed and concurred with by the COE (and SE for DSN Subsystem Software).

c. Approval. The SOM is approved prior to the Acceptance Readiness Review by the COE (and also by the SE for DSN Subsystem Software); other required approvals (line managements, etc.) are specified in the SRD, as required. Approval indicates that the SOM can support acceptance testing. Requested changes resulting from acceptance testing activities require the concurrence of both the CDE and COE (and SE for DSN Subsystem Software).

SECTION IV
PREPARATION AIDS

A. SOM FORMATTING CONVENTIONS

The CDE is directed to Chapter 4 for detailed information on formatting conventions to be used for SOMs (as adopted for the other surviving DSN documents — the SSD and STT). The SOM is given an identification number by prefixing "SOM" to the program number that was assigned by the DSN Program Library.

B. GUIDELINES FOR SUPPLEMENTARY USER INSTRUCTIONS

User instructions are needed in addition to operational instructions whenever the user of a program is not the operator of the program, and perhaps at other times as well. The user generates or prepares data (or causes them to be generated or prepared), submits them (or has them submitted) for operations (either conversationally or in batch), and uses the output (if any) for an intended task, including the task of gaining insight into a problem. In cases where a user causes a data base to be updated, that output may not be immediate, or may not even be the result of operating the same program. Typical items needed in these instances are summarized below. Further detail on user instructions and information is presented in Reference 1.

1. Description and Background

This section describes the program usage, perhaps where it fits into a software system, what major functions are performed, how much runs cost (dollars or execution time), major limitations, and other items of a background nature.

2. Program Capabilities and Use

This possibly lengthy section describes how to prepare data or instructions to "operations" in order to apply the program to problems which the program can handle, or to achieve a desired output. This includes such information as

(a) Acceptable input data units.
(b) Features of the program, processing available, output data generated.
(c) Modes of operation.
(d) Interpretation of results.
(e) Typical application guidelines.
(f) Restrictions or limitations on applications.
(g) How to diagnose errors in runs.

3. Operations Interface

This section describes the user-operations protocols necessary to submit input, run the program, and receive the output. Included are such things as

(a) Initial protocols (account codes, passwords, file assignments, etc.).
(b) Control data and selection of options.
(c) Source data creation/input/formatting/update.
(d) Output handling (e.g., routing).
(e) Error handling.
(f) Interrupt/recovery.
(g) Turn-around time.
(h) Interactive vs batch protocols.

4. Theory of Operations

This section, when needed, would include not the theory of the software itself but rather the theory behind the model or "black box" specification that the software implements. This would be the case, for example, for programs implementing a management science model. However, if the method of solution is necessary to understand the numerical accuracy, then this should be included.

CHAPTER 6

THE STANDARD PRACTICE FOR THE SOFTWARE TEST AND TRANSFER DOCUMENT

SECTION I
INTRODUCTION

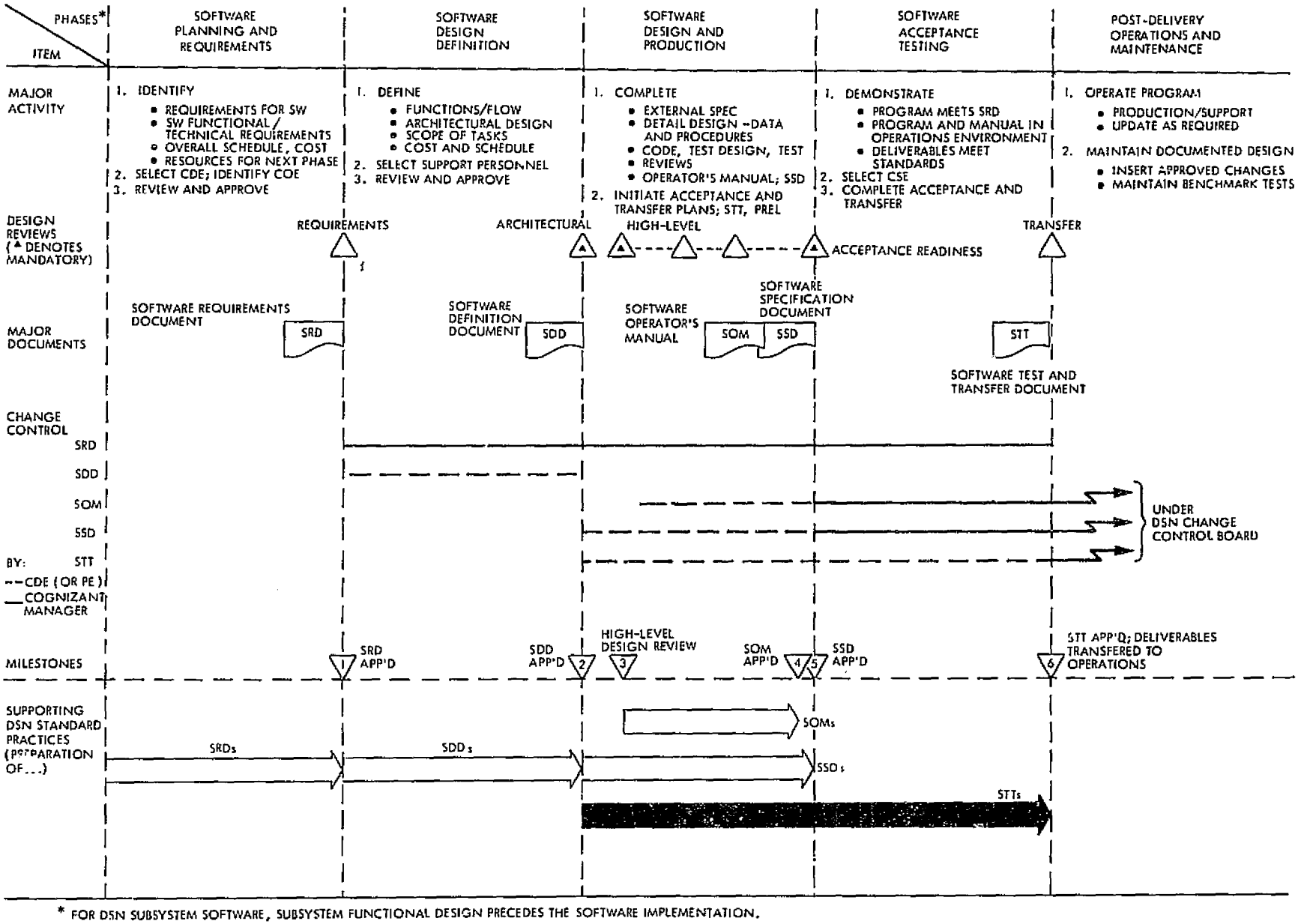
A. PURPOSE OF THIS STANDARD PRACTICE

The Software Acceptance Testing Phase and its relationship to the overall software implementation process are shown in Figure 6-1. As shown in the figure, the acceptance testing and transfer activities are initiated after the Software Definition Document is approved and continue in parallel with the program construction activities. Program as-built and operating information is documented concurrently with the program construction itself in the Software Specification Document and in the Software Operator's Manual. STT documentation of the program acceptance, transfer, and as-tested information is addressed further in this chapter.

B. SCOPE OF SOFTWARE TEST AND TRANSFER DOCUMENTS

Acceptance test planning and proposed tests are documented in a preliminary STT and reviewed at the Acceptance Readiness Review prior to the start of formal acceptance testing. Throughout acceptance testing, the STT is updated with actual test conditions and test results. Following acceptance testing, the STT, reflecting the as-tested program and containing key acceptance test information, is approved.

An agreement on the acceptability and conditions of all software deliverables is then documented in a Software Transfer Agreement. Upon approval of the Transfer Agreement, the software products are transferred from implementation status to operational status; the responsibility for and custody of the software transfers from the Implementing Organization to the Operations Organization, with changes implemented only by approved Engineering Change Orders. The implementation project is then complete.



* FOR DSN SUBSYSTEM SOFTWARE, SUBSYSTEM FUNCTIONAL DESIGN PRECEDES THE SOFTWARE IMPLEMENTATION.

Figure 6-1. DSN Software Management and Implementation Plan (STT Software Design and Production, Software Acceptance Testing)

The preliminary STT typically contains the following information for concurrence, before acceptance testing begins:

(1) Implementation and exception status at the time of the Acceptance Readiness Review.
(2) Quality Assurance "as-built" Inspection Report.
(3) Acceptance Test Criteria and Test Plans.
(4) Transfer preparations and plans.
(5) Proposed acceptance test procedures, input data, and predicted results (appended).
(6) Provision for retaining actual test results after acceptance testing (appended).

The final, approved STT typically contains, for reference throughout the life of the computer program, the above six items, updated from the preliminary STT to reflect the actual acceptance testing, especially:

(1) Acceptance test procedures and data (appended).
(2) Summaries of acceptance test results and anomalies (appended).
(3) Provision for maintaining an Engineering Change File after transfer (appended).

SECTION II
STT CONTENTS

A. CONTENT OUTLINE

A typical STT content outline is shown in Figure 6-2, along with an identification of personnel responsible for preparing, concurring with, and approving the STT. Introductory information on the STT and program testing is followed by key implementation and acceptance information. Then, since transfer preparations begin before acceptance testing starts, possibly deficient transfer-deliverables and other key items that might be at issue during the transfer process are identified. Appended to the preliminary STT are the proposed acceptance test procedures, input data, and predicted test results. The information is revised and test results from the acceptance tests are appended to preserve the key as-tested information and results.

The outline covers the major items needed by management and other reviewers to assess the readiness for acceptance testing and maintenance after transfer. For a given program, the actual contents may vary and will depend on the complexity of the program interfaces. However, items starred (*) in Figure 6-2 are considered always necessary. Conversely, there could well be items in the table of contents other than those listed. The SRD can be used for identifying exactly what items of an optional nature should appear in the STT. At a minimum, the information and procedures contained in the STT should be sufficient for verifying the quality, accuracy, and completeness of the software deliverables in meeting the SRD requirements.

For Review information and criteria, refer to Section III, Paragraph B, Review and Approval.

*1. INTRODUCTION

- *1.1 Purpose and Scope of the STT
- *1.2 Program Acceptance Overview
- *1.2.1 Functions, Interfaces, Environment
- 1.3 Reference Material

*2. ACCEPTANCE READINESS

- *2.1 Implementation Status and Exception Identification
- *2.2 QA Inspection Status
- *2.2.1 Module-by-Module Code Audit

*3. ACCEPTANCE BASIS

- *3.1 Acceptance Test Objectives, Philosophy, and Approach
- *3.2 Acceptance Test Criteria
- *3.3 Acceptance Test Plan

*4. TRANSFER PREPARATIONS

- *4.1 Deliverable Items
- 4.2 Transfer Status Projections

*5. APPENDIXES

- *5.1 Acceptance Test Procedures, Data, and Sample Results
- *5.2 Test Result Summaries and Anomaly Summary Sheets
- 5.3 Change Engineering File (ECOs After Transfer)

*Item considered always necessary.

SIGNATURES	
Prepared by:	CDE (Helpers)
Concurred by:	COE, Cognizant Line Management, and SE, for DSN Subsystem Software (Note)
Approved by:	(See, Concurred by)

Note: Concurrence at the Acceptance Readiness Review (for Test Readiness) and final approval after acceptance testing/evaluation. Transfer Agreement then approved by cognizant management.

Figure 6-2. Typical Outline for a Software Test and Transfer Document

C-3

B. OUTLINE DISCUSSION

1. Introduction

a. Item 1.1, Purpose and Scope of the STT. The purpose of the STT is stated, along with the scope of coverage and a brief summary of the document.

b. Item 1.2, Program Acceptance Overview. An overview of the program and its acceptable operation are briefly described in terms of major program functions, operational interfaces, and its acceptance testing and operations environments. The functions to be tested are correlated with the requirements in the SRD. Differences between the test and operations interfaces and environments that could affect program operation or assessment of its acceptability are described. Acceptability of documentation is also addressed.

c. Item 1.3, Reference Material. Reference material consists of a list of any documentation that is not included as part of the STT or as an appendix, but that is needed to support the acceptance testing and transfer activities. All referenced documents must be readily available.

2. Acceptance Readiness

a. Item 2.1, Implementation Status and Exception Identification. The implementation status (i.e., the completeness of the program and its supporting Software Specification Document and Software Operator's Manual, and the status of acceptance test preparations) is provided. The implementation status information is used in the Acceptance Readiness Review to help assess readiness for proceeding to the formal acceptance testing and transfer activities.

b. Item 2.2, QA Inspection Status. A summary of the results of the ongoing module-by-module code audits and reaudits conducted by QA throughout the program construction is presented. Deficient or marginal items (both

code and documentation) present at the time of the Acceptance Readiness Review are identified. Following acceptance testing, a QA Certification is performed and reported in the Software Transfer Agreement.

3. Acceptance Basis

a. Item 3.1, Acceptance Test Objectives, Philosophy, and Approach. The basic acceptance test objectives which form the basis for the detailed acceptance test procedures are stated. Additional discussion of test philosophy and rationale, as needed, is also included. The overall test approach, including a summary of acceptance test methods and modus operandi are presented (such as the use of special equipment or test software. Any further helpful guidelines, suggestions, background information, special items of emphasis, or overriding considerations of the specific implementation that facilitate the acceptance and transfer activities should also be presented.

b. Item 3.2, Acceptance Test Criteria. Acceptance test criteria are generated by the initiator, with inputs from the Cognizant Operations Engineer and Cognizant Development Engineer, following SRD Approval. Criteria typically cover items of limits, tolerances, go/no-go, fail-safe, fail-soft, calibrations, standards, etc. They are derived by adding detail to or "expanding" the SRD acceptance requirements. The purpose here is to identify and list (or reference, if extensive) the governing criteria for acceptance, and to update them, if needed, from information gained throughout the program detailed design and construction activities. If the criteria are included by reference, a high-level summary of the underlying principles is provided for reader orientation to the detailed testing and basis of acceptance.

c. Item 3.3, Acceptance Test Plan. The plan provides a general description of the types of tests needed, their priorities, and test support needed, taking into account the actual status of the implemented program, its documentation, and the availability of needed acceptance test equipment, facilities, and other resources. The test plan is presented in outline, but enough detail

is included to show the planned means for accomplishing the testing and to provide specific guidelines for development of comprehensive tests and detailed test procedures.

4. Transfer Preparations

a. Item 4.1, Deliverable Items. The Software Transfer Agreement form, as discussed in Paragraph A.2.b. of Section III, contains an inventory list of deliverable items considered always necessary for transfer; it also has provisions for other items. For particular software implementations, there may be additional items, unique to a specific implementation that must be addressed. Also, a specific software project may have justification for waiving a particular deliverable that is listed as a necessary item on the Transfer Agreement form. Therefore, this section allows any such project-unique items (typically identified in the SRD) to be explained and reviewed before acceptance testing begins.

For the typical case, where all items of the Transfer Agreement apply and there are no additional items needed, a statement to this effect is appropriate here. This provides the acceptance-readiness reviewers with assurance that the necessary look-ahead activity, oriented towards transfer, has been accomplished.

b. Item 4.2, Transfer Status Projections. This paragraph lists and describes any anticipated deficiencies or exceptions that are likely to exist after completion of acceptance testing. These are also indicated on the draft Transfer Agreement, which is reviewed at the Acceptance Readiness Review. The intent here is to identify and initiate action to avoid or minimize these instances (or their impact) by the time of transfer.

5. Appendices

a. Item 5.1, Acceptance Test Procedures, Data, and Sample Results. The purpose of this Appendix is to preserve the acceptance test procedures, input data, and sample (observed) results upon which program acceptance was based. These procedures and data should provide a comprehensive test and

demonstration of the program capabilities. The intent is to facilitate reverification, as may be needed due to future program maintenance, modification, or updating.

1) Acceptance Test Procedures. These should define the software and system configuration and the resources required. Test inputs and sources are identified, and the step-by-step sequence of actions is prescribed that leads to specified results, which are described along with criteria that can be used for their assessment. An overall test for adequacy of the acceptance test procedures is that an individual, not knowledgeable of the specific implementation, could, by using only the procedures, set up and run the tests. If the tests are extensive, the test documents and results may be referenced and only summarily described in the appended material of the STT.

2) Data. An identification and detailed description of program input data, their sources, and key characteristics, such as the data accuracy, range, validity, flowrate, randomness, etc., are provided. A subset of input data may also be provided directly for certain tests.

3) Sample Results. A description of selected observed results, or a sample set of outputs for a given set of inputs, should be provided. The combined information in this Appendix on acceptance test procedures, input data, and sample results should provide the capability needed for future benchmarking; that is, for reverification of original capability, as transferred, using original key tests as the control reference.

b. Item 5.2, Test Result Summaries and Anomaly Summary Sheets. After acceptance testing, summaries of all test results are appended. Differences between expected and actual results are reconciled, and anomalies are identified and documented. Also, any procedure modifications or additional tests that were incorporated are included in this Appendix.

c. Item 5.3, Change Engineering File. This Appendix provides a running log of ECO information, as approved modifications are implemented after transfer.

SECTION III
STT PREPARATION, REVIEW, AND APPROVAL

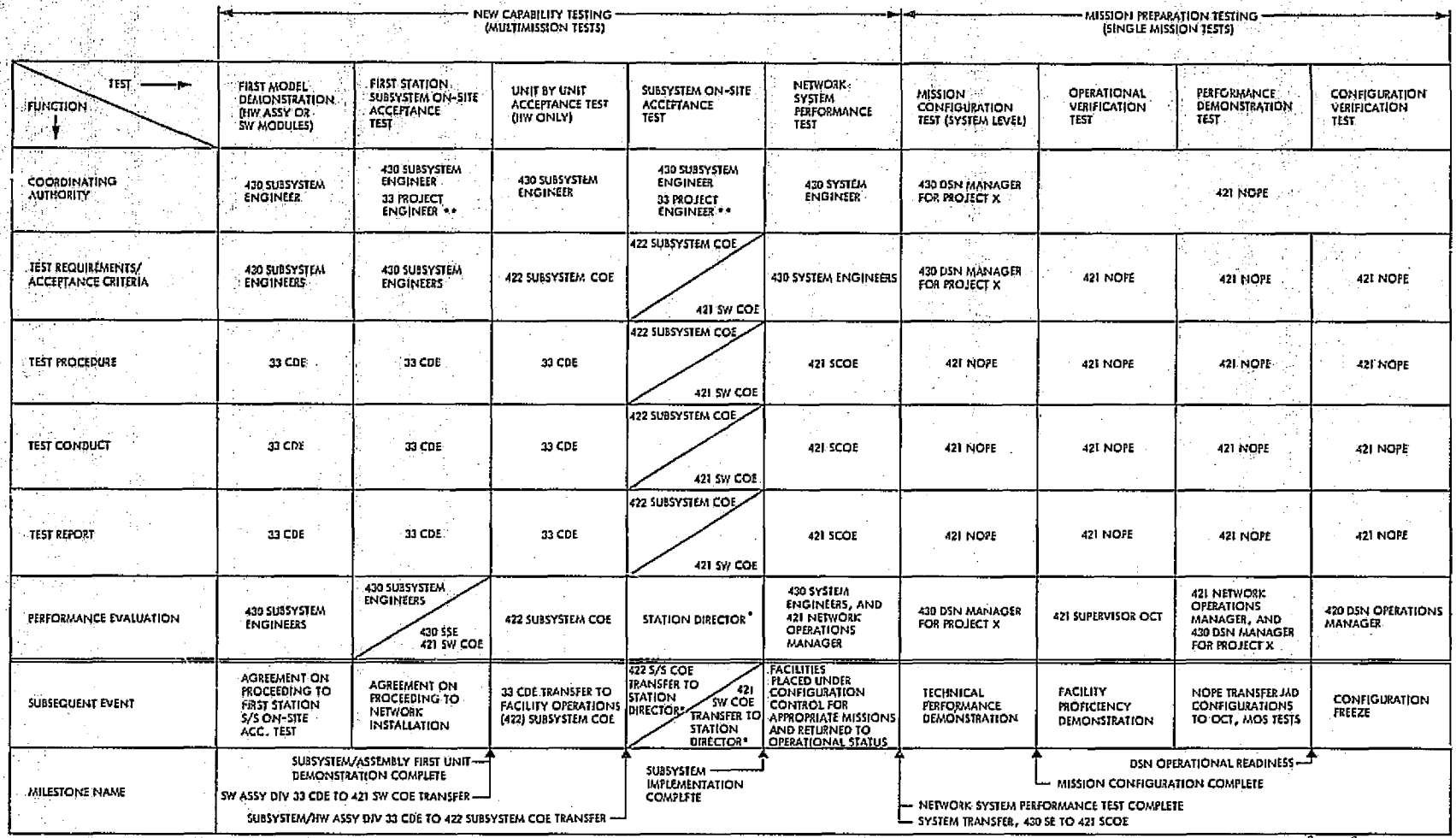
A. PREPARATION ACTIVITIES

The activities and documentation involved in the acceptance process and the transfer process are discussed below. Software implementation functions, including those involved in acceptance and transfer, are identified and described in Section IV of Chapter 1, along with specific responsibilities of each functional task, the functional team structure and organization, and the functional operational interactions. For DSN Subsystem Software, the DSN functional working relations relative to acceptance test and transfer activities were investigated, and results (responsibility matrix) are shown in Figure 6-3. Additional background material on acceptance and transfer of software products is presented in Reference 1.

1. Acceptance

a. General. Basic to the guidelines discussed in Chapter 1 is the acceptance activity, which leads to the transfer of the software from implementation to operational status. Acceptance information and test procedures are reviewed prior to acceptance testing and preserved following acceptance through the use of the STT.

b. Documentation. Using the STT content guidelines of Section II, program acceptance test planning and transfer information is documented in a preliminary version of the STT. After STT concurrence at the Acceptance Readiness Review, the acceptance test procedures govern the actual acceptance testing. Modifications, as may be needed, are under internal project control and at this point are authorized jointly by the CDE and COE. Concurrent with the acceptance testing and data evaluation activities, the STT is updated to reflect the as-tested program and, when completed, is submitted for approval by the line management of the implementing and operations organizations.



LEGEND

- TEST RESPONSIBILITY FOR HARDWARE
- TEST RESPONSIBILITY FOR SOFTWARE
- TEST RESPONSIBILITY INDEPENDENT OF HARDWARE VERSUS SOFTWARE

* OR GCF AND/OR NCS EQUIVALENT
** MAJOR STATION UPGRADE
SSE SUBSYSTEM ENGINEER
CDE COGNIZANT DEVELOPMENT ENGINEER
COE COGNIZANT OPERATIONS ENGINEER
SCOE SYSTEM COGNIZANT OPERATIONS ENGINEER
NOPE NETWORK OPERATIONS PROJECT ENGINEER
OCT OPERATIONS CONTROL TEAM
MOS MISSION OPERATIONS SYSTEM
33, 430, 422, ETC. - DIVISION OR SECTION/ORGANIZATION NO.

APPROVED:
L.W. RANDOLPH
DSN IMPLEMENTATION MANAGER
R.K. MALLIS
DSN OPERATIONS MANAGER
N.A. RENZETTI
DSN SYSTEMS ENGINEERING MANAGER

Figure 6-3. Relationship of Tests, Responsibilities, and Milestones

The STT is updated and maintained throughout the useful life of the program, and formality of formatting is prescribed to aid readability and change engineering. The CDE is directed to Chapter 4 for detailed information on formatting conventions to be used for STTs (as adopted for the other maintained software documents – the SSD and SOM). The STT is given an identification number by prefixing "STT" to the program number that was assigned by the Program Library. Following transfer, changes to the STT are implemented only by approved Engineering Change Orders.

2. Transfer

a. General. Following acceptance testing, an agreement is reached between the involved parties to transfer the responsibility for the software products from the implementation organization to the operations organization. The agreement explicitly identifies the software products and declares their state of completeness in meeting the requirements of the SRD and the extent of their compliance to quality standards; deficiencies, if any, are identified and dispositioned. Upon approval of the Software Transfer Agreement by cognizant management, the software is transferred to operations, delivered to the Program Library, and placed under Change Control.

b. Documentation. The agreement and conditions for transfer are documented using a Software Transfer Agreement. A sample of this three-page form is shown in Figures 6-4, 6-5, and 6-6. Also, an addendum form, as discussed later, is used to document requested changes to an existing Software Transfer Agreement. Preparation of the forms and their approval are discussed below. Using the document numbering system, the Transfer Agreement is given an identification number by prefixing "TA" to its program number (assigned by the Program Library). After approval, the Software Transfer Agreement is delivered to the Program Library, along with the other deliverable items.

SOFTWARE TRANSFER AGREEMENT

TA-_____, Page 1 of _____

FOR _____ AS OF _____
Computer program ID — Date initiated

USE DESTINATION _____ (If applicable)

SYSTEM FOR USE _____ (If applicable)

The cognizant individuals/organizations, listed below, are satisfied that this software (computer program and its supporting documentation) is complete, subject to the exceptions identified, and agree to transfer responsibility for this software from Implementation to the Cognizant Operations organization. Changes to the software after transfer are under DSN Engineering Change Control.

CDE for the Software	Date
DSN QA Section	Date
Subsystem Engineer	Date
COE for the Software	Date
Other	Date

FOR LIBRARY USE

Date received _____

Storage location ID _____

Date of addendum _____

_____

_____

Date deleted _____

Other:

We concur with the above agreement, thereby approving the Transfer:

CDE Section Manager	Date
COE Section Manager	Date
DSN Implementation Manager	Date
Mission Support Operations (or User Organization) Manager	Date
Other	Date

JPL 2743-1 11/76

Figure 6-4. Sample "Page 1" of the Software Transfer Agreement

SOFTWARE TRANSFER AGREEMENT TA-_____ ,Page 2 of _____

FOR _____ AS OF _____

Item 1. The software meets the requirements, criteria, and conditions specified in:

1) SRD-_____

a) Functional requirements of FRD-_____

b) Other (refer to Documentation listed on Page 3 of this Software Transfer Agreement)

_____

2) STT-_____

a) Acceptance test procedure of ATP-_____

EXCEPTIONS:

Item 2. The deliverable software items are ready for transfer to operations and delivery to the DSN Program Library.
NOTE: Refer to Deliverable Item Inventory List, Page 3 of this Software Transfer Agreement.

EXCEPTIONS:

ITEM 3. All software program elements have been audited by JPL Quality Assurance (QA). A QA Certification (Cert. # _____) including the certification date, has been issued and filed with related Inspection Reports for all open or closed QA discrepancies.

EXCEPTIONS OR COMMENTS:

JPL 2743-2 11/76

Figure 6-5. Sample "Page 2" of the Software Transfer Agreement

SOFTWARE TRANSFER AGREEMENT TA-_____, Page 3 of _____

FOR _____ AS OF _____

DELIVERABLE ITEM INVENTORY LIST
(* Item Considered Always Necessary)

DELIVERABLE ITEMS	IDENTIFICATION No.	RELEASE DATE
Computer Code		
* Source Program		
* Executable (Object)		
Other		
Planning & Requirements Documentation		
* SRD	SRD-	
FRD	FRD-	
TRD	TRD-	
Other		
Architectural Design Documentation		
* SDD	SDD-	
Other		
Design & Production Documentation		
* SSD	SSD-	
* SOM	SOM-	
Other		
Acceptance Documentation		
* STT	STT-	
ATP	ATP-	
ATR	ATR-	
Other		

COMMENTS:

JPL 2743-3 11/76

Figure 6-6. Sample "Page 3" of the Software Transfer Agreement

1) Software Transfer Agreement. The Software Transfer Agreement form is filled in by the CDE as information becomes available from the implementation activities. The three pages of the form contain the approval signatures, the software completeness and quality items, and the Deliverable Item Inventory List.

a) Approval Signatures (Page 1). After information is complete and all involved individuals and organizations agree as to the software acceptability and its readiness for operational use, the Software Transfer Agreement is signed by the designated personnel, in the spaces provided on Page 1 of the form (see Figure 6-4), to indicate their approval. Approval places the software under change control and transfers responsibility for the delivered software products from the implementing to the operations organization.

All computer software transferred to operations, including the Software Transfer Agreement itself, is placed in the Program Library. In general, the Transfer Agreement is approved concurrently with delivery to the Program Library. Computer software is subsequently reproduced and distributed by the Program Library, as appropriate. It is intended that a Software Transfer Agreement be approved before the software is required or committed to the use and support of operations.

b) Software Completeness and Quality Items (Page 2). The spaces provided on Page 2 of the form (see Figure 6-5) are filled in to indicate the software's compliance with the SRD requirements and its STT acceptance test procedure, its completeness and readiness for delivery, and the identification of its QA certification report.

If any deliverable item is deficient, the deficiency is described under the "Exceptions" heading, along with the date that the deficiency will be corrected. If several deficiencies exist, they can be briefly identified and numbered on the form and then described in more detail, using additional blank pages that can be attached after Page 3 of the form.

The implementer is responsible for completion of all exceptions noted on the approved Transfer Agreement, including the rescheduling, if needed, of due dates prior to expiration of those presently accepted and scheduled. It should be noted that if an exception is not corrected on or before the due date, the Software Transfer Agreement becomes subject to review and cancellation by the Change Control Board. Use of the Addendum, as discussed below, facilitates the changing of the due date, if needed. To be effective, it must be initiated prior to the present due date.

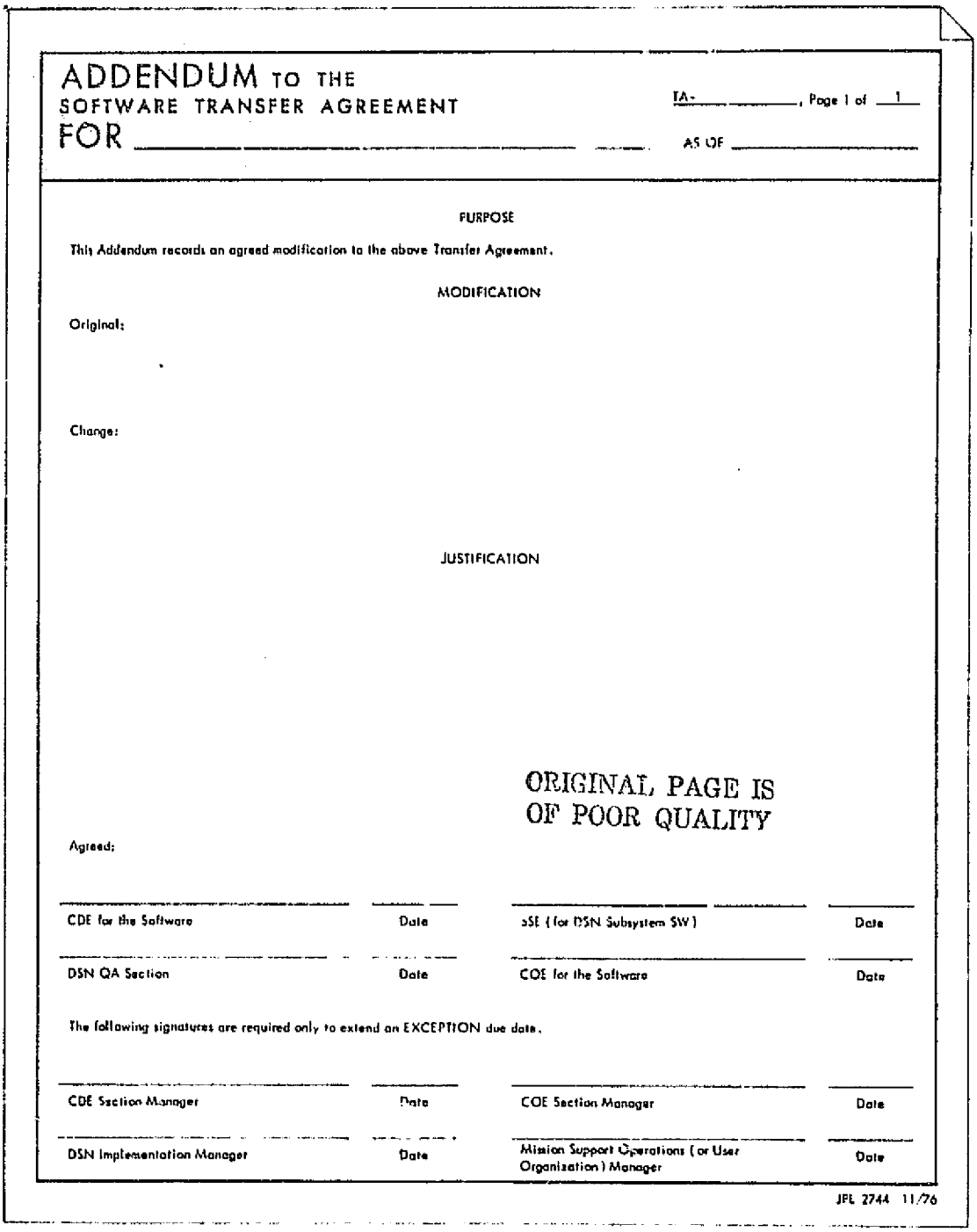
If it is agreed by all concerned that an identified discrepancy need not be corrected, this should be noted, along with the description of the discrepancy. No further action by the implementer is required in these instances.

c) Deliverable Item Inventory List (Page 3). Provision for logging the release dates of individual deliverable items is provided on Page 3 (see Figure 6-6). This supplies supporting information for completing Page 2 of the Software Transfer Agreement. The inventory list includes space for additional or other (such as project-peculiar or special) deliverable items, above those normally required for transfer. Items that do not apply should be so noted. Section IV provides summary information on the deliverables considered to be always necessary for transfer and delivery to operations.

2) Addendum. Figure 6-7 presents a sample Addendum that is used by the implementer to request a modification to an existing Software Transfer Agreement. Modifications are typically needed either

(a) To indicate completion of an exception or exceptions described on the approved Software Transfer Agreement, or

(b) To extend the due date for completion of an exception or exceptions from the current due date scheduled on the approved Software Transfer Agreement or subsequent Addenda.



ADDENDUM TO THE
SOFTWARE TRANSFER AGREEMENT
FOR _____

TA-_____, Page 1 of 1

AS OF _____

PURPOSE

This Addendum records an agreed modification to the above Transfer Agreement.

MODIFICATION

Original:

Change:

JUSTIFICATION

ORIGINAL PAGE IS
OF POOR QUALITY

Agreed:

CDE for the Software	Date	SSE (for DSN Subsystem SW)	Date
DSN QA Section	Date	COE for the Software	Date

The following signatures are required only to extend an EXCEPTION due date.

CDE Section Manager	Date	COE Section Manager	Date
DSN Implementation Manager	Date	Mission Support Operations (or User Organization) Manager	Date

JPL 2744 11/76

Figure 6-7. Sample "Addendum" to the Software Transfer Agreement

Implementer responsibility terminates upon completion of all exceptions and approval of the corresponding Addenda. The approved Addenda are furnished to the Program Library, where they are attached to the Software Transfer Agreement. This provides a current record of the transferred software with respect to the exceptions noted at the time of transfer.

All modifications to the transferred software (excluding the exceptions discussed above) are made only by approved Engineering Change Orders; typically Addenda sheets are sufficient to update the Software Transfer Agreement to reflect the current status of the transferred software. In some instances, the Operations organization may instead request a new Transfer Agreement. Also, for certain major extensions* to existing programs, a new SRD may be required.

The following apply for preparing the Addendum for approval:

(a) The software identification should be the same as recorded on the original Software Transfer Agreement.
(b) The modification is summarized in terms of the original conditions and the needed changes.
(c) If applicable, the old and new exception due dates are listed.
(d) A brief justification for the modification is provided.
(e) Designated signatories indicate their agreement.

B. REVIEW AND APPROVAL

The STT is reviewed in preliminary form in the Acceptance Readiness Review after program construction is complete and then in final form after acceptance testing is complete. An optional Transfer Review may be held to facilitate the review of the final software deliverables. The reviews are briefly discussed below.

*As determined by the Change Control Board, but typically major extensions require at least one person-year of effort.

1. Acceptance Readiness Review

The Acceptance Readiness Review is held after the program detailed design and construction activities have been completed. The purpose is to assess both the status of the program and its as-built specification (SSD) and operator's manual (SOM) and the status of the acceptance test preparations, plans, and procedures to determine if meaningful, effective acceptance testing can begin.

The CDE has overall responsibility for this review. This includes setting the time, place, and agenda and also issuance of review meeting notices and material in advance, and minutes of the review after the meeting. The CDE (or the CDE's management) conducts the review. The Acceptance Readiness Review board participants should represent, at a minimum, the initiator, the operations organization, the user organization (if different from operations), the funding organization (or representative of the program office), the implementing organization, and QA.

Chapters 4 and 5 provide further detail on the program specification and operator's manual items; program acceptance and transfer review items are presented below.

a. Agenda. Software Test and Transfer Document and transfer-related items for the Acceptance Readiness Review typically include:

(1) Any carry-over acceptance planning items from the High-Level Design Review.
(2) Implementation status and deficiencies.
(3) Deficiency workarounds and completion dates.
(4) QA inspection status.
(5) Acceptance Test criteria and plans.
(6) Proposed detailed test procedures, data, and predicted test results (appended to the preliminary STT).
(7) Draft Transfer Agreement.
(8) Other pertinent items.

b. Criteria. The following STT and transfer-related criteria can be used as aids in determining the readiness to begin acceptance testing:

(1) Are all relevant items for starting the formal acceptance testing available at the Acceptance Readiness Review (including the completed program, its SOM, the SSD as-built program specifications, and the preliminary STT)? If not, do workaround plans and scheduled completion dates reduce program acceptance and transfer risks to acceptable levels?
(2) Are the test criteria for acceptance complete, responsive to the SRD requirements, and adequately documented?
(3) Are the proposed test procedures responsive to the test criteria and test plans?
(4) Does the draft Transfer Agreement indicate any anticipated exceptions or deficient items to be present after acceptance testing is completed?

c. Approval. Authorization to proceed to acceptance testing is indicated at the Acceptance Readiness Review by cognizant line management approval of the SOM and SSD, and concurrence with the preliminary STT.

2. Transfer Review

The Transfer Review is optional and is held after program acceptance testing has been completed and the final STT has been approved by cognizant line management. The request for a Transfer Review is typically made prior to or during the Acceptance Readiness Review. The main purposes of the Transfer Review, when held, are to assess the quality, completeness, and operational readiness of the program and its as-built, as-tested documentation, and to approve the Software Transfer Agreement.

The requester (or COE) has overall responsibility for the Transfer Review. This includes setting the time, place, and agenda, and also issuance of review meeting notices and material in advance, and minutes of the review after the

meeting. The Transfer Review Board participants will typically be the same persons (or their representatives) that participated in the Acceptance Readiness Review.

a. Agenda. The Transfer Review agenda items typically include

(1) Any updated or carryover items from the Acceptance Readiness Review.
(2) Summary of program and documentation changes that resulted from the acceptance testing activities.
(3) Approved SSD and SOM, reflecting all changes.
(4) Approved STT, including

(a) Acceptance Test Procedures (updated)
(b) Summary of Acceptance Test results
(c) Anomaly Summary Sheets

(5) Completed Software Transfer Agreement (ready for approval signatures), including all exceptions and their scheduled completion dates.
(6) Other pertinent items.

b. Transfer Criteria. The following criteria can be used as aids in determining the readiness for transfer to operations:

(1) Have all Acceptance Readiness Review concerns been adequately resolved and included in the final acceptance activities?
(2) Have the acceptance t sts been run per the procedures documented in the STT? If not, are the deviations documented and acceptable on an individual basis (such as variations in test conditions, configurations, etc.)?
(3) Have all test results been analyzed, documented, and compared to specified results?
(4) Has the STT been updated (to reflect as-tested information) and approved?

(5) Have required modifications, during acceptance, to the program, the SSD, and the SOM been documented for traceability and for concurrence by the COE?

(6) Has QA certified the SSD, and when required, the SOM and STT, indicating that the as-built, as-tested software meets DSN standards of quality?

(7) Is the Software Transfer Agreement complete and ready for approval signatures? Are all deficiencies described, along with their scheduled completion dates?

(8) Are all items to be transferred to operations available for delivery to the Program Library?

c. Approval. The Software Transfer Agreement is approved by the designated cognizant personnel and their management (approval level not required above that of Division Manager). Approval of the Software Transfer Agreement signifies that the software deliverables are operational and are available for delivery to the Program Library.

This completes the implementation project.

SECTION IV
REQUIRED DELIVERABLES TO BE TRANSFERRED

A transfer (to Operations) Design Review may be held at completion of the acceptance testing to ensure that all requirements of the implementation project have been met. A list of criteria that can be used for this review is presented in Section III, Paragraph B. 2, Transfer Review.

A Software Transfer Agreement, initiated by the CDE, is approved by the CDE, QA, Subsystem Engineer, COE, and both the cognizant implementing and operations management (not above the level of Division Manager). The deliverable items described below represent the items considered always necessary to be transferred to operations and delivered to the Program Library. Special requests for any additions, substitutions, or deletions to this set should be included in the SRD, for approval.

*(1) Software Requirements Document; prepared according to the Standard Practices and approved by cognizant management, not above ALD level.

*(2) Software Definition Document; prepared according to the Standard Practices and approved by cognizant line management.

(3) Software Specification Document; prepared according to the Standard Practices and approved by cognizant line management. The concurrence signatures of the CSE and COE assure the usability of the SSD for change and support engineering, without recourse to the CDE or to the Implementation Organization. Correctness test design and results are a part of the SSD, as well as program flowcharts (or equivalent), narrative, and source code listings.

---

*Archived for Reference Information

(4) Software Operator's Manual; prepared according to the Standard Practices and approved by cognizant line management. The concurrence signature of the COE assures the usability of the SOM for operating the program and for preparing instructions for remote facility use without recourse to the Implementation Organization. For certain programs, a User's Guide may also be needed; its requirements are identified in the SRD.

(5) Software Test and Transfer Document; prepared according to the Standard Practices and approved by cognizant line management. The STT contains the acceptance test plan, procedures, and results of the acceptance testing. Following transfer to operations, all ECRs and ECOs are appended.

(6) Software Transfer Agreement; prepared according to the Standard Practices and approved by cognizant management, not above Division Manager level. The agreement and conditions for transfer of responsibilities from the implementing to the operations organization are included, as well as the QA certification that

   (a) The design as documented in the relevant portions of the SSD agrees with the implemented source code and to code listings in the SSD.

   (b) The design in the SSD uses only approved structures, and conforms to other workmanship standards such as page size, indenting of code, etc., and conforms to any project-unique standards identified in the SRD (and SSD).

   (c) The design description of striped modules in the SSD is adequate for hierarchic expansion and correctness assessment.

(7) Code; includes machine-compatible source and object code that has been QA certified.

ABBREVIATIONS

ALD	Assistant Laboratory Director
ANSI	American National Standards Institute
ASCII	American Standard Code for Information Interchange
ATP	Acceptance Test Procedure
ATR	Acceptance Test Report
CCB	Change Control Board (DSN)
CDE	Cognizant Development Engineer
CHKSUM	Submodule of SUMS
CM	Cognizant Management
CMF	Communications Monitor & Formatter Assembly
COE	Cognizant Operations Engineer
COMPUT	Submodule of WADTOT
CSE	Cognizant Sustaining Engineer
CTA 21	Compatibility Test Area, JPL, Pasadena, California
DIVSUM	Submodule of PDRPTS
DMC	DSS Monitor & Control Subsystem
DSN	Deep Space Network
DSPRPT	Submodule of WADRPT
DSS	Deep Space Station
DST	Data System Terminal Assembly
DTM	DSS Telemetry Subsystem
ECO	Engineering Change Order
ECR	Engineering Change Request

FORMSTER	Submodule of SORTDSP
FRD	Functional Requirements Document
FTS	Frequency & Timing Subsystem
GCF	Ground Communications Facility
GHS	GCF High-Speed Data Subsystem
GTPRNT	Submodule of CHKSUM
HIPO	Heirarchy Plus Input-Process-Output
HSP	High-speed printer
I/O	Input-output
JBCNTL	Submodule of WADRPT
JPL	Jet Propulsion Laboratory
KB	Keyboard
MDA	Metric Data Assembly
MDD	Environmental "Machine-Dependent Design" interface routines
MDDB	MDD batch processor routines
MDDX	MDD extension (to full MBASIC language)
MDS	MARK III DSN Data Subsystems Implementation Project
MIB	MBASIC "MACHINE-Independent batch processor"
MID	MBASIC "Machine-Independent Design" fundamental processor
MIDX	MID extension (to full MBASIC language)
NOCC	Network Operations Control Center
ODR	Original Data Record
OS	Operating system

PDRPTS	Submodule of WADRPT
PE	Project Engineer
PPOS	Submodule of COMPUT
PREPROC	Preprocessor Module for WADTOTRPT
QA	Quality assurance
QUERY	Submodule of JBCNTL
RPTPRT	Submodule of DSPRPT
SDD	Software Definition Document
SE	System Engineer
SOM	Software Operator's Manual
SORTDSP	Submodule of PREPROC
SRD	Software Requirements Document
SSA	Symbol Synchronizer Assembly
SSD	Software Specification Document
SSE	Subsystem Engineer
STT	Software Test and Transfer Document
SUMS	Submodule of WADTOT
SW	Software
SYMEQU	System Equates Module
TA	Software Transfer Agreement
TIDD	Technical Information and Documentation Division (JPL)
TLMOS	Telemetry Operating System Module
TODR	Temporary Original Data Records

TRD	Technical Requirements Document
TYPRPT	Submodule of RPTPRT
USEMAC	User Macros Module
WAD	Work Authorization Document
WADRPT	Report Generator Module for WADTOTRPT
WADTOT	Processor Module for WADTOTRPT
WADTOTRPT	WAD Report Writer Program
WBS	Work Breakdown Structure

REFERENCES

1. Tausworthe, R. C., Standardized Development of Computer Software, Prentice-Hall, Englewood, New Jersey, 1977.

2. MBASIC Manual, Volumes I and II, Jet Propulsion Laboratory, Pasadena, California, August 1975 (JPL internal document).

3. American National Standard Vocabulary for Information Processing, ANSI X3.12, American National Standards Institute, New York, 1970.

4. American National Standard Flowchart Symbols and Their Usage in Information Processing, ANSI X3.5, American National Standards Institute, New York, 1970.

5. Aaron, J.D., "Estimating Resources for Large Systems," in Software Engineering Techniques, NATO Science Committee, Rome, Italy, 1969.

6. HIPO – A Design Aid and Documentation Technique, International Business Machines Manual GC20-1851-1, IBM Technical Document Center, White Plains, New York, May 1975.