DSN Standard Practices for Software Implementation

D. C. Preska
TDA Planning Office

DSN Standard Practices for implementing software are being applied to all new computer programs and modifications to be transferred to DSN Operations. This article presents an overview of the DSN methodology and prescribed practices. The methodology is based on established good engineering practices combined with newer software technologies involving structured programming, top-down principles, and concurrent construction and documentation. The practices provide the DSN with a means of producing maintainable software which is correct, on schedule, and within cost limitations. The ultimate goal is to permit DSN engineers and managers to concentrate their efforts not on details of machines and languages but rather on the tracking and data acquisition problem to be solved.

I. Introduction

To more effectively produce software, elements of the DSN Programming System have been defined to allow refinement and application of new methods in an orderly manner. The ultimate goal is to permit DSN engineers and managers to concentrate their efforts not on details of machines and languages but rather on the tracking and data acquisition problem to be solved. The DSN Programming System includes (1) implementation methodology, (2) Standard Practices for applying adopted methods, (3) language and data base standards, and (4) implementation aids. In conjunction with the Programming System, a DSN Software Management Seminar with wide participation was established and several pilot software implementation projects were undertaken. This report presents an overview of the Standard Practices (Ref. 1) developed from both the methodology research and continuous input from the Software Management Seminar.

The prescribed methodology combines proven, well-established engineering practices with emerging techniques that are mathematically sound and have been demonstrated by many independent workers involved in software development. The elements of the methodology and the management practices involved in their application are discussed, followed by a set of brief statements

that sum up the essence of the Standard Practices for software implementation.

II. Methodology

The methodology used for implementing DSN operational software makes use of the theorems of structured programming and its consequent enabling of top-down construction of programs. Application of structured programming and top-down practices further allows design and documentation to be performed concurrently with the coding and correctness testing in orderly, hierarchical builds. These terms will be defined below. The elements of the methodology are:

(1) Principles of Structured Programming

(2) Top-Down Construction

(3) Concurrent Design and Documentation, Coding, and Testing

A. Principles of Structured Programming

Structured programming is a program organizing and implementing technique that involves the concept of representing the control logic of arbitrary computer programs with iterations of a limited set of primary program (or flowchart) structures. The objectives of structured programming are increased clarity, increased commonality (interchangeability/module-to-module, program-to-program, project-to-project), reduced complexity, and simplified maintenance. Also, structured programming allows the top-level main program control to be designed and documented while leaving the lower levels of detailed computational design to be rigorously designed later.

The Structured Programming Theorem states that any program, arbitrarily large and complex, can be expressed by primary structures that need include only the operations for performing (a) functional sequencing, (b) conditional branching, and (c) conditional iteration (looping). Application of the concept of iterating only a small number of standard structures results in a highly organized and structured representation of the design and subsequent code that is easy to implement, read, test, and understand. Coding of the design by using prescribed primary code structures is referred to as "structured coding" which typically also encompasses code indentation and line spacing to display the program flow and modularity clearly to persons reading the code.

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 to a DSN standard language, however, is the full portability of DSN applications programs from one machine to another.

Limiting the complexity of a flowchart enhances its clarity and readability. This can be achieved by using the "striped module" technique (Ref. 2) to indicate that further detail is provided elsewhere. Unstriped modules contain sufficient detail to be translated directly into code.

B. 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. This provides the capability for engineers and managers 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 formal implementation. An orderly, more serial process of program production or construction follows, and includes: 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 is referred to as *top-down construction.*

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 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. The expansion can be completed in some paths before it is completed or even started in others.

External interfaces are defined, negotiated, and implemented early, and then used in the subsequent develop-

ment. This minimizes 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 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.

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 provide 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 work through the various phases, and for managing and conducting design reviews.

C. Concurrent Design and Documentation, Coding, and Testing

Concurrent activities of design and documentation, along with coding and testing in *hierarchical builds* (module-by-module program expansions, each increasing in lower level detail) are enhanced by the use of structured programming and top-down practices. Timely documentation, resulting from use of these practices, provides ready access to current program detail and status. Management of the effort and provision for Quality Assurance are both facilitated. Potential problem areas can be identified early for timely action. Also, a large-scale separate documentation effort at the end of the project, with its attendant problems, is avoided.

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 which are needed to run the rest of the program along other paths). The dummy stubs are intended to work for one or more special test cases. There is a correctness 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 by 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 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.

Figure 1 represents the process followed in doing concurrent design and documentation, coding, and testing from the top down. 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 results of the test and code audit, formally completes the build at this level. Symbols for scheduling as shown are provided for planning and monitoring the implementation progress within each build.

III. Management

The methodology enables effective application and management of the practices and concepts, and also enables effective team operations for large efforts.

The DSN software implementation process extends from an initial program justification activity through program transfer to operations and involves (1) planning and specifying requirements, (2) design definition, (3) design and production, (4) acceptance, and (5) operations and maintenance. Major milestones are defined mainly at the end of each phase of activity, above. Figure 2 presents an overview of a typical DSN software implementation sequence, and Figure 3 summarizes the DSN software management and implementation plan, as derived from Fig. 2. The figures are self-explanatory; however, further description of the management and implementation activities can be found in Ref. 1. Supporting DSN Standard Practices shown in Fig. 3 are being produced.

The implementation process using the prescribed methods is flexible and can accommodate both large and small efforts. For large efforts, an implementation team can be established, which is directed by a Cognizant Development Engineer (CDE) who is responsible for the implementation. The CDE is provided with functional support as shown in Fig. 4, such as (but not limited to): a Programming Secretariat for administrative assistance,

project communications, and data handling; Quality Assurance services; and technical specialist support as required, spanning the disciplines of design, coding, and test design, conduct, and evaluation. The functional tasks are independent but are not necessarily related on a one-to-one basis to team personnel. The Programming Secretariat maintains the project's development records, controls the completed code, and serves as the center of communications for the entire team. Specialists, as required, are functionally separated but work as a team on a module-by-module basis and provide needed expertise to the implementation, as well as providing, to a degree, "checks and balances" to the total process. Quality Assurance provides independent code auditing services to the team on a module-by-module basis. Reference 1 provides additional detail on these and other functional tasks involved, their interaction, and a functional task statement of responsibilities for each.

Smaller efforts may require that the functional tasks be spread among fewer available personnel, which tends to lessen both the independence of each task and the inherent self-checking of the implementation process. This can be partially compensated for during the design reviews, where independent views can be factored in. Also, as for large efforts, independent code auditing on a module-by-module basis is required, and provides some of the checking needed.

IV. Summary

The following brief statements summarize the baseline software implementation guidelines that are applicable to DSN operational programs. Further comments, explanations, and examples of their application can be found in Ref. 1. Figure 5 displays the Table of Contents of Ref. 1 to show the overall organization of information upon which these guidelines are based.

(1) *Adherence to DSN Software Standard Practices.* DSN Standard Practices governing software implementation and documentation are to be followed for all computer programs to be transferred to operations.

(2) *Adherence to Referenced Practices.* Referenced practices, standards, and requirements are considered to be part of the DSN Standard Practices and are to be followed as an extension of Item (1), above.

(3) MBASIC/*Standard Language for Nonreal-Time DSN Computer Programs.* It is the intent of the DSN to use MBASIC as the standard language for nonreal-time DSN computer programs.

(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.

(5) *Use of Structured Programming Principles.* Only prescribed programming structures will be used for flowcharting the design and translating the design into computer code.

(6) *Modular Implementation.* By applying Items (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 × 11-in. (22 × 28-cm)) flowchart by using the Striped Module technique of hierarchical expansion.

(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

(8) *Standard Project Milestones.* Standard project milestones will be used for technical and management planning and control of the software implementation.

(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.

(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.

(11) *Project Documentation.* Documentation will be compiled concurrently with the implementation progress and will be available on a continuous basis for use and review.

(12) *Quality Assurance.* Quality Assurance (QA) will provide an independent check on DSN software quality.

Acknowledgment

The author is indebted to the members of the DSN Software Management Seminar, whose support emanated from JPL Organizations 15, 33, 39, 65, and 91, as well as from the DSN itself.

References

1. *Software Implementation Guidelines and Practices*, DSN Standard Practice 810-13, Aug. 15, 1975 (JPL internal document).
2. Tausworthe, R. C., *Standardized Development of Computer Software*, Jet Propulsion Laboratory, Pasadena, Calif. (in preparation).

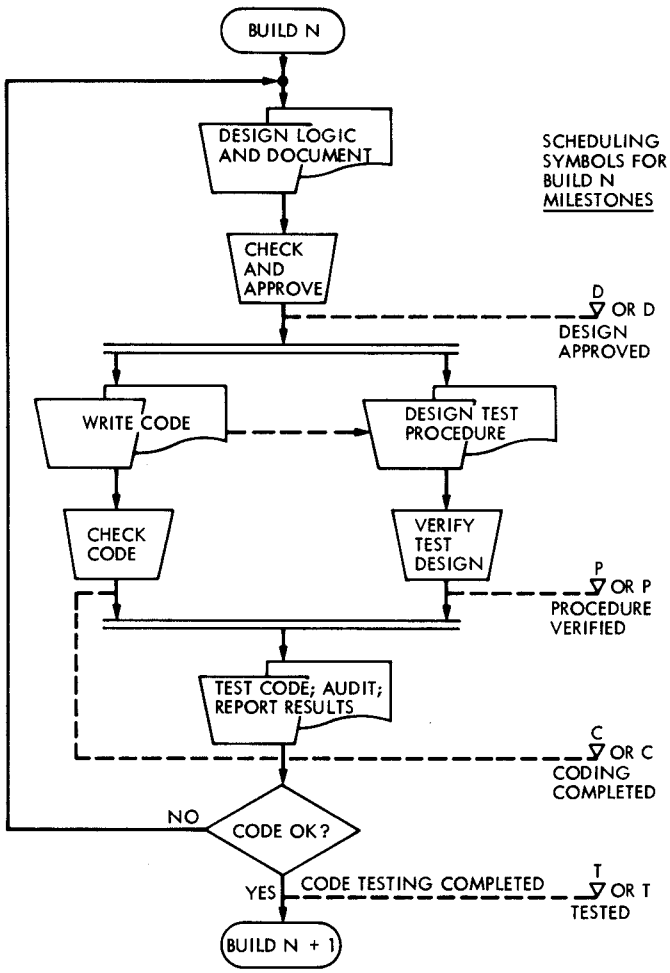

Fig. 1. Top-down, concurrent construction process for Build N

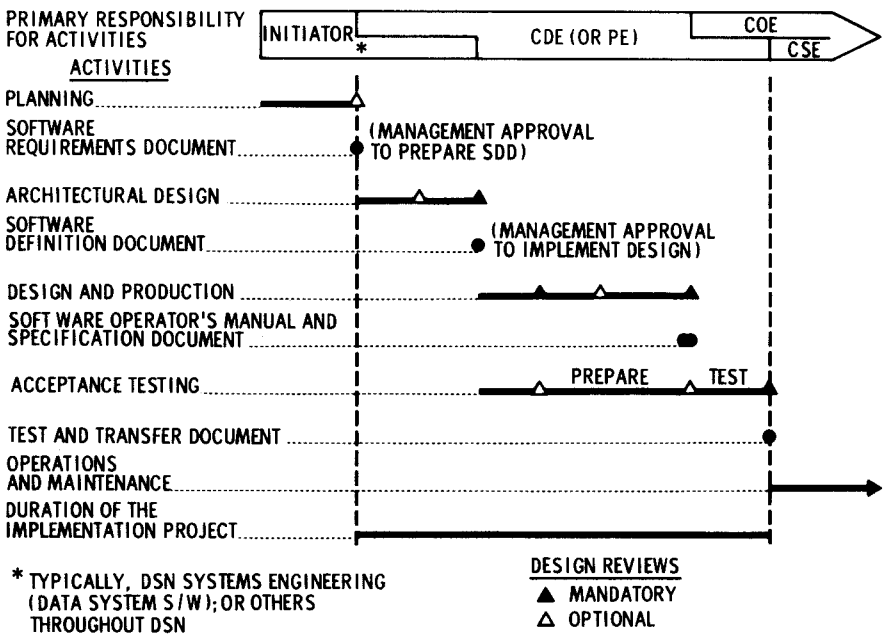

Fig. 2. Sequence of activities in a typical DSN software implementation project

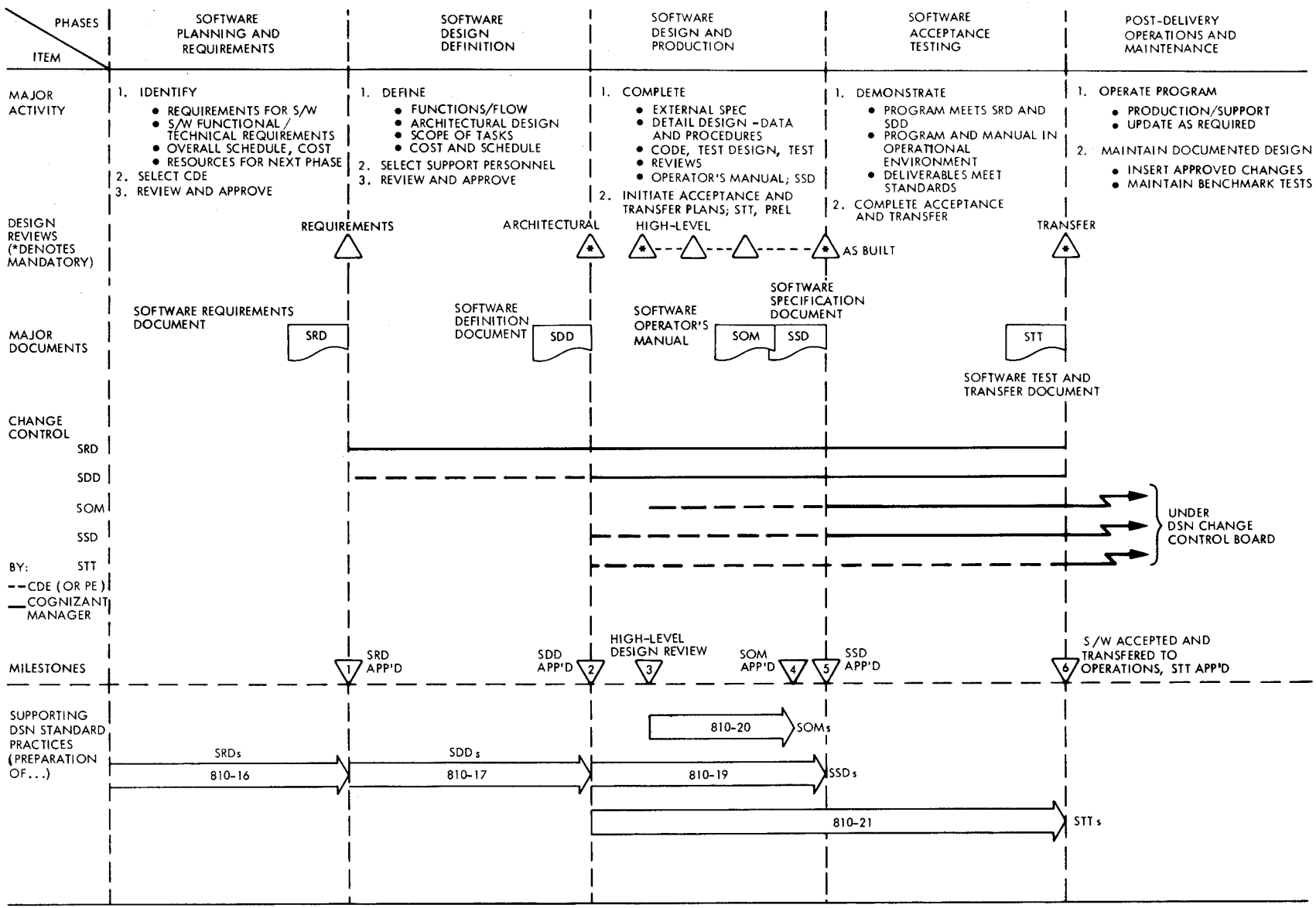

Fig. 3. DSN software management and implementation plan

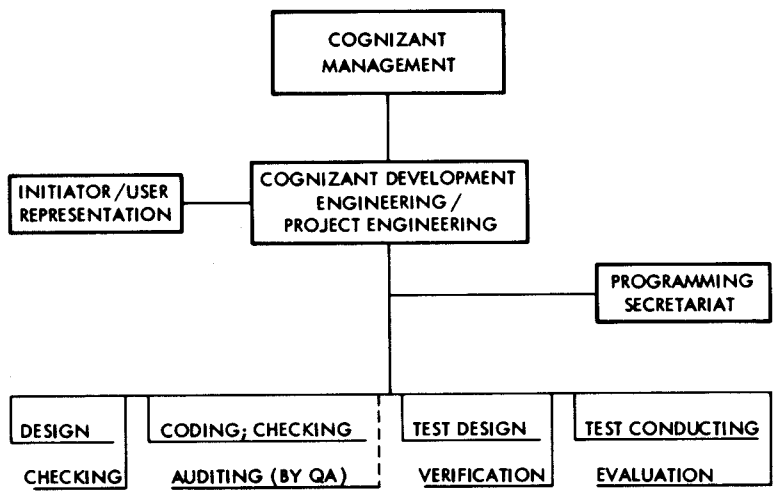

Fig. 4. Software implementation functional tasks

CONTENTS

Fig. 5. Contents of DSN Standard Practice 810-13, "Software Implementation Guidelines and Practices"

CONTENTS (cont'd)

Fig. 5 (contd)

CONTENTS (cont'd)

Fig. 5 (contd)