NASA

Technical Memorandum 82148

(NASA-TM-82148) SOFTWARE ENGINEERING STANDARDS AND PRACTICES (NASA) 49 p
HC A03/MF A01 CSCL 09B

N81-26748

63 Unclas
09/61 28867

Software Engineering Standards and Practices

Ronald W. Durachka

JUNE 1981

National Aeronautics and Space Administration

Goddard Space Flight Center
Greenbelt, Maryland 20771

TM 82148

Software Engineering Standards and Practices

Ronald W. Durachka
Data Set Preparation Section
Code 931.1
Information Management Branch
Information Extraction Division
Goddard Space Flight Center
Greenbelt, Maryland

June 1981

Goddard Space Flight Center
Greenbelt, Maryland

PREFACE

This document presents techniques and concepts for software engineering standards and practices throughout the software development life cycle. These ideas are not new; they have been documented in great detail in text books, technical papers, and in several GSFC software engineering documents. The guidelines presented in this document are those which are most applicable to the Information Extraction Division's software development activities.

TABLE OF CONTENTS

LIST OF FIGURES

1. INTRODUCTION

1.1 Objectives

This document describes the various phases of a software development project throughout its life cycle and general software engineering standards and practices to be followed during each phase.

This document is also intended to provide guidelines for the preparation of a software development plan, which includes a specification of the standards and practices to be followed from project inception to project end. This software development plan will aid in:

- Defining the software engineering standards and practices to be used by the designer;
- providing an objective basis for measuring the project's progress; and
- ensuring high quality software.

The software development plan will aid software designers in:

- Providing a model of the software development cycle which will be used as a basis for planning, measuring, and controlling the software throughout its life cycle,
- providing a basis for communications with the project, the user, and the operations and maintenance personnel,
- providing more usable, operable, maintainable and transportable software.

Section 2 of this document describes the step-by-step development of a software project. Section 3 discusses each of these steps in more detail. The Appendices contain additional information on the characteristics of detailed design modules and their prologs.

1.2 References

The material presented in this document is based on the documents listed below. Certain parts were extracted verbatim; other parts were modified or added to more properly reflect the activities of the Information Extraction Division.

1. "Guidelines for Documentation of Computer Programs and Automated Data Systems", FIPS PUB 38.

2. "Mission Operations Division (MOD) Software Engineering Standards and Practices (SESP) Volumes 1 and 2 - Software System Life Cycle", prepared for GSFC by Computer Sciences Corporation, May 1979.

3. "POCCNET Software Engineering Standard Approach Recommendation", SCPB Informal Memorandum I-76-34, prepared for GSFC by Computer Sciences Corporation, December 1976.

4. "Telemetry Computation Branch Quality Assurance Procedures--Programming and Documentation Standards", prepared for GSFC by Computer Sciences Corporation, February 1978.

5. "Telemetry Computation Branch Quality Assurance Procedures--Software Development Procedures" (Revision 1), prepared for GSFC by Computer Sciences Corporation, August 1977.

2. SOFTWARE DEVELOPMENT LIFE CYCLE

Figure 2-1 identifies and briefly describes the phases, major activities, and major products of the software system life cycle. Figure 2-2 identifies deliverables and illustrates a proven, risk-reducing software system development cycle which emphasizes the formal specification of requirements and strict control of changes to the specification, implementation, and testing procedures.

This software system development cycle is divided into nine areas: requirements, functional design, detailed design, test plans, configuration control, implementation, testing, operations, and maintenance procedures.

The requirements document is the first phase of the system life cycle. The purpose of this phase is to provide a basis for mutual understanding between the project, users, and designers of the initial definition of the software to be developed.

The functional design document presents the preliminary software system design proposed to satisfy the requirements. It provides a system-level description of the proposed software capabilities in light of the currently stated project requirements.

The detailed design document presents the unit-level software system design proposed to satisfy the requirements. It describes, at a detailed level, the structure of all the software identified and presented in the functional design document. The detailed information necessary to develop the basic individual software components, or units, of each program is provided. In addition to program specifications, this document includes the detailed descriptions of all data files and external interfaces utilized within the processing environment. The detailed design also includes a detailed milestone software build schedule and implementation

FIGURE 2-1. SOFTWARE DEVELOPMENT CYCLE

FUNCTION	REQUIREMENTS	FUNCTIONAL DESIGN	DETAILED DESIGN	TEST PLANS	CONFIGURATION CONTROL	IMPLEMENTATION	TESTING	OPERATION	MAINTENANCE
PURPOSE	Identify, specify and define the overall environment the system will function and operate in	Describe system philosophy and how the requirements are to be satisfied	Specify in detail the system described in the functional design	Define the tests and demonstrations necessary to qualify the system for operational use	Control new requirements, changes to requirements, and software problems	Develop code to implement the software builds	Demonstrate required responses to both normal and abnormal data. Benchmarks	Operate the system	Correct system problems. Implement enhancements.
DOCUMENTATION	Requirements document, including performance, interfaces and operating environment	Functional design document including data flows, processing descriptions, software structure, input output, error handling, resources, interfaces	Detailed design document specifying inputs and outputs, data formats, software builds, information processing narratives, mathematical expressions, diagrams, timing req., PDL	Testing criteria developed for unit tests, build tests, integration and system tests	Configuration control form	Implementation table or matrix showing processing requirements versus system functions versus test procedures	Demonstration and/or written results of each test including function being tested input/output and evaluation of the test	Operations manual and user's guide including operating procedures, output descriptions diagnostic messages, error handling utilities	Use and update of existing documentation

FIGURE 2-2. SYSTEM DEVELOPMENT CYCLE

	REQUIREMENTS PHASE	DESIGN PHASE	IMPLEMENTATION AND TESTING	OPERATIONS, MAINTENANCE AND ENHANCEMENTS
ACTIVITIES	Definition of functional requirements	Functional Design Detailed Design Test plans	Code modules for build 1, ..., code modules for build N Test build 1,..., test build N Prepare users manual Prepare operational manual Prepare maintenance manual Integration and acceptance testing	Use the system Operate the system Maintain and enhance the system Analyze system performance
CONFIGURATION CONTROL				
REVIEWS	Requirements review and approval	Functional design review and approval Detailed design review and approval Test plan review and approval	Demonstration of each build Demonstration of total system and acceptance	System performance
PRODUCTS	Functional requirements document Development plan Budget	Functional design document Detailed design document Test plan document Milestone schedule Implementation Matrix	Test analysis reports Users manual Operations manual Maintenance manual	Updates to system documentation

matrix to be used in conjunction with the test plan during the implementation phase.

The test plans specify the proposed techniques and mechanics to be employed in the validation and verification of the software during the implementation phase of the system development cycle.

Configuration control is a systematic and disciplined approach to applying administrative direction and maintaining surveillance over software related activities. It is used to control and account for changes to software, and to verify that deliverables contain only approved changes.

The implementation phase develops the system in accordance with the requirements, design, test plan, and configuration procedures.

The testing strategy is based on the fact that systems are developed and tested incrementally in a series of builds proceeding from a minimal capability in a series of well planned stages. This type of testing procedure of "build-a-little, test-a-little, use-a-little" is very successful and has the following advantages:

- o Interface problems can be discovered early.
- o Increments of system capabilities can be demonstrated before completion of the total system.
- o It allows user feedback to validate the requirements before the end of implementation.

The operational phase begins when the completed system is installed in the user's environment and accepted by the project.

The maintenance phase consists of making modifications to the code, the documentation, and/or data for the purpose of correcting errors in the system. System enhancements are also included in this phase and consist of responding to changes in the system requirements or other user needs.

3. SOFTWARE DEVELOPMENT ACTIVITIES

3.1 Requirements

The activity of the requirements phase is to specify the requirements that the system must satisfy and to gain project approval of the specification of these requirements. Inputs to this phase may include results from previous efforts, such as feasibility study results, concept descriptions, and trade-off study results.

Proper execution of requirements specification activities is critical to providing a system that meets user requirements and minimizes overall life cycle costs. User requirements are captured, analyzed, and synthesized to provide clear, concise, unambiguous, and complete requirements statements that can be verified, tested, and/or demonstrated. These statements are organized and documented in the Functional Requirements Document. This document is absolutely essential for the success of a development effort, since it forms the basis of communications about the developing system. Omissions, errors, or misunderstandings at this stage can later cause major problems. When several individuals or organizations are involved in development projects, it is critical that all requirements be recorded in writing.

The requirements are defined through an iterative dialog between the system users and the development group, in which general requirements are gradually translated into specific functions. The requirements document produced by this phase describes what is to be accomplished by the software system. In contrast, follow-on design documents describe how the requirements will be satisfied and show the required software modules/routines, their functions and interrelationships.

The written requirements specification will be the definitive reference for all future development activity and must be updated periodically (e.g., after each design review) to reflect changes or additions. The specification should describe areas of uncertainty or missing information, as well as what is certain. The requirements specifications must be properly maintained, since they may be used by the project/task managers, design personnel, and other project personnel (e.g., the testing group), and users. An accurate requirements specification is also essential for meaningful testing and verification of the completed software system.

The Functional Requirements Document divides requirements into logically related groups, such as those presented in Figure 3-1. A well-organized presentation of requirements states what is to be done and provides a means for evaluating the results; that is, the statement of requirements provides both the basic "design to" guide for the development group and the standard by which the manager can evaluate the resulting system.

The outline presented in Figure 3-1 describes one possible means of specifying requirements. Individual projects may develop more suitable outlines and better methods of presentation. Although the format of requirements specifications may vary, the contents should basically include those areas covered in Figure 3-1 which are applicable to that project. Most important, however, is that the requirements be uniquely identified and collected into logically related groups.

FIGURE 3-1. Functional Requirements Document Outline

1. General Information

1.1 Summary - Summarize the general nature of the system to be developed

1.2 Environment - Identify the project, developer, users, and computer complex where the system is to be implemented

1.3 References

2. Overview

2.1 Background - Present the purpose and scope of the system

2.2 Objectives - State the major performance objectives of the system, types of users, data bases, interfaces, and operational changes which are to be supported.

2.3 Existing Methods and Procedures - Describe the current methods and procedures for satisfying the existing objectives.

2.4 Proposed Methods and Procedures - Describe the proposed system and its capabilities. Identify techniques and procedures from other systems that will be used or that will become part of the proposed system. Identify the requirements that will be satisfied by the proposed system.

3. Requirements

3.1 Functions - State the functions required of the software in quantitative and qualitative terms, and how these functions will satisfy the performance objectives.

3.2 Performance - Specify the performance requirements, such as accuracy, validation, timing, and flexibility

3.3 Inputs/Outputs - Specify and show examples of the various data inputs and outputs

3.4 Data Characteristics - Describe the data and estimate total storage requirements based on the system's expected growth

3.5 Failure Contingencies - Specify the possible failures of the system, alternative courses of action, and consequences. Include back-up, fallback, and recovery and restart in the failure analysis.

4. Operating Environment

4.1 Equipment - Identify the equipment required for the operation of the system.

4.2 Support Software - Identify the needed support software.

4.3 Interfaces - Describe the interfaces with other systems

4.4 Security and Privacy - Describe the overall security and privacy requirements imposed on the system

4.5 Controls - Describe the operational controls imposed on the system

3.2 Functional Design

The functional or preliminary design phase consists of analyzing the requirements specified in the functional requirements document and designing a software system, at the highest level, which will perform those functions. That is, the functional design describes how the proposed system will perform the required functions. Processing strategies and data structures are considered, and subsystems and their interfaces and interactions are described. The principal product of the functional design phase is a functional design document or set of document sections for incorporation into a later document. The outline of the design document can serve as an organizational mechanism for design activity, since designing the system and writing the document are nearly synonymous.

The purpose of a formal design document is to allow review by all affected organizations to ensure that a workable system has been planned which will satisfy all requirements. User acceptability is a primary consideration during the functional design phase. User or other operations interfaces should be described in enough detail to allow potential users to evaluate the system from their point of view.

The Functional Design Document consists of the following sections--Introduction, Overview, and Subsystems.

A. Introduction

The Introduction briefly identifies the system, its purposes, users, environment, and identifies the related requirements document and any other relevant information.

B. Overview

The Overview section contains the following:

1) A block diagram of the system which illustrates and concisely names

all software subsystems, I/O devices and associated files, and special purpose hardware.

2) A data flow diagram which depicts the flow of all data throughout the system, from raw input through data base to user, and concisely identifies all data types and output products. (This diagram may be combined with the system block diagram, depending on complexity.)

3) A description of the overall system in terms of how it meets the functional requirements. It briefly describes the subsystems shown in the block diagram and how they interact; discusses and justifies major design decisions where alternate approaches were considered; describes the input data, all data files, and output products; describes user interaction and operational procedures; that is, how the user would perform specific functions mentioned in the functional requirements document; and discusses theoretical approaches taken in solving specific analysis problems.

C. Subsystems

This section provides a logical flow chart for each subsystem and discusses each subsystem's function. It describes, using text and tables, the interface to each subsystem, e.g., subroutine arguments, common blocks, inter-task communication packets, and contents of I/O records. (Note that a subsystem may be defined as a subroutine or as an independent program, depending on the size of the system.) It describes, for each subsystem, any specific analysis algorithms, computational accuracy considerations, speed requirements, etc., as necessitated by the functional requirements.

3.3 Detailed Design

The detailed design phase consists of refining the functional design into a complete, detailed system blueprint, essentially a code-to specification. The detailed design is formally documented in order to allow review, to communicate the design to the programmers, and to serve as the basis for final system documentation.

The Detailed Design Document consists of the following sections--Introduction, Subsystems, Formats, and Schedules.

A. Introduction

The Introduction briefly identifies the system, its purpose, users, environment, and identifies the related Functional Design Document and any other relevant information.

B. Subsystems

This section describes in detail each software subsystem as defined in the Functional Design Document. Each subsystem description includes the following:

- Logical flow chart illustrating subsystem function
- Block diagram showing and naming each module within the system
- Overview of the subsystem briefly describing each module and inter-module interactions and describing how the subsystem function is achieved
- Description of each module, to include a prolog as defined in Appendix B, a description of the module in Program Design Language (PDL) and, optionally, a flow chart.

Prologs may be in the form of computer listings appended to the design document. All modules must conform to the module characteristics as described in Appendix A.

C. Formats

This section describes, to the level necessary for programming, the formats of all I/O files and records, all output products (punched cards, television displays, plots, graphs, listings, etc.), all global storage areas (COMMON blocks), inter-task communication packets, interactive user language and menu formats, and any other software constructs other than actual code. Formats may be in the form of tables, diagrams and text.

D. Schedules

System development is achieved as a top-down sequence of software builds. This section of the Detailed Design Document describes each build by listing the modules contained in it, the tests to be performed, and the functional capabilities. (The final build is the completed system.) A milestone schedule is included in this section which specifies the delivery date for each build.

3.4 Test Plans

There are three phases of test plans necessary for the staged system implementation and acceptance: (1) module test plans; (2) build test plans (system increments); and (3) integration and acceptance test plans. The technical inputs to the test plans are the Functional Requirements Document, the Functional Design Document, and the Detailed Design Document. The management inputs to the test plans are the software development plan generated during the requirements phase and the software build schedule and implementation matrix generated during the detailed design phase.

The module test plan preparation and specification coincides with the preparation of the module prolog and logic specification. The formats for prologs are defined in the detailed design section 3.3. and Appendices A and B.

The build test is composed mainly of the combination of the module tests of the modules contained in the build. It identifies the individual test cases that are required to demonstrate that the build performs its functions and to show that the software and data requirement has been implemented. The build test procedures describe the input, the expected output, and the step-by-step procedures required to perform each specified build test case.

The objective of the integration test plan is to integrate the modules of the current build with the modules from the previous builds. The primary inputs to the integration tests are the tested modules of the current build with any builds previously integrated and tested. Having met the above criteria the new build is integrated by testing the new capabilities and determining that they satisfy the system requirements.

Once all the software builds have been successfully integrated, the system is then ready for acceptance testing. Acceptance testing consists of running the build tests, demonstrating the system capabilities to project personnel and users, and thus obtaining approval and acceptance of the system.

A general outline of the contents of a typical test plan is shown in Figure 3-2. This outline is based in part on FIPS publication 38.

FIGURE 3-2. Outline of Test Plan

1. General Information

1.1 Summary - Summarizes the functions of the software and the tests to be performed

1.2 Environment - Describes the test environment, summarizes the needed equipment, inputs, etc.

2. Plan

2.1 Test Description - Briefly identifies the inputs, outputs, and functions of the software being tested, specifies the step-by-step procedures to accomplish the test, including test setup, initialization, and termination.

2.2 Milestones - Gives the projected test date

2.3 Testing

2.3.1 Requirements - Lists the functional requirements the test will satisfy, and identifies the equipment and other software needed to support the test.

2.3.2 Testing Materials - Lists the materials needed for the test, such as documentation, input data, and output data.

3. Specifications and Evaluation

3.1 Specifications and Evaluation - Lists the tests to be performed and relates them to the functions of the software module or build.

3.2 Methods and Constraints

3.2.1 Methodology - Describes the general method or strategy of the test.

3.2.2 Conditions - Specifies the input data to be used as well as the data volume.

3.2.3 Data Recording - Discusses the method to be used for recording the test results

3.2.3 Constraints - Indicates anticipated limitations on the test due to interfaces, data bases, functions being completed in another module or build, etc.

3.3 Evaluation - Describes the criteria and techniques used to evaluate the test results.

3.5 Configuration Control

Configuration control is comprised of three areas: the configuration control board, configuration management, and the configuration management procedure.

The configuration control board is composed of personnel from the project/designers/users/operations and maintenance organizations. It is responsible for reviewing proposed changes to the system, assessing the impact of the change request, and either approving, disapproving, or delaying the implementation of the proposed change.

Configuration management encompasses change control, configuration accounting and reporting, and configuration reviews.

Configuration management procedures are necessary because as a rule data processing systems undergo considerable change and evolution during development; it is, therefore, desirable to control changes to the system requirements and design. This is made possible by well-defined procedures for coding (Section 3.3), along with a formal mechanism for requesting software changes and reporting software problems.

The formal mechanism for requesting a design change or reporting a software problem or documentation discrepancy is the configuration control form (CCF), illustrated in Figure 3-3.

The CCF is a four part form for reporting and/or requesting system changes to requirements, design, software, and documentation. The CCF is classified as one of three types: (1) software problem, for program malfunctions; (2) design change requests, for changes in requirements or design; and (3) documentation, for updates and changes to existing documentation.

The change control process of the configuration management procedure is described in the items below.

- The CCF may be initiated by any individual involved in the system. The initiator will complete the "INITIATOR Description" portion of the CCF, and submit the form along with sufficient information to justify the request.
- The analyst reviews the request and assigns a problem type designation in the space provided. The analyst determines the amount and kind of information needed, performs the analysis, and includes this information in the "analysis" portion of the CCF, with additional sheets as required. The analyst describes the required modifications and the anticipated impact on the hardware/software resources, schedules, manpower, and cost. When the analysis has been completed it is submitted for approval.
- If the change is approved by the configuration control board, it is sent to the implementation personnel for inclusion in the development schedule. The change could also be deferred or rejected by the control board.
- Change approval means that the implementation group is expected to work the new activity into the development schedule. When the change and testing has been completed, the "change" portion of the CCF is filled in. Descriptions of modifications to the software must be specific because the information is required for verification activities. The supervisor reviews the modifications for correctness and conformance to software standards and signs the CCF on the line denoted "work reviewed by:".

- The implementation is reviewed by the software system technical monitor to determine what additional tests if any are required for final verification. When the verification actions have been completed, the technical monitor signs the CCF and returns it to the control board for final disposition.
- The system version is filled in with date of system availability, and the change is advertised to the user community as completed and ready for use.

CONFIGURATION CONTROL FORM

REQUEST NUMBER: _____ PROJECT: _____
YR • MON • DAY • SEQ

STATUS:

Received: _____

Analysis: _____

Implementation: _____

Verification: _____

PROBLEM TYPE:

☐ Software Problem

☐ Design Change

☐ New Requirement

INITIATOR DESCRIPTION:

INITIATOR: _____
Name Organization Phone No.

STATEMENT OF PROBLEM TYPE: _____

additional pages ☐

ANALYSIS: _____
Name Date

Software/Hardware _____

Schedule; _____

Resources: _____

Other (Documentation, etc.): _____

additional pages ☐

CHANGE (Description of work performed, include test scores used for verification): _____

Software/Hardware Affected: _____

Work Performed by: _____
Name Date

Work Reviewed by (supervisor): _____
Name Date

VERIFICATION: _____
Name Date

Availability of Change (System/Version) _____
Name Date

COPIES: INITIATOR/ANALYST/CONFIGURATION CONTROLLER/VERIFIER

3.6 Implementation

The implementation phase includes coding of software modules, module testing, integration of modules into builds, build testing, and integration and acceptance testing. This phase also includes the development of the User, Operations, and Maintenance manuals.

This phase of software development has traditionally had low management visibility and as a result has presented opportunities for uncontrolled schedule slips. To avoid or at least anticipate schedule problems in the implementation phase, top down system development methodology is used and the system is broken into software builds, where at least one visible, verifiable event is scheduled each month.

The top down methodology consists of developing the highest level software modules first, i.e., the main control module followed by the modules called by the main program. Following this, the next level of modules are then developed. This process continues until the lowest level modules are finished. At each level, the modules to be called on the next lowest level are represented by stubs. A stub consists of a prolog, output message, and a return to the calling module. Using the top down method the system is tested from the start of development as each software build is finished.

A build is a software subset which fulfills some of the system requirements. The advantages of builds are:

- Software functions become visible early during development
- Testing and development proceed concurrently
- If development is halted, the status of the system is well defined
- Management can more easily track the system development.

The objective of the build concept (Figure 3-4) is to reduce dependence on final testing and to develop a usable subset of the system capabilities with the delivery of each build. Thus, a build can be functionally tested at a high level and can be operationally demonstrated to the user. From the developer's view, the build consists of a number of modules which have been individually tested and then integrated. The build plan is developed during the detailed design phase. An example of a build implementation matrix is given in Figure 3-5. The build implementation matrix shows all the software builds across the top. The left side of the matrix indicates the subsystems and the requirements fulfilled by each subsystem. The right side of the matrix indicates the test performed to satisfy the requirement. The asterisks indicate in which build the requirement will be available for operational use.

Build test, integration and acceptance test plans are required under this development methodology and are defined in the test plan section 3.4. The build test plan defines test cases needed to demonstrate that the build is internally correct. Integration and acceptance test plans are designed to demonstrate the correctness of interfaces, data flows, and to verify that the system performs all the capabilities defined in the design.

The system documentation developed during the implementation phase are the User, Operation and Maintenance guides. These guides are defined in sections 3.8 and 3.9.

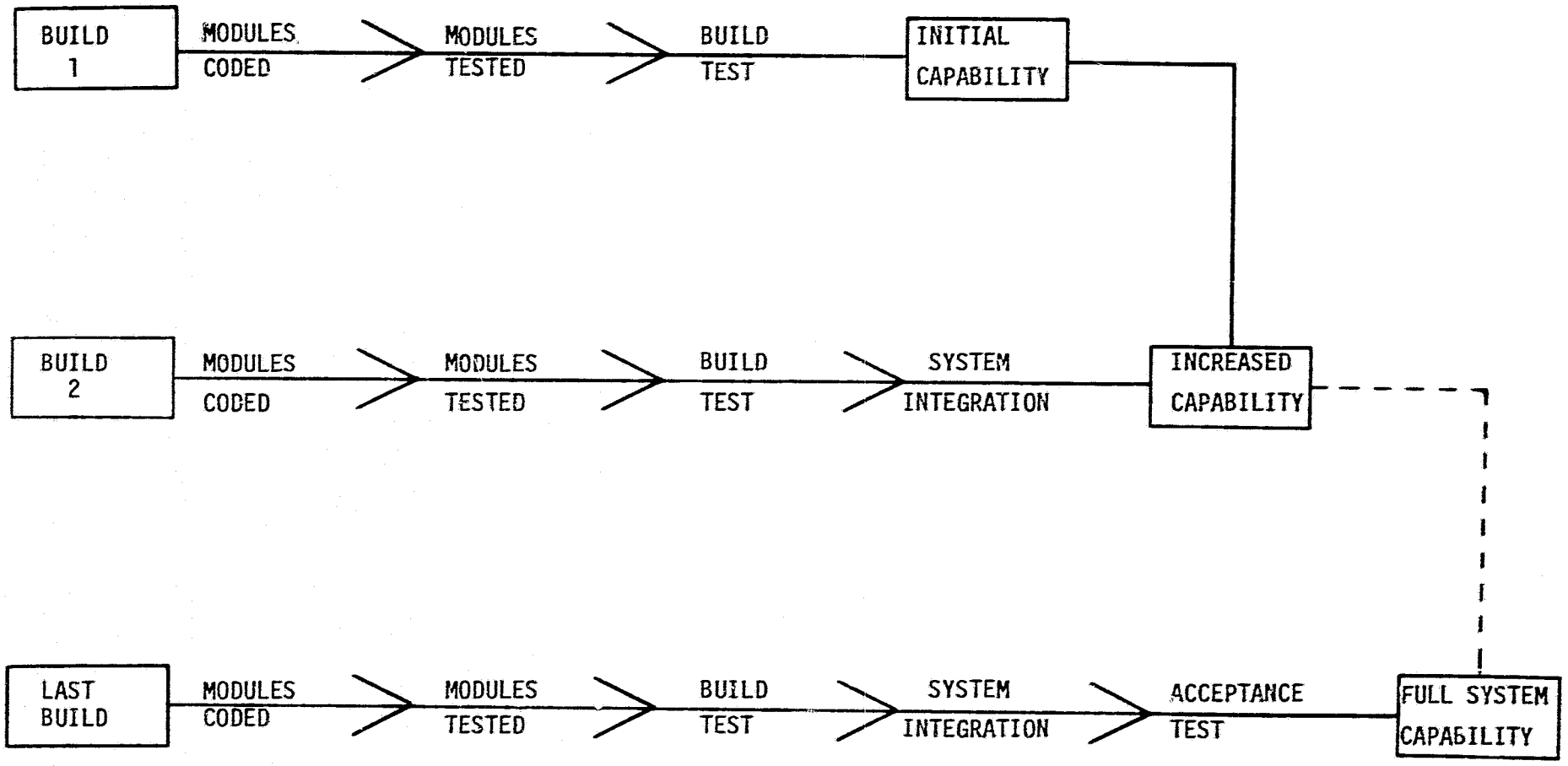

FIGURE 3-4 BUILD IMPLEMENTATION CONCEPT

BUILDS / SUBSYSTEM REQUIREMENT	BUILD 1 INITIALIZE SYSTEM	BUILD 2 READ FORMATTED DATA TAPE	. . . . .	BUILD N OUTPUT FORMATTED DATA TAPES	BUILDS / TEST PROCEDURE
Subsystem 1					
Requirement 1	*				Test 1
Requirement 2		*			Test 2
Subsystem 2					
Requirement 3		*			Test 3
			. . .		
Subsystem M					
Requirement N		*			Test N
Requirement N+1				*	Test N+1

Figure 3-5 Build Implementation Matrix

3.7 Testing

The test phase, which begins during the implementation phase, carries out the plans as previously developed (Section 3.4). The tests performed occur at the module, build, integration, and acceptance levels. By using the strategy of testing at various levels the system is developed, tested, and used incrementally from modules to builds to final system, beginning with minimal capability and ending with full system capability. This testing procedure of "build-a-little, test-a-little, use-a-little" is very successful and has the advantages of early discovery of problems, early demonstratable partial system, and early user interaction.

3.7.1 Module Testing

The purpose of module testing is to verify the correctness of the module design and the module code. Since module testing is used as an internal means of verification it does not have to be a formal process. The only requirement is that the module pass a set of standard tests. These tests certify that the module is ready to be integrated into a software build and subjected to build testing.

The standard module test verifies that:

- The module provides the software capability allocated to it in the design document
- All processing paths between successive decision points identified in the module design function correctly.
- All the module interfaces to the data base are correct
- All processing of user and computer inputs is done correctly
- Correct responses are generated for user and computer inputs.

3.7.2 Build Testing

The purpose of build testing is to verify that the build as produced reflects the design. When build testing is successfully carried out it signifies that the integration test is ready to commence. The build configuration is subjected to a series of tests which verify that the build as produced reflects the design as described in the detailed design document.

The standard build test verifies that:

- All the module software capabilities specified in the design document are provided for in the build
- All the software interfaces to records, groups, and subgroups in the data base are correct
- All the software interfaces between modules are correct
- All user input and the resulting responses specified in the User's Manual (Section 3.8) are correctly processed and generated by the build
- All computer operations input and the resulting responses specified in the Operations Manual (Section 3.8) are correctly processed and generated by the build.

3.7.3 Integration and Acceptance Testing

The purpose of integration and acceptance testing is to demonstrate that the system as designed and coded meets the capabilities as defined in the requirements and design documents. The standard integration and acceptance test verifies that:

- All capabilities specified in the requirements and design documents are provided for by the system
- All the performance requirements established in the requirements document are met by the system
- All data base requirements defined in the requirements and design documents are met by the system
- All the quality criteria set forth in the requirements and design document are met by the system
- All the interface requirements between the system and other systems, as specified in the requirements and design documents, are satisfied.

3.7.4 Build, Integration, and Acceptance Test Analysis Reports

The purpose of a test analysis report is to document the test results and findings, to describe the demonstrated capabilities, and to provide a basis for preparing a statement of system readiness.

Figure 3-6 presents a general outline for the test analysis report.

FIGURE 3-6. Outline of Test Analysis Report

1. Identifies the test.
2. Identifies and presents the test results, including data input and output results.
3. Identifies and describes the findings for each function the test is fulfilling, including performance evaluation, and identifies the deficiencies, limitations, and constraints detected in the software during testing.
4. Analysis Summary
   - 4.1 Describes the capabilities of the software as demonstrated by the tests.
   - 4.2 Describes the deficiencies of the software as demonstrated by the tests.
   - 4.3 Describes the readiness of the software for inclusion into the system.

3.8 Operations

The operational phase of the system life cycle has two main objectives: (1) to use the system to perform its required functions, and (2) to provide the operational support needed in the use of the system.

3.8.1 Users

The major activities performed by the system users are:

- Perform user functions for which the system was developed
- Identify errors or deficiencies which inhibit full use of the system and file a problem report (Section 3.5)
- Evaluate and report on system capabilities from the point of view of user satisfaction
- Suggest system improvements.

A user manual is developed to describe sufficiently the functions performed by the system, so that the user community can determine how to use the system. The manual serves as a reference document for preparation of input data and parameters and interpretation of results by the users.

Figure 3-7 presents a sample outline for a Users Manual.

FIGURE 3-7. Outline of Users Manual

1. General Information

1.1 Introduction - Summarizes the applications and general functions of the software system.

1.2 Overview - Identifies the user organization, the computer center, and general data flow.

1.3 References

2. Application

2.1 Description - Describes when and how the software is used and the unique support provided to the users.

2.2 Operation - Shows the operating relationships between the system capabilities and the users. Describes security and privacy considerations.

2.3 Equipment - Describes the hardware on which the system is running and the degree of system transportability.

2.4 Structure - Shows the structure of the software and describes the role of each component in the operation of the system.

2.5 Performance - Describes the performance capabilities of the system.

2.6 Data Base - Describes all data files in the data base that are referenced, supported, or kept current by the system, and indicates the purpose of each file.

2.7 Input, Processing, and Outputs - Describes the inputs, flow of data through the data processing, and the resultant outputs.

3. Procedures and Requirements

This section provides information about initiation procedures and the preparation of data and parameter inputs for the system. This information is logically arranged and sufficiently precise and complete to enable the user to prepare all required inputs. This section also explains in detail the characteristics and meaning of all system outputs. It also describes error, recovery, and query procedures.

3.1 Initiation - Describes step-by-step procedures required to initiate processing.

3.2 Input - Describes the preparation for preparing input data and parameters.

3.3 Output - Describes the content, format, and intepretation of each system output.

3.4 Error and Recovery - Lists error codes or conditions generated by the system and corrective action to be taken by the user. Indicates procedures to be followed by the user to ensure that any restart and recovery capability can be used.

3.8.2 Operational Support

The system is considered to be operational once it has passed all integration and acceptance testing, and an Operations Manual is made available to the operations personnel. The operations personnel have the responsibility to:

- Operate the system.
- Identify system operational problem areas, errors, and deficiencies and file a problem report about them (Section 3.5).
- Monitor, evaluate, and report on system performance.
- Make recommendations to solve operational problems and to improve the system.

The Operations Manual provided with the system is intended to provide system operations personnel with a description of the software and of the operational environment. Figure 3-8 presents a sample outline for an Operations Manual.

FIGURE 3-8. Outline of Operations Manual

1. General Information

1.1 Introduction - Summarizes the general functions of the software

1.2 Environment - Identifies the user organization and the computer center

1.3 References

2. Overview

2.1 System Organization - Provides a diagram showing the inputs, outputs, data files, and sequence of operations of the system

2.2 Program Inventory - Identifies each program

2.3 File Inventory - Identifies each permanent file that is referenced, created, or updated by the system.

3. Description of Tasks

3.1 Task Inventory - Lists the various tasks possible, summarizes the purpose of each task, and shows the programs that are executed during each task.

3.2 Task Progression - Describes the manner in which progression advances from one task to another so that the entire task cycle is shown

3.3 Task Description - Organizes the information on each task into the most useful presentation for the operations personnel involved

4. Non-Routine Porcedures

Provides all information necessary concerning emergency or non-routine operations, such as switchover to a back-up system and procedures for turnover to maintenance personnel.

3.9 Maintenance and Enhancements

The maintenance and enhancement phase is the last phase of the system life cycle and consists of making modifications to the code, the documentation, and/or data for the purpose of correcting errors and improving the system.

A Maintenance Manual is provided to complement the Operations Manual and User's Manual. The Maintenance Manual provides the information necessary to understand the system, its operating environment, and its maintenance procedures. Continued additions to this document, such as new operating procedures, new processing algorithms, or any other specific technical enhancement or modification to the system will keep the documentation current and abreast with the system changes.

Figure 3-9 presents a sample outline to be used as a guide in developing a Maintenance Manual.

FIGURE 3-9. Outline of Maintenance Manual

1. General Information

1.1 Introduction - Summarizes the general nature of the software to be maintained

1.2 References

2. Program Descriptions

Describes the programs in the system to be maintained

2.1 Program Attributes for Programs

2.1.1 Programming language

*2.1.2 Input (reference User's Guide)

*2.1.3 Processing features (variables, algorithms, error handling, linkages, etc.) (reference User's Guide)

*2.1.4 Outputs (reference User's Guide)

*2.1.5 Interfaces (reference Detailed Design Document)

*2.1.6 Operating procedures to run the program (reference User's Guide)

2.N Program Attributes for Program N (repeat 2.1)

3. Program Environment

Identifies the resources necessary for running the programs

3.1 Resources for Programs

3.1.1 Program size (memory used)

3.1.2 Data storage on line and off line

3.1.3 Input/Output devices used (tape and disk drives, etc.)

3.1.4 Transmission lines

3.N Resources for Program N (repeat 3.1)

FIGURE 3-9 continued

4. Maintenance Procedures

*4.1 Programming Conventions (refer to detailed design Section 3.3)

*4.2 Verification Procedures (refer to test plans and testing Sections 3.4 and 3.7)

4.3 Error Correction Procedures - Describes all error conditions, their sources, and procedures for correcting them

4.4 Special Maintenance Procedures - Describes any special procedures required for the maintenance of the programs

4.5 Program Listings

*This feature is described in detail in the referenced document or section and is only mentioned here for completeness.

APPENDIX A

DETAILED DESIGN MODULE CHARACTERISTICS

Definition

A module is the smallest software element in the system hierarchy, i.e. subroutine, and performs a maximum of one function.

Module Characteristics

A module shall conform to the following characteristics:

1. It shall perform only one function.
2. It shall have a unique name.
3. It shall constitute a single compilation or assembly entity.
4. As a guideline it should consist of no more than 100 executable statements.
5. It shall have only one point of entry and one point of exit.
6. When invoked by another module, it shall return to the point immediately following the invocation.
7. It shall not relinquish control to another module other than by invocation or by return to the module which invoked it.
8. It shall be documented as a unit.
9. Its internal logic shall be independent of the internal logic of all other modules.
10. It shall be serially reusable, i.e. no assumptions shall be made regarding its previous execution.
11. The organization of the module shall be as follows:
    a. Purpose of the module

b. Module prolog

c. Module flow description statements interspersed among executable statements.

APPENDIX B

DETAILED DESIGN MODULE PROLOG CHARACTERISTICS

B.1 Prolog Contents

All headings listed below shall be a part of every module prolog from the beginning of module design. Those items for which information is not available at that time shall contain the entry "NOT AVAILABLE". As coding progresses and the information becomes available, it shall be entered into the appropriate place within the prolog. When coding is completed, the prolog shall be checked for completeness, for accuracy, and for consistency with the code.

1. Module Name: The module invocation name and the English language name (full name or phrase from which the module invocation name is derived).
2. Function: A concise description of the function of the module. The function is the net effect of one execution module; it is not an explanation of how the module accomplishes its function. A module shall have only one function.
3. Author Names: The names (first initial and last name) and organizational affiliations of the module designer(s) and module programmer(s).
4. Calling Sequence: A list and brief description of the parameters of the calling sequence. Any constraints on size or range of values shall be noted here. Each parameter shall be labeled as an input parameter (used inside the module without initialization) or an output parameter (modified by this module). The use of parameters on both input and output parameters shall be avoided.

5. Method: A brief narrative describing the module processing shall be specified. This shall be written so that the reader can easily relate the narrative to the module function (item 2) and to the module flow (item 15).

6. Error Processing: All error or exception conditions which can be detected by this module shall be identified along with the response to each condition.

7. File References: Each file, including data base elements, shall be listed by name, purpose of reference, and type of reference (e.g., Write, Read, Store, Retrieve, Update, Create, Delete).

8. Global Storage References: All global storage blocks (e.g., named COMMON blocks in Fortran) which are referenced by this module shall be listed by name, purpose of reference, and type of reference (i.e., Store or Retrieve). Include list and description of each variable contained in the global storage blocks.

9. External Module/Macro References: All references to external modules or macros shall be listed by name and purpose of reference.

10. Internal Tables and Variables: The format of each nontrivial table, buffer, or work area shall be described. Include list and purpose of each major internal variable.

11. Dependencies: Any known dependencies on a specific computer, operating system, or compiler shall be described.

12. Assumptions/Restructions: All assumptions (especially those which result in algorithm simplification) and restrictions (e.g., unit conventions, size limitations) regarding processing logic or data shall be clearly specified.

13. Reference: References to any additional information regarding the module not covered above, such as file formats, data structures, or mathematical/scientific techniques shall be clearly identified.

14. Change History: All changes shall be listed here. Each entry shall contain the following information: name (first initial, last name) and organization affiliation of the person making the change, date of implementation of the change, project identification information (project name, contract number, task number, etc.), references to pertinent change control documents, and a short (one line) description of the purpose of the change. Change history data shall be listed in reverse order of implementation, i.e., the most recent change shall be listed first and the oldest change listed last.

15. Module Flow: A detailed description of the module processing will be provided. The description will use structured design constructs, such as If-Then-Else, Do-While, in combination with an English language vocabulary. The description will address each decision path and loop and will clearly indicate decision path and loop dependencies.

B.2 Prolog Maintenance

The prolog represents the final documentation of the module. Because other personnel will use the prolog for interface information and for understanding the module, the prolog shall be updated each time the module is modified.