NASA TM- 85396

NASA-TM-85396
19830024084

SOFTWARE ENGINEERING LABORATORY SERIES SEL-81-205

RECOMMENDED APPROACH TO SOFTWARE DEVELOPMENT

APRIL 1983

NASA

National Aeronautics and
Space Administration

Goddard Space Flight Center
Greenbelt, Maryland 20771

FOREWORD

The Software Engineering Laboratory (SEL) is an organization sponsored by the National Aeronautics and Space Administration Goddard Space Flight Center (NASA/GSFC) and created for the purpose of investigating the effectiveness of software engineering technologies when applied to the development of applications software. The SEL was created in 1977 and has three primary organizational members:

NASA/GSFC (Systems Development and Analysis Branch)
The University of Maryland (Computer Sciences Department)
Computer Sciences Corporation (Flight Systems Operation)

The goals of the SEL are (1) to understand the software development process in the GSFC environment; (2) to measure the effect of various methodologies, tools, and models on this process; and (3) to identify and then to apply successful development practices. The activities, findings, and recommendations of the SEL are recorded in the Software Engineering Laboratory Series, a continuing series of reports that includes this document. A version of this document was also issued as Computer Sciences Corporation document CSC/TM-83/6019.

The contributors to this document include

Frank McGarry	(Goddard Space Flight Center)
Jerry Page	(Computer Sciences Corporation)
Suellen Eslinger	(Computer Sciences Corporation)
Victor Church	(Computer Sciences Corporation)
Phillip Merwarth	(Goddard Space Flight Center)

Single copies of this document can be obtained by writing to

Frank E. McGarry
Code 582.1
NASA/GSFC
Greenbelt, Md. 20771

ABSTRACT

This document presents a set of guidelines for an organized, disciplined approach to software development, based on data collected and studied by the Software Engineering Laboratory (SEL) since 1977 for 46 flight dynamics software development projects. It describes methods and practices for each phase of a software development life cycle that starts with requirements analysis and ends with acceptance testing; maintenance and operation is not addressed. For each defined life cycle phase, this document presents guidelines for the development process and its management, and the products produced and their reviews. This document is a major revision of SEL-81-105.

PRECEDING PAGE BLANK NOT FILMED

TABLE OF CONTENTS

PRECEDING PAGE BLANK NOT FILMED

TABLE OF CONTENTS (Cont'd)

TABLE OF CONTENTS (Cont'd)

LIST OF ILLUSTRATIONS

LIST OF TABLES

1 INTRODUCTION

INTRODUCTION

The Software Engineering Laboratory (SEL) was established in 1977 by the National Aeronautics and Space Administration Goddard Space Flight Center (NASA/GSFC) to investigate the effectiveness of software engineering techniques applied in developing ground support flight dynamics systems. The investigation's goals are (1) to understand the software development process in a particular environment; (2) to measure the effects of various development techniques, models, and tools on this development process; and (3) to identify and apply improved methodologies in the GSFC environment. The results of SEL research should enable GSFC to produce better, less costly software.

This document presents a set of software development guidelines for a disciplined approach to software development, with special emphasis on management considerations. These recommendations are based on data collected and studied by the SEL since 1977 for 46 flight dynamics software development projects. Most of the software development projects studied by the SEL were performed by an independent contractor. For some projects, however, the software development team consisted of both GSFC and contractor personnel. Developing a flight dynamics software system involves diverse skills and many staff-years of effort in specifying, designing, implementing, integrating, and testing complex computer programs. This document describes the software development life cycle from the technical manager's perspective and provides useful techniques for managing software development.

This document is neither a manual on applying the technologies described here nor a tutorial on monitoring a Government contract. Instead, it describes the methodologies and tools that the SEL recommends for use in each life cycle phase to produce reliable, cost-effective software.

This document is primarily for technical managers of software development efforts. However, it is also intended for higher level managers who are concerned with schedules and budgets and for senior technical personnel (such as designers, analysts, and programmers) who are responsible for implementing the recommended procedures. The recommended software development guidelines are appropriate for both GSFC and contractor personnel. Although the recommended guidelines have been formulated based on the development of flight dynamics systems, there is no reason to believe that the recommended guidelines are not applicable to any software development project.

The SEL continually monitors and studies flight dynamics support software, including software developed by both GSFC employees and contractor personnel. It anticipates that data will continue to be collected and analyzed in the future. The recommendations in this document will be refined and enhanced as more knowledge is obtained about the production of better, less costly software and as more progress is made toward achieving the goals of the SEL.

1.1 DOCUMENT OVERVIEW

Section 1 describes the document's purpose and its intended audience and establishes the general background for the remainder of the document. Sections 1.2 and 1.3 briefly describe the SEL and the flight dynamics software development environment.

Section 2 describes the typical software development life cycle in the flight dynamics area, identifying the major phases and their characteristics.

Section 3 describes in detail the recommended guidelines for software development. It discusses the major activities, end products, methodologies, tools, and measures applicable to each life cycle phase. In addition, it recommends management activities for each life cycle phase.

Section 4 discusses management and control of the development process, and Section 5 summarizes key aspects of successful and unsuccessful projects. Appendix A discusses formal reviews, Appendix B presents the content and format of documents, and Appendix C provides a brief example of some steps in organizing a project. Appendix D summarizes the key information of the document.

A glossary, references, and a bibliography of SEL literature are also included.

1.2 SOFTWARE ENGINEERING LABORATORY

The SEL monitors and studies all software developed by the Systems Development Section at NASA/GSFC, which is responsible for producing flight dynamics support software for GSFC-supported missions. To date, 46 projects developed by both GSFC and contractor employees have been studied. They range in size from 1,500 lines of source code to more than 110,000 lines. Much of the data is collected on a series of forms completed by project personnel throughout the development effort. Data is also collected through computer accounting monitoring, personal interviews, automated tools, and summary management reviews. From investigating projects totaling more than 1.5 million lines of source code, the SEL has gained insight into the software development process and has begun to identify trends in the effects of applying various techniques to the software development projects.

The SEL approach to software engineering research is based on the high-level software development model shown in Figure 1-1. The four components of this model are a problem statement, an environment, a process or activity, and a product (software). The development process is divided into seven sequential phases of activity. These phases are described in more detail in Section 2.

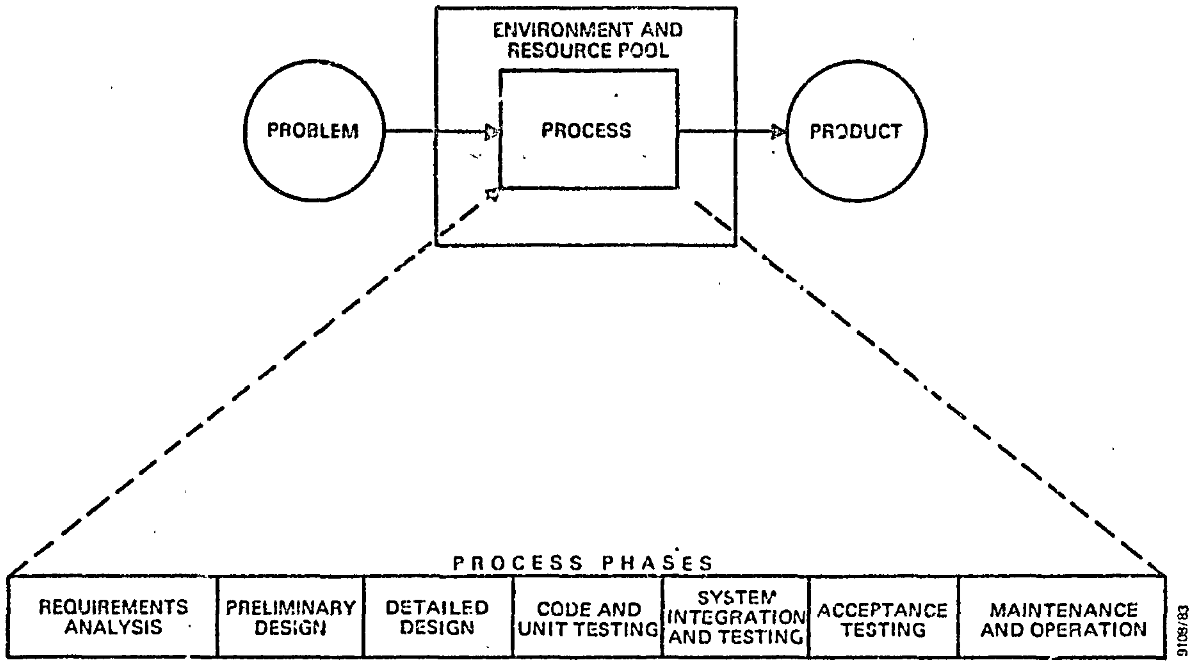


Figure 1-1. Software Development Model

Introduction

SEL research first attempts to understand the software development process currently in operation and its environment. This understanding provides a baseline for measuring the effects of attempted improvements. Next, the SEL tries to improve that process and environment to produce high-quality software with fewer errors at a lower cost. To achieve these goals, the SEL identifies the development techniques available, evaluates these techniques to determine the most effective ones, adapts the "best" techniques for optimal performance, and applies the customized techniques to the software development process. The recommendations in this document are based on the application of this four-step procedure to a large set of software development technologies.

The technologies studied by the SEL may be classified into four major areas of concern: methodologies, tools, models, and measures. Methodologies are systematic applications of prescribed principles to the development process. Tools are software aids used during the development process to make it easier for development team members to do their work. Models explain and/or predict some aspect of the development process and are usually formulated as mathematical equations relating two or more quantitative factors. Measures define, explain, and predict software development qualities; they may be objective or subjective.

Section 3 identifies the technologies in these four areas that the SEL recommends for application to the software development process during the development life cycle phases.

Reference 1 contains additional information on the activities of the SEL and further details on the results of SEL research.

1.3 FLIGHT DYNAMICS ENVIRONMENT

The empirical basis for this document is the studies of flight dynamics software conducted by the SEL since 1977. Flight dynamics software includes applications to support attitude determination, attitude control, maneuver planning, orbit adjustment, and general mission analysis. The attitude systems, in particular, form a large and homogeneous group of software that has been studied extensively.

The attitude determination and control systems are designed similarly for each mission using a standard executive support package, the Graphic Executive Support System (GESS), as the controlling system. All these systems are designed to run in batch and/or interactive graphic mode. Depending on mission characteristics (for example, the type of data available and the accuracy required), these systems may range from 30,000 to approximately 120,000 lines of code. The percentage of reused code ranges from 10 percent to nearly 70 percent, with the average system reusing about 30 percent.

The applications developed in the flight dynamics area are mostly scientific and mathematical in nature, with moderate reliability requirements. Severe development time constraints are imposed by the spacecraft launch date; the software must be completed (through acceptance testing) 90 days before the scheduled launch, and the requirements and functional specifications are normally made available less than 2 years before the scheduled launch.

Most flight dynamics software projects use a group of IBM S/360 computers for development. All resources on these machines are extremely limited, and the hardware is very unreliable.[1] Because the machines are shared among the

[1]The mean time between failures for the primary development machine is approximately 6 to 8 hours of operation (see Figure 3-4 of Reference 1).

analysis, software development, and operations areas, software development schedules are affected when simulations, launches, and maneuvers occur. In addition to the IBM S/360s, a DEC PDP-11/70 and a DEC VAX-11/780 are occasionally used to develop utilities and support systems for the flight dynamics area.

Additional information about the flight dynamics software development environment may be found in Reference 1. Section 3.1 of that document presents the results of profile analysis of several large software development projects studied by the SEL. These statistical profiles characterize the software development process, environment, and products for these flight dynamics projects.

2 SOFTWARE DEVELOPMENT LIFE CYCLE

SOFTWARE DEVELOPMENT LIFE CYCLE

The flight dynamics software development process is divided into the following seven sequential phases, collectively referred to as the software development life cycle:

1. Requirements analysis
2. Preliminary design
3. Detailed design
4. Implementation (code and unit testing)
5. System testing (system integration and testing)
6. Acceptance testing
7. Maintenance and operation

This division is shown in Figure 1-1 (page 1-5), which illustrates the SEL software development model.

For the purpose of this document, the software development life cycle is divided into calendar phases rather than activity phases; that is, the seven life cycle phases subdivide the software development effort into seven sequential periods of time that do not overlap. Each calendar phase of the software development life cycle is characterized by specific activities and the products produced by those activities.

Activities that are characteristic of one calendar phase, however, may be performed in other phases. For instance, analyzing the requirements, which makes up most of the effort during the requirements analysis phase, continues at a lower level throughout the software development life cycle. Once a development activity is started, it continues throughout the life cycle; however, the level of effort for early life cycle activities continually decreases. SEL data shows that the estimated size of large flight dynamics systems grows between 15 and 40 percent after implementation starts (that is, after completion of the detailed design) because of uncertainty in the estimation process[1], new requirements, and requirements changes. This growth causes requirements analysis and design activities to continue during subsequent life cycle phases. Figure 2-1 illustrates the activities performed during each (calendar) life cycle phase as a percentage of the total staff effort.

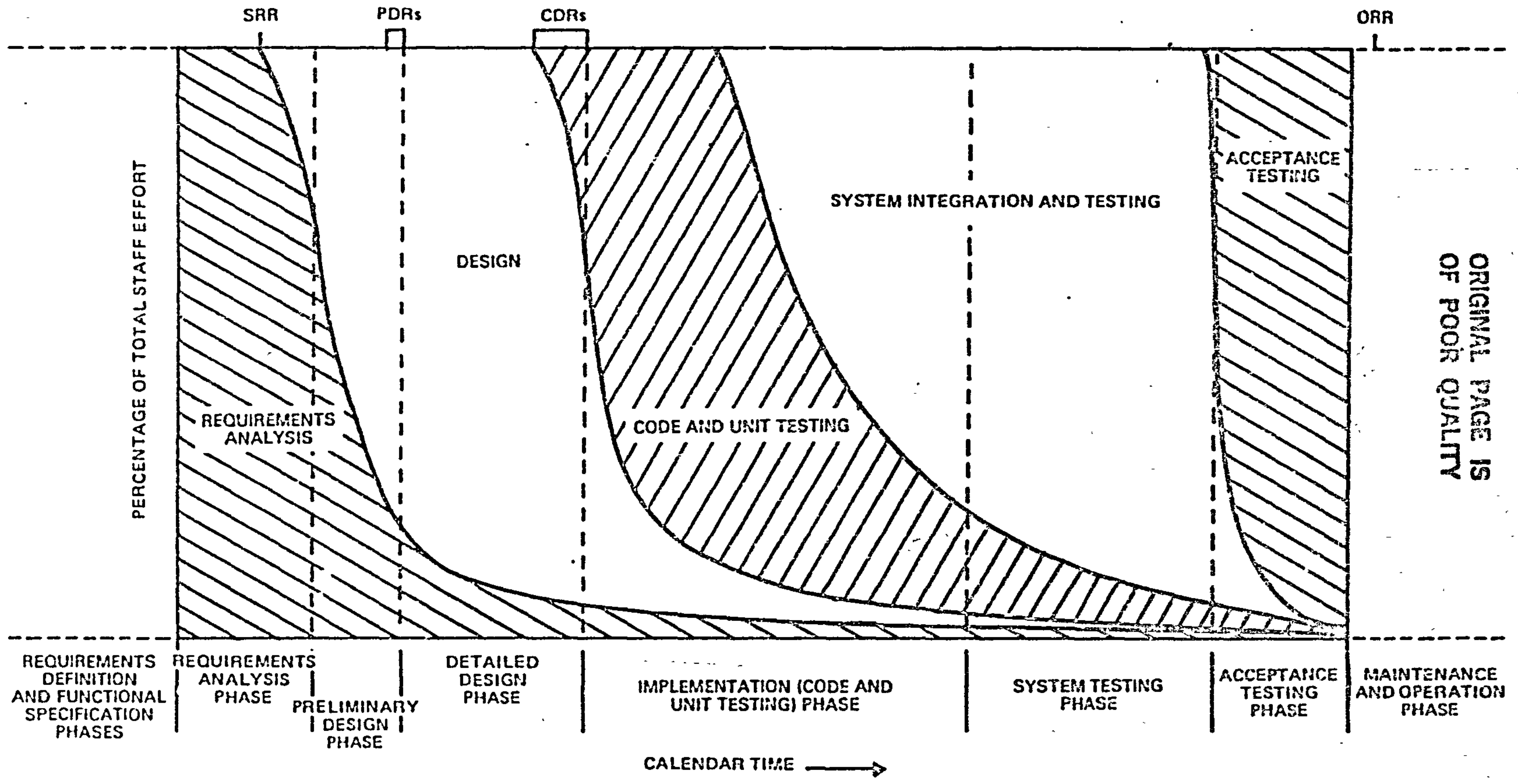
The following subsections define the seven software development life cycle phases, and Section 3 describes them in more detail.

2.1 REQUIREMENTS ANALYSIS

Before the requirements analysis phase begins, a team other than the development team defines the requirements and produces a functional specifications and requirements document. The requirements analysis phase begins when the requirements definition team completes the draft of the functional specifications and requirements document. It is the first phase of the software development life cycle, and in it, the functional specifications are translated from mission terms into a software-supportable form.

In this phase, the development team analyzes the functional specifications and requirements document from a software system viewpoint and recasts the requirements in terms suitable for software design. The development team assesses the completeness and feasibility of the requirements, identifies missing or to-be-determined (TBD) requirements, specifies all external interfaces, and makes the initial determination and allocation of resources. Close interaction with the requirements definition team is necessary for the development team to clarify and amplify the requirements. The development team gives the results of the requirements analysis

[1]See Table C-1 on page C-4 of Appendix C and Figure C-1 on page C-8 for information on uncertainty limits.



NOTE: FOR EXAMPLE, AT THE END OF THE IMPLEMENTATION PHASE (4TH DASHED LINE), APPROXIMATELY 79% OF THE STAFF ARE INVOLVED IN SYSTEM INTEGRATION AND TESTING; APPROXIMATELY 2% ARE ADDRESSING REQUIRMENTS CHANGES OR PROBLEMS; APPROXIMATELY 2% ARE DESIGNING MODIFICATIONS; AND APPROXIMATELY 17% ARE CODING AND UNIT TESTING CHANGES.

9103-(51)-33

Figure 2-1. Activities by Percentage of Total Development Staff Effort

to the requirements definition team for incorporation into the final version of the functional specifications and requirements document. They also prepare a summary report of the results as a basis for preliminary design. When the final version of the functional specifications and requirements document has been completed, a system requirements review (SRR) is held to evaluate the completeness of the requirements.

2.2 PRELIMINARY DESIGN

During the preliminary design phase, the development team defines the software system architecture based on the requirements given in the functional specifications and requirements document. The team translates this architecture into software requirements in the requirements analysis summary report. During this phase, the development team specifies major functional subsystems, input/output interfaces, processing modes, and implementation strategy. The requirements evaluated during the requirements analysis phase are translated into functional capabilities and are organized into major subsystems. All internal and external interfaces are completely defined to the subsystem level, and the design is refined to two levels below the subsystem drivers. Figure 2-2 illustrates the hierarchical levels of a software system baseline diagram (treechart). The development team documents the functional design of the system in the preliminary design report. The preliminary design phase culminates in the preliminary design review (PDR), where the development team formally presents the functional design for review. The preliminary design is considered complete when responses to the PDR comments and criticisms have been incorporated in the functional design.

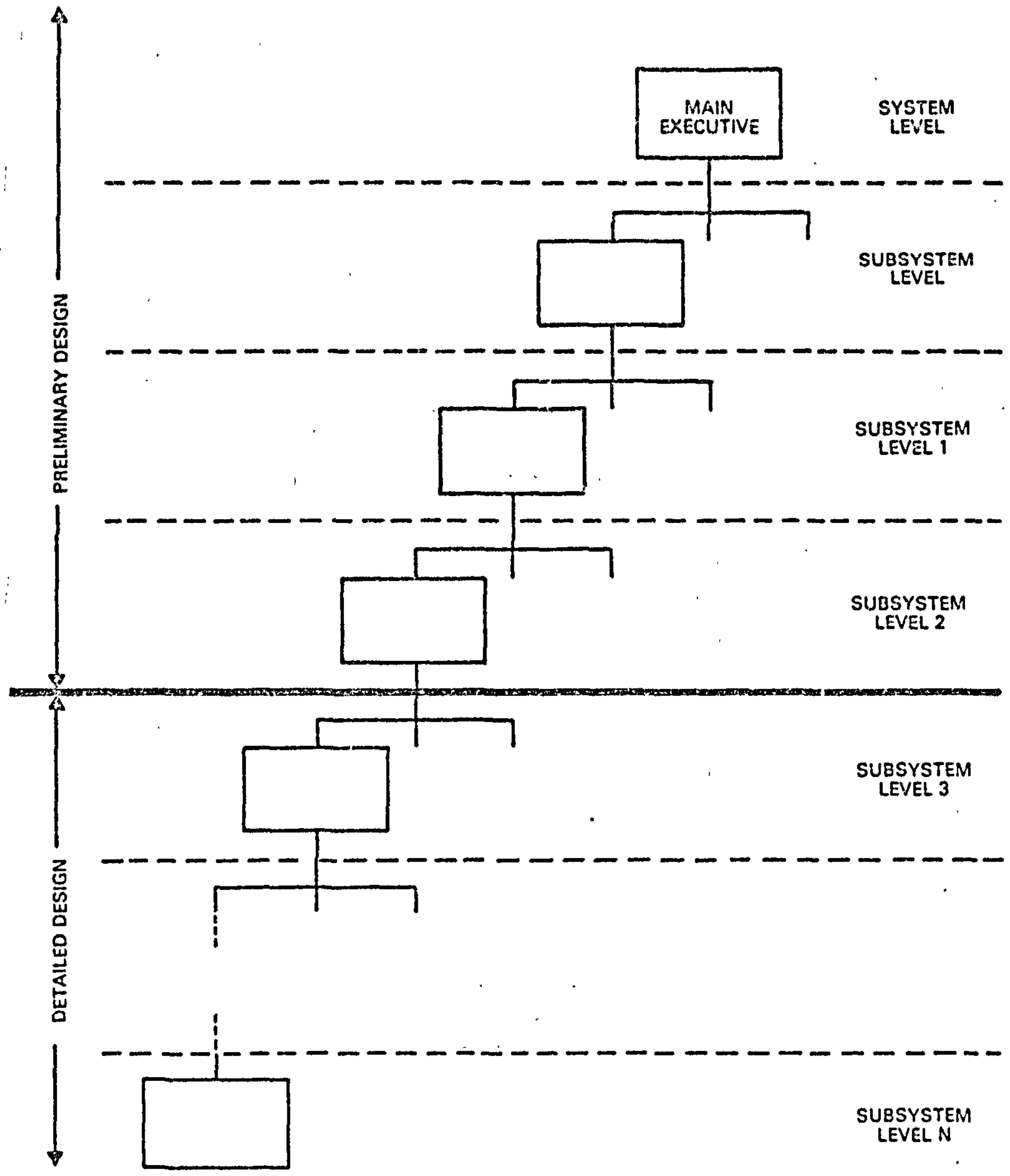


Figure 2-2. Hierarchical Levels of a Software System Baseline Diagram (Treechart)

2.3 DETAILED DESIGN

During the detailed design phase, the development team extends the system architecture defined in preliminary design to the subroutine level. By successive refinement techniques, they elaborate the preliminary design to produce "code-to" specifications for the system.

All formalisms for the system design specifications are produced, including functional and procedural descriptions of the system; data flow descriptions; complete descriptions of all user input, system output (for example, screen, printer, and plotter), and input/output files; operational procedures; functional and procedural descriptions of each module; complete descriptions of all internal interfaces between modules; and build/release capabilities. The team documents these design specifications in the detailed design document, which forms the basis for implementation. The detailed design phase culminates in the critical design review (CDR), where the development team formally presents the "code-to" specifications for review. The detailed design is considered complete when the responses to the CDR comments and criticisms have been incorporated in the detailed design document.

2.4 IMPLEMENTATION (CODE AND UNIT TESTING)

In the implementation (code and unit testing) phase, the developers code new modules from the design specifications or revise old code to meet new requirements, integrate each module into the growing system, and perform unit and integration testing to ensure that the newly added capabilities function correctly.

In a typical project, the developers build several subsystems simultaneously from individual components. The team repeatedly tests each subsystem as new components are coded and integrated into the evolving software. At intervals, they combine subsystem capabilities into a complete working system for testing end-to-end processing capabilities. The sequence in which functions are coded and integrated into executable subsystems and the process of combining these builds into systems are defined in the implementation plan produced during detailed design by the development managers.

The team also produces a system test plan and drafts of the user's guide and system description documents during this phase in preparation for the system integration and testing phase that follows. Implementation is considered complete when all code for the system is produced, tested, and integrated into the system.

An independent acceptance test team prepares the acceptance test plan based on the information in the functional specifications and requirements document. The acceptance test team usually consists of analysts who will use the system, and it frequently includes members of the organization that specified the requirements.

2.5 SYSTEM TESTING (SYSTEM INTEGRATION AND TESTING)

During the system testing (system integration and testing) phase, the development team validates the completely integrated system produced by the implementation phase. This means that functional testing of end-to-end system capabilities is performed according to the system test plan developed during the preceding phase. The system test plan is based on requirements set forth in the functional specifications and requirements document. Successfully completing the tests specified in the test plan demonstrates that the system satisfies the requirements.

In this phase, the developers correct any errors uncovered by system tests. They also update the draft documentation to reflect the system as it exists when system testing is complete. System testing is considered complete when all tests specified in the system test plan have been run successfully.

2.6 ACCEPTANCE TESTING

During the acceptance testing phase, the system is tested by an independent acceptance test team to ensure that the software meets all requirements. Testing by an independent team (one that does not have the developers' preconceptions about the functioning of the system) provides assurance that the system satisfies the intent of the official requirements.

During this phase, the development team assists the acceptance test team and usually executes the acceptance tests under their direction. Any errors uncovered by the acceptance tests are corrected by the development team. Acceptance testing is considered complete when all tests specified in the acceptance test plan have been run successfully. After the successful completion of acceptance testing, the development team delivers final versions of the software and the system documentation to the customer and an operational readiness review (ORR) is held to evaluate the readiness of the system to support operations.

2.7 MAINTENANCE AND OPERATION

At the end of acceptance testing, the system becomes the responsibility of a maintenance and operation group. This marks the beginning of the maintenance and operation phase. The nature of the activities during maintenance and operation is highly dependent on the type of software involved. For most flight dynamics software, this phase typically lasts the lifetime of the spacecraft and involves relatively few changes to the software. For some support software, however, this phase may be much longer and more active as the software is modified to respond to changes in the requirements and environment.

The maintenance and operation phase is outside the scope of this document. However, enhancements and error corrections also progress through a development life cycle but at a much lower level of effort than original development. Therefore, the recommendations made for original development are, for the most part, applicable to development during the maintenance and operation phase.

3 RECOMMENDED SOFTWARE DEVELOPMENT GUIDELINES

This section presents the SEL's recommended software development guidelines. Each major subsection presents the software development guidelines that are of primary importance to the basic development team[1] for a particular software development life cycle phase.[2] Each major subsection begins with two summaries. The first summarizes the actions and transactions of basic development team members, i.e., their major activities, the end products they produce, and the methodologies and tools they use for the life cycle's primary development activity. The second summarizes special actions and transactions of the development team managers, i.e., their special activities and the measures they use during the life cycle phase. The activities in the management summary are listed in blocks. The first block of activities are those that are specific to the phase and are also of interest to higher level managers. Subsequent blocks list those activities that are frequently assumed to be going on or have occurred.

Although the approach is described in terms of calendar phases, it applies to specific development activities whenever they occur. For instance, the methodologies and tools recommended for use during the detailed design phase apply to detailed design activities when they are performed during subsequent life cycle phases. This overlap of activities is discussed in Section 2 (see especially Figure 2-1 on page 2-4).

[1] The basic development team consists of the customer interface, who monitors resources and progress; the project manager, who serves as technical consultant and manages project resources; the project leader, who provides technical direction and day-to-day supervision; and the developers, who do the technical work. See Table C-8 on pages C-16 and C-17 of Appendix C and Table 1-1 and Figure 1-2 of Reference 1.

[2] The maintenance and operation phase is not addressed in this document.

3.1 REQUIREMENTS ANALYSIS

DEVELOPMENT	ACTIONS AND TRANSACTIONS
ACTIVITIES	Software Development Plan Prepared Functional Requirements Amplified Performance Requirements Analyzed Operational Requirements Determined External Interfaces Identified Report and Display Requirements Determined TBD Requirements Identified System Size Estimated Reusable Software Identified Computer Resources Determined Hardware Selected Requirements Analysis Summary Report Prepared SRR Held
END PRODUCTS	Software Development Plan Requirements Analysis Summary Report
METHODOLOGIES	Project Notebook Data Collection Librarians Unit Development Folders Requirements Question and Change Records Structured Analysis (Complex Data Processing)
TOOLS	Configuration Analysis Tool (CAT)

9103-(51)-83

3.1 REQUIREMENTS ANALYSIS

MANAGEMENT	ACTIONS AND TRANSACTIONS
ACTIVITIES	Software Development Plan Reviewed Schedule and Staffing Planned Requirements Analysis Summary Report Reviewed Preliminary Design Transition Planned Team Trained Standards and Procedures Enforced Progress Monitored Visibility Promoted System Size Estimated Resources and Cost Estimated Team Interaction Coordinated
MEASURES	TBD Requirements Requirements Questions and Answers Requirements Changes Subjective Evaluations

9103-(51)-83

The recommended software development guidelines for the requirements analysis phase are described in detail below.

3.1.1 MAJOR ACTIVITIES

Requirements analysis begins when the requirements definition team completes the functional specifications and requirements document. This document presents the functional requirements of the system, including system input and output, algorithms, and timing and accuracy requirements. The development team analyzes the contents of this document for completeness, consistency, clarity, and feasibility, and then translates the requirements into a form suitable for beginning the design. During the requirements analysis phase, the team

- Amplifies and clarifies the functional requirements and the algorithms specified to satisfy those requirements
- Analyzes the algorithms, mathematical formulations, error and stability requirements, and timing and accuracy requirements for completeness and feasibility
- Determines operational requirements (scenarios)
- Ensures that all requirements from the mission needs statement, the mission problem statement, and the end users have been addressed
- Identifies all external interfaces (both input and output)
- Determines report and display specifications
- Identifies requirements that are missing or yet to be determined (collectively known as <u>TBD requirements</u>)
- Identifies any existing software that can be used

Requirements Analysis

- Determines computer resource requirements and availability
- Participates in the hardware selection process
- Communicates findings to the requirements definition team
- Prepares a summary report as the basis for beginning preliminary design
- Participates in the system requirements review (SRR)

During the requirements analysis phase, the development team works closely with the requirements definition team, who must answer their questions about the requirements and respond to their requests for requirements changes. Generally, the two teams hold requirements review meetings where they clarify requirements, discuss problems, and identify items needing action. The development team provides the requirements definition team with the results of their analysis for incorporation into the final version of the functional specifications and requirements document.

This phase culminates in a final requirements review meeting of the requirements definition team and the development team and their managers. Maintenance and operation personnel and their managers may also be present. The purpose of this meeting is to ensure the correctness and completeness of the requirements from the viewpoints of all those concerned and to identify and assess the impact of any remaining TBD requirements. Comments and criticisms resulting from this meeting are given to the requirements definition team so that those issues will be addressed in the final version of the functional specifications and requirements document.

When the final version of the functional specifications and requirements document has been completed, an SRR is held to evaluate the completeness of the requirements. The SRR is attended by the requirements definition team and its managers, the development team and its managers, and others involved with the system. See Section A.1 of Appendix A for details about the SRR.

Another important function of the requirements analysis phase is to produce an initial estimate of the system's size and of the schedule and staffing required for development. This topic is addressed more fully in Section 3.1.6, page 3-13, under "Key Management Activities."

3.1.2 END PRODUCTS

The requirements analysis summary report is the primary product. This report summarizes the results of requirements analysis and establishes a basis for beginning preliminary design. See Section B.3 of Appendix B for information on the format and contents of the requirements analysis summary report.

3.1.3 METHODOLOGIES

The recommended methodologies are

- Project notebook
- Data collection
- Librarians
- Unit development folders
- Formal recording mechanisms for requirements questions and changes
- Structured analysis for complex data processing

These methodologies are discussed in the following subsections.

3.1.3.1 Project Notebook

The project notebook is established and maintained by the development managers to provide readily accessible summary information on the key aspects and phases of the project. The notebook is part of the project's files. The information kept in the project notebook is current; i.e., it is updated weekly, biweekly, or monthly depending on the type of information. See Section B.2 of Appendix B for information on the format and contents of the project notebook.

3.1.3.2 Data Collection

To understand the development process and to monitor the progress of a project, software engineering data must be collected throughout the development life cycle. The SEL recommends that managers make software engineering data collection a natural byproduct of their managing techniques. This topic is treated in more detail in the Software Manager's Handbook (Reference 2).

3.1.3.3 Librarians

At GSFC, the librarians are a separate group of personnel who are responsible for certain clerical and data entry functions. The relationship between the librarians and the development team is described in Section 1.4 of Reference 1. During a project, the librarians maintain the project library (or project notebook), which is a repository of all project information. They also maintain online project libraries, enter code, and operate various software tools in support of project activities.

During the requirements analysis phase, the librarians establish the project library. In it, they include such items as the functional specifications and requirements document (draft and final versions as available), questions on requirements and responses to questions, requests for requirements changes and responses to requests, and the requirements analysis summary report. In general, the project library contains any written material produced by the development team for the purpose of recording decisions or communicating information. Necessary management information, such as schedules and staffing plans, are also included. Management information produced during the requirements analysis phase is discussed in Section 3.1.6, page 3-13.

Librarian functions during this phase are generally limited to the operation of software tools, which are discussed in Section 3.1.4, page 3-10.

3.1.3.4 Unit Development Folders

The project library is organized according to functional units so that all information pertaining to one topic can be found in one location in a unit[1] development folder. During the requirements analysis phase, the most logical organization of the materials collected from the development team is into the general categories (units) of the system's functional requirements (for example, input, output, and algorithms). For the project library to be useful throughout the development life cycle, it must be established at the beginning of the requirements analysis phase; it must contain all material pertinent to the project; and it must be logically organized. A description of unit development folders is contained in Reference 3, for example.

3.1.3.5 Formal Recording Mechanisms

As a part of configuration management procedures, the development team uses formal recording mechanisms to communicate requirements questions and requirements changes. One

[1]A unit is defined by the development manager for convenience; e.g., units may be schedules, system size estimates, resource estimates, external interfaces, subsystem details, development plans, implementation plans, user's guide, and system description.

mechanism is the requirements question form. The development team uses this form to question the requirements definition team about the requirements. Responses to requirements questions must be in written form. The requirements definition team managers use the form to assign personnel and due dates for their team's response to the developers. The forms are also used to track TBD requirements.

Another procedure is used for requirements modifications. A request for a requirements modification is made using a requirements change request form (or engineering change request (ECR)), on which the requested change and its justification are described. Requested changes to the requirements must be approved by the manager of the requirements definition team and the Configuration Control Board (CCB). After approving a change, the manager adds it to the functional specifications and requirements document.

3.1.3.6 Structured Analysis

No particular methodology is recommended for requirements analysis. However, structured analysis is useful for systems or subsystems with large quantities of input or output data or complex data processing requirements. Structured analysis is described in Reference 4, for example.

3.1.4 TOOLS

An online configuration management tool is recommended for use in configuration management throughout the project. During the requirements analysis phase, managers can use such a tool to track requirements questions, TBD requirements, and requests for requirements changes. The online Configuration Analysis Tool (CAT) program was developed for the SEL for use in flight dynamics projects. CAT is documented in Reference 5.

The SEL recognizes the need for an automated requirements language,[1] but has been unable to identify one that is adequate and cost effective for the flight dynamics environment.

3.1.5 MEASURES

The following subsections describe various measures and evaluation criteria for managers to use to assess the results of the requirements analysis phase and to determine whether enough progress has been made to begin design.

3.1.5.1 Objective Measures

Managers can monitor the progress of requirements analysis by examining the number of requirements questions, responses to questions, and requirements changes. Several signals should alert the manager to problems. For example, a growing gap between the number of questions submitted and the number of responses received or a large number of requirements changes due to errors may indicate problems with the clarity, correctness, or completeness of the requirements as presented in the functional specifications and requirements document. Managers can use data from similar past projects to assess the meaning of the relative sizes of these numbers.

The number of TBD requirements is the most important measure to be examined during this phase, since unresolved TBD requirements can necessitate severe design changes later in the project. The TBD requirements must be categorized according to their severity. TBD requirements concerning external interfaces are the most critical, especially if they involve system input. Internal algorithms are generally not

[1]Examples of requirements languages include Problem Statement Language/Problem Statement Analyzer (PSL/PSA) (Reference 6) and Multi-Level Expression Design Language - Requirements Level (MEDL-R) (Reference 7).

as severe, unless they concern data processing requirements. Output requirements are, in general, not as severe unless they concern data being transmitted to other systems.

A TBD requirement is considered severe if it could affect the functional design of one or more subsystems or of the high-level data structures needed to support the data processing algorithms. Preliminary design should not proceed until all severe TBD requirements have been resolved. A TBD requirement is considered nominal if it affects a portion of a subsystem involving more than one module. Preliminary design can proceed unless large numbers of TBD requirements exist in one functional area (for example, more than 5). However, these TBD requirements must be resolved during preliminary design. An incidental TBD requirement is one that affects only the internals of one module. Incidental TBD requirements must be resolved by the end of detailed design.

For each TBD requirement, managers must estimate the effect on system size, required effort, cost, and schedule. Often the information necessary to resolve a TBD requirement is not available until later, and design must begin to meet fixed deadlines. These estimates will help predict the uncertainty in the development schedule due to unresolved TBD requirements.

3.1.5.2 Evaluation Criteria

To determine whether or not the development team is ready to proceed with preliminary design, managers must consider the following questions:

- Is all external input and output completely defined?
- Have all TBD requirements been identified and their impact assessed?

- Are all necessary algorithms identified? Are the identified algorithms complete and correct? Are they optimal? Are error and stability limits defined?
- Are the environmental constraints (for example, timing, memory, and accuracy) clear?
- Are the requirements feasible, given the environmental constraints? Are sufficient computer resources available?
- Are the requirements traceable? Does the functional specifications and requirements document provide a basis for defining acceptance tests?

3.1.6 KEY MANAGEMENT ACTIVITIES

Planning is the manager's primary activity during the requirements analysis phase. During this phase, the development team managers produce the software development plan. Toward the end of the phase, the transition to the preliminary design phase must be planned. These planning activities are in addition to other activities such as monitoring progress, ensuring cooperation among all groups involved, and reviewing the results of requirements analysis. These key activities are discussed in further detail in the following subsections.

3.1.6.1 Software Development Plan

The software development plan contains specific information about the technical and management approaches of the current project throughout its life cycle. The development team managers prepare this document during the requirements analysis phase. Because of the primary importance of this plan, it is described in great detail in the Software Manager's Handbook (Reference 2).

The SEL often uses the flight dynamics software development projects as experiments in software engineering research. Thus, frequently, a specific software engineering approach is applied to a project to evaluate its effectiveness. Therefore, data collection procedures and details concerning the application of the specified approach must be established at the beginning of the project life cycle.

3.1.6.2 Resource and Cost Estimates

At the beginning of this phase, the managers must estimate the amount of code to be developed and the effort required to develop it. Sufficient information is not usually available to use a sophisticated resource estimation model, but a rough model can be applied. Historical knowledge of similar systems can be used to estimate the size of the system, and historical productivity figures can be used to estimate the amount of effort required. From these estimates, the amount of time and effort necessary for each life cycle phase is allocated, and the first cost figures and schedules are produced.

3.1.6.3 Phase Transition Plans

Toward the end of the requirements analysis phase, managers must plan an orderly transition to the preliminary design phase. They must convey to the development team members the parts of the software development plan that apply to preliminary design (for example, design standards and configuration management procedures) and instruct them in the specific software engineering approach to use during design. They must ensure that the development team is trained in design technologies at the beginning of the preliminary design phase.

3.1.6.4 Other Management Functions

During the requirements analysis phase, the managers must also

- Monitor adherence to planned schedules and resource expenditures.
- Ensure adherence to data collection, quality assurance, and configuration management procedures.
- Review the results of the requirements analysis process.
- Ensure cooperation among the various groups involved (that is, development team, requirements definition team, user organization, and librarians).
- Schedule and participate in the requirements analysis review and ensure that all pertinent groups participate.
- Ensure that all facets of the project are completely visible (that is, know exactly where the project is and where it is going at all times). Project visibility is critically important. Managers must know at all times the exact status of all task activities and the detailed plans for development completion. This is necessary so that problems can be dealt with when they occur rather than later in the process, when their impact is likely to be greater.
- Participate in the SRR.

The Software Manager's Handbook (Reference 2) presents further information about the manager's activities throughout the development life cycle.

3.2 PRELIMINARY DESIGN

DEVELOPMENT	ACTIONS AND TRANSACTIONS
ACTIVITIES	System Partitioned Processing Options Defined Alternative Designs Examined External Interfaces Defined Subsystems Partitioned Subsystem Interfaces Defined Error Processing and Recovery Defined TBD Requirements Resolved Reusable Software Identified Preliminary Design Report Prepared PDR Held
END PRODUCTS	Preliminary Design Report Software Development Plan Update
METHODOLOGIES	Project Notebook Data Collection Librarians Unit Development Folders Requirements Question and Change Records Design Formalisms Design Decision and Change Records Configuration Management Design Walkthroughs Iterative Enhancement Information Hiding Data Abstraction PDL
TOOLS	PDL Processor Source Code Library Management System CAT Resource Estimation

9108-(51)-83

PRECEDING PAGE BLANK NOT FILMED

3.2 PRELIMINARY DESIGN

MANAGEMENT	ACTIONS AND TRANSACTIONS
ACTIVITIES	TBD Requirements Resolved Requirements Changes Reviewed and Assessed Design Reviewed and Walked Through Detailed Design Transition Planned Team Trained Standards and Procedures Enforced Progress Monitored Visibility Promoted System Size Estimated Resources and Cost Estimated Team Interaction Coordinated
MEASURES	TBD Requirements Requirements Changes Requirements Questions and Answers Design Changes Interfaces Design Completion Checklist Subjective Evaluations

9160-(51)-83

The recommended software development guidelines for the preliminary design phase are described in detail below.

3.2.1 MAJOR ACTIVITIES

Preliminary design begins at the end of the requirements analysis phase. At this point, the development team has completed the requirements summary report; the requirements definition team has incorporated the results of the requirements analysis and comments from the requirements review into the functional specifications and requirements document; an SRR has been held; and the development managers have determined that the team understands the requirements well enough to begin design.

During preliminary design, the development team defines the software system architecture based on the requirements given in the functional specifications and requirements document and amplified during the requirements analysis phase. Specifically, the team

- Structurally and functionally partitions the system into major subsystems
- Determines operational scenarios for major processing options (that is, operating procedures)
- Examines alternative design strategies
- Defines all external interfaces to the system--that is, completely specifies all system input and output, including
  - Data set layouts (record content and file structure)
  - User input
  - Screen, printer, and plotter output

Preliminary Design

- Defines all interfaces between subsystems--that is, completely specifies all input and output for each subsystem, including
  - Data set layouts (record content and file structure)
  - Data transferred in memory
  - User input
  - Screen, printer, and plotter output
- Refines the subsystem design to two levels below the subsystem drivers, including preparation of functional baseline diagrams (treecharts) through two levels below subsystem drivers and module prologs and Process Design Language (PDL) through one level below subsystem drivers (see Figure 2-2 on page 2-6)
- Determines error processing and recovery strategy, especially with respect to handling system input/output errors
- Resolves as many remaining TBD requirements as possible; assesses the impact of those not resolved
- Examines all requirements to ensure that they are met by the functional capabilities of the subsystems defined in the preliminary design
- Identifies all existing software to be used in the system
- Prepares preliminary design documentation as a basis for the preliminary design review (PDR)
- Participates in the PDR and then incorporates changes recommended at the PDR into the preliminary design

The preliminary design phase culminates in the PDR, attended by the development team and its managers, the requirements definition team and its managers, and others involved with the system. At the PDR, the development team presents the functional design of the system and the rationale for choosing that design over alternatives. The presentation is based on the preliminary design documentation and may require a series of meetings if the system is large. See Section A.2 of Appendix A for details about the PDR.

For the PDR presentation, the participants evaluate the functional design of the system for completeness and correctness and give comments and criticisms to the development team during and immediately after the presentation. The preliminary design is complete when the development team has adjusted the preliminary design documentation to respond to comments and criticism expressed at the PDR.

3.2.2 END PRODUCTS

The preliminary design report is the primary product. It presents the functional design of the system and forms the basis for the detailed design document produced during the next life cycle phase. See Section B.4 of Appendix B for the format and contents of the preliminary design report.

3.2.3 METHODOLOGIES

The SEL recommends the following methodologies for use during the preliminary design phase:

- Project notebook
- Data collection
- Librarians
- Unit development folders
- Formal recording of design decisions and changes
- Configuration management procedures
- Design walkthroughs

- Iterative refinement
- Information hiding and data abstraction
- PDL

Data collection and maintenance of the project notebook must continue throughout the development life cycle. The remaining methodologies for the preliminary design phase are elaborated in the following subsections.

3.2.3.1 Librarians and Unit Development Folders

The librarians continue to maintain the project library. They add to the library such items as design decision notes, design change forms, and all preliminary design documentation, as well as pertinent management materials. (The management information produced during the preliminary design phase is discussed in Section 3.2.6, page 3-28). Since requirements analysis continues throughout the development life cycle, the development team may produce more requirements questions and requests for requirements changes during this phase. The librarians also add these inquiries and their responses to the project library.

During this phase, the development team managers organize the project library materials in the unit development folders by major subsystems and by functional areas within each subsystem. This organization corresponds to the first level below the subsystem drivers on the functional baseline diagrams (see Figure 2-2 on page 2-6).

The librarians enter module prologs[1] and PDL as well as operate CAT, which is discussed in Section 3.2.4, page 3-25.

[1]Module comments describing the module's purpose, operation, calling sequence arguments, external references, etc.

3.2.3.2 Formal Recording Mechanisms and Configuration Management Procedures

As part of the configuration management of the project, the developers use formal recording mechanisms to document design decisions and changes. When a design decision is made, it is recorded as a design decision note. Because these notes document the design process--particularly the evaluation and selection of alternatives--they are a valuable reference for the developers throughout the software life cycle.

Once design decisions have been finalized by the development team managers, design change forms are used to record further changes. The use of formal design change procedures enables the team managers to control design changes and to ensure that all team members are kept informed of the current state of the design.

The use of formal recording mechanisms for particular life cycle activities must continue throughout the life cycle whenever those activities occur. For instance, requirements questions forms and requirements change request forms continue to be used after the requirements analysis phase for requests for clarification or changes to the requirements.

The procedures for configuration management must be strictly adhered to throughout the development life cycle. These procedures specify the forms to be used for recording various inquiries, decisions, and requests for changes and address the processing of such forms (for example, responsibility for response, authority for approval, and distribution). Strict procedures are especially important in the area of change control.

Configuration management procedures also specify the control of online project libraries. The procedures must specify the point at which modules are moved from the individual developer's jurisdiction and placed in the project libraries. At that time, the modules are placed under configuration control. Any changes are performed by a specially designated person or group of people (usually the librarians) and must be recorded formally by means of a change report form. These procedures for the control of online libraries apply to the preliminary design phase only if a decision is made to place module prologs and PDL under configuration management.

3.2.3.3 Design Walkthroughs

Design walkthroughs are held throughout the preliminary design phase by development team personnel and their managers to review design elements and to identify problem areas and TBD requirements. This peer review is an important quality assurance procedure. After design walkthroughs, managers assign personnel to resolve problems and schedule their response as part of configuration management. The development team leader records any decisions made at a design walkthrou.. in design decision notes.

Design walkthroughs are also used to identify the point at which design elements are placed under configuration control by the development team manager. A design element is usually put under configuration control when it has been incorporated in the preliminary design documentation or presented in a design walkthrough or at the PDR, depending on the status of the design. After that, changes to the design must be made according to the change control procedures and must be recorded by means of design change forms. A description of design walkthroughs is contained in Reference 8, for example.

3.2.3.4 Design Technologies

The SEL recommends iterative refinement (for example, Reference 9) as the primary method for producing the system design. When a substantial amount of existing design and/or code is to be reused (for example, 20 percent or more), iterative refinement is recommended to functionally partition the system into modules. This process is preferred to strict successive refinement (used in top-down design) when adapting the design to the structure of the existing design and code.

In the functional partitioning process, the SEL recommends the principles of information hiding (for example, Reference 10) and data abstraction (for example, Reference 11). An example of these techniques is the use of common interface routines for performing input or output operations so that the format and structure of each external data set are known to only one routine and are transparent to the rest of the system.

3.2.3.5 Process Design Language (Program Design Language)

The SEL highly recommends the use of PDL during design as a very beneficial and cost-effective methodology. Comparable to the blueprint in hardware, PDL communicates the concept of the software design in all necessary detail. It provides a complete, formal, algorithmic specification for a software component. Its use enables the designer to communicate the exact intent of the design and thus reduces errors due to misinterpretation of the design by reviewers and coders. A description of PDL is contained in Reference 12.

3.2.4 TOOLS

An online configuration management tool (for example, CAT) is used throughout the development life cycle. In this phase, it can be used to maintain detailed schedule information at the subsystem and module levels.

One new tool is recommended:

- An automated PDL processor (for example, Reference 13) (if one is available). This tool enforces consistency of PDL usage among development team members and also performs syntax-checking operations.

3.2.5 MEASURES

The following subsections present various measures and evaluation criteria for managers to use in assessing the preliminary design phase.

3.2.5.1 Objective Measures

During this phase, managers continue to use the same objective measures as during requirements analysis. In particular, they monitor

- Number of requirements questions, responses to questions, and requirements changes. The number of design changes must also be examined. Numerous design changes not attributable to requirement changes should alert the manager to problems; these changes may indicate that the development team does not really understand the requirements.

- Number of TBD requirements (see Section 3.1.5.1, page 3-11). Managers must assess how each TBD requirement will affect system size, required effort, cost, and schedule. By the end of this phase, only incidental TBD requirements can be left unresolved.

- Number of interfaces. The number of interfaces per subsystem is an indication of that subsystem's complexity: a subsystem with a large number of interfaces relative to its size will require more time for implementation and thorough testing. Data from past projects of a similar nature can be used to interpret the relative sizes of these numbers.

To help monitor the progress of the preliminary design, managers can produce and use

- <u>A detailed checklist of design formalisms to be produced</u>. Because preliminary design documentation contains all design formalisms produced during preliminary design, all items on the checklist must be completed before the PDR.

3.2.5.2 <u>Evaluation Criteria</u>

To evaluate the correctness and completeness of the preliminary design and determine whether the development team is ready to proceed with detailed design, managers must consider the following questions:

- Have all requirements been mapped into functional capabilities of specific subsystems?
- Have alternative design approaches been examined and rationally discounted, and has the simplest design been chosen?
- Is the partition into subsystems sensible? Are functions and capabilities allocated logically? Does the design minimize the transfer of control information? Does the design have low coupling between subsystems and high cohesion within each subsystem? (Coupling and cohesion are defined in Reference 14, for example.)
- Are all interface descriptions complete at both the system and subsystem level?
- Are the data set layouts for all external data sets completely specified?
- Are the required baseline diagrams and module prologs and PDL provided to a sufficient level of detail?

- Is the error handling and recovery strategy comprehensive?
- Is the estimate of resources adequate and the schedule reasonable? Has time been allocated for contingencies, training, and the like?
- Has the impact of any remaining TBD requirements been assessed?

3.2.6 KEY MANAGEMENT ACTIVITIES

During this phase, the managers' focus begins to change from planning to monitoring. Specifically, managers

- Provide required training for the development team
- Ensure adherence to
  - Design standards
  - Configuration management procedures
  - Reporting procedures
  - Data collection procedures
  - Quality assurance procedures
- Review the design produced, participate in design walkthroughs, and resolve TBD requirements
- Monitor adherence to planned schedule and expenditure of resources, and update cost and resource estimates and schedules
- Ensure that all facets of the project are completely visible
- Coordinate communication between the development team and the other groups with which they must interact (for example, the librarians and the requirements definition team)
- Plan transition to the detailed design phase

- Schedule and participate in the PDR, and ensure that all pertinent groups participate

Further details on the refinement of resource and cost estimates and on phase transition planning are discussed in the following subsections.

3.2.6.1 Resource and Cost Estimates

During the preliminary design phase, managers must monitor the development team's adherence to cost and resource estimates and the schedules in the software development plan (see Reference 2). The percentages of effort and time actually expended versus the percentages of the quantities planned to be expended in terms of the work accomplished are good measures to examine for monitoring progress.

By the end of this phase, the managers can refine and update resource and cost estimates made during the requirements analysis phase. System size is better known, as are resources expended and progress made to date. Enough information is usually available to use a formal resource estimation model. It is important for the manager to use a model that is tuned to the specific environment and corresponds well with the resources expended for similar past projects. The Meta-Model has been developed using SEL data (see Reference 15). However, managers must never completely rely on any formal resource estimation model. Rather, they must use the results of the model, together with historical knowledge of similar systems, to update resource and cost estimates. The new estimates are more accurate because they are based on additional information and model support.

From these new estimates, managers prepare schedules and staffing plans. Schedules are refined to reflect the subsystem division established in the preliminary design. The managers add these new estimates and schedules to the software development plan to form the basis for monitoring progress during the next life cycle phase. The process of monitoring actual progress versus planned progress and updating the plan as more detailed information becomes available continues throughout the project life cycle.

3.2.6.2 Phase Transition Plans

Toward the end of the preliminary design phase, managers must plan the transition to the detailed design phase; i.e., they must plan the detailed design phase so that major subsystems are designed concurrently. In addition, they must determine the staffing levels and assignments necessary to perform the detailed design. Managers usually add personnel to the development team at the beginning of detailed design. They must ensure that these personnel receive any training required and that new members are informed of work assignments, design standards, software engineering approaches, and quality assurance and configuration management procedures. The online libraries must also be established to store module prologs, PDL, and reused code during detailed design.

3.3 DETAILED DESIGN

DEVELOPMENT	ACTIONS AND TRANSACTIONS
ACTIVITIES	Single Functions Refined Baseline Diagrams Prepared I/O Specified PDL and Prologs Specified COMMON Blocks Specified Internal Interfaces Specified TBD Requirements Resolved Reusable Software Identified Detailed Design Document Prepared Implementation Strategy Planned CDR Held
END PRODUCTS	Detailed Design Document Software Development Plan Update Implementation Plan
METHODOLOGIES	Project Notebook Data Collection Librarians Unit Development Folders Requirements Question and Change Records Design Formalisms Design Decision and Change Records Configuration Management Design Walkthroughs Iterative Enhancement Information Hiding Data Abstraction PDL
TOOLS	PDL Processor Source Code Library Management System CAT Resource Estimation

9103-(51)-83

3.3 DETAILED DESIGN

MANAGEMENT	ACTIONS AND TRANSACTIONS
ACTIVITIES	Implementation Strategy Reviewed Implementation Transition Planned TBD Items Resolved Requirements Changes Reviewed and Assessed Design Reviewed and Walked Through Team Trained Standards and Procedures Enforced Progress Monitored Visibility Promoted System Size Estimated Resources and Cost Estimated Team Interaction Coordinated
MEASURES	TBD Items Requirements Changes Requirements Questions and Answers Design Changes Interfaces Design Completion Checklist Design Growth Rate Module Strength Module Coupling Subjective Evaluations

9103-(51)-83

The recommended software development guidelines for the detailed design phase are described in detail below.

3.3.1 MAJOR ACTIVITIES

Detailed design begins after the PDR unless comments and criticism expressed at the PDR indicate serious problems or deficiencies with the preliminary design. During detailed design, the team elaborates the system architecture defined by the preliminary design to the subroutine level. The detailed design process is an extension of the activities begun during preliminary design until "code-to" specifications are complete. Specifically, the team

- Successively refines each subsystem until each component performs a single function and can be coded as a single module
- Prepares detailed baseline diagrams (treecharts) to the subroutine level
- Finishes specifying detailed formats of all system and subsystem input and output
- Completes prologs and PDL for all modules
- Specifies COMMON blocks and internal interfaces between modules
- Specifies the staged implementation plan, including capabilities to be included in each build/release and the detailed milestone schedule for each build/release
- Prepares detailed design documentation as a basis for the critical design review (CDR)
- Participates in the CDR and incorporates changes recommended at the CDR into the detailed design

Detailed Design

The detailed design documentation contains all design formalisms and must be distributed to everyone attending the CDR before the CDR meetings. The design formalisms must be prepared in accordance with the guidelines and standards specified (that is, size, complexity, functionality of modules, prolog contents, and PDL usage).

The detailed design phase culminates in the CDR, attended by the development team and its managers, the requirements definition team and its managers, and others involved with the system. At the CDR, the development team presents the detailed design of each subsystem for critical review. This presentation is based on the detailed design documentation and may require a series of meetings if the system is large. See Section A.3 of Appendix A for details about the CDR.

For the CDR presentation, the participants evaluate the detailed design of the system to determine whether the design is correct and complete enough to begin implementation. They also review build/release capabilities and schedule for feasibility. The detailed design is complete when the development team has adjusted the detailed design to respond to comments and criticism expressed at the CDR.

3.3.2 END PRODUCTS

The detailed design document is the primary product. This document is an extension of the preliminary design report. See Section B.5 of Appendix B for the format and contents of the detailed design document.

3.3.3 METHODOLOGIES

Because the activities of detailed design are an extension of those performed during preliminary design, the same methodologies are used (see Section 3.2.3, page 3-21). They are repeated below:

- Project notebook
- Data collection
- Formal recording of design decisions and changes
- Configuration management procedures
- Design walkthroughs
- Iterative refinement
- Information hiding and data abstraction
- PDL

New activities for this phase are described below:

- Librarians. The librarians begin to transfer existing code to be used in the implementation into the project's online source code libraries. They continue their activities of preliminary design, including adding all materials produced during the detailed design phase to the project library. The organization into unit development folders according to subsystem (started during preliminary design) is continued and refined during detailed design.

- Unit development folders.. A chart is added to the unit development folder for each subsystem, showing each module in the subsystem and the planned and actual starting and ending dates for each of the major phases (that is, design, code, and test) for the module. The librarians update these charts to reflect current development status for each module throughout the remainder of the project life cycle.

3.3.4 TOOLS

The same tools recommended for use in the preceding phases are used (see Section 3.2.4, page 3-25):

- An online configuration management tool--for example, CAT
- An automated PDL processor, if one is available

A new tool for detailed design is

- An online source code library management system, which is to be used to manage the project libraries. Such a tool, if available, is an important part of the configuration management procedures because it can be used to enforce strict change control procedures on the project libraries containing PDL and source code that have been placed under configuration control.

3.3.5 MEASURES

The measures and evaluation criteria used during detailed design are similar to those used for preliminary design. Further explanation is given in the following subsections.

3.3.5.1 Objective Measures

As specified for preliminary design (see Section 3.2.5.1, page 3-26), managers monitor the following objective measures, repeated below:

- Number of requirements questions.
- Number of responses to requirements questions.
- Number of requirements changes.
- Number of design changes.
- Number of interfaces.
- Number of TBD requirements. The number of TBD requirements is the most important quantity to be examined. Ideally, all TBD requirements must be resolved by the end of this phase. If this goal is impossible to achieve, the managers must assess how the remaining TBD requirements will affect system size, required effort, cost, and schedule.
- A detailed checklist of all design formalisms. This list can be used to evaluate the design's completeness. Because the detailed design documentation contains all the

design formalisms produced for detailed design, all items on the checklist must be completed before the CDR.

One new measure can be used by the managers to monitor progress:

- An updated estimate of the number of lines of code in the system. By the end of detailed design, managers know the projected number of modules in the system. The budgeted effort rate can then be examined by computing the number of lines of code per (budgeted) effort unit and the number of modules per (budgeted) effort unit. Managers then can compare these figures with the same figures for similar past projects to determine whether or not enough effort has been budgeted to complete development.

3.3.5.2 Evaluation Criteria

To evaluate the correctness and completeness of the design and to determine whether the development team is ready to proceed with implementation, managers must consider the following questions:

- Have all items on the checklist of required design formalisms been completed? For example, are all external data sets completely defined and all baseline diagrams (treecharts) provided to the subroutine level?
- Is the design correct? Will the transformations specified produce the correct output from the input?
- Is the design robust? Is user input examined for potential errors before processing continues?
- Is the design testable?
- Have all design guidelines and standards specified been followed?

Detailed Design

- Are the descriptions of each component clear enough and sufficiently unambiguous so that implementers can proceed autonomously?
- Have all TBD requirements been resolved? If not, how will the remaining TBD requirements affect system size, required effort, cost, and schedule?
- Is the build/release schedule structured to provide early testing of end-to-end system capabilities? Is the schedule reasonable and feasible for implementing the design?
- Is the estimate of resources adequate for completing development?

Managers can evaluate the quality of the design by considering the following factors:

- The level of information hiding (that is, how well have data usage and access been localized. Are modules secretive in the way in which they perform their functions?)
- The degree of coupling between modules (that is, intramodule dependencies are minimized)
- The cohesiveness of the lowest level components (that is, each module has a single purpose)

3.3.6 KEY MANAGEMENT ACTIVITIES

During this phase, the manager's concerns are identical to those for preliminary design (see Section 3.2.6, page 3-28) and are repeated below. The activities include both planning and monitoring. Specifically, the managers

- Ensure adherence to
  - Design standards
  - Configuration management procedures, especially change control

  - Reporting procedures
  - Data collection procedures
  - Quality assurance procedures

- Review the design produced, participate in .esign walkthroughs, and resolve TBD requirements
- Monitor adherence to planned schedules and expenditure of resources, and update cost and resource estimates and schedules
- Ensure that all facets of the project are completely visible
- Ensure cooperation between the development team and the other groups with which thev must interact
- Plan transition to the implementation phase
- Schedule and participate in the CDR, and ensure that all pertinent groups participate

For the transition to the implementation phase, it is usually necessary to increase the size of the development team substantially to handle the simultaneous implementation of the builds for each subsystem. Managers must inform the development team of the software engineering approaches to be used during implementation and must provide required training. Also, the members of the development team must understand the code and testing standards, the quality assurance procedures, and the configuration management procedures to be followed in addition to their individual areas of responsibility.

Managers must also ensure that the online project libraries are established, that the strict change control procedures concerning these libraries are followed, and that the job control language (JCL) for building and testing the system is prepared for the developers so that they can start implementation immediately after the CDR.

3.4 IMPLEMENTATION

DEVELOPMENT	ACTIONS AND TRANSACTIONS
ACTIVITIES	Job Control Language Prepared Command Procedures Prepared New Modules Coded Reusable Modules Revised Units Integrated and Tested Build/Release Test Plans Prepared Data Prepared Build/Release Test Plans Executed Discrepancies Resolved System Integrated System Test Plan Prepared Acceptance Test Plan Prepared User's Guide Prepared System Description Prepared
END PRODUCTS	System Code Supporting Data and System Files Build/Release Test Plans and Results System Test Plan Acceptance Test Plan Draft User's Guide Draft System Description Software Development Plan Update
METHODOLOGIES	Project Notebook Data Collection Librarians Unit Development Folders Requirements Question and Change Records Design Decision and Change Records Coding Standards Structured Code Code Reading Code Change Records Configuration Management Builds/Releases Top-Down Implementation Formal Test Plans Functional (Thread) Testing
TOOLS	PDL Processor Source Code Library Management System Structured Coding Language CAT Resource Estimation

9103-(51)-83

PRECEDING PAGE BLANK NOT FILMED

3.4 IMPLEMENTATION

MANAGEMENT	ACTIONS AND TRANSACTIONS
ACTIVITIES	Build/Release Test Plans Reviewed Build/Release Test Plan Results Reviewed Discrepancies Resolved System Test Plan Reviewed Draft User's Guide Reviewed Draft System Description Reviewed System Testing Transition Planned TBD Items Resolved Requirements Changes Reviewed and Assessed Design Changes Reviewed and Walked Through Code Changes Reviewed Team Trained Standards and Procedures Enforced Progress Monitored Visibility Promoted System Size Estimated Resources and Cost Estimated Team Interaction Coordinated
MEASURES	TBD Items Requirements Changes Requirements Questions and Answers Design Changes Code Changes Code/Test Completion Checklists Code Growth Rate Error/Change Growth Rates Discrepancies/Resolutions Growth Rates Computer Usage Growth Rate Team/Individual Productivity Rates Subjective Evaluations

9103-(51)-83

The recommended software development guidelines for the implementation phase are described in detail below.

3.4.1 MAJOR ACTIVITIES

Implementation begins after the CDR unless comments and criticism expressed at the CDR indicate serious problems or deficiencies with the detailed design. In implementation, the development team

- Completes preparation of JCL and command procedures necessary to build and test the system
- Codes new modules from the detailed design specifications and revises old routines required to meet the requirements
- Integrates new modules into the growing system or subsystem
- Prepares data for performing unit/integration and release testing
- Performs unit/integration testing to ensure that newly added capabilities function properly
- Prepares test plans for each build/release
- Executes tests specified by the test plan for each build/release and reviews test results
- Prepares the system test plan for use during the system integration and testing phase
- Prepares drafts of the user's guide and system description documents, based on the material in the detailed design document

The system is implemented according to the staged implementation plan prepared by the developers during the detailed design phase. For each release, individual developers code and test the modules identified as belonging to a particular build of each subsystem. At the same time, members of the development team prepare the test plan for the release comprising the builds under development. This test plan is designed to test the functional capabilities of the release and is reviewed for correctness and completeness by development team members and their managers.

When the developers have completed all coding and unit testing for the release, they rebuild the system from source code and execute the tests specified in the release test plan. The development team and its managers carefully review test results to identify discrepancies.

For each release, the test plan evaluates the functional capabilities of the release as it is defined in the staged implementation plan. A sampling of tests from previous releases, called regression tests, is included in each test plan to ensure that the newly added capabilities have not affected the functioning of the previously implemented capabilities. During implementation of the last release, the development team prepares the system test plan in addition to the test plan for the last release. The system test plan is the basis for system testing performed during the next life cycle phase. It is designed to test the functional capabilities of the system as specified in the requirements documentation.

An independent acceptance test team prepares the acceptance test plan based on the information in the functional specifications and requirements document. The acceptance test team usually consists of analysts who will use the system. This team frequently includes members of the organization that prepared the functional specifications and requirements document.

3.4.2 END PRODUCTS

During implementation, the development team produces the following products:

- Completed code for the system
- Supporting files necessary for building and executing the system (for example, JCL, command procedures, and load modules)
- Test plans and results for each build/release
- System test plan
- Draft user's guide
- Draft system description

The test plans are generally produced as informal documents. Each one contains a set of tests designed to test the functional capabilities of a particular release or of the entire system. See Section B.6 of Appendix B for the format and contents of test plans.

The user's guide and the system description may be produced as two separate documents or combined into one. During this phase, this material is prepared in draft form. Most of the information needed is already available in the detailed design document. See Sections B.7 and B.8 of Appendix B for the format and contents of the user's guide and system description.

An independent acceptance test team produces the acceptance test plan.

3.4.3 METHODOLOGIES

The SEL recommends the following methodologies for implementation.

- Project notebook
- Data collection

- Librarians
- Unit development folders
- Formal recording of changes
- Configuration management procedures

Data collection and maintenance of the project notebook continue as is recommended in the preceding life cycle phases. New applications of the others are described in Sections 3.4.3.1 and 3.4.3.2 below.

In addition, the following new methodologies are also used:

- Coding standards
- Structured code
- Code reading
- Top-down implementation
- Builds/releases
- Functional (thread) testing
- Formal test plans

The remaining methodologies are described in more detail in Sections 3.4.3.3 through 3.4.3.6 below.

3.4.3.1 Librarians and Unit Development Folders

During implementation, the librarians support the development team by entering newly developed code, entering modifications for reusable code, and operating the software tools discussed in Section 3.4.4 below. The librarians also update the project's permanent source code libraries, incorporating changes made to the source code after it has been placed under configuration control. In this function, the librarians become an important part of the configuration management procedure.

The librarians maintain the central project library and keep it organized into unit development folders by subsystem. They add all materials produced during implementation to the project library: test plans and results for each build/release, and drafts of user's guide and system description information. They also add change reports for changes made to any parts of the system that are under configuration control (for example, the functional specifications and requirements document, the detailed design document, and the project's permanent source code libraries). The librarians also update the charts (started during detailed design) that show the exact status of each module in the system.

3.4.3.2 Formal Recording Mechanisms and Configuration Management Procedures

Configuration management procedures must be strictly adhered to during this phase. Source code for a module is placed under configuration control when the individual developer has coded, compiled, and tested the module successfully. At that point, the module is moved from the developer's jurisdiction into a permanent project source code library. Any further changes to the module must be approved by the development team leader before they are made by the librarians. These changes must be recorded on development change report forms.

In addition, the formal recording mechanisms used in the preceding life cycle phases for requirements questions and changes, and design decisions and changes, are used for requirements analysis and design activities that occur during implementation.

3.4.3.3 Structured Code and Coding Standards

The SEL recommends use of structured code (that is, using only the basic structured constructs) in implementing the design of the modules. These constructs correspond to those in the module's PDL. The principles of structured programming are described in Reference 16, for example.

The code must conform to the coding standards specified. Quality assurance procedures must be enforced by the managers to ensure that the developers adhere to those standards.

3.4.3.4 Code Reading

After a developer codes and successfully compiles a module, another member of the development team reads the code to verify that it performs the functions specified in the design and to check for common coding errors. The reader must review and return the code within half a day so that the developer is not delayed. Code reading identifies errors in the implementation of the design before testing begins. This review procedure is usually adequate. Occasionally, however, the development team may hold more formal walkthroughs for high-level or very complex modules, but this is unnecessary for most modules. Details on code walkthroughs are contained in Reference 17, for example.

3.4.3.5 Implementation Technologies

Implementation proceeds according to the builds and releases defined during detailed design in the staged implementation plan. A build is a portion of a subsystem that performs certain designated functions; a release is a portion of the system, composed of one or more builds, that has certain end-to-end functional capabilities. The modules in each subsystem build and the builds in each release are specified in the staged implementation plan.

Each subsystem build is implemented in a top-down fashion: i.e., if the baseline diagram is pictured as a map with North at the top, modules in the build are coded and tested in the order in which they appear in a northwest-to-southeast sweep of the baseline diagram (from the highest level to the lowest level and simultaneously from left to right). Developers test modules by integrating them into the growing subsystem and using the existing, previously tested subsystem as a test bed. Modules not yet implemented exist in the subsystem as stubs (that is, fully executable modules containing no executable instructions except to write a message that the module was entered and has returned control to the calling module).

Top-down implementation tests both the module's integration into the growing subsystem and its internal code. It also exercises the higher level and data input modules more fully and eliminates building test drivers that themselves require testing. Some modules may require unit testing in an isolated environment before they are integrated into the subsystem, but this should be necessary only in special cases (for example, to verify a particular algorithm).

3.4.3.6 Functional Testing and Formal Test Plans

After the builds of a particular release are completed and integrated into the system, the release's end-to-end processing capabilities (called "threads") are tested by the developers. An important part of this functional testing process is the formal test plan, which specifies the functional capabilities to be tested and the criteria for determining whether or not the test is successful. This is done for each test in the release test plan. The use of a formal test plan thus allows release testing to proceed in a logically organized manner and facilitates agreement among managers and developers as to when release testing is satisfactorily completed. The system test plan, prepared during the implementation phase, serves the same purpose during the system integration and testing phase that follows. Testing is described in Reference 18, for example.

3.4.4 TOOLS

The development team uses

- An online configuration management tool (for example, CAT). The tool is important in configuration management of the project's permanent source code libraries to track development changes. It is also used to maintain the detailed schedule for the development of each module in the system. In this phase it is very useful for maintaining information about discrepancies identified during testing. During testing of each release, discrepancies between how the system works and how it is supposed to work are identified. For large systems, the number of discrepancies that must be rectified can be substantial. Managers must keep track of these discrepancies, assign personnel to resolve them, set dates for resolution, and verify that the discrepancies have been resolved. A tool such as CAT makes this task easier.

- An online source code library management system.

- A structured FORTRAN preprocessor. This tool, which translates structured constructs into valid FORTRAN code, allows the programmer direct use of the standard structured constructs and thus facilitates structured programming. A structured preprocessor (SFORT) (Reference 19) is available in the SEL environment. Some versions of FORTRAN (for example, those conforming to the FORTRAN 77 language standards) already contain the structured constructs as part of the language and therefore do not require the use of a structured preprocessor to provide those capabilities.

3.4.5 MEASURES

The following subsections present various measures and evaluation criteria that may be used to assess the implementation phase.

3.4.5.1 Objective Measures

As in preceding life cycle phases, managers monitor the number of requirements questions, responses to requirements questions, requirements changes, and design changes. Managers also ensure that all TBD requirements are resolved by the beginning of the implementation phase. If this is not possible, managers must reassess how remaining TBD requirements will affect system size, required effort, cost, and schedule.

Managers must also monitor the following additional objective measures during the implementation phase:

- Productivity rates (number of lines of code, number of modules, and number of pages of documentation per effort unit). As implementation progresses, managers can obtain more accurate estimates of the number of lines of code and number of modules. Then they can update estimates of the budgeted productivity or effort rates (that is, number of lines of code per budgeted effort unit and number of modules per budgeted effort unit) to determine whether enough effort has been allocated to complete the development.

Managers can compute actual productivity rates to compare the pace of implementation with that experienced in past projects or with that budgeted. Productivity factors might include the number of lines of code in the projects' permanent source code libraries, the number of coded modules in the project libraries, or the number of completed pages of documentation per effort unit since the beginning of the implementation phase.

- Growth rate of the number of lines of code. The growth rate of the number of lines of code in the project libraries is another indication of the pace of the project.

- Error rate (number of errors per 1000 lines of code).

- Number of changes to code in the project's permanent source code libraries. Managers can use the error rate and the number of changes made to code after it has been placed under configuration control as indications of the code's reliability and stability. Excessively high figures for these measures (in comparison to past projects) might be caused by inadequate design specifications or insufficient testing by developers.

- Number of identified discrepancies versus number of resolved discrepancies. The number of discrepancies identified in release testing is also a measure of the system's reliability. A widening gap between the number of discrepancies identified and the number of discrepancies resolved as implementation progresses probably indicates problems requiring the manager's attention.

- Computer usage rate (number of minutes per 1000 lines of code). A computer usage rate much lower or much higher than previous projects may indicate problems in development, such as insufficient testing or excessive numbers of diagnostic test runs.

The SEL recommends the use of all these concrete measures. The SEL does not advocate the use of the more abstract measures of the development product (for example, the McCabe and Halstead measures) because a clear understanding of their meaning has not yet been obtained.

Managers must monitor the progress of the development throughout the staged implementation process. The detailed chart maintained by the librarians as part of the unit development folders, showing the exact status of each module in the system, contains the information necessary to assess how complete each build/release is. At all times throughout the implementation process, managers must know where the project is (that is, its exact status) and where the project is going (that is, the detailed schedule for completing the project).

3.4.5.2 Evaluation Criteria

To evaluate the quality and completeness of the products of implementation, managers must consider the following questions:

- For source code
  - Is the code complete?
  - Does the code adhere to the design?
  - Does the code adhere to the coding standards?
  - How reliable is the code? What is the confidence level of the system performing without failure?
  - Is the code maintainable? How easily can changes be introduced, tested, and verified?
  - How stable has the code been?
- For test plans and results
  - Are the test plans complete? Is all necessary information provided for each test? (See Section B.6 of Appendix B.)
  - Are the tests specified in the test plans repeatable? If two different groups execute the test plans, will the same tests be performed?
  - Do the test plans cover the key functional capabilities of the system?
  - Have the results of release tests been reviewed by developers and managers for discrepancies?

- For documentation
  - Does the documentation contain the key information?
  - Is the documentation as brief as possible?
  - Is the documentation clear and easy to understand? Can it be used by someone not familiar with the system? That is, is each document styled for its intended audience?

3.4.6 KEY MANAGEMENT ACTIVITIES

Several key management considerations during the implementation phase are identical to those in the preceding life cycle phases and are repeated below. The activities include both planning and monitoring. Specifically, managers

- Ensure adherence to
  - Reporting procedures.
  - Data collection procedures.
  - Quality assurance procedures.
  - Coding standards.
  - Configuration management procedures. These procedures--especially change control on the project's permanent source code libraries--must be enforced during the implementation phase when the staff is at its peak size and a large amount of code is being produced.

- Monitor adherence to the planned schedule, monitor expenditure of resources, and update cost and resource estimates and schedules. As implementation progresses, it becomes easier for managers to estimate the size of the system. Actual resources expended and progress during implementation can also be obtained to update cost and resource estimates with a resource estimation model (like the SEL Meta-Model). Updating cost and resource estimates, with resulting updates to schedules and staffing plans, is necessary several times in this phase as various builds and releases are completed.

- Ensure that all facets of the project are completely visible (that is, know exactly where the project is and where it is going at all times). Project visibility is critically important. Managers must know at all times the exact status of all task activities and the detailed plans for development completion. This is necessary so that problems can be dealt with when they occur rather than late in the process, when their impact is likely to be greater.

New management activities specific to implementation include the following:

- Review the release and system test plans and participate in the test result reviews for each build/release.
- Resolve discrepancies identified by the build/release testing.
- Plan the transition to the system testing phase. Managers must ensure that the data is available to perform the tests specified in the system test plan and that arrangements have been made to provide all computer resources required for system testing. They must inform development team personnel of the testing procedures to be followed and provide them with required training. Special emphasis is placed on enforcing the strict change control procedures for the project's online source code libraries during final release testing and system testing activities.

3.5 SYSTEM TESTING

DEVELOPMENT	ACTIONS AND TRANSACTIONS
ACTIVITIES	System Created System Test Plan Executed Discrepancies Resolved User's Guide Reviewed and Revised System Description Reviewed and Revised Acceptance Testing Planned
END PRODUCTS	System Code Update Supporting Data and System Files Update System Test Plan Results User's Guide Update System Description Update Software Development Plan Update
METHODOLOGIES	Project Notebook Data Collection Librarians Unit Development Folders Requirements Question and Change Records Design Decision and Change Records Code Change Records Configuration Management Formal Test Plan Functional (Thread) Testing
TOOLS	PDL Processor Source Code Library Management System Structured Coding Language CAT Resource Estimation

9103-(51)-83

PRECEDING PAGE BLANK NOT FILMED

3.5 SYSTEM TESTING

MANAGEMENT	ACTIONS AND TRANSACTIONS
ACTIVITIES	System Test Plan Results Reviewed Discrepancies Resolved Acceptance Test Plan Reviewed Acceptance Testing Transition Planned User's Guide Reviewed System Description Reviewed TBD Items Resolved Requirements Changes Reviewed and Assessed Design Changes Reviewed and Walked Through Code Changes Reviewed Team Trained Standards and Procedures Enforced Progress Monitored Visibility Promoted System Size Estimated Resources and Cost Estimated Team Interaction Coordinated
MEASURES	TBD Items Requirements Changes Requirements Questions and Answers Design Changes Code Changes Test Completion Checklist Code Growth Rate Error/Change Growth Rates Discrepancies/Resolutions Growth Rates Computer Usage Growth Rate Team/Individual Productivity Rates Subjective Evaluations

8103-(51)-83

The recommended software development guidelines for the system integration and testing phase are described in detail below.

3.5.1 MAJOR ACTIVITIES

System integration and testing begins at the end of the implementation phase. At this point, all code for the system is complete, and the release test plan for the last system release has been executed satisfactorily by the developers. In this phase, the developers validate the completely integrated system by functional testing of the end-to-end system capabilities according to the system test plan prepared during the preceding life cycle phase. Specifically, the development team

- Builds the system from the project's permanent source code libraries
- Performs the tests specified by the system test plan
- Reviews the test results
- Corrects code to fix any errors identified by the system tests
- Revises the drafts of the user's guide and system description, if necessary, so that the documentation reflects the final state of the system
- Prepares for the acceptance testing phase

System testing, which proceeds according to the system test plan, is performed like the testing of each release during the implementation phase. The development team and its managers, including customer and contractor personnel, carefully review the test results to identify any discrepancies between the way the system works and the way it is supposed to work. The developers then correct the errors in the code that are causing these discrepancies. The system testing phase is complete when all tests in the sys· test plan have been executed successfully. Since the system test plan must specify the expected output and the criteria for determining whether or not the test was successful (see Section B.6 of Appendix B), the conditions for system integration and testing completion are not ambiguous.

Toward the end of this phase, the development team must prepare for the beginning of acceptance testing. They must become familiar with the acceptance test plan and the acceptance test procedures. They must obtain the computer resources necessary for acceptance testing and modify the JCL, command procedures, and so on, to perform the acceptance tests. The development team must also begin to instruct the acceptance test team--by demonstrations and documentation--in the system's operation.

3.5.2 END PRODUCTS

At the end of the system testing phase, the completed system is available. The only new product of this phase is

- Test results from the system test plan

The remaining products are updated versions of products produced during implementation:

- Completed code for the system, including changes made to correct discrepancies identified by system testing
- Supporting files necessary for building and executing the system (for example, JCL, command procedures, and load modules)
- Updated drafts of the user's guide and system description, reflecting the state of the system at the completion of system testing

3.5.3 METHODOLOGIES

The methodologies used during system testing are a subset of those used during the implementation phase:

- Project notebook.
- Data collection.
- Librarians.
- Unit development folders.
- Formal recording of changes.
- Functional (thread) testing.
- Configuration management procedures. Strict adherence is essential. Because all code is under configuration control at this time, any changes to the code in the permanent source code libraries must be made according to the established procedures and must be recorded by means of development change forms. The configuration control procedures used must ensure that the load modules being tested correspond to the code in the project's libraries. Although requirements and design changes are not frequent this late in the life cycle, when they do occur, the same formal recording mechanisms for requirements questions and changes and design questions and changes must be used as is recommended in the preceding life cycle phases.
- Formal test plans. The system test plan is the basis for system testing. The tests specified are designed to verify the system's end-to-end functional processing capabilities or threads. The system test plan is written and carried out by the developers. The system test plan frequently contains a number of tests specified in the build/release test plans (see Section 3.4.3.6, page 3-49).

3.5.4 TOOLS

Managers continue to use the following tools:

- An online configuration management tool (for example, CAT)
- An online source code library management system, if available

3.5.5 MEASURES

The following subsections present various measures and evaluation criteria for assessing system testing.

3.5.5.1 Objective Measures

The objective measures that managers must monitor during system testing are the same as those of the implementation phase (see Section 3.4.5.1, page 3-51):

- Actual productivity rates for the completed system versus planned productivity rates (number of lines of code, number of modules, and number of pages of documentation per effort unit).

- Error rate (number of errors per 1000 lines of code).

- Number of changes to code in the project's permanent source code libraries.

- Number of identified discrepancies versus number of resolved discrepancies.

- Computer usage rate (number of minutes per 1000 lines of code).

- Actual size of completed system versus planned size (number of lines of code, number of modules, and number of pages of documentation). Comparing actual versus planned system size and productivity rates enables managers to evaluate the accuracy of the process they used to estimate system size, resources, cost, and schedules. This information adds to existing historical knowledge about the estimation process and can help to make this process more accurate for future projects. The actual computer usage rate for the completed system can also be useful in estimating required computer resources for future projects.

3.5.5.2 Evaluation Criteria

Because the products of this phase are basically updated versions of those produced during implementation, the subjective criteria for evaluating their quality and completeness are similar to those used in the preceding life cycle phase (see Section 3.4.5.2, page 3-53). Managers can consider the following questions:

- How reliable is the code? What is the confidence level of the system performing without failure?
- Is the code maintainable? How easily can changes be introduced, tested, and verified?
- How stable has the code been?
- Have the configuration management procedures for the system been strictly followed? Are the source code and load modules consistent?
- Have the results of the system tests been thoroughly reviewed by developers and managers for discrepancies?
- Do the test results meet the system requirements for each test in the system test plan? Does the system satisfy all requirements?
- Is the documentation complete and correct? Does it reflect the state of the completed system?

3.5.6 KEY MANAGEMENT ACTIVITIES

The manager's primary concerns during this phase are identical to those in the preceding life cycle phase and include both planning and monitoring. Specifically, managers

- Ensure adherence to
  - Reporting procedures
  - Data collection procedures
  - Quality assurance procedures
  - Guidelines/standards
  - Configuration management procedures, especially change control
- Monitor adherence to the planned schedule and expenditure of resources, and update cost and resource estimates and schedules
- Ensure that all facets of the project are completely visible
- Review test results for each test in the system test plan
- Review system documentation
- Resolve discrepancies identified by system testing
- Plan the transition to the acceptance testing phase. Managers must ensure that the data is available to perform the tests specified in the acceptance test plan and that arrangements have been made to provide all computer resources required for acceptance testing. Transition planning is especially important for the acceptance testing phase because the development team must work with two different groups (the acceptance test team and maintenance and operation personnel). Managers must ensure that the procedures to be followed during acceptance testing are well defined

and understood by the development team. Managers must also supervise the instruction of the acceptance test team and operators in the system's operation. Providing this instruction is the developers' responsibility. Special emphasis is placed on enforcing the strict change control procedures for the project's online source code library during system testing and acceptance testing activities.

3.6 ACCEPTANCE TESTING

DEVELOPMENT	ACTIONS AND TRANSACTIONS
ACTIVITIES	System Created Users and Operators Trained Acceptance Test Plan Executed Discrepancies Resolved User's Guide Reviewed and Revised System Description Reviewed and Revised System Delivery Planned ORR Held Software Development History Prepared
END PRODUCTS	System Code Supporting Data and System Files Acceptance Test Plan Results User's Guide System Description Archived System (Tapes) and Documentation Software Development History
METHODOLOGIES	Project Notebook Data Collection Librarians Unit Development Folders Requirements Question and Change Records Design Decision and Change Records Code Change Records Configuration Management Formal Test Plan Functional (Thread) Testing
TOOLS	PDL Processor Source Code Library Management System Structured Coding Language CAT Resource Estimation

9103-(51)-83

PRECEDING PAGE BLANK NOT FILMED

3.6 ACCEPTANCE TESTING

MANAGEMENT	ACTIONS AND TRANSACTIONS
ACTIVITIES	Acceptance Test Plan Results Reviewed Discrepancies Resolved System Delivery Reviewed User's Guide Reviewed System Description Reviewed TBD Items Resolved Requirements Changes Reviewed and Assessed Design Changes Reviewed and Walked Through Code Changes Reviewed Team Trained Standards and Procedures Enforced Progress Monitored Visibility Promoted System Size Estimated Resources and Cost Estimated Team Interaction Coordinated
MEASURES	TBD Items Requirements Changes Requirements Questions and Answers Design Changes Code Changes Test Completion Checklist Code Growth Rate Error/Change Growth Rates Discrepancies/Resolutions Growth Rates Computer Usage Growth Rate Team/Individual Productivity Rates Subjective Evaluations

9103-(51)-83

The recommended software development guidelines for the acceptance testing phase are described in detail below.

3.6.1 MAJOR ACTIVITIES

Acceptance testing begins at the end of the system testing phase, when all tests in the system test plan have been executed satisfactorily by the developers. Before acceptance testing begins, an acceptance test plan is prepared by the acceptance test team, based on the information in the functional specifications and requirements document. The system is then tested according to this plan.

During acceptance testing, an independent acceptance test team tests the system to validate that the software meets all requirements. The development team assists the acceptance test team. Specifically, the development team

- Builds the system from the project's permanent source code libraries
- Provides training for users and operators
- Sets up and executes tests as specified in the acceptance test plan at the direction of the acceptance test team
- Participates with the acceptance test team in the review of the test results to identify discrepancies
- Corrects the code to fix any errors identified by the acceptance tests
- Provides user assistance to the acceptance test team
- Completes the final versions of the user's guide and system description
- Delivers the final system to the customer

Four important activities are described in more detail below.

- Agree on test procedures. Before acceptance testing begins, the procedures for acceptance testing must be agreed on by the managers of both the development and the acceptance test teams and given to the team members. The procedures must specify whether all tests will be run before code is changed to resolve discrepancies. If modifications to the code are allowed as testing progresses, the effect of these modifications on the testing process must be addressed. (For example, will acceptance testing start over after each modification or will all tests be completed before they are rerun?) The procedures must also specify the respective responsibilities of the development and the acceptance testing team members and the lines of communication between these two teams, their managers, and the operations personnel with whom the teams must work to perform the acceptance testing.

- Understand the test plan. The acceptance test plan prepared by the acceptance test team is similar to the release and system test plans prepared by the development team (see Section B.6 of Appendix B). For each test to be performed, the acceptance test plan must specify the purpose of the test (that is, the specific functional capabilities or requirements being tested), detailed descriptions of the input and required environment, and the operational procedure to be used (also see Reference 20). The acceptance test team supplies test data for the acceptance testing and provides the development team with all required external data sets. The development team sets up and performs the tests according to the specifications in the acceptance test plan.

The acceptance test plan must also specify the expected output for each test to be performed and the criteria for determining whether or not the test results are acceptable. The development and acceptance test teams must be able to agree on which discrepancies identified by the testing must be corrected before the system is accepted. The acceptance testing phase is considered complete when all tests specified by the acceptance test plan have executed successfully. Because the acceptance test plan contains pass/fail criteria for each test, the conditions for acceptance testing completion are not ambiguous.

- <u>Deliver the system</u>. After the successful completion of acceptance testing, the developers formally deliver the accepted system to the customer. They clean up all files and they prepare and deliver a system delivery tape. They also deliver the final versions of the user's guide and system description (see Section 3.4.2, page 3-45). After the system has been formally delivered, it becomes the responsibility of a maintenance and operation group. The maintenance and operation phase of the software development life cycle is not addressed in this document.

- <u>Participate in the operational readiness review (ORR)</u>. After the successful completion of acceptance testing, an ORR is held to evaluate the readiness of the system to support operations. The ORR is attended by the development team managers, the acceptance test team managers, the maintenance and operation team managers, and others involved with the system. See Section A.4 of Appendix A for details about the ORR.

3.6.2 END PRODUCTS

At the end of the acceptance testing phase, the accepted system is delivered to the customer. Some products of this phase are updated versions of products previously begun:

- Completed code for the accepted system, including changes made to correct discrepancies identified by acceptance testing
- Supporting files necessary for building and executing the system (for example, JCL, command procedures, and load modules)
- Final version of the user's guide
- Final version of the system description

Three products are new:

- Test results from the acceptance test plan.
- System delivery tape.
- Software development history. Within 3 months after system delivery, the program manager, with input from the project manager, writes a software development history for the project. This report summarizes development and evaluates the technical and managerial aspects of the project from a software engineering point of view. The purpose of the report is to allow development managers to become familiar with successful and unsuccessful practices and to provide them with a basis for improving the development process and product. See Section B.9 of Appendix B for the format and contents of the software development history.

3.6.3 METHODOLOGIES

For acceptance testing, the SEL recommends the same methodologies as for the system testing phase (Section 3.5.3, page 3-61).

3.6.4 TOOLS

The tools recommended for use in the acceptance testing phase are the same as those recommended throughout the software life cycle: CAT and an online source code library management system.

3.6.5 MEASURES

The objective measures and evaluation criteria recommended by the SEL for use during the acceptance testing phase are the same as those recommended for use during the system testing phase (Sections 3.5.5.1 and 3.5.5.2, pages 3-62 and 3-63).

3.6.6 KEY MANAGEMENT ACTIVITIES

Monitoring is the key activity for managers during this phase. Several management activities are identical to those in the preceding life cycle phase. Specifically, the managers

- Ensure adherence to
  - Reporting procedures.
  - Data collection procedures.
  - Quality assurance procedures.
  - Guidelines/standards.
  - Configuration management procedures, especially change control. The managers must ensure that all changes are coordinated with acceptance testing activities and are made according to established acceptance testing procedures.

- Monitor adherence to the planned schedule and expenditure of resources, and update cost and resource estimates and schedules. At the end of this phase, development is complete, and the actual cost and resource expenditures throughout the project are known. Managers can compare these figures to the estimates produced during each life cycle phase to understand the estimation process better so that they can make more accurate predictions for future projects.

- Ensure that all facets of the project are completely visible.

These activities are specific to the acceptance testing phase:

- Participate in the review of acceptance test results
- Resolve discrepancies identified by acceptance testing
- Ensure cooperation among the various groups involved during this phase (for example, development team personnel, acceptance test team personnel, maintenance and operation support personnel, and librarians) and their adherence to established acceptance testing procedures
- Schedule and participate in the ORR, and ensure that all pertinent groups participate

4 MANAGEMENT AND CONTROL

Directing and controlling the execution of the development plan is the most important aspect of the development manager's job once a feasible plan is produced. By definition, requirements change. For some complex requirements, designing an implementation may take several tries. In some cases, the design may not be implementable because of a change in computer hardware configuration or limitations. Certainly the development project staff changes, and the personality of individuals may change. In short, the development process is very dynamic. Therefore, the successful execution of even the most complete plans involves

- Carefully measuring or assessing development progress and team performance (see Section 4.1)
- Recognizing the danger signals or warning signs of problems that will prevent proper execution of the plan (see Section 4.2)
- Taking appropriate steps to solve problems once they have been accurately identified so that the real problems are addressed--not their symptoms (see Section 4.3)

Data Collection

If the development manager expects to manage a development project successfully, he or she must measure it in some concrete way to assess general and specific progress. Measuring the performance of team members is also essential, since it will indicate team strengths and weaknesses, areas for training, and potential problem areas. To measure the process in a concrete manner, the managers must collect appropriate data, monitor its collection, and then use the data (1) for comparison with past projects, (2) in predictive models, and/or (3) with conventional techniques of progress assessment.

The SEL stresses the importance of collecting and archiving data throughout the software development process, not only for monitoring the development project, but also for understanding the environment and evaluating the effects of new technologies on productivity and reliability. Key data types to be collected throughout the project include

- Basic project statistics (estimated and actual numbers of modules and source lines of code and the start and end dates for each life cycle phase)
- Resource data (weekly expenditures of resources for technical staff, managers, secretaries, librarians, publications personnel, and computer usage)
- Change and growth data (the number of source lines of code and number of modules entered into the system by week and the number of changes made to the system by week)
- Activity data (weekly summary of hours spent in various project activities by each member of the technical staff)
- Change data (change report forms for all changes to the design and code after it has been placed under configuration control)
- Subjective evaluation data from managers after completion of project phases

The SEL's Guide to Data Collection (Reference 21) provides further recommendations on this topic.

4.1 INDICATORS OF DEVELOPMENT STATUS

- Frequency of Schedule/Milestone Changes
- Consistency in Organizational Structure Compared With Original Plans
- Fluctuation in Project Staff Level and System Size Estimates
- History of Number and Type of TBD Items for Requirements and Design
- Ease of Access to Information on Project Status, Schedules, and Plans
- Frequency and Amount of Unusually Long Hours Required or Planned To Attain Certain Objectives
- Level of Detail (Both Technical and Managerial) Understood and Controlled by the Project Manager and the Project Leader
- Discrepancies in Staff Level and Workload
- Discrepancies in Planned Weekly Staff Level and Computer Usage or Compared With Past Projects

9103-(51)-83

PRECEDING PAGE BLANK NOT FILMED

Status Indicators

The preceding page contains a brief list of measures that the SEL has found beneficial in monitoring the status of development projects. Continually lower frequencies of change, smaller fluctuations in estimates, and smaller discrepancies in planned-versus-actual reports (i.e., convergence toward goals) as development ;rogresses are strong indicators that the development team has the project under control. Opposite tendencies usually indicate the development of problems.

The development status indicators are described below.

Frequency of schedule/milestone changes. During the development life cycle, the development managers are able to make better estimates of system size and the effort required for development; therefore, they are able to make better estimates of completion dates. Although the estimates are expected to change periodically, the frequency and magnitude of the changes should continually decrease throughout the development life cycle. By monitoring this simple data point closely, especially once implementation starts, a manager may be able to identify problems early.

Consistency in organizational structure compared with original plans. The development managers usually organize their personnel before a project starts and make minor adjustments while preparing the development plan during requirements analysis and preliminary design. Substantial changes to those plans usually indicate problems. One common example is the unplanned appearance of a senior group of personnel--more senior than the development team--whose advertised task is quality assurance, a difficult design problem, or an independent assessment. A second example is the presence of a senior person who appears to be running the operation but who has no official role. A third example is diffusion of the project manager's or leader's responsibilities, i.e., delegation of nearly total responsibility for pieces of the system to other development team members.

<u>Fluctuation in project staff level and system size estimates</u>. As work progresses throughout the development life cycle, the development managers are able to make better estimates of system size, required effort, cost, and schedule. The uncertainty in these estimates will decrease and the confidence in them will increase after each phase of development (see Figure C-1 on page C-8 of Appendix C). Estimates that reach or exceed the normal limits of uncertainty indicate problems with plans, understanding of the project, or staff composition.

<u>History of number and type of TBD items for requirements and design</u>. A large number of TBD items or a small number of severe TBD items for requirements usually indicates a system definition problem. During design, large numbers of TBD items or severe TBD items indicate a lack of understanding by, or inexperience of, the development team.

<u>Ease of access to information on project status, schedules, and plans</u>. All development managers prepare a development plan and maintain a project notebook, which are kept up to date in a central repository. Yet it is not uncommon to solicit information from a project manager or leader and receive the response "I'll have to check." The longer it takes the team leaders to respond, the more suspicious higher level managers should be about the quality and usefulness of project plans and records.

<u>Frequency and amount of unusually long hours required or planned to attain certain objectives</u>. The list of reasons for overtime hours is long and includes getting the most up-to-date material together for formal reviews, meeting major milestones, recovering from late software/interface

deliveries or hardware failures, and covering for staffing problems. Sometimes the overtime hours are necessary and expected; however, overuse of this practice is frequently indicative of problems with the staff's qualifications, the staff level, or the team's leadership. (Inexperienced project managers/leaders, who are the most qualified to perform most development tasks, frequently do certain functions themselves because they think it is faster that way, rather than enlisting the team's help.)

Level of detail (both technical and managerial) understood and controlled by the project manager and project leader. All development managers prepare a development plan and maintain an up-to-date project notebook. The project manager's responsibilities are technical consultation and management of the development plan (technical, management, and configuration control approaches). The project leader's responsibilities are technical direction and day-to-day supervision of project activities. Yet it is not uncommon for the team leaders to be unable to respond to queries at status meetings. The more frequently the team leaders are unable to respond, the more suspicious higher level managers should be about the level of control that the team leaders have over the project.

Discrepancies in staff level and workload. All development organizations use some algorithm for determining staff levels based on the type and amount of each type of work to be done. The workload and staff levels, which are recorded in the development plan, change throughout the development life cycle. Discrepancies between these and the plan indicate problems.

Discrepancies in planned weekly staff level and computer usage or in comparison with past projects. A decrease in the weekly staff level may indicate a temporary or permanent loss of personnel from the project to another project; an increase may indicate an attempt to remedy a deficiency. A decrease or slow start in using the computer may indicate that the development team is engaged in some other activity (for example, design) rather than testing.

4.2 DANGER SIGNALS

- Scheduled Capabilities Delayed to Later Build/Release
- Coding Started Too Early (Staff Too Large Too Early)
- Numerous Changes Made to Initial Software Development Plan
- Guidelines or Planned Procedures Deemphasized or Deleted
- Sudden Changes in Staffing (Magnitude) Suggested or Made
- Excessive Documentation and Paperwork That Have Little Direct Bearing on Required Documentation Prepared
- Continual Increase in Numbers of TBD Items and ECRs Measured
- Decrease in Estimated Effort for System Testing Suggested or Made
- Reliance on Other Sources for Soon-To-Be-Available Software

8108-(51)-83

PRECEDING PAGE BLANK NOT FILMED

Danger Signals

The SEL has monitored many software development projects, some of which were considered very successful and some of which were considered less than successful. From this experience, the SEL has compiled a brief list of indicators (preceding page) that often characterize serious problems within the project.

These indicators are described below.

Scheduled capabilities delayed to a later build/release. Assuming that a build/release approach to implementation is being followed, the single most important signal of serious problems during implementation is rescheduling capabilities from one build/release to a later one. Although it is sometimes necessary to reschedule capabilities, the consistent rescheduling of capabilities from one build/release to another as a completion date nears often indicates serious problems.

Coding started too early. It is not uncommon for development projects to be fully staffed too early or overstaffed, although it seems as though most projects are understaffed or staffed too late. Starting coding too early, i.e., before a design from which coding can start has been approved, is a signal that the development team will end up building on the structure of a system that is not best suited to satisfy the total requirements.

Numerous changes made to the initial software development plan. When the development managers make numerous changes to the initial development plan, it is a signal that they are inexperienced and are reacting to internal problems rather than solving them.

Guildelines or planned procedures deemphasized or deleted. When the development managers suggest that the deletion or deemphasis of a method or procedure will save time and help to make a deadline, it is a nearly certain signal that the deadline will be made in a questionable manner, if at all, and that makeup work will be needed later to complete the activity correctly.

Sudden changes in staffing (magnitude) suggested or made. A sure signal of serious problems is the sudden suggestion or application of unplanned staff increases.

Excessive documentation and paperwork with little direct bearing on required documentation prepared. When development managers suggest the complete, detailed, formal documentation of each activity, it is a signal that they are inexperienced and that cost and schedule problems are imminent. Complete, detailed, formal documentation does not ensure success when the team leaders' effort is diverted from managing the technical aspects of the project.

Continual increase in numbers of TBD items and ECRs measured. A continual increase in the number of TBD items is a clear signal that technical problems are not being resolved. A continual increase in the number of TBD requirements and engineering change requests (ECRs) is a signal that the requirements are not adequately defined or stated.

Decrease in estimated effort for system testing suggested or made. When the development managers suggest or make a decrease in the estimated effort for system testing, it is a signal that they are inexperienced or they are excessively scheduling to success. With acceptance testing, system testing is the most sequential phase in the development process. Little can be done to compress the full testing phases. Assuming that testing can be compressed to make up for slippages in earlier phases leads to a loss of credibility in the development organization when the system is not ready on time or is flawed in operation.

Reliance on other sources for soon-to-be-available software. Every experienced developer and manager recognizes the cost benefits of using existing software or soon-to-be-existing software. However, all management levels of the development team must be especially concerned when the successful execution of their development plans depends on other sources for their system's software capability. Managers' concern should increase inversely proportionately to the level of control they have over the source who is developing the capability. For example, a loosely related project in the same organization is greater cause for concern than a closely related project in the same development organization; another contractor or a vendor is an even greater concern. The obvious problems with externally developed software are (1) it is always late and (2) it is never fully checked out.

4.3 CORRECTIVE MEASURES

- Stop Current Activities and Review/Complete Predecessor or Problem Activity
- Decrease Staff to Manageable Level
- Replace Junior With Senior Personnel
- Increase and Tighten Management Procedures
- Increase Number of Intermediate Deliverables
- Decrease Scope of Work and Define a Manageable, Doable Thread of the System
- Audit Project with Independent Personnel and Act on Their Findings

8106-(51)-83

Once the development manager has recognized that there is a problem, he or she must correct the problem. The preceding page contains a brief list of corrective measures that the SEL has found effective. Depending on the problem, one or more of these corrective measures may be necessary.

Frequently, when development managers find their projects in difficulty, they have a tendency to

- Shortcut procedures, i.e., to cut out the presumed "busy" or nonessential work and to concentrate on the "real" work
- Add staff (usually junior level) to help bail out
- Plunge ahead to meet milestones with some kind of product

The SEL's experience, however, shows that, more often than not, these steps compound problems. The corrective measures (preceding page) suggested by the SEL are usually counter to the normal tendency; that is, they increase and tighten procedures, reduce staff levels (or replace junior with senior staff), and slow down the process to get a better handle on it or to better define the objective. The following subsections list the basic development problem areas and the suggested steps to correct them.

4.3.1 BASIC PROBLEM AREAS

The following is a list of the basic problem areas:

1. Development plan problems
2. Requirements or design problems
3. Confusion with
    a. Development plan
    b. Requirements or design
    c. Development plan execution

4. System growth problems because of

   a. Poor direction
   b. Staff ability
   c. Major requirements changes
   d. Many minor requirements changes
   e. Incomplete facets of project

5. Changes or decrease in scope of plans

6. Configuration problems

7. Schedule problems

4.3.2 STEPS FOR CORRECTIVE ACTION

Following the outline above (Section 4.3.1), this subsection presents the steps to follow to correct problems in each of the basic problem areas.

1. <u>When there are serious problems with the development plan</u>,

   a. Stop development activity.
   b. Complete and/or review plans.
   c. Follow through with plans.

2. <u>When there are serious problems with requirements or design</u>,

   a. Stop staff growth.

   b. Decide which are appropriate:

      (1) Decrease the scope of the system.
      (2) Solve problems before proceeding.
      (3) Replace junior personnel with senior personnel.

3. <u>When there is confusion</u>,

   a. Obtain an accurate assessment of the cause.

3. When there is confusion (continued),

   b. <u>When the development plan is the cause of confusion,</u>

      (1) Stop development activity.
      (2) Complete and/or review plans.
      (3) Follow through with plans.

   c. <u>When requirements or design is the cause of confusion,</u>

      (1) Stop staff growth.

      (2) Decide which are appropriate:

         (a) Decrease the scope of the system.

         (b) Solve problems before proceeding.

         (c) Replace junior personnel with senior personnel.

   d. <u>When plan execution is the cause of confusion,</u>

      (1) Decide which are appropriate:

         (a) Decrease staff size to a manageable level.

         (b) Replace junior team leaders with senior leaders.

         (c) Replace junior team members with senior people.

      (2) Create intermediate products and milestones for review.

      (3) Increase status reviews to improve direction.

      (4) Follow through with plans.

4. <u>When there is inadequate system growth (progress),</u>

   a. Obtain an accurate independent assessment (audit) of the problem.

4. When there is inadequate system growth (continued),

   b. <u>When poor direction inhibits system growth</u>,

      (1) Decide which is appropriate:

         (a) Decrease staff size to a manageable level.

         (b) Replace junior team leaders with senior leaders.

      (2) Create intermediate products and milestones for review.

      (3) Increase status reviews to improve direction and to tighten management procedures.

      (4) Follow through with plans.

   c. <u>When staff ability inhibits system growth</u>,

      (1) During design, replace junior team members with senior personnel to complete design.

      (2) During implementation, add intermediate- to senior-level personnel to complete implementation.

      (3) During testing, add senior personnel to solve problems and to improve direction.

   d. <u>When major requirements changes inhibit system growth</u>,

      (1) Stop staff growth.

      (2) Decide which are appropriate:

         (a) Decrease the scope of the system.

         (b) Solve problems before proceeding.

         (c) Replace junior personnel with senior personnel.

4. When there is inadequate system growth (continued),

    e. <u>When many minc. requirements changes inhibit system growth</u>,

        (1) During design and early implementation,

            (a) Stop staff growth.

            (b) Decide which are appropriate:

                i. Decrease the scope of the system.

                ii. Solve problems before proceeding.

                iii. Replace junior personnel with senior personnel.

        (2) During implementation, hold changes and complete implementation of a build of the system first.

        (3) During testing, hold changes and complete testing of a version of the system first.

    f. <u>When an incomplete facet inhibits system growth</u>,

        (1) Decide which is appropriate:

            (a) Redirect senior personnel from less important or low-priority work to complete design or implementation.

            (b) Add senior personnel to complete design or implementation.

        (2) During testing, add senior personnel to solve problems.

5. <u>When there is a significant change or decrease in the scope of the development plan</u>,

    a. Obtain an accurate assessment of motivation.

5. When there is a significant change or decrease in the scope of the development plan (continued),

   b. When confusion causes change of plan,

      (1) When the development plan is the cause of confusion,

         (a) Stop development activity.
         (b) Complete and/or review plans.
         (c) Follow through with plans.

      (2) When requirements or design is the cause of confusion,

         (a) Stop staff growth.

         (b) Decide which are appropriate:

            i. Decrease the scope of the system.

            ii. Solve problems before proceeding.

            iii. Replace junior personnel with senior personnel.

      (3) When development plan execution is the cause of confusion

         (a) Decide which are appropriate:

            i. Decrease staff size to a manageable level.

            ii. Replace junior team leaders with senior leaders.

   c. When inadequate system growth (progress) causes change of plan,

      (1) Obtain an accurate independent assessment (audit) of the problem.

      (2) When poor direction inhibits system growth, follow steps (1) through (4) in item 4b.

c. When inadequate system growth (progress) causes change of plan (continued),

(3) When staff ability inhibits system growth, follow steps indicated in item 4c.

(4) When major requirements changes inhibit system growth, follow steps (1) and (2) in item 4d.

(5) When many minor requirements changes inhibit system growth, follow steps indicated in item 4e.

(6) When an incomplete facet inhibits system growth, follow steps indicated in item 4f.

6. When there are problems with configuration control,

a. Obtain an accurate assessment of weak areas.

b. Firm up and tighten configuration management procedures.

c. Follow through with plans.

7. When there are problems in maintaining schedules,

a. Obtain an accurate independent assessment (audit) of cause.

b. When confusion causes schedule slippage,

(1) When the development plan is the cause of confusion,

(a) Stop development activity.
(b) Complete and/or review plans.
(c) Follow through with plans.

b. When confusion causes schedule slippage (continued),

(2) When requirements or design is the cause of confusion,

(a) Stop staff growth.

(b) Decide which are appropriate:

i. Decrease the scope of the system.

ii. Solve problems before proceeding.

iii. Replace junior personnel with senior personnel.

(3) When development plan execution is the cause of confusion, decide which are appropriate:

(a) Decrease staff size to a manageable level.

(b) Replace junior team leaders with senior leaders.

c. When inadequate system growth (progress) causes schedule slippage,

(1) Obtain an accurate independent assessment (audit) of the problem.

(2) When poor direction inhibits system growth, follow steps (1) through (4) in item 4b.

(3) When staff ability inhibits system growth, follow steps indicated in item 4c.

(4) When major requirements changes inhibit system growth, follow steps (1) and (2) in item 4d.

(5) When many minor requirements changes inhibit system growth, follow steps indicated in item 4e.

(6) When an incomplete facet inhibits system growth, follow steps indicated in item 4f.

5 ASPECTS OF SUCCESSFUL PROJECTS

ASPECTS OF SUCCESSFUL PROJECTS

The preceding sections of this document present the software development and management practices, techniques, and aids that the SEL has found beneficial. This section identifies key aspects of successful software development projects and discusses the application of the recommended approach.

The following subsections contain three lists identifying key aspects of software development:

- Ten key "Do's" for project success
- Ten key "Don'ts" for project success
- Ten key points for assessing the quality of a project

These lists are derived from SEL experience using the recommended approach.

Section 5.1 lists and describes the 10 most important guidelines for managing a successful development project; Section 5.2, the 10 most important things to avoid in managing a development project. Section 5.3 highlights the 10 key points most useful in evaluating or assessing (auditing) a software development project. No particular order of priority is implied in any of these lists.

Section 5.4 discusses the application of the recommended approach to software development.

5.1 TEN "DOs" FOR PROJECT SUCCESS

- Use a Small Senior Staff for the Early Life Cycle Phases
- Develop and Adhere to a Software Development Plan
- Define Specific Intermediate and End Products
- Examine Alternative Approaches
- Use Formal Testing
- Use a Central Repository
- Keep a Detailed List of TBD Items
- Update System Size, Required Effort, Cost, and Schedule Estimates
- Allocate 30 Percent of Effort for Integration and Testing
- Experiment

8103-(51)-83

10 DOs

The SEL's 10 most important "DOs" for project success are described below.

Use a small senior staff for the early life cycle phases. A small group of experienced senior personnel is better equipped to determine the approach, to prepare the software development plan, to set priorities and organize the work, and to establish reasonable schedules. With a large team, there is a tendency to begin design or coding to keep people busy before the actual problem is known.

Develop and adhere to a software development plan. This plan defines project organization and responsibilities; life cycle phases, approaches, intermediate and end products; approach guidelines and standards; product completion and acceptance criteria; configuration management procedures; mechanisms for accounting status; product and progress reviews; cost and schedule reviews; and contingency plans. All development team members must know the plan and adhere to it.

Define specific intermediate and end products. Specific intermediate and end products for each life cycle phase give the devel̈opment team well-focused short-term goals, provide the team with a sense of accomplishment, and provide a means to measure and evaluate progress.

Examine alternative approaches. Alternative approaches, and the rationale for them, must be considered and evaluated in terms of project objectives and constraints, such as schedule, cost, team skill mix, availability of resources, and existing software. This is especially important during design. Do not assume that there is only one way of performing the task; seriously examine at least one other approach to the design.

Use formal testing. Because all testing (unit, system integration, acceptance) makes up 40 to 60 percent of a completed project's effort, cost, and schedule, it must be a well-organized and efficient process. Avoid a haphazard approach to testing; develop a test plan and follow it.

Use a central repository. Keep all development records and materials available in a central location so that the development process and progress are visible to management. Keep the repository organized and up to date throughout the project.

Keep a detailed list of TBD items. Classify TBD items by severity of impact in terms of system size, required effort, cost, and schedule and set priorities for their resolution. Assign appropriate personnel to resolve TBD items and follow their progress closely to ensure timely resolution.

Update system size, required effort, cost, and schedule estimates. Do not insist on maintaining original estimates of the system size, required effort, cost, and schedule. Requirements do change, the composition of the development team changes, and problems are encountered throughout the project. Most important, more information is learned about the size and complexity of the problem as the project progresses. Each phase of the life cycle provides new and refined information to improve the estimates and to plan the project more effectively.

Allocate 30 percent of effort for integration and testing. The activities in the system integration and testing and the acceptance testing phases are the most sequential in the development process. These phases account for 20 to 40 percent of a completed project's total effort, cost, and schedule; and little can be done to compress or reduce the work required in these phases. The code must be complete and unit tested before entering the system integration and testing phase. Avoid the common error of assuming that the integration and testing effort can be compressed to make up for slippages in the schedule during design and implementation.

Experiment. In an age of increasingly scarce resources, review effectiveness, identify areas for improvement, and take steps to make the improvements. Acquire new skills, examine alternative approaches, and test and evaluate changes. Try new techniques. Using the same methods this year that were used 2 or 3 years ago indicates a lack of growth.

5.2 TEN "DON'Ts" FOR PROJECT SUCCESS

- Don't Overstaff
- Don't Allow Team Members To Proceed in an Undisciplined Manner
- Don't Delegate Technical Details to Team Members
- Don't Assume That a Rigid Set of Standards Ensures Success
- Don't Assume That a Large Amount of Documentation Ensures Success
- Don't Deviate from the Approved Design
- Don't Assume That Relaxing Standards Reduces Costs
- Don't Assume That the Pace Will Increase Later in the Project
- Don't Assume That Intermediate Schedule Slippage Can Be Absorbed in a Later Phase
- Don't Assume That Everything Will Fit Together Smoothly at the End

0103-(51)-83

10 Don'ts

The SEL's 10 most important "Don'ts" for project success are described below.

Don't overstaff (especially dangerous early in development). A small group of senior personnel is better equipped than a large staff to organize and determine the direction of a project. When a large staff is assigned at the beginning of the project, the staff members usually begin designing some aspect of the system before the actual problem is known. After a significant amount of the budget is spent, managers frequently are reluctant to admit that a mistake was made and that the work performed is unusable. Because of this unwillingness to discard the work and start over, the remainder of the project will be based on an invalid design that causes further problems throughout the project.

Don't allow team members to proceed in an undisciplined manner. Developing very reliable high-quality software at low cost is not a creative art. Rather, it is a very disciplined application of a set of refined principles, methods, practices, and techniques. Apply them.

Don't delegate technical details to team members. First-line managers must know the technical details of the project. Do not delegate this aspect of the project to the members of the development team, especially to those on a junior level.

Don't assume that a rigid set of standards ensures success. Success is not guaranteed by any development methodology, practice, or technique. These standards promote discipline and consistency in the process and facilitate design walk-throughs, code reading, and test evaluation. However, the experienced judgments and decisions of the project manager, the development team leader, and other senior technical personnel are necessary to ensure success of the project.

Don't assume that a large amount of documentation ensures success. Each phase of the life cycle does not necessarily require a formally produced document to provide a clear starting point for the next phase. The level of formality and amount of detail to be provided in the documentation must be determined by the project size, the life cycle duration, and the lifetime of the system. For example, intermediate-sized projects (4 to 12 staff-years of effort) of 18 months' duration or less do not require a formally produced preliminary design document. By the time the material is prepared (edited, typed, reviewed, and so on), the design document is obsolete.

Don't deviate from the approved design. As development progresses, developers may tend to implement a slightly different design that still satisfies the requirements. The managers must control this tendency by holding design walkthroughs. Modifications by individual developers may be correct in the local sense but not for the system as a whole.

Don't assume that relaxing standards reduces cost. When a failure to meet a deadline seems imminent, managers and developers sometimes attempt shortcuts by relaxing configuration control procedures, data collection procedures, design formalism, or coding standards. In the long run, panic actions cause greater problems and added expense and do not usually succeed in making the deadline anyway.

Don't assume that the pace will increase later in the project. When design, implementation, or testing is progressing slowly, assign additional senior personnel to help and/or make schedule adjustments. The workrate for a given activity is characteristic of the particular development team--it generally does not change within a short period of time. Do not assume that the team will work faster later on.

Don't assume that intermediate schedule slippage can be absorbed in a later phase. When some part of the design must be completed during implementation, or when some part of the implementation must be completed during system testing, the later phase will not be completed on time unless extra staff is added well before its scheduled completion. It is a common mistake of managers and overly optimistic developers to assume that the team will be more productive later on. The workrate of the team cannot be changed appreciably because the project is approaching completion of a phase, especially in the later phases of development, when the process is most sequential. Because little can be done to compress the schedule during the later life cycle phases, the managers must change the schedule or apply additional staff as soon as the problem is known.

Don't assume that everything will fit together smoothly at the end. Managers erroneously assume that late pieces of design, code, or testing will contain few or no errors and will fit into the system with minimal integration effort. The work of the developers will not be of higher quality later in the project than it was earlier.

5.3 ASSESSING PROJECT QUALITY

- Is a Written Software Development Plan Being Followed?
- Are Life Cycle Phases and Products Defined?
- Is Someone in Charge?
- Does the Staff Size Match the Workload?
- Do Team Members Know Where the Project Is and Where It Is Going?
- Is a Configuration Control Plan Being Followed?
- Is There a Single Complete List of TBD Items With Assessments?
- Is There a Commonly-Adhered-To Methodology?
- Have Alternative Designs and Approaches Been Considered?
- Are There Contingency Plans for Rationally Solving Problems?

9100-(50a)-03

Project Quality

The SEL's experience in assessing the quality of an active project results from close review of monitored SEL projects and from conducting audits or independent evaluations of other projects. Ten key items for assessing the quality of a project are explained below.

Software development plan. Is there a written software development plan? Do all team members know it, and are they adhering to it?

Life cycle phases and products. Have the project managers and team leaders defined a life cycle with specific intermediate and end products? Are there centrally located lists detailing what these phases and products are?

Managers and leaders. Is there someone in charge? Does the development team leader know 90 percent of the technical details; the status of all major pieces of the software; the status of critical, major, and nominal problem areas; and future needs and potential problems? Does the project manager know the status of all major pieces of the software; two alternatives being considered to solve critical and major problems; one alternative being considered to solve nominal problems; the impact of critical and major TBD items; and the likelihood of cost and schedule perturbations in terms of workrate and workload?

Staff size. Are the correct number of people working on the project based on the projected workload and the development team's workrate? Are staffing changes planned to match projected increases or decreases in workload? Do the project workload and staff workrate projections match the schedule projections?

Project objectives and status. Do development team members know how their individual work fits in the project? Do they know their own deadlines and the objectives of those deadlines? Do they know when to expect data or interfaces to be established? Do the senior members of the team know the overall objectives of the project and how it fits in with the work being done by other groups? Do they know the probability of timely interfaces and the impact and contingency plans if they are not established?

Configuration control. Is there a written configuration control plan? Do all development team members know and follow it?

List of TBD items. Is there a single complete list of TBD items? Are they classified by severity of impact in terms of system size, required effort, cost, and schedule? Who sets the priorities for the resolution of TBD items, and how is their resolution scheduled and tracked?

Adherence to methodology. Do the development team members follow a specific methodology? Do all team members have the same understanding of what the methodology is?

Alternative approaches. Has the development team seriously considered at least one other design? Have they written down and justified the rationale for selecting the current design over alternatives?

Contingency plans. Are there written contingency plans on how to continue project work if, for example, a severe or major TPD item breaks a development sequence, an interface is several weeks or months late, the computer is down or malfunctioning for an extended period of time, the configuration of the software system is lost, or a key team member leaves the team prematurely? Are the manager and team leader aware of a potential problem and do they have a plan to minimize its effect?

5.4 APPLYING THE RECOMMENDED APPROACH

- Review the Recommended Approach
- Apply the Recommended Approach as Set Forth Unless There Is an Obvious and Founded Reason for Modifying It
- Make Additions to the Approach To Include Established Principles, Methods, Practices, and Techniques That Have Proved Beneficial in the User's Environment
- Standardize the Resulting Approach
- Refine This Standard Approach Based on Results of Completed Projects and Knowledge of New and Refined Technological Experiments
- Judiciously Adjust Selected Facets of This Standard Approach by Means of the Software Development Plan, Depending on the Particular Characteristics of the Individual Project, Such as Type and Size

9108-(51)-83

Recommended Approach

The SEL does not expect users to transfer the SEL's recommended approach to software development and apply it to their own environment without modification. The recommended approach is standard in the SEL environment for one class of software--flight dynamics systems. Even in the SEL environment, however, the recommended approach is tuned to the characteristics of individual projects and modified to reflect successful new technologies.

A SOFTWARE REVIEWS

SOFTWARE REVIEWS

This appendix summarizes suggested formats and contents of reviews scheduled during the software life cycle. All the reviews are part of the recommended approach to software development. For each review--SRR, PDR, CDR, and ORR--the following areas are addressed:

- Review presentation material
- Review format
- Review hardcopy material

where the review presentation material is a subset of the review hardcopy material. Following each hardcopy material summary is a brief description of the contents.

A.1 SYSTEM REQUIREMENTS REVIEW (SRR)

SRR PRESENTATION MATERIAL

- Introduction and Agenda
- Requirements Summary
- Analysis Overview
- Functional Specifications
  - Environmental Considerations
  - Operational Requirements
    - (1) Operating Scenarios
    - (2) Data Flow Analysis
    - (3) Performance Requirements
    - (4) Interface Requirements
  - Requirements Relationships
- Derived System Requirements
- Requirements Management Plan
- Personnel Organization and Interfaces
- Software Performance and Testing Requirements
- Issues, TBD Items, and Problems
- Milestones and Suggested Development Schedule

SRR FORMAT

PRESENTERS	Requirements Definition Team
PARTICIPANTS	Development Team Representatives Quality Assurance Representatives User Representatives Customer Representatives
TIME	After Functional Specifications Completed and Before Functional Design Started
HARDCOPY DISTRIBUTION	Minimum of 5 Days Before SRR

A.1 SYSTEM REQUIREMENTS REVIEW (SRR)

SRR HARDCOPY MATERIAL

1. Introduction
2. Requirements Summary
3. Analysis Overview
4. Functional Specifications
   a. Environmental Considerations
   b. Operational Requirements
      (1) Operating Scenarios
      (2) Data Flow Analysis
         (a) System Input
         (b) Processing Requirements
         (c) System Output
      (3) Performance Requirements
      (4) Interface Requirements
   c. Requirements Relationships
5. Derived System Requirements
6. Utility, Support, and Test Programs
7. Reusable Software Summary
8. Data Set Definitions
9. Requirements Management Plan
   a. Personnel Assignments
   b. Description of Required Documents
   c. Configuration Control Approach
   d. Enhancement/Maintenance Procedures
   e. Reporting and Testing Evaluation Procedures
10. Personnel Organization and Interfaces
11. Software Performance and Testing Requirements
   a. Analytical
   b. System
   c. Interface
   d. Acceptance
12. Issues, TBD Items, and Problems
13. Milestones and Suggested Development Schedule

9103-(51a)-83

The SRR hardcopy material contains

1. Introduction--Purpose of system, background of project, and outline of review material

2. Requirements summary--Review of top-level (basic) requirements that are developed to form the functional specifications

   a. Background of requirements--Overview of project characteristics, major events, support

   b. Derivation of requirements--Diagram showing

      (1) Project Office input used to formulate the requirements for the support organization--support instrumentation requirements document (SIRD), memorandums of information (MOIs), memorandums of understanding (MOUs), etc.

      (2) Support organization input used to formulate requirements for the system engineering organization, e.g., SIRD, MOIs, MOUs, and support organization's constraints, assumptions, and guidelines

      (3) System engineering organization input used to formulate the requirements (functional specifications and requirements document (FSRD)) for the software engineering organization--Analytical studies, software system analysis, etc.

   c. Type of requirements

      (1) Evolution of support requirements

         (a) Typical support
         (b) Critical support
         (c, Special support
         (d) Contingency support

(2) Operational support scenarios

(3) Relationship of requirements matrix--Relationship of top-level requirements to operational support scenarios

d. Constraints, assumptions, and guidelines

(1) Organizational interfaces--Organizations that provide system and support input and receive system output

(2) Data availability for the operational support scenarios--Frequency, volume, format

(3) Facilities--Target computing hardware, environment characteristics, communications protocols, etc.

(4) General software considerations--High-level description of computer storage, graphics, and failure/recovery requirements; operator interaction requirements; system error recovery and diagnostic output requirements; etc.

(5) Support and test software considerations--High-level description of requirements for data simulators, test programs, support utilities

e. Overview of functional specifications and requirements document (FSRD)

(1) History of evolution--Draft dates and reviews

(2) Outline of contents

3. Analysis overview--Mathematical and logical framework necessary for design, implementation, and testing of the system

a. Introduction--Project overview, support firsts, bases for analysis

b. Analysis approach--Major areas of analysis necessary to produce FSRD

c. Special studies and results--Overview of purpose of studies and conclusions

4. Functional specifications

a. Environmental considerations--Target computing hardware, special computing capabilities (e.g., graphics), operating system limitations, computer facility operating procedures and policies, support software limitations, data base constraints, resource limitations, etc.

b. Operational requirements

(1) Operational scenarios--High-level diagrams of operational support goals and concepts, including all interfaces

(2) Data flow analysis--Diagrams showing input, processes, and output, including all interfaces

(a) System input--Data availability, frequency, volume, coordinates, units, formats

(b) Processing requirements--What functions must be performed to transform some input into some output

(c) System output--Data frequency, volume, coordinates, units, formats

(3) Performance requirements--System processing speed, system response time, system failure recovery time, output data availability

(4) Interface requirements--Summary of human, special-purpose hardware, and automated system

interfaces, including references to interface agreement documents (IADs) and interface control documents (ICDs)

c. Relationship of requirements matrix, e.g., relationship of requirements to operational scenarios

5. Derived system requirements--Structured, enumerated list of the requirements derived in formulating the functional specifications

6. Utility, support, and test programs--Rationale for partitioning system into smaller programs to support operational scenarios; to support data processing volume, frequency, or special conditions; and to make use of reliable existing software; also, rationale for data simulators and test programs

7. Reusable software summary--Identification of existing software components that satisfy specific system functional specifications exactly or that will satisfy them after specified modifications

8. Data set definitions for

   a. Interfaces external to the system, including

      (1) Format and description of items in header, data, and other records

      (2) File structure (blocking and access methods)

      (3) Storage requirements in all processing modes

   b. Interfaces between specified utilities and support programs--Format of data, description of data flow, etc.

9. Requirements management plan

   a. Personnel assignments--Requirements definition and analysis team organization; key personnel and their responsibilities and start dates; etc.

b. Description of required documents--Name, form, and contents of documents; production standards; date scheduled for delivery; name of organization (person) responsible; internal, external, and quality assurance review procedures; authorization procedure for release; etc.

   (1) System functional specifications and requirements

   (2) Software system

   (3) Operational support

c. Specifications/requirements change control procedures--Initiation, forms, reviews, approval, authorization, distribution

d. System enhancement/maintenance procedures--Initiation, forms, reviews, approval, authorization

e. Reporting and testing evaluation procedures--Forms, reviews, approval, authorization, distribution

10. Personnel organization and interfaces--Diagram showing organizational interfaces, their points of contact, and their responsibilities

11. System performance and testing requirements--Organization responsible; testing philosophy and procedures; forms; internal, external, and quality assurance reviews; approval

    a. Analytical tests
    b. System tests
    c. Interface tests
    d. Acceptance tests

12. Issues, TBD items, and problems--A characterization of all those things that affect plans for preliminary design and the state of the requirements, an assessment of their effect on progress, and a course of action to resolve them, including required effort, schedule, and cost

13. Milestones and suggested development schedule--Reviews; delivery of interfaces, documents, and externally developed software; data flows for readiness preparation and training

A.2 PRELIMINARY DESIGN REVIEW (PDR)

PDR PRESENTATION MATERIAL

- Introduction and Agenda
- Design Overview
- High-Level Diagrams of Operating Scenarios
- High-Level Diagrams of System Structure
- Major Software Components
  - High-Level Diagrams of Subsystems
  - High-Level I/O Specifications and Interfaces
  - Functional Baseline Diagrams (Treecharts)
- Design Team Assessment
- System Size, Required Effort, Cost, and Schedule Estimates
- Resource Allocation and External Support
- Development Management Plan
- Personnel Organization and Interfaces
- Testing Strategy
- Issues, TBD Items, and Problems
- Milestones and Schedules

PDR FORMAT

PRESENTERS	Software Development Team
PARTICIPANTS	Requirements Definition Team Quality Assurance Representatives From Both Groups Customer Interfaces for Both Groups
TIME	After Functional Design Completed and Before Detailed Design Started
HARDCOPY DISTRIBUTION	Minimum of 5 Days Before PDR

9103-(51f)-83

A.2 PRELIMINARY DESIGN REVIEW (PDR)

PDR HARDCOPY MATERIAL

1. Introduction
2. Design Overview
3. High-Level Diagrams of Operating Scenarios
4. High-Level Diagrams of System Structure
5. Critique of Alternative Designs
6. Major Software Components
   a. High-Level Diagrams of Subsystems
   b. High-Level I/O Specifications and Interfaces
   c. Functional Baseline Diagrams (Treecharts)
   d. Screen, Printer, and Plotter Formats
7. Hardware Interfaces
8. Internal Data Set Definitions
9. Reusable Code Summary
10. Design Team Assessment
11. System Size, Required Effort, Cost, and Schedule Estimates
12. Resource Allocation and External Support
13. Development Management Plan
    a. Life Cycle and Products
    b. Methodologies
    c. Models and Tools
    d. Configuration Control Approach
14. Personnel Organization and Interfaces
15. Testing Strategy
    a. General Approach
    b. Extent
    c. Control Mechanisms
16. Issues, TBD Items, and Problems
17. Milestones and Schedules

9108-(51g)-83

The PDR hardcopy material contains

1. Introduction--Purpose of the system and outline of review material
   a. Requirements summary--Origin and format of requirements; list of major system components, including the top-level (basic) requirements which they satisfy and which are covered in the review
   b. Additional derived software requirements--Requirements derived by the development team during the requirements analysis phase
      (1) Operating scenario requirements--Data handling, execution frequency, turnaround time, checkpoint/restart capabilities, graphics needs, etc.
      (2) Environment considerations--Target computing machine, operating system, computer system support software, graphics packages and hardware, etc.
      (3) Software legacy (past experiences and history)
         (a) Cost factors--Experience history, design models, reusable code, etc.
         (b) Schedule factors--Deadlines; hardware, software, and support dependencies; etc.
2. Design overview
   a. Requirements summary--List cross-referencing top-level (basic) requirements to major system components presented at SRR
   b. Performance requirements--Cross-reference list of performance requirements that led to partitioning of system into major components

c. Design drivers--Primary factors that influenced the development team's design, e.g., operating scenarios, environmental considerations, and software legacy

3. High-level diagrams of operating scenarios--Input stimulus, processing, output stimulus, and interfaces to show how requirements are met

4. High-level diagrams of system structure--Internal and external data and hardware interfaces, etc.

5. Critique of alternative designs or approaches

6. Major software components--For each subsystem or major functional breakdown (in each processing mode),

   a. High-level diagrams of subsystems--Internal and external data and hardware interfaces, etc.

   b. High-level input and output specifications, including frequency and volume

   c. Functional baseline diagrams (treecharts) expanded to two levels below the subsystem driver, showing interfaces, dat[illegible] .low, and how requirements are met

   d. Facsimiles of I/O graphics displays (screens) and printer and plotter output

   e. Error processing and recovery strategy

7. Hardware interfaces

8. Internal data sets

   a. Format and description of items in header, data, and other records

   b. File structure (blocking and access methods)

   c. Storage requirements in all processing modes

9. Summary of existing code that may be reused

10. Design team assessment

    a. List of constraints and their effects on design

    b. List of assumptions and possible effects on design if they are wrong

    c. List of concerns and problem areas--Deterrents of progress

    d. List of TBD requirements and an assessment of their impact on system size, required effort, cost, and schedule

    e. List of priority areas

11. Estimates of system size, required effort, cost, and schedule

    a. Sizing and resources for major system components or subsystems--Number of modules, source lines of code, computer hours, effort units, etc.

    b. Life cycle expenditures--Time and effort breakdown for life cycle phases

    c. Staffing plan--Allocation of personnel by type for life cycle phases

12. Resource allocation and external support

    a. Summary of how system functions will be performed, i.e., by hardware, firmware, software, or human

    b. Rationale for selecting computers, e.g., speed, memory, storage, and reliability of mainframes, minicomputers, or microcomputers

    c. Summary of what the development team will do and what they need to do it, e.g., analysis support, librarian support, computer access and information, support documentation, interface access, integration support

13. Development management plan

    a. Life cycle phases and products produced

    b. Methodologies used by phase

    c. Models and tools used by phase

    d. Configuration control approach followed by phase--Controlled items, forms, procedures, approval, authorization, distribution

14. Personnel organization and interfaces--Diagram showing organizational interfaces, their points of contact, and their responsibilities

15. Testing strategy

    a. General approach to testing (methods)

    b. Extent--Responsibility and procedures for unit tests, build/release tests integration tests, system tests, acceptance tests

    c. Control mechanisms--Internal, external, and quality assurance review procedures; approval; authorization; configuration integrity procedures

16. Issues, TBD items, and problems--A characterization of all those things that affect plans for detailed design, an assessment of their effect on progress, and a course of action to resolve them, including required effort, schedule, and cost

17. Milestones and schedules--Including delivery of interfaces and externally developed software for integration and testing

A.3 CRITICAL DESIGN REVIEW (CDR)

CDR PRESENTATION MATERIAL

- Introduction and Agenda
- Design Overview
- High-Level Diagrams of Operating Scenarios
- High-Level Diagrams of System Structure
- Major Software Components
  - High-Level Diagrams of Subsystems
  - High-Level I/O Specifications and Interfaces
  - Functional Baseline Diagrams (Treecharts)
  - Error Processing and Recovery Strategy
  - Restrictions of Processing Modes
  - Internal Storage Requirements
- Design Team Assessment
- Implementation Strategy and Traceability
- System Size, Required Effort, Cost, and Schedule Estimates
- Resource Allocation and External Support
- Development Management Plan
- Personnel Organization and Interfaces
- Testing Strategy
- Issues, TBD Items, and Problems
- Milestones and Schedules

CDR FORMAT

PRESENTERS	Software Development Team
PARTICIPANTS	Requirements Definition Team Quality Assurance Representatives From Both Groups Customer Interfaces for Both Groups
TIME	After Detailed Design Completed and Before Implementation Started
HARDCOPY DISTRIBUTION	Minimum of 5 Days Before CDR

A.3 CRITICAL DESIGN REVIEW (CDR)

CDR HARDCOPY MATERIAL

1. Introduction
2. Design Overview
3. High-Level Diagrams of Operating Scenarios
4. High-Level Diagrams of System Structure
5. Major Software Components
   a. High-Level Diagrams of Subsystems
   b. High-Level I/O Specifications and Interfaces
   c. Functional Baseline Diagrams (Treecharts)
   d. Error Processing and Recovery Strategy
   e. Restrictions of Processing Modes
   f. Internal Storage Requirements
   g. Detailed I/O Specifications
      (1) Processing Control Parameters
      (2) Screen, Printer, and Plotter Formats
6. Hardware Interfaces
7. Internal Data Set Definitions
8. Reusable Code Summary
9. Design Team Assessment
10. Implementation Strategy and Traceability
11. System Size, Required Effort, Cost, and Schedule Estimates
12. Resource Allocation and External Support
13. Development Management Plan
    a. Life Cycle and Products
    b. Methodologies
    c. Models and Tools
    d. Configuration Control Approach
14. Personnel Organization and Interfaces
15. Testing Strategy
    a. General Approach
    b. Extent
    c. Control Mechanisms
16. Issues, TBD Items, and Problems
17. Milestones and Schedules

9103-(51)-83

The CDR hardcopy material contains

1. Introduction--Purpose of the system and outline of review material

    a. Requirements summary--Origin and format of requirements; list of major system components, including the top-level (basic) requirements which they satisfy and which are covered in the review

    b. Additional derived software requirements--Requirements derived by the development team during the requirements analysis phase

        (1) Operating scenario requirements--Data handling, execution frequency, turnaround time, checkpoint/restart capabilities, graphics needs, etc.

        (2) Environment considerations--Target computing machine, operating system, computer system support software, graphics packages and hardware, etc.

        (3) Software legacy (past experiences and history)

            (a) Cost factors--Experience history, design models, reusable code, etc.

            (b) Schedule factors--Deadlines; hardware, software, and support dependencies; etc.

2. Design overview

    a. Requirements summary--List cross-referencing top-level (basic) requirements to major system components presented at SRR

    b. Performance requirements--Cross-reference list of performance requirements that led to partitioning of system into major components

   c. Design drivers--Primary factors that influenced the development team's design, e.g., operating scenarios, environmental considerations, and software legacy

3. High-level diagrams of operating scenarios--Input stimulus, processing, output stimulus, and interfaces to show how requirements are met

4. High-level diagrams of system structure--Internal and external data and hardware interfaces, etc.

5. Major software components--For each subsystem or major functional breakdown (in each processing mode),

   a. High-level diagrams of subsystems--Internal and external data and hardware interfaces, etc.

   b. High-level input and output specifications, including frequency and volume

   c. Baseline diagrams (treecharts) expanded to the subroutine level, showing interfaces, data flow, interactive control, interactive input and output, and how requirements are met

   d. Error processing and recovery strategy

   e. Restrictions in each processing mode

   f. Internal storage requirements--Description of arrays, their size, their data capacity in all processing modes, and implied limitations of processing

   g. Detailed input and output specifications

      (1) Processing control parameters--NAMELISTs, etc.

      (2) Facsimiles of I/O graphics displays (screens) and printer and plotter output

6. Hardware interfaces

7. Internal data sets

   a. Format and description of items in header, data, and other records

   b. File structure (blocking and access methods)

   c. Storage requirements in all processing modes

8. Summary of existing code that may be reused

9. Design team assessment

   a. List of constraints and their effects on design

   b. List of assumptions and possible effects on design if they are wrong

   c. List of concerns and problem areas--Deterrents of progress

   d. List of TBD requirements and an assessment of their impact on system size, required effort, cost, and schedule

   e. List of priority areas

10. Implementation strategy and traceability

   a. Build/release overview and schedule, indicating establishment of internal and external data interfaces for both connection tests and data flow tests and showing delivery of interfaces and externally developed software

   b. Build/release capabilities--List of capabilities implemented in each build/release by subsystem

   c. Requirements traceability--Cross-reference list of build/release capabilities to basic and derived software requirements

11. Estimates of system size, required effort, cost, and schedule

    a. Sizing and resources for major system components or subsystems--Number of modules, source lines of code, computer hours, effort units, etc.

    b. Life cycle expenditures--Time and effort breakdown for life cycle phases

    c. Staffing plan--Allocation of personnel by type for life cycle phases

12. Resource allocation and external support

    a. Summary of how system functions will be performed, i.e., by hardware, firmware, software, or human

    b. Rationale for selecting computers, e.g., speed, memory, storage, and reliability of mainframes, minicomputers, or microcomputers

    c. Summary of what the development team will do and what they need to do it, e.g., analysis support, librarian support, computer access and information, support documentation, interface access, integration support

13. Development management plan

    a. Life cycle phases and products produced

    b. Methodologies used by phase

    c. Models and tools used by phase

    d. Configuration control approach followed by phase--Controlled items, forms, procedures, approval, authorization, distribution

14. Personnel organization and interfaces--Diagram showing organizational interfaces, their points of contact, and their responsibilities

15. Testing strategy

    a. General approach to testing (methods)

    b. Extent--Responsibility and procedures for unit tests, build/release tests, integration tests, system tests, acceptance tests

    c. Control mechanisms--Internal, external, and quality assurance review procedures; approval; authorization; configuration integrity procedures

16. Issues, TBD items, and problems--A characterization of all those things that affect plans for detailed design, an assessment of their effect on progress, and a course of action to resolve them, including required effort, schedule, and cost

17. Milestones and schedules--Including delivery of interfaces and externally developed software for integration and testing

A.4 OPERATIONAL READINESS REVIEW (ORR)

ORR PRESENTATION MATERIAL

- Introduction and Agenda
- System Requirements Summary
- Analysis Overview
- Support System Overview
  - Major Software Components
  - System Testing Philosophy
  - System Testing and Performance Evaluation Results
  - Requirements Verification Philosophy
  - System Software and Documentation Status
- Operations and Support Plan
  - Personnel Assignments and Responsibilities
  - Organizational Interfaces
  - Data Availability
  - Facilities
  - Operating Scenarios
- System Management Plan
- Personnel Organization and Interfaces
- Issues, TBD Items, and Problems
- Contingency Plans
- Milestones and Timeline of Events

ORR FORMAT

PRESENTERS	Operations and Support Team
PARTICIPANTS	User Acceptance Test Team Requirements Definition, Software Development, and Software Maintenance Representatives Quality Assurance Representatives From All Groups Customer Interfaces for All Groups
TIME	After Acceptance Testing Completed and 90 Days Before Operations Start
HARDCOPY DISTRIBUTION	Minimum of 5 Days Before ORR

9103-(51b)-83

PRECEDING PAGE BLANK NOT FILMED

A.4 OPERATIONAL READINESS REVIEW (ORR)

ORR HARDCOPY MATERIAL

1. Introduction
2. System Requirements Summary
3. Analysis Overview
4. Support System Overview
   - a. Major Software Components
   - b. System Testing Philosophy
   - c. System Testing and Performance Evaluation Results
   - d. Requirements Verification Philosophy
   - e. System Software and Documentation Status
5. Operations and Support Plan
   - a. Personnel Assignments and Responsibilities
   - b. Organizational Interfaces
   - c. Data Availability
   - d. Facilities
     - (1) Normal Operations
     - (2) Critical Operations
     - (3) Emergency Operations
     - (4) Contingency Operations
   - e. Operating Scenarios
     - (1) Support Requirements
     - (2) Timeline of Events
     - (3) Operating Procedures
     - (4) Resources Required
6. System Management Plan
   - a. Personnel Assignments
   - b. Description of Required Products
   - c. Configuration Control Approach
   - d. Enhancement/Maintenance Procedures
   - e. Reporting and Testing Evaluation Procedures
   - f. System Performance Evaluation Procedures
7. Personnel Organization and Interfaces
8. Issues, TBD Items, and Problems
9. Contingency Plans
10. Milestones and Timeline of Events

9108-(51c)-83

The ORR hardcopy material contains

1. Introduction--Purpose of the system and outline of review material

2. System requirements summary--Review of top-level (basic) requirements

    a. Background of requirements--Overview of project characteristics, major events, and support

    b. Derivation of requirements--Diagram showing

        (1) Project office input
        (2) Support organization input
        (3) System engineering input
        (4) Software engineering input

    c. Type of requirements

        (1) Evaluation of support requirements

            (a) Typical support
            (b) Critical support
            (c) Special support
            (d) Contingency support

        (2) Operational support scenarios

        (3) Relationship of requirements matrix, e.g., relationship of top-level requirements to operational support scenarios

    d. Constraints, assumptions, and guidelines

        (1) Organizational interfaces--Organizations that provide system and support input and receive system output

        (2) Data availability for the operating scenarios--Frequency, volume, format

        (3) Facilities--Computing hardware, environment characteristics, communications protocols, etc.

   (4) General system considerations--High-level description of computer storage, graphics, and failure/recovery requirements; operator interaction requirements; system error recovery and diagnostic output requirements; etc.

   (5) Support and test software considerations--High-level description of requirements for data simulators, test programs, and support utilities

3. Analysis overview--Summary of all analysis leading to an operational system

   a. Introduction--Project overview, support firsts, bases for analysis

   b. Major areas of analysis and findings

   c. Special studies and results

4. Support system overview

   a. Major software components--Purpose, general characteristics, and operating scenarios supported by programs and subsystems

   b. System testing philosophy for each type of testing--Unit, build/release, integration, checkout, and acceptance

   c. System testing and performance evaluation results--Summary of test results and evaluation of system performance measured against performance requirements

   d. Requirements verification philosophy--Demonstration of methods used to ensure that the software satisfies all system requirements

e. System software and documentation status--Summary of completed work packages and list of incomplete work packages with scheduled completion dates and explanation of delays

5. Operations and support plan

   a. Personnel assignments and responsibilities--Operations and support team organization, key personnel and their responsibilities, etc.

   b. Organizational interfaces--Diagrams and tables indicating organizational interfaces, their points of contact and their responsibilities; data flow and medium (forms, tapes, voice, log)

   c. Data availability--Nominal schedule of input and output data by type, format, frequency, volume, response time, turnaround time

   d. Facilities--Nominal schedule of accessibility to computers, support hardware, special-purpose hardware, operating systems, and support software for

      (1) Normal operations
      (2) Critical operations
      (3) Emergency operations
      (4) Contingency operations

   e. Operating scenarios--Step-by-step operating procedures, including personnel assignments, points of contact for decisions, authorization, and external interfaces, timelines, contingencies, and emergencies

      (1) Support requirements--What has to be done?
      (2) Timeline of events--When do things get done?
      (3) Operating procedures--How do things get done?

(4) Resources required for operations--What is necessary to get things done?

(a) Hardware--CPUs, disks, tapes, printers, graphic devices, etc.

(b) Memory considerations--Program storage, array storage, data set buffers, etc.

(c) Timing considerations--CPU and wallclock time in terms of samples and cycles processed and I/O time in terms of data sets used and type of processing

(d) Peripheral space considerations--Data storage and printout

(e) Staffing considerations--Number and type of people in terms of processing steps and shifts

(f) Physical space considerations--Desks, chairs, shelves, storage, etc.

6. System management plan

a. Personnel assignments--Operations and support team organization; key personnel and their responsibilities and start dates; etc.

b. Description of required products--Name, form, and contents of products; production standards; date scheduled for delivery; name of organization (person) responsible; internal, external, and quality assurance review procedures; authorization procedures for release; etc.

c. Configuration control procedures--Explanation of step-by-step procedures for maintaining system integrity, recovering from loss, fixing faults, and enhancing system

   d. Enhancement/maintenance procedures--Initiation, forms, reviews, approval, authorization

   e. Reporting and testing evaluation procedures--Forms, reviews, approval, authorization, distribution

   f. System performance evaluation procedures--Approach for ongoing evaluation of system performance

7. Personnel organization and interfaces--Diagram showing organizational interfaces, their points of contact, and their responsibilities

8. Issues, TBD items, and problems--A characterization of all those things that affect intended normal operations as perceived by the developers and users, an assessment of their effect on operations, and a course of action to resolve them, including required effort, schedule, and cost

9. Contingency plans--Prioritized list of things that could prevent normal operations, including the steps necessary to work around the problems, the defined levels of operations during the workarounds, and the procedures to attempt to regain normal operations

10. Milestones and timeline of events--Diagrams, tables, and scripts of events; operating scenarios; maintenance; enhancement; reviews; training; etc.

B DEVELOPMENT DOCUMENTS

DEVELOPMENT DOCUMENTS

This appendix presents suggested contents and formats for various documents produced during the software development life cycle. Following each document format summary is a brief description of the contents.

B.1 SOFTWARE DEVELOPMENT PLAN

SOFTWARE DEVELOPMENT PLAN FORMAT

1. Introduction
2. Problem Statement
3. Approach
   - a. Technical
   - b. Management
   - c. Configuration Control
4. Resource Estimates and Staffing Profile
5. Schedules and Milestones
6. Items Required From the Customer
7. Items To Be Delivered to the Customer

9103-(51)-03

Software Development Plan

The software development plan (see Reference 2) is completed during the requirements analysis and preliminary design phases. It contains

1. Introduction, including purpose, background, origin of requirements, personnel organization, and summary of plan
2. Problem statement, i.e., what has to be done and what steps are necessary, including what is different about this project compared with a typical development project
3. Approach
   - a. Technical approach (by life cycle phase)
     - (1) Assumptions and constraints
     - (2) Anticipated and unresolved problems
     - (3) Major activities, including methods and tools used
     - (4) Products produced, including contents and standards for production
   - b. Management approach (by life cycle phase)
     - (1) Concerns
     - (2) Resource estimates, including system size in terms of code origin and language as well as support such as project personnel, computer time, external groups, and training
     - (3) Personnel assignments and schedules
     - (4) Progress accounting procedures, i.e., data collection, progress measurement, and progress reporting methods
     - (5) Quality assurance procedures
     - (6) Internal and external review methods

   c. Configuration management procedures for maintaining requirements, design, code, and documentation integrity

4. Summary of resources required and cost

5. Milestone charts showing the development life cycle, delivery of required external interfaces, scheduled integration of externally developed software, delivery of required information, delivery of development products, and scheduled reviews

6. List of items (dated) required from the customer

7. List of items (dated) to be delivered to the customer

B.2 PROJECT NOTEBOOK

PROJECT NOTEBOOK FORMAT

1. Description and List of Major Components
2. Key Personnel and Their Responsibilities
3. Description of Capabilities in Each Build/Release
4. Deliverable Products
5. History of Events, Schedules, and Milestones
6. History of System Size, Required Effort, and Cost Estimates
7. History of Source Code Changes
8. History of Accomplishments
9. History of Outstanding TBD Items, Changes, Errors, and Problems
10. History of Verification of System Requirements

8103-(51)-33

PRECEDING PAGE BLANK NOT FILMED

Project Notebook

The project notebook is started at the beginning of the project and maintained throughout the development life cycle. It contains

1. Description and list of major components
   a. Description of project
   b. List of major components of the system
2. Key personnel and their responsibilities
   a. Name, title, organization, telephone number, start date, and responsibility of key personnel
   b. Name, title, organization, telephone number, start date, and responsibility of points of contact for interfacing groups
3. Description of functional capabilities in each build/release listed in relation to the requirements
4. Deliverable products
   a. Name of product, form of product, date scheduled for delivery, and actual date delivered for each product
   b. Name of person responsible for each product
5. History of events, schedules, and milestones
   a. Milestone chart identifying tasks and originally scheduled completion dates
   b. Biweekly updates to milestone charts
6. History of system size, required effort, and cost estimates
   a. Graph of number of modules in operational version of system versus time in weeks, including estimates of final number in end product

b. Graphs of number of source lines of code (with and without comments) and executable lines of code in operational version of system versus time in weeks, including estimates of final numbers in end product

c. Graph of staff-months of effort expended versus time in weeks, including estimates of planned expenditures (weekly totals and accumulated)

7. History of source code changes--Graph of number of source code changes and change reports versus time in weeks for each development life cycle phase and overall

8. History of accomplishments--Monthly list of achieved milestones, completed tasks, and delivered products

9. History of outstanding TBD items, changes, errors and problems, i.e., list of those things that are unresolved

10. History of verification of system requirements, i.e., list of functional capabilities that have entered the system and were tested by build/release, system testing, and acceptance testing with date of test and test result

B.3 REQUIREMENTS ANALYSIS SUMMARY REPORT

REQUIREMENTS ANALYSIS SUMMARY REPORT FORMAT

1. Introduction
2. System Constraints
3. Development Assumptions
4. Areas of Concern and TBD Requirements
5. Analysis of Basic and Derived System Requirements
6. Analysis of Basic and Derived Requirements by Subsystem
7. Data Interfaces
8. Summary of Reusable Software
9. System Size, Required Effort, Cost, and Schedule Estimates

9103-(51)-63

PRECEDING PAGE BLANK NOT FILMED

Requirements Analysis Summary Report

The requirements analysis summary report is completed at the end of the requirements analysis phase. It contains

1. Introduction, including purpose and background of project
   a. Overall system concepts
   b. Discussion and high-level pictures (diagrams) of system showing hardware interfaces, external data interfaces, and data flow
   c. Discussion and high-level pictures (diagrams) of operating scenarios with interfaces and data flow
2. System constraints
   a. Hardware availability
      (1) Execution
      (2) Storage
      (3) Peripherals
   b. Operating system limitations
   c. Support software limitations
3. Development assumptions
4. Areas of concern and TBD requirements
   a. List of concerns and problem areas, i.e., deterrents of progress
   b. List of TBD requirements and an assessment of their impact on system size, required effort, cost, and schedule
   c. List of priority areas
5. Analysis of basic and derived requirements for system, including level of importance of key issues and completeness

a. Stimulus for input

(1) Frequency

(2) Volume

(3) Coordinates and units

(4) Timing

b. Processing

(1) Functionality

(2) Accuracy

(3) Timing

(4) Error handling

c. Stimulus for output

(1) Frequency

(2) Volume

(3) Coordinates and units

(4) Timing

6. Analysis of basic and derived requirements for subsystems or major functional breakdowns, including level of importance of key issues and completeness. Same as for item 5 above except for subsystems or major functional breakdown

7. Data interfaces--For each interface,

a. Description, including name, function, frequency, coordinates, units, and computer type, length, and representation

b. Format

(1) Organization, e.g., physical sequential

(2) Transfer medium, e.g., 9-track tape, printout

   (3) Layout of frames, samples, records, blocks, and/or transmissions

   (4) Storage requirements

8. Summary of existing code that may be reused

9. Estimates of system size, required effort, cost, and schedule

B.4 PRELIMINARY DESIGN REPORT

PRELIMINARY DESIGN REPORT FORMAT

1. Introduction
   a. Overall System Concepts
   b. High-Level Pictures of System
   c. Operating Scenarios
   d. Design Status
   e. Critique of Alternative Designs
2. Subsystems
   a. High-Level Pictures
   b. Description of Input and Output
   c. Description of Processing
   d. Functional Baseline Diagrams
   e. Prologs and PDL for First Level
3. Resource Requirements
   a. Hardware
   b. Data Definitions
   c. Peripheral Space Considerations
   d. Memory Considerations
4. Data Interfaces
   a. Description
   b. Format
5. Summary of Reusable Software

9103-(51)-83

Preliminary Design Report

The preliminary design report is completed at the end of the preliminary design phase. It contains

1. Introduction, including purpose and background of project
   a. Overall system concepts
   b. Discussion and high-level pictures (diagrams) of system showing hardware interfaces, external data interfaces, and data flow
   c. Discussion and high-level pictures (diagrams) of operating scenarios with interfaces and data flow
   d. Design status
      (1) List of constraints and their effects on design
      (2) List of assumptions and possible effects on design if they are wrong
      (3) List of concerns and problem areas, i.e., deterrents of progress
      (4) List of TBD requirements and an assessment of their impact on system size, required effort, cost, and schedule
      (5) List of priority areas
   e. Critique of alternative designs
2. For each subsystem or major functional breakdown,
   a. Discussion and high-level pictures (diagrams) of subsystem, including interfaces, data flow, and communications for each processing mode
   b. High-level description of input and output
   c. High-level description of processing keyed to operator-specified input and actions in terms of points of control, functions performed, and results

obtained (both normal and abnormal, i.e., error processing and recovery)

d. Functional baseline diagrams (treecharts) expanded to two levels below the subsystem driver

e. Prologs[1] and PDL for each module through first level below subsystem driver

3. Resource requirements--Discussion, high-level pictures, and tables for system and subsystem

   a. Hardware

   b. Data definitions, i.e., data groupings and names

   c. Peripheral space considerations

      (1) Data storage
      (2) Printout

   d. Memory considerations

      (1) Program storage
      (2) Array storage
      (3) Data set buffers

4. Data interfaces--For each internal and external interface,

   a. Description, including name, function, frequency, coordinates, units, and computer type, length, and representation

   b. Format

      (1) Organization, e.g., physical sequential

      (2) Transfer medium, e.g., tape

[1]Module comments to describe the module's purpose, operation, calling sequence arguments, external references, etc.

   (3) Layout of frames, samples, records, blocks, and/or transmissions

   (4) Storage requirements

5. Summary of existing code that may be reused

j. Description of COMMON Areas

k. Prologs and PDL

3. Resource Requirements

4. Data Interfaces

5. Summary of Reusable Software

6. Results of System Modeling

B.9 SOFTWARE DEVELOPMENT HISTORY

SOFTWARE DEVELOPMENT HISTORY FORMAT

1. Project Description and Background
2. Development History
3. Project Assessment
4. Functional Specifications and Requirements
5. Summary
6. References

obtained (both normal and abnormal, i.e., error processing and recovery)

e. Baseline diagrams (treecharts) expanded to the subroutine level showing interfaces, data flow, interactive control, interactive input and output, and hardcopy output

f. Restrictions in each processing mode

g. Internal storage requirements, i.e., description of arrays, their size, their data capacity in all processing modes, and implied limitations of processing

h. Detailed input and output specifications

    (1) Processing control parameters, e.g., NAMELISTs

    (2) Facsimiles of graphic displays for interactive graphic systems

    (3) Facsimiles of hardcopy output

i. List of numbered error messages with description of system's and user's actions

j. Description of COMMON areas

k. Prologs[1] and PDL for each subroutine

3. Resource requirements--Discussion, high-level pictures, and tables for system and subsystems

    a. Hardware

    b. Data definitions, i.e., data groupings and names

    c. Peripheral space considerations

        (1) Data storage

        (2) Printout

[1]Module comment to describe the module's purpose, operation, calling sequence arguments, external references, etc.

d. Memory considerations

(1) Program storage

(2) Array storage

(3) Data set buffers

4. Data interfaces--For each internal and external interface,

a. Description, including name, function, frequency, coordinates, units, and computer type, length, and representation

b. Format

(1) Organization, e.g., direct access

(2) Transfer medium, e.g., disk

(3) Layout of frames, samples, records, blocks, and/or transmissions

(4) Storage requirements

5. Summary of existing code that may be reused, including list of code with level of modification

B.6 TEST PLANS

TEST PLAN FORMAT

1. Introduction
   a. Purpose
   b. Type and Level of Testing
   c. Schedule
2. For Each Test
   a. Purpose
   b. Detailed Specification of Input
   c. Required Environment
   d. Operational Procedure
   e. Detailed Specification of Output
   f. Pass/Fail Criteria
   g. Discussion of Results

8108-(51)-83

Test Plans

A test plan is completed before the period in which it will be used, with sufficient time for testers to review it. A test plan contains

1. Introduction, including purpose, type and level of testing, and schedule
2. For each test,
   a. Purpose of test, i.e., specific capabilities or requirements tested
   b. Detailed specification of input
   c. Required environment, e.g., data sets required, computer hardware necessary
   d. Operational procedure, i.e., how to do the test
   e. Detailed specification of output, i.e., the expected results
   f. Criteria for determining whether or not test results are acceptable
   g. Section for discussion of results, i.e., for explaining deviations from expected results and identifing cause of deviation or for justifying deviation

B.7 USER'S GUIDE

USER'S GUIDE FORMAT

1. Introduction
   a. Overall System Concepts
   b. High-Level Pictures
   c. Operating Scenarios
2. Subsystems
   a. Overall Capability
   b. Assumptions and Restrictions
   c. High-Level Pictures
   d. Description of Input and Output
   e. Description of Processing
3. Requirements for Execution
   a. Resources
   b. Run Information
   c. Control Parameter Information
4. Detailed Description of Input and Output
   a. Facsimiles of Graphic Displays
   b. Facsimiles of Hardcopy Output
   c. List of Messages

9108-(50a)-83

User's Guide

Formalization of the user's guide is started during the implementation phase using the design document as a starting point. Reorganization is completed for the beginning of system testing, and a typed draft is completed for the beginning of acceptance testing. The user's guide is completed by the end of acceptance testing. It contains

1. Introduction, including purpose and background
    a. Overall system concepts
    b. Discussion and high-level pictures (diagrams) of system showing hardware interfaces, external data interfaces, and data flow
    c. Discussion and high-level pictures (diagrams) of operating scenarios with interfaces and data flow
2. For each subsystem or major functional breakdown,
    a. Overall subsystem capability
    b. Assumptions and restrictions to processing
    c. Discussion and high-level pictures (diagrams) of subsystems, including interfaces, data flow, and communications for each processing mode
    d. High-level description of input and output
    e. Detailed description of processing keyed to operator-specified input and actions in terms of points of control, functions performed, and results obtained (both normal and abnormal, i.e., error processing and recovery)

3. Requirements for execution

   a. Resources--Discussion, high-level pictures (diagrams), and tables for system and subsystems

      (1) Hardware

      (2) Data definitions, i.e., data groupings and names

      (3) Peripheral space considerations

         (a) Data storage
         (b) Printout

      (4) Memory considerations

         (a) Program storage
         (b) Array storage
         (c) Data set buffers

      (5) Timing considerations

         (a) CPU time in terms of samples and cycles processed

         (b) I/O time in terms of data sets used and type of processing

         (c) Wallclock time in terms of samples and cycles processed

   b. Run information--Control statements for various processing modes

   c. Control parameter information--By subsystem, detailed description of all control parameters (e.g., NAMELISTs), including name, computer type, length, and representation, and description of parameter with valid values, default value, units, and relationship to other parameters

4. Detailed description of input and output by system and subsystem

   a. Facsimiles of graphic displays for interactive graphic systems in the order in which they appear for each processing mode

   b. Facsimiles of hardcopy output in the order in which it is produced, annotated to show what parameters control it

   c. List of numbered messages with explanation of system's and user's actions annotated to show subroutines that issue the message

B.8 SYSTEM DESCRIPTION

SYSTEM DESCRIPTION FORMAT

1. Introduction
2. Subsystems
   a. Overall Capability
   b. Assumptions and Restrictions
   c. High-Level Pictures
   d. Description of Input and Output
   e. Baseline Diagrams
3. Requirements for Creation
   a. Resources
   b. Creation Information
   c. Program Structure Information
4. Detailed Description of Input and Output
5. Internal Storage Requirements
6. Data Interfaces
7. Description of COMMON Areas
8. Prologs and PDL
9. List of Software from Support Libraries

9108-(50a)-83

System Description

Formalization of the system description is started at the beginning of system testing using the design document as a starting point. It is completed by the end of acceptance testing. The system description contains

1. Introduction, including purpose and background of project
    a. Overall system capabilities
    b. Discussion and high-level pictures (diagrams) of system showing hardware interfaces, external data interfaces, and data flow
    c. Discussion and high-level pictures (diagrams) of operating scenarios with interfaces and data flow
2. For each subsystem or major functional breakdown,
    a. Overall subsystem capability
    b. Assumptions and restrictions to processing
    c. Discussion and high-level pictures of subsystem, including interfaces, data flow, and communications for each processing mode
    d. High-level description of input and output
    e. Detailed baseline diagram at subroutine level showing interfaces, data flow, interactive control, interactive input and output, including messages, and hardcopy output
3. Requirements for creation
    a. Resources--Discussion, high-level pictures (diagrams), and tables for system and subsystems
        (1) Hardware
        (2) Support data sets

(3) Peripheral space considerations

(a) Source code storage
(b) Scratch space
(c) Printout

(4) Memory considerations

(a) Program generation storage
(b) Data set buffers

(5) Timing considerations

(a) CPU time in terms of compile, build, and execute benchmark test

(b) I/O time in terms of the steps to create the system

b. Creation information--Control statements for various steps

c. Program structure information--Control statements, e.g., for overlay or for addressing and loading

4. Detailed description of input and output by step

a. Source code libraries for system and subsystems
b. Object code libraries
c. Execution code libraries
d. Auxiliary libraries, e.g., support tables

5. Internal storage requirements, i.e., description of arrays, their size, their data capacity in all processing modes, and implied limitations of processing

6. Data interfaces--For each internal and external interface,

a. Description, including name, function, frequency, coordinates, units, and computer type, length, and representation

b. Format

(1) Organization, e.g., indexed

(2) Transfer medium, e.g., drum

(3) Layout of frames, samples, records, blocks, and/or transmissions

(4) Storage requirements

7. Description of COMMON areas

8. Prologs and PDL for each subroutine (usually produced in a separate volume)

9. Alphabetical list of subroutines from support data sets, including--for each subroutine--a reference to the support data set from which it comes and a description of the subroutine's function

B.9 SOFTWARE DEVELOPMENT HISTORY

SOFTWARE DEVELOPMENT HISTORY FORMAT

1. Project Description and Background
2. Development History
3. Project Assessment
4. Functional Specifications and Requirements
5. Summary
6. References

9100-(31)-03

Software Development History

The software development history is completed within 3 months of software acceptance. It contains

1. Project description and background
    a. Problem statement and list of key requirements
    b. Origin of requirements
    c. Customer of system
    d. Purpose of system
    e. Key dates (actuals)
        (1) Availability of functional specifications and requirements
        (2) Development phase dates (start and finish), i.e., requirements analysis, preliminary design, detailed design, implementation, system testing, acceptance testing
        (3) Event dates, e.g., SRR, PDR, CDR, ORR
    f. Key products produced
        (1) All software, i.e., the system, simulators, and test programs
        (2) All documents, i.e., development plan, project notebook, requirements analysis summary report, preliminary design report, detailed design document, test plans and results, user's guide, and system description
    g. Development organization
    h. System characteristics
        (1) Total, new, and reused number of source lines of code (with and without comments) of end product

(2) Total, new, and reused number of modules of end product

(3) Total, managerial, programmer, and support service hours required for development

2. Development history

   a. Original and updated estimates of system size, required effort, schedule, and cost

   b. Organizational structure and key personnel

   c. Specified approaches, e.g., methods, practices, standards, and tools

   d. Unique approaches, e.g., independent verification and validation team, prototyping

   e. Target development machine and programming languages

   f. Special problems encountered

   g. Build/release history

   h. Test history

   i. Configuration control start dates for requirements, design, and code

3. Project assessment

   a. Substantiated major strengths of the development process and product

   b. Substantiated major weaknesses of the development process and product

   c. Major problem areas

   d. Development plan timeliness and usefulness

   e. Adherence to development plan

   f. Adherence to standards and practices

   g. Design timeliness, completeness, and quality
   h. Code timeliness, completeness, and quality
   i. Test timeliness, completeness, and quality
   j. Personnel adequacy (number and quality)

4. Functional specifications and requirements
   a. Origin and timeliness
   b. Completeness and adequacy for design
   c. Change history and stability
   d. Clarity (i.e., were there misinterpretations?)

5. Summary
   a. List of shortcomings of the development process and product
   b. List of successful aspects of the development process and product
   c List of things that should be done differently for future projects
   d. List of things that should be done similarly for future projects
   e. List of the major causes of errors

6. References
   a. List of relevant background documents, e.g., functional specifications and requirements, software development plan, test strategy and plans
   b. List of reports, e.g., requirements analysis summary report, preliminary design report, detailed design document, test plans with results, change histories
   c. List of any other necessary reference material

C BRIEF EXAMPLE OF SOME STEPS TO ORGANIZE A PROJECT

BRIEF EXAMPLE OF SOME STEPS TO ORGANIZE A PROJECT

This appendix contains a brief example of some steps the development manager must take to organize a project.

Senior-level development managers usually have, in their minds, the benefits of experience; unfortunately, this experience is not usually written down. When the benefits of senior personnel's experience are documented, they are usually summarized in a way that is difficult for the junior-level development manager to interpret and apply. Without similar experience, a junior-level manager will often find a more experienced manager's experience summary to be confusing or unintelligible.

The example here is an illustration for the more junior-level manager. It is not necessarily universally applicable; however, the example illustrates steps and factors that a development manager must consider when organizing a project. The following subsections are interrelated because the organization of the approach depends on the constraints of the problem; i.e., the order in which steps are taken depends on the problem.

C.1 ESTIMATING SIZE AND EFFORT

One of the most critical aspects of organizing a development project is estimating the amount of software to be developed and the effort required to develop it. Poor estimates, either high or low, cause a loss of confidence in the eyes of the customer as well as within the development organization.

In this example, the development organization develops software systems for a broad application of related scientific, data base, and data support systems. New applications (project types) and computing facilities (environment types) are introduced every few years. The development organization develops systems for both the old and new computing facilities and for the same and different, but usually related, applications. That is, the organization not only builds on past experience for continuing support of its charter but also branches out into new aspects of its charter because of changing technology. Therefore, in general, a moderate amount of code is reused from project to project.

C.1.1 SIZE OF THE SYSTEM

At each phase of the development life cycle, the development organization uses essential, supplemental, and archived information to produce an estimate of the system's size with acceptable and understood uncertainty limits (see Reference 22). Table C-1 lists some high-level information. For example, in this example, the development organization uses 3040 x (number of general subsystems) to compute the number of executable lines of code (LOC).

Table C-1. Effort Estimators and Uncertainty Limits by Phase

End of Phase	Effort Estimators	Limits of Uncertainty as Percent of Estimated Effort[a]
Preproject	Similar projects, general subsystems	±100
Requirements Analysis	General subsystems	±70
Preliminary design	Specific subsystems	±50
Detailed design	Actual subsystems, modules, code, documentation	±30
Implementation	Modules, code, tests, documentation	±12
System testing	Code, tests, documentation	±5

[a]Upper limit = (Effort estimate)x(1. + uncertainty in decimal)
Lower limit = (Effort estimate)/(1. + uncertainty in decimal)

C.1.2 EFFORT REQUIRED TO DEVELOP THE SYSTEM

In this example, using the size estimate from any life cycle phase, Equation (C-1) predicts the total effort[1] required for complete development[2]

$$E^{Tot} = 8\ f_1 f_2 f_3 \ldots f_k\ L^{1.05} \qquad \text{(C-1)}$$

[1]Total development effort includes administrative and technical managers, developers, and support personnel, i.e., secretaries, librarians, and technical publications. See Table C-8 on page C-16.

[2]Complete development encompasses the software development life cycle phases from requirements analysis through acceptance testing.

where E^{Tot} is effort in staff-months; the f_i's are factors that increase or decrease required effort because of problem complexity, team experience, schedule, methodology used, and so on; and

$$L = (0.8N + 0.2) \times (\text{estimate of total system size in thousands of executable LOC}) \quad \text{(C-2)}$$

where N is the fraction of newly developed and extensively modified code in the system size estimate.

Basically, to start, the manager need only consider the complexity of the problem, the development team's experience, and the schedule. As the development organization establishes itself, other factors (such as methodology usage) can be added to fine-tune the estimation process and to apply it to a wider range of software development problems.

For this example, Table C-2 provides the development organization with a simple guideline for adjusting effort with a

Table C-2. Complexity Guideline[a]

Project Type[b]	Environment Type[c]	Effort Factor (f_1)
Old	Old	0.45
Old	New	0.65
New	Old	0.65
New	New	1.00

[a]Based on SEL data and other data available to the SEL.

[b]Application, e.g., orbit determination, data base. The project type is old when the development team has more than 2 years of experience with it.

[c]Computing environment, e.g., IBM S/370, VAX-11/780, Intel. The environment type is old when the development team has more than 2 years of experience with it.

complexity factor. Table C-3 provides a simple guideline for adjusting effort with a team experience factor. Table C-4 provides a simple guideline for adjusting effort with a schedule factor.

Table C-3. Development Team Experience Guideline

Team Years of Applicable Experience[a]	Effort Factor (f_2)
10	0.5
8	0.6
6	0.8
4	1.0
2	1.4
1	2.5

[a]Sum of products of fraction of team member participation with his/her years of applicable experience (requirements/specification definition, development, maintenance, and operation).

Table C-4. Schedule Guideline

Schedule Characterization	Effort Factor (f_3)
Fast	1.15
Optimum	1.00
Slow	0.85

To illustrate, assume that the development organization must develop a system estimated at 25,000 executable LOC. It is similar to ones they have developed before (old project type), but it is being developed for a new computing facility (new environment type); therefore, $f_1 = 0.65$.

Fifty percent of the code can be reused without modification ($N = 0.5$). The development organization has a team in mind whose weighted applicable experience is 6 years ($f_2 = 0.8$) and the luxury of a slow schedule ($f_3 = 0.85$). Then, assuming that all other factors are normal ($f_i = 1$, $i = 4$ to k),

$$\begin{aligned} E^{Tot} &= 8.0(0.65)(0.8)(0.85)\ \{[(0.8)(0.5) + 0.2]\ 25.0\}^{1.05} \\ &= 60.7 \text{ staff-months} \qquad \text{(C-3)} \end{aligned}$$

Since it is the beginning of the project, there is an uncertainty limit of ±100 percent in the system size estimate. Therefore,

$$\begin{aligned} E^{Tot} \text{ (upper limit)} &= 3.536\ \{15.0\}^{1.05} \times (1.0 + 1.0) \\ &= 121.5 \text{ staff-months} \qquad \text{(C-4)} \end{aligned}$$

$$\begin{aligned} E^{Tot} \text{ (lower limit)} &= 3.536\ \{15.0\}^{1.05} / (1.0 + 1.0) \\ &= 30.4 \text{ staff-months} \qquad \text{(C-5)} \end{aligned}$$

Table C-1 (page C-4) lists the effort uncertainty limits that can be expected at the end of each life cycle phase. Figure C-1 illustrates the effort uncertainty limits as a function of phase. It is not sufficient for managers to merely state uncertainty limits. They must be able to explain why these limits can be reached, i.e., what factors are uncertain, how they can increase or decrease the estimate, and by how much.

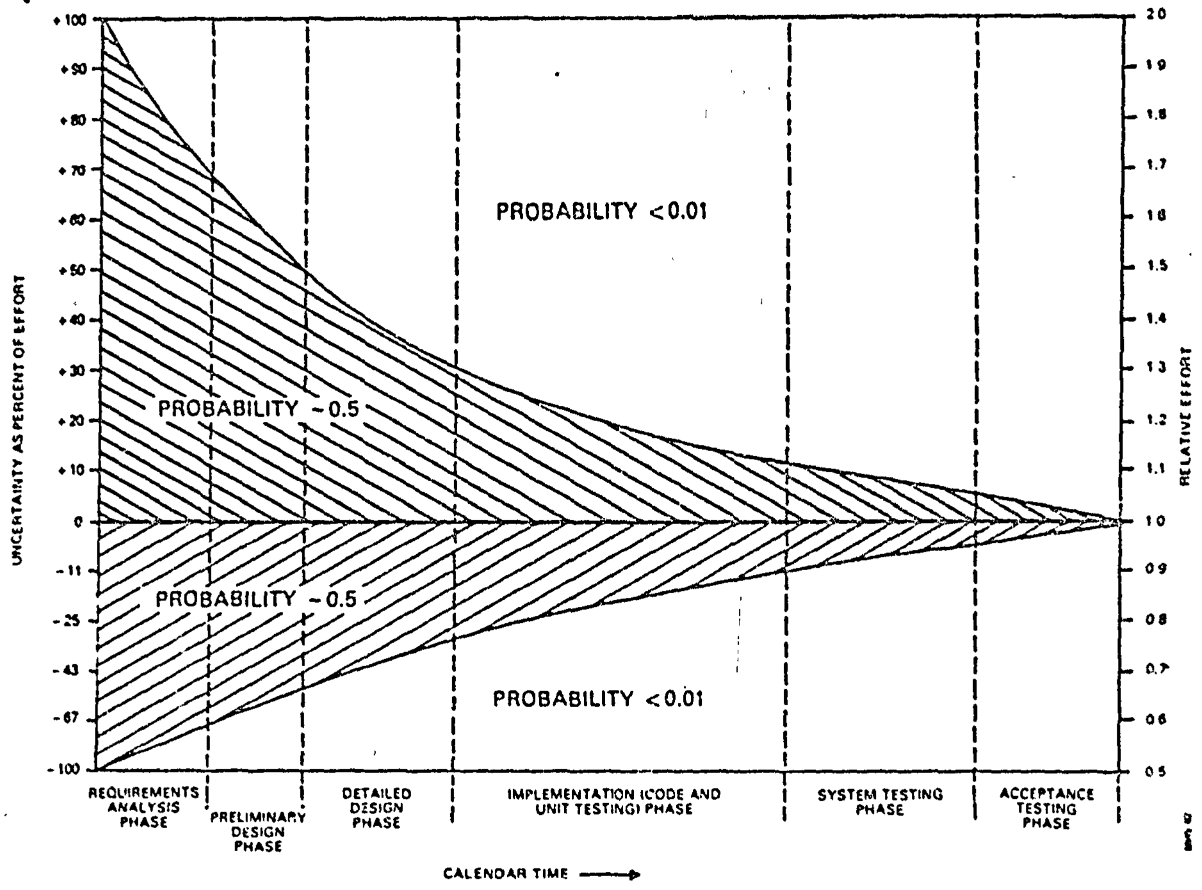


Figure C-1. Effort Uncertainty Limits as a Function of Phase

ORIGINAL PAGE IS
OF POOR QUALITY

C.2 ESTABLISHING REALISTIC SCHEDULES

The literature contains many models for resource estimation that are simple to implement but take some time to understand and calibrate for a particular environment. Few, however, explain in any detail how to establish schedules from the information produced. Therefore, this subsection is probably more important for the more junior-level manager. To establish realistic schedules, the development managers must basically consider the effort required for each activity, the team size, and the experience of the team leader.

C.2.1 REQUIRED EFFORT FOR LIFE CYCLE PHASES

In this example, the development organization computes the fraction of effort required for the design, coding, and testing activities from Equations (C-6) through (C-8), respectively.

$$f^{DAct} = 0.36N + 0.04 \qquad (C-6)$$

$$f^{CAct} = 0.08N + 0.02 \qquad (C-7)$$

$$f^{TAct} = 0.36N + 0.14 \qquad (C-8)$$

where N is the fraction of newly developed and extensively modified code in the system size estimate and DAct, CAct, and TAct are abbreviations for the design, coding, and testing activities.

The development organization computes the effort required for the design, coding, and testing activities from Equations (C-9) through (C-12), respectively.

ORIGINAL PAGE IS
OF POOR QUALITY

$$f = f^{DAct} + f^{CAct} + f^{TAct} \qquad \text{(C-9)}$$

$$E^{DAct} = (f^{DAct}/f)\ E^{Tot} \qquad \text{(C-10)}$$

$$E^{CAct} = (f^{CAct}/f)\ E^{Tot} \qquad \text{(C-11)}$$

$$E^{TAct} = (f^{TAct}/f)\ E^{Tot} \qquad \text{(C-12)}$$

The development organization computes the effort required for each life cycle phase from Equations (C-13) through (C-18), respectively.

$$E^{RAPh} = 0.15\ E^{DAct} \qquad \text{(C-13)}$$

$$E^{PDPh} = 0.20\ E^{DAct} \qquad \text{(C-14)}$$

$$E^{DDPh} = 0.40\ E^{DAct} \qquad \text{(C-15)}$$

$$E^{IPh} = 0.19\ E^{DAct} + 0.94\ E^{CAct} + 0.57\ E^{TAct} \qquad \text{(C-16)}$$

$$E^{STPh} = 0.05\ E^{DAct} + 0.05\ E^{CAct} + 0.33\ E^{TAct} \qquad \text{(C-17)}$$

$$E^{ATPh} = 0.01\ E^{DAct} + 0.01\ E^{CAct} + 0.10\ E^{TAct} \qquad \text{(C-18)}$$

where RAPh, PDPh, DDPh, IPh, STPh, and ATPh are abbreviations for the requirements analysis, preliminary design, detailed design, implementation, system testing, and acceptance testing phases, respectively.

Equations (C-13) through (C-18) indicate that 75 percent of the design activity effort is expended to get to CDR; nearly 20 percent of the design activity effort is expended during implementation to respond to functional specification, design, and implementation errors; and smaller amounts of the design activity effort are expended during system and acceptance testing for the same reasons. The equations also show that nearly 60 percent of the testing activity effort is expended during implementation, 33 percent during system testing, and 10 percent during acceptance testing.

C.2.2 REALISTIC SCHEDULES

Once the effort for each phase is known, the development manager must determine how fast th development team can be staffed, how large it can get, and how fast team members can be released without losing control of discipline and order. This determination has to be made based on the project leader's experience level. In this example, Table C-5 provides the development organization with a simple guideline for determining team size in terms of the experience of the team leaders. Table C-6 provides a simple guideline for producing a staffing pattern.

Using these guidelines, the manager will find that team size peaks at the beginning of implementation. When the team is too large for a less senior project leader, the development manager can replace the project leader with a more senior project leader or extend the schedule. When the team size is too large for a senior project leader, the manager must extend the schedule or partition the development effort into several smaller projects. Forming smaller projects, of course, will present the manager with software integration problems and additional management and support charges because each smaller project will have a project leader and its own reporting and support requirements.

Table C-5. Team Size Guideline

Minimum Years of Experience[a]						Maximum Team Size Excluding Team Leaders
Project Manager			Project Leader			
App.	Org.	Leader	App.	Org.	Leader	
8	6	5	6	4	3	7 ±2
7	5	3	5	3	1	4.5 ±1.5
6	4	2	4	2	0	2 ±1

[a] App. = Applicable experience, i.e., requirements/ specification definition, development, maintenance, and operation.

Org. = Experience with organization.

Leader = Experience as project leader or manager.

ORIGINAL PAGE IS
OF POOR QUALITY

Table C-6. Staffing Pattern Guideline

PROJECT LEADER		SCHEDULE TYPE	DEVELOPMENT TEAM MEMBERS		
TYPE	LEAD TIME (WEEKS)[a]		PHASE IN INCREMENT (WEEKS)[b]	PHASE OUT INCREMENT (WEEKS)[c]	MINIMUM LENGTH OF PARTICIPATION (WEEKS)
SENIOR	4	FAST	1	2	6
	6	OPTIMUM	2	3	6
	6	SLOW	3	4	6
INTERMEDIATE	5	FAST	2	3	7
	6	OPTIMUM	3	4	7
	7	SLOW	4	5	7
JUNIOR	6	FAST	3	4	8
	7	OPTIMUM	4	5	8
	8	SLOW	5	6	8

[a]TIME THAT THE PROJECT MANAGER AND LEADER NEED TO ORGANIZE AND PLAN PROJECTS BEFORE OTHER TEAM MEMBERS JOIN THE PROJECT

[b]FASTEST RATE AT WHICH TEAM MEMBERS ARE ADDED TO THE PROJECT SO THAT THE PROJECT LEADER CAN MAINTAIN ORDER

[c]FASTEST RATE AT WHICH TEAM MEMBERS LEAVE THE PROJECT SO THAT THE DETAILS OF THEIR ASSIGNMENTS CAN BE ABSORBED BY THE TEAM AND MINIMIZE CALLBACK.

C.3 ALLOCATING PROPER RESOURCES

Junior first-line development managers do not usually have much influence in forming the development team. A higher level manager in the development organization generally selects personnel based on his/her judgment and experience with the problem and discussions with the first-line manager. The first-line manager, however, must become familiar with the judgments made and the experience base available, so that he/she can organize the project effectively.

In this example, Table C-7 provides the development organization with a simple guideline for determining the type and experience level of personnel needed in terms of the complexity of the problem. Table C-8 provides a simple guideline for allocating resources in terms of manager, developer, and other support charges.

Table C-7. Development Team Staffing Guideline

Project Type[a]	Environment Type[a]	Percentage of Senior Personnel[b]	Percentage of Analysts[c]
Old	Old	25-33	25-33
Old	New	33-50	25-33
New	Old	33-50	33-50
New	New	50-67	33-50

[a]The project and environment types are old when the development team has more than 2 years of experience with them.

[b]Senior personnel are those with more than 5 years of experience in development-related activities.

[c]Analysts are those personnel who have training and an educational background in problem definition and solution with the application (project type) or the computers (environment type) depending on the problem.

Table C-8. Development Staff Composition Guideline[1] (1 of 2)

Personnel Category	Percentage of Staff	Function
MANAGERS	21.5	
Project Manager	5.0	First-line development manager responsible, with project leader, for organization and planning of project. Provides technical consultation and manages project resources.
Project Leader	7.5	Lead developer responsible, with project manager, for organization and planning of project. Provides technical direction and day-to-day supervision of project activities.
Administrative	4.5	Second-, third-, fourth-line development organization (higher level) managers and project control office personnel responsible for administrative aspects of projects, such as financial reporting and quality assurance of documentation and progress report formats and procedures.
Customer Interface	4.5	First- or second-line manager from customer's organization responsible for monitoring resources and progress. Is primary point of contact with external customer, support, and contractor groups.

[1]This breakdown is for development charges. Other charges to the overall cost of a project come from staffs with analogous breakdowns, e.g., the requirements team, the independent verification and validation team, the maintenance and operation team.

Table C-8. Development Staff Composition Guideline[1] (2 of 2)

Personnel Category	Percentage of Staff	Function
DEVELOPERS	63.5	Programmer/analysts responsible for requirements analysis, design, implementation, testing, and documentation of software.
SUPPORT	15.0	
Librarian	5.5	Personnel responsible for development-related clerical work and source code maintenance.
Secretarial	4.0	Personnel responsible for development-related and development organization office clerical work.
Publications	5.5	Reproduction, composition, graphics, and editorial personnel responsible for formal production of documents and reports.

[1]This breakdown is for development charges. Other charges to the overall cost of a project come from staffs with analogous breakdowns, e.g., the requirements team, the independent verification and validation team, the maintenance and operation team.

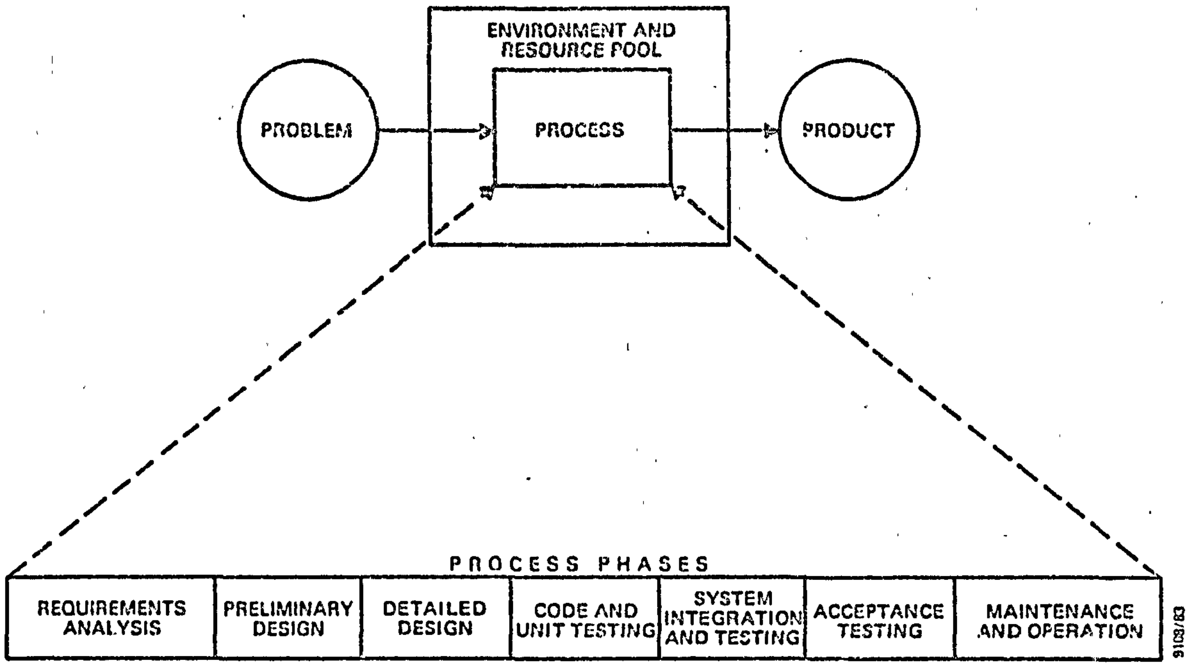
C.4 SUMMARY

Although the example in this appendix is not a recommended or even a suggested procedure for organizing aspects of a project, it provides the junior-level development manager with basic information about organization that is not available with simple estimation models. The type of information emphasizes the need for managers to record development information and judgments through the development plan, data collection, and postmortem evaluations.

Managers must realize that guidelines are just that and must not let their education obstruct their common sense. Blindly applying guidelines to solve a particular problem frequently leads to undesirable situations. Guidelines should only provide a starting point from which a manager makes common sense adjustments to suit project-specific conditions.

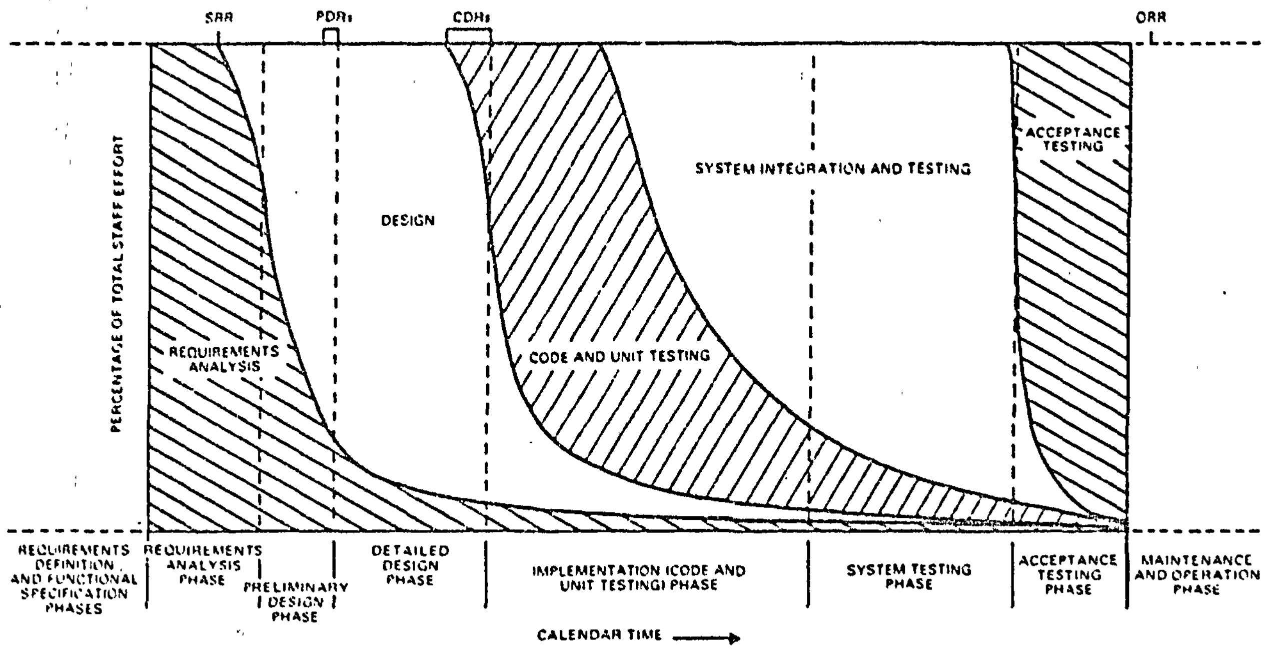
D SUMMARY OF KEY INFORMATION

1.2 SOFTWARE DEVELOPMENT MODEL



ORIGINAL PAGE IS
OF POOR QUALITY

2.0 SOFTWARE DEVELOPMENT ACTIVITIES AND LIFE CYCLE PHASES



NOTE FOR EXAMPLE AT THE END OF THE IMPLEMENTATION PHASE (4TH DASHED LINE), APPROXIMATELY 79% OF THE STAFF ARE INVOLVED IN SYSTEM INTEGRATION AND TESTING. APPROXIMATELY 2% ARE ADDRESSING REQUIRMENTS CHANGES OR PROBLEMS, APPROXIMATELY 2% ARE DESIGNING MODIFICATIONS AND APPROXIMATELY 17% ARE CODING AND UNIT TESTING CHANGES

8100-(51)-83

3.1 REQUIREMENTS ANALYSIS

DEVELOPMENT	ACTIONS AND TRANSACTIONS
ACTIVITIES	Software Development Plan Prepared Functional Requirements Amplified Performance Requirements Analyzed Operational Requirements Determined External Interfaces Identified Report and Display Requirements Determined TBD Requirements Identified System Size Estimated Reusable Software Identified Computer Resources Determined Hardware Selected Requirements Analysis Summary Report Prepared SRR Held
END PRODUCTS	Software Development Plan Requirements Analysis Summary Report
METHODOLOGIES	Project Notebook Data Collection Librarians Unit Development Folders Requirements Question and Change Records Structured Analysis (Complex Data Processing)
TOOLS	Configuration Analysis Tool (CAT)

9103-(51)-83

3.1 REQUIREMENTS ANALYSIS

MANAGEMENT	ACTIONS AND TRANSACTIONS
ACTIVITIES	Software Development Plan Reviewed Schedule and Staffing Planned Requirements Analysis Summary Report Reviewed Preliminary Design Transition Planned Team Trained Standards and Procedures Enforced Progress Monitored Visibility Promoted System Size Estimated Resources and Cost Estimated Team Interaction Coordinated
MEASURES	TBD Requirements Requirements Questions and Answers Requirements Changes Subjective Evaluations

9108-(51)-03

3.2 PRELIMINARY DESIGN

DEVELOPMENT	ACTIONS AND TRANSACTIONS
ACTIVITIES	System Partitioned Processing Options Defined Alternative Designs Examined External Interfaces Defined Subsystems Partitioned Subsystem Interfaces Defined Error Processing and Recovery Defined TBD Requirements Resolved Reusable Software Identified Preliminary Design Report Prepared PDR Held
END PRODUCTS	Preliminary Design Report Software Development Plan Update
METHODOLOGIES	Project Notebook Data Collection Librarians Unit Development Folders Requirements Question and Change Records Design Formalisms Design Decision and Change Records Configuration Management Design Walkthroughs Iterative Enhancement Information Hiding Data Abstraction PDL
TOOLS	PDL Processor Source Code Library Management System CAT Resource Estimation

9103-(51)-83

3.2 PRELIMINARY DESIGN

MANAGEMENT	ACTIONS AND TRANSACTIONS
ACTIVITIES	TBD Requirements Resolved Requirements Changes Reviewed and Assessed Design Reviewed and Walked Through Detailed Design Transition Planned Team Trained Standards and Procedures Enforced Progress Monitored Visibility Promoted System Size Estimated Resources and Cost Estimated Team interaction Coordinated
MEASURES	TBD Requirements Requirements Changes Requirements Questions and Answers Design Changes Interfaces Design Completion Checklist Subjective Evaluations

9136-(51)-83

3.3 DETAILED DESIGN

DEVELOPMENT	ACTIONS AND TRANSACTIONS
ACTIVITIES	Single Functions Refined Baseline Diagrams Prepared I/O Specified PDL and Prologs Specified COMMON Blocks Specified Internal Interfaces Specified TBD Requirements Resolved Reusable Software Identified Detailed Design Document Prepared Implementation Strategy Planned CDR Held
END PRODUCTS	Detailed Design Document Software Development Plan Update Implementation Plan
METHODOLOGIES	Project Notebook Data Collection Librarians Unit Development Folders Requirements Question and Change Records Design Formalisms Design Decision and Change Records Configuration Management Design Walkthroughs Iterative Enhancement Information Hiding Data Abstraction PDL
TOOLS	PDL Processor Source Code Library Management System CAT Resource Estimation

9103-(51)-83

3.3 DETAILED DESIGN

MANAGEMENT	ACTIONS AND TRANSACTIONS
ACTIVITIES	Implementation Strategy Reviewed Implementation Transition Planned TBD Items Resolved Requirements Changes Reviewed and Assessed Design Reviewed and Walked Through Team Trained Standards and Procedures Enforced Progress Monitored Visibility Promoted System Size Estimated Resources and Cost Estimated Team Interaction Coordinated
MEASURES	TBD Items Requirements Changes • Requirements Questions and Answers Design Changes Interfaces Design Completion Checklist Design Growth Rate Module Strength Module Coupling Subjective Evaluations

9103-(51)-83

3.4 IMPLEMENTATION

DEVELOPMENT	ACTIONS AND TRANSACTIONS
ACTIVITIES	Job Control Language Prepared Command Procedures Prepared New Modules Coded Reusable Modules Revised Units Integrated and Tested Build/Release Test Plans Prepared Data Prepared Build/Release Test Plans Executed Discrepancies Resolved System Integrated System Test Plan Prepared Acceptance Test Plan Prepared User's Guide Prepared System Description Prepared
END PRODUCTS	System Code Supporting Data and System Files Build/Release Test Plans and Results System Test Plan Acceptance Test Plan Draft User's Guide Draft System Description Software Development Plan Update
METHODOLOGIES	Project Notebook Data Collection Librarians Unit Development Folders Requirements Question and Change Records Design Decision and Change Records Coding Standards Structured Code Code Reading Code Change Records Configuration Management Builds/Releases Top-Down Implementation Formal Test Plans Functional (Thread) Testing
TOOLS	PDL Processor Source Code Library Management System Structured Coding Language CAT Resource Estimation

9108-(51)-83

3.4 IMPLEMENTATION

MANAGEMENT	ACTIONS AND TRANSACTIONS
ACTIVITIES	Build/Release Test Plans Reviewed Build/Release Test Plan Results Reviewed Discrepancies Resolved System Test Plan Reviewed Draft User's Guide Reviewed Draft System Description Reviewed System Testing Transition Planned TBD Items Resolved Requirements Changes Reviewed and Assessed Design Changes Reviewed and Walked Through Code Changes Reviewed Team Trained Standards and Procedures Enforced Progress Monitored Visibility Promoted System Size Estimated Resources and Cost Estimated Team Interaction Coordinated
MEASURES	TBD Items Requirements Changes Requirements Questions and Answers Design Changes Code Changes Code/Test Completion Checklists Code Growth Rate Error/Change Growth Rates Discrepancies/Resolutions Growth Rates Computer Usage Growth Rate Team/Individual Productivity Rates Subjective Evaluations

9103-(51)-83

3.5 SYSTEM TESTING

DEVELOPMENT	ACTIONS AND TRANSACTIONS
ACTIVITIES	System Created System Test Plan Executed Discrepancies Resolved User's Guide Reviewed and Revised System Description Reviewed and Revised Acceptance Testing Planned
END PRODUCTS	System Code Update Supporting Data and System Files Update System Test Plan Results User's Guide Update System Description Update Software Development Plan Update
METHODOLOGIES	Project Notebook Data Collection Librarians Unit Development Folders Requirements Question and Change Records Design Decision and Change Records Code Change Records Configuration Management Formal Test Plan Functional (Thread) Testing
TOOLS	PDL Processor Source Code Library Management System Structured Coding Language CAT Resource Estimation

9108-(51)-83

3.5 SYSTEM TESTING

MANAGEMENT	ACTIONS AND TRANSACTIONS
ACTIVITIES	System Test Plan Results Reviewed Discrepancies Resolved Acceptance Test Plan Reviewed Acceptance Testing Transition Planned User's Guide Reviewed System Description Reviewed TBD Items Resolved Requirements Changes Reviewed and Assessed Design Changes Reviewed and Walked Through Code Changes Reviewed Team Trained Standards and Procedures Enforced Progress Monitored Visibility Promoted System Size Estimated Resources and Cost Estimated Team Interaction Coordinated
MEASURES	TBD Items Requirements Changes Requirements Questions and Answers Design Changes Code Changes Test Completion Checklist Code Growth Rate Error/Change Growth Rates Discrepancies/Resolutions Growth Rates Computer Usage Growth Rate Team/Individual Productivity Rates Subjective Evaluations

9103-(51)-83

3.6 ACCEPTANCE TESTING

DEVELOPMENT	ACTIONS AND TRANSACTIONS
ACTIVITIES	System Created Users and Operators Trained Acceptance Test Plan Executed Discrepancies Resolved User's Guide Reviewed and Revised System Description Reviewed and Revised System Delivery Planned ORR Held Software Development History Prepared
END PRODUCTS	System Code Supporting Data and System Files Acceptance Test Plan Results User's Guide System Description Archived System (Tapes) and Documentation Software Development History
METHODOLOGIES	Project Notebook Data Collection Librarians Unit Development Folders Requirements Question and Change Records Design Decision and Change Records Code Change Records Configuration Management Formal Test Plan Functional (Thread) Testing
TOOLS	PDL Processor Source Code Library Management System Structured Coding Language CAT Resource Estimation

9103-(51)-83

3.6 ACCEPTANCE TESTING

MANAGEMENT	ACTIONS AND TRANSACTIONS
ACTIVITIES	Acceptance Test Plan Results Reviewed Discrepancies Received System Delivery Reviewed User's Guide Reviewed System Description Reviewed TBD Items Resolved Requirements Changes Reviewed and Assessed Design Changes Reviewed and Walked Through Code Changes Reviewed Team Trained Standards and Procedures Enforced Progress Monitored Visibility Promoted System Size Estimated Resources and Cost Estimated Team Interaction Coordinated
MEASURES	TBD Items Requirements Changes Requirements Questions and Answers Design Changes Code Changes Test Completion Checklist Code Growth Rate Error/Change Growth Rates Discrepancies/Resolutions Growth Rates Computer Usage Growth Rate Team/Individual Productivity Rates Subjective Evaluations

9103-(51)-83

4.1 INDICATORS OF DEVELOPMENT STATUS

- Frequency of Schedule/Milestone Changes
- Consistency in Organizational Structure Compared With Original Plans
- Fluctuation in Project Staff Level and System Size Estimates
- History of Number and Type of TBD Items for Requirements and Design
- Ease of Access to Information on Project Status, Schedules, and Plans
- Frequency and Amount of Unusually Long Hours Required or Planned To Attain Certain Objectives
- Level of Detail (Both Technical and Managerial) Understood and Controlled by the Project Manager and the Project Leader
- Discrepancies in Staff Level and Workload
- Discrepancies in Planned Weekly Staff Level and Computer Usage or Compared With Past Projects

9103-(51)-83

4.2 DANGER SIGNALS

- Scheduled Capabilities Delayed to Later Build/Release
- Coding Started Too Early (Staff Too Large Too Early)
- Numerous Changes Made to Initial Software Development Plan
- Guidelines or Planned Procedures Deemphasized or Deleted
- Sudden Changes in Staffing (Magnitude) Suggested or Made
- Excessive Documentation and Paperwork That Have Little Direct Bearing on Required Documentation Prepared
- Continual Increase in Numbers of TBD Items and ECRs Measured
- Decrease in Estimated Effort for System Testing Suggested or Made
- Reliance on Other Sources for Soon-To-Be-Available Software

9103-(51)-83

4.3 CORRECTIVE MEASURES

- Stop Current Activities and Review/Complete Predecessor or Problem Activity
- Decrease Staff to Manageable Level
- Replace Junior With Senior Personnel
- Increase and Tighten Management Procedures
- Increase Number of Intermediate Deliverables
- Decrease Scope of Work and Define a Manageable, Doable Thread of the System
- Audit Project with Independent Personnel and Act on Their Findings

9103-(51)-83

5.1 TEN "DOs" FOR PROJECT SUCCESS

- Use a Small Senior Staff for the Early Life Cycle Phases
- Develop and Adhere to a Software Development Plan
- Define Specific Intermediate and End Products
- Examine Alternative Approaches
- Use Formal Testing
- Use a Central Repository
- Keep a Detailed List of TBD Items
- Update System Size, Required Effort, Cost, and Schedule Estimates
- Allocate 30 Percent of Effort for Integration and Testing
- Experiment

9108-(51)-83

5.2 TEN "DON'Ts" FOR PROJECT SUCCESS

- Don't Overstaff
- Don't Allow Team Members To Proceed in an Undisciplined Manner
- Don't Delegate Technical Details to Team Members
- Don't Assume That a Rigid Set of Standards Ensures Success
- Don't Assume That a Large Amount of Documentation Ensures Success
- Don't Deviate from the Approved Design
- Don't Assume That Relaxing Standards Reduces Costs
- Don't Assume That the Pace Will Increase Later in the Project
- Don't Assume That Intermediate Schedule Slippage Can Be Absorbed in a Later Phase
- Don't Assume That Everything Will Fit Together Smoothly at the End

3103-(51)-83

5.3 ASSESSING PROJECT QUALITY

- Is a Written Software Development Plan Being Followed?
- Are Life Cycle Phases and Products Defined?
- Is Someone in Charge?
- Does the Staff Size Match the Workload?
- Do Team Members Know Where the Project Is and Where It Is Going?
- Is a Configuration Control Plan Being Followed?
- Is There a Single Complete List of TBD Items With Assessments?
- Is There a Commonly-Adhered-To Methodology?
- Have Alternative Designs and Approaches Been Considered?
- Are There Contingency Plans for Rationally Solving Problems?

3108-(50a)-83

A.1 SYSTEM REQUIREMENTS REVIEW (SRR)

SRR PRESENTATION MATERIAL

- Introduction and Agenda
- Requirements Summary
- Analysis Overview
- Functional Specifications
  - — Environmental Considerations
  - — Operational Requirements
    - (1) Operating Scenarios
    - (2) Data Flow Analysis
    - (3) Performance Requirements
    - (4) Interface Requirements
  - — Requirements Relationships
- Derived System Requirements
- Requirements Management Plan
- Personnel Organization and Interfaces
- Software Performance and Testing Requirements
- Issues, TBD Items, and Problems
- Milestones and Suggested Development Schedule

SRR FORMAT

PRESENTERS	Requirements Definition Team
PARTICIPANTS	Development Team Representatives Quality Assurance Representatives User Representatives Customer Representatives
TIME	After Functional Specifications Completed and Before Functional Design Started
HARDCOPY DISTRIBUTION	Minimum of 5 Days Before SRR

9103-(51d)-83

A.1 SYSTEM REQUIREMENTS REVIEW (SRR)

SRR HARDCOPY MATERIAL

1. Introduction
2. Requirements Summary
3. Analysis Overview
4. Functional Specifications
   a. Environmental Considerations
   b. Operational Requirements
      (1) Operating Scenarios
      (2) Data Flow Analysis
         (a) System Input
         (b) Processing Requirements
         (c) System Output
      (3) Performance Requirements
      (4) Interface Requirements
   c. Requirements Relationships
5. Derived System Requirements
6. Utility, Support, and Test Programs
7. Reusable Software Summary
8. Data Set Definitions
9. Requirements Management Plan
   a. Personnel Assignments
   b. Description of Required Documents
   c. Configuration Control Approach
   d. Enhancement/Maintenance Procedures
   e. Reporting and Testing Evaluation Procedures
10. Personnel Organization and Interfaces
11. Software Performance and Testing Requirements
   a. Analytical
   b. System
   c. Interface
   d. Acceptance
12. Issues, TBD Items, and Problems
13. Milestones and Suggested Development Schedule

9108 (51a) 83

A.2 PRELIMINARY DESIGN REVIEW (PDR)

PDR PRESENTATION MATERIAL

- Introduction and Agenda
- Design Overview
- High-Level Diagrams of Operating Scenarios
- High-Level Diagrams of System Structure
- Major Software Components
  - High-Level Diagrams of Subsystems
  - High-Level I/O Specifications and Interfaces
  - Functional Baseline Diagrams (Treecharts)
- Design Team Assessment
- System Size, Required Effort, Cost, and Schedule Estimates
- Resource Allocation and External Support
- Development Management Plan
- Personnel Organization and Interfaces
- Testing Strategy
- Issues, TBD Items, and Problems
- Milestones and Schedules

PDR FORMAT

PRESENTERS	Software Development Team
PARTICIPANTS	Requirements Definition Team Quality Assurance Representatives From Both Groups Customer Interfaces for Both Groups
TIME	After Functional Design Completed and Before Detailed Design Started
HARDCOPY DISTRIBUTION	Minimum of 5 Days Before PDR

A.2 PRELIMINARY DESIGN REVIEW (PDR)

PDR HARDCOPY MATERIAL

1. Introduction
2. Design Overview
3. High-Level Diagrams of Operating Scenarios
4. High-Level Diagrams of System Structure
5. Critique of Alternative Designs
6. Major Software Components
    a. High-Level Diagrams of Subsystems
    b. High-Level I/O Specifications and Interfaces
    c. Functional Baseline Diagrams (Treecharts)
    d. Screen, Printer, and Plotter Formats
7. Hardware Interfaces
8. Internal Data Set Definitions
9. Reusable Code Summary
10. Design Team Assessment
11. System Size, Required Effort, Cost, and Schedule Estimates
12. Resource Allocation and External Support
13. Development Management Plan
    a. Life Cycle and Products
    b. Methodologies
    c. Models and Tools
    d. Configuration Control Approach
14. Personnel Organization and Interfaces
15. Testing Strategy
    a. General Approach
    b. Extent
    c. Control Mechanisms
16. Issues, TBD Items, and Problems
17. Milestones and Schedules

9108-(51g)-83

A.3 CRITICAL DESIGN REVIEW (CDR)

CDR PRESENTATION MATERIAL

- Introduction and Agenda
- Design Overview
- High-Level Diagrams of Operating Scenarios
- High-Level Diagrams of System Structure
- Major Software Components
  - High-Level Diagrams of Subsystems
  - High-Level I/O Specifications and Interfaces
  - Functional Baseline Diagrams (Treecharts)
  - Error Processing and Recovery Strategy
  - Restrictions of Processing Modes
  - Internal Storage Requirements
- Design Team Assessment
- Implementation Strategy and Traceability
- System Size, Required Effort, Cost, and Schedule Estimates
- Resource Allocation and External Support
- Development Management Plan
- Personnel Organization and Interfaces
- Testing Strategy
- Issues, TBD Items, and Problems
- Milestones and Schedules

CDR FORMAT

PRESENTERS	Software Development Team
PARTICIPANTS	Requirements Definition Team Quality Assurance Representatives From Both Groups Customer Interfaces for Both Groups
TIME	After Detailed Design Completed and Before Implementation Started
HARDCOPY DISTRIBUTION	Minimum of 5 Days Before CDR

A.3 CRITICAL DESIGN REVIEW (CDR)

CDR HARDCOPY MATERIAL

1. Introduction
2. Design Overview
3. High-Level Diagrams of Operating Scenarios
4. High-Level Diagrams of System Structure
5. Major Software Components
   a. High-Level Diagrams of Subsystems
   b. High-Level I/O Specifications and Interfaces
   c. Functional Baseline Diagrams (Treecharts)
   d. Error Processing and Recovery Strategy
   e. Restrictions of Processing Modes
   f. Internal Storage Requirements
   g. Detailed I/O Specifications
      (1) Processing Control Parameters
      (2) Screen, Printer, and Plotter Formats
6. Hardware Interfaces
7. Internal Data Set Definitions
8. Reusable Code Summary
9. Design Team Assessment
10. Implementation Strategy and Traceability
11. System Size, Required Effort, Cost, and Schedule Estimates
12. Resource Allocation and External Support
13. Development Management Plan
    a. Life Cycle and Products
    b. Methodologies
    c. Models and Tools
    d. Configuration Control Approach
14. Personnel Organization and Interfaces
15. Testing Strategy
    a. General Approach
    b. Extent
    c. Control Mechanisms
16. Issues, TBD Items, and Problems
17. Milestones and Schedules

A.4 OPERATIONAL READINESS REVIEW (ORR)

ORR PRESENTATION MATERIAL

- Introduction and Agenda
- System Requirements Summary
- Analysis Overview
- Support System Overview
  - Major Software Components
  - System Testing Philosophy
  - System Testing and Performance Evaluation Results
  - Requirements Verification Philosophy
  - System Software and Documentation Status
- Operations and Support Plan
  - Personnel Assignments and Responsibilities
  - Organizational Interfaces
  - Data Availability
  - Facilities
  - Operating Scenarios
- System Management Plan
- Personnel Organization and Interfaces
- Issues, TBD Items, and Problems
- Contingency Plans
- Milestones and Timeline of Events

ORR FORMAT

PRESENTERS	Operations and Support Team
PARTICIPANTS	User Acceptance Test Team Requirements Definition, Software Development, and Software Maintenance Representatives Quality Assurance Representatives From All Groups Customer Interfaces for All Groups
TIME	After Acceptance Testing Completed and 90 Days Before Operations Start
HARDCOPY DISTRIBUTION	Minimum of 5 Days Before ORR

A.4 OPERATIONAL READINESS REVIEW (ORR)

ORR HARDCOPY MATERIAL

1. Introduction
2. System Requirements Summary
3. Analysis Overview
4. Support System Overview
   a. *Major* Software Components
   b. System Testing Philosophy
   c. System Testing and *Performance* Evaluation Results
   d. Requirements Verification Philosophy
   e. System Software and Documentation Status
5. Operations and Support Plan
   a. Personnel Assignments and Responsibilities
   b. Organizational Interfaces
   c. Data Availability
   d. Facilities
      (1) Normal Operations
      (2) Critical Operations
      (3) Emergency Operations
      (4) Contingency Operations
   e. Operating Scenarios
      (1) Support Requirements
      (2) Timeline of Events
      (3) Operating Procedures
      (4) Resources Required
6. System Management Plan
   a. Personnel Assignments
   b. Description of Required Products
   c. Configuration Control Approach
   d. Enhancement/Maintenance Procedures
   e. Reporting and Testing Evaluation Procedures
   f. System Performance Evaluation Procedures
7. Personnel Organization and Interfaces
8. Issues, TBD Items, and Problems
9. Contingency Plans
10. Milestones and Timeline of Events

9108-(51c)-83

B.1 SOFTWARE DEVELOPMENT PLAN

SOFTWARE DEVELOPMENT PLAN FORMAT

1. Introduction
2. Problem Statement
3. Approach
    a. Technical
    b. Management
    c. Configuration Control
4. Resource Estimates and Staffing Profile
5. Schedules and Milestones
6. Items Required From the Customer
7. Items To Be Delivered to the Customer

9108-(51)-83

B.2 PROJECT NOTEBOOK

PROJECT NOTEBOOK FORMAT

1. Description and List of Major Components
2. Key Personnel and Their Responsibilities
3. Description of Capabilities in Each Build/Release
4. Deliverable Products
5. History of Events, Schedules, and Milestones
6. History of System Size, Required Effort, and Cost Estimates
7. History of Source Code Changes
8. History of Accomplishments
9. History of Outstanding TBD Items, Changes, Errors, and Problems
10. History of Verification of System Requirements

9103-(51)-83

B.3 REQUIREMENTS ANALYSIS SUMMARY REPORT

REQUIREMENTS ANALYSIS SUMMARY REPORT FORMAT

1. Introduction
2. System Constraints
3. Development Assumptions
4. Areas of Concern and TBD Requirements
5. Analysis of Basic and Derived System Requirements
6. Analysis of Basic and Derived Requirements by Subsystem
7. Data Interfaces
8. Summary of Reusable Software
9. System Size, Required Effort, Cost, and Schedule Estimates

9103-(51)-83

B.4 PRELIMINARY DESIGN REPORT

PRELIMINARY DESIGN REPORT FORMAT

1. Introduction
   a. Overall System Concepts
   b. High-Level Pictures of System
   c. Operating Scenarios
   d. Design Status
   e. Critique of Alternative Designs
2. Subsystems
   a. High-Level Pictures
   b. Description of Input and Output
   c. Description of Processing
   d. Functional Baseline Diagrams
   e. Prologs and PDL for First Level
3. Resource Requirements
   a. Hardware
   b. Data Definitions
   c. Peripheral Space Considerations
   d. Memory Considerations
4. Data Interfaces
   a. Description
   b. Format
5. Summary of Reusable Software

9108-(51)-83

B.5 DETAILED DESIGN DOCUMENT

DETAILED DESIGN DOCUMENT FORMAT

1. Introduction
2. Subsystems
    a. Overall Capability
    b. High-Level Pictures
    c. Description of Input and Output
    d. Description of Processing
    e. Baseline Diagrams
    f. Restrictions
    g. Internal Storage Requirements
    h. Detailed Input and Output Specifications
    i. List of Messages
    j. Description of COMMON Areas
    k. Prologs and PDL
3. Resource Requirements
4. Data Interfaces
5. Summary of Reusable Software
6. Results of System Modeling

9108-(51)-83

B.6 TEST PLANS

TEST PLAN FORMAT

1. **Introduction**
   - a. **Purpose**
   - b. **Type and Level of Testing**
   - c. **Schedule**
2. **For Each Test**
   - a. **Purpose**
   - b. **Detailed Specification of Input**
   - c. **Required Environment**
   - d. **Operational Procedure**
   - e. **Detailed Specification of Output**
   - f. **Pass/Fail Criteria**
   - g. **Discussion of Results**

9108-(51)-83

B.7 USER'S GUIDE

USER'S GUIDE FORMAT

1. Introduction
   - a. Overall System Concepts
   - b. High-Level Pictures
   - c. Operating Scenarios
2. Subsystems
   - a. Overall Capability
   - b. Assumptions and Restrictions
   - c. High-Level Pictures
   - d. Description of Input and Output
   - e. Description of Processing
3. Requirements for Execution
   - a. Resources
   - b. Run Information
   - c. Control Parameter Information
4. Detailed Description of Input and Output
   - a. Facsimiles of Graphic Displays
   - b. Facsimiles of Hardcopy Output
   - c. List of Messages

9108-(50a)-83

B.8 SYSTEM DESCRIPTION

SYSTEM DESCRIPTION FORMAT

1. Introduction
2. Subsystems
   a. Overall Capability
   b. Assumptions and Restrictions
   c. High-Level Pictures
   d. Description of Input and Output
   e. Baseline Diagrams
3. Requirements for Creation
   a. Resources
   b. Creation Information
   c. Program Structure Information
4. Detailed Description of Input and Output
5. Internal Storage Requirements
6. Data Interfaces
7. Description of COMMON Areas
8. Prologs and PDL
9. List of Software from Support Libraries

9108-(50a) 83

B.9 SOFTWARE DEVELOPMENT HISTORY

SOFTWARE DEVELOPMENT HISTORY FORMAT

1. Project Description and Background
2. Development History
3. Project Assessment
4. Functional Specifications and Requirements
5. Summary
6. References

9105-(51)-83

GLOSSARY

ORIGINAL PAGE IS OF POOR QUALITY

Act	activity
App	applicable
AT	acceptance testing
ATPh	acceptance testing phase
C	coding
CAct	coding activity
CAT	Configuration Analysis Tool
CCB	Configuration Control Board
CDR	critical design review
CPU	Central Processor Unit
D	design
DAct	design activity
DD	detailed design
DDPh	detailed design phase
DEC	Digital Equipment Corporation
E	effort
ECR	Engineering change request, requirements modification request, functional specifications modification request, specifications modification request
f	factor
FSRD	functional specifications and requirements document
GESS	Graphic Executive Support System
GSFC	Goddard Space Flight Center
I	implementation, code and unit testing
IAD	interface agreement document
IBM	International Business Machines
ICD	interface control document
Intel	Intel computer
IPh	implementation phase, code and unit testing phase
I/O	input and output
JCL	Job Control Language
K	one thousand

ORIGINAL PAGE IS
OF POOR QUALITY

L	adjusted lines of code
LOC	lines of code
M&DOD	Mission and Data Operations Directorate
MEDL-R	Multi-Level Expression Design Language - Requirements Level
MOI	memorandum of information
MOU	memorandum of understanding
NASA	National Aeronautics and Space Administration
NSP	NASA Support Plan
Org	organization
ORR	operational readiness review
PD	preliminary design
PDL	Process Design Language, Program Design Language, pseudocode, metacode
PDP	DEC computer
PDPh	preliminary design phase
PDR	preliminary design review
Ph	phase
POC	point of contact
PSA	Problem Statement Analyzer
PSL	Problem Statement Language
RA	requirements analysis
RAPh	requirements analysis phase
RID	review item disposition, review item discrepancy
S	system
SAP	FORTRAN Static Source Code Analyzer Program
SEL	Software Engineering Laboratory
SFORT	Structured FORTRAN Preprocessor
SIRD	support instrumentation requirements document
SLOC	source lines of code
SRR	system requirements review
ST	system testing, system integration and testing
STPh	system testing phase, system integration and testing phase
T	testing

ORIGINAL PAGE IS
OF POOR QUALITY

TAct	testing activity
TBD	"to be determined" used as an adjective (TBD items) or as a noun (TBDs)
Tot	total
UDF	unit development folder
VAX	DEC computer

REFERENCES

1. Software Engineering Laboratory, SEL-81-104, The Software Engineering Laboratory, D. N. Card, F. E. McGarry, G. Page, et al., February 1982

2. --, Software Managers Handbook (to be published)

3. TRW, TRW-SS-76-11, The Unit Development Folder (UDF): An Effective Management Tool for Software Development, F. S. Ingrassia, October 1976

4. E. Yourdon and L. Constantine, Structured Design. New York: Yourdon Press, 1979

5. Software Engineering Laboratory, SEL-80-104, Configuration Analysis Tool (CAT) System Description and User's Guide (Revision 1), W. J. Decker and W. A. Taylor, December 1982

6. D. Teichroew and E. A. Hershey III, "PSL/PSA: A Computer-Aided Technique for Structured Documentation and Analysis of Information Processing Systems," IEEE Transactions on Software Engineering, SE-3:1, January 1977

7. Martin Marietta Aerospace, Multi-Level Expression Design System (MEDSys) - Requirements Level Language (MEDL-R) Description Manual, P. Scheffer and A. Musser, Revised February 1979

8. M. Page-Jones, The Practical Guide to Structured Design. New York: Yourdan Press, 1980; App. B: Walkthroughs

9. R. Tausworthe, Standardized Development of Computer Software, Vol. I. Englewood Cliffs, New Jersey: Prentice-Hall, Inc., 1977

10. D. L. Parnas, "On the Criteria To Be Used in Decomposing Systems Into Modules," Communications of the ACM, vol. 5, no. 2, pp. 1053-1058

11. E. W. Dijkstra "The Structure of the "THE" - Multiprogramming System", Communications of the ACM, May 1968, vol. 11, no. 5, pp. 341-346

12. Computer Sciences Corporation, CSC/TM-76/6189, Telemetry Computation Branch Quality Assurance Procedures: Program Design Language, R. Bieri, November 1976

ORIGINAL PAGE IS
OF POOR QUALITY

13. S. H. Caine and E. K. Gordon, "PDL - A Tool for Software Design," Proceedings of the 1975 National Computer Conference, vol. 44, pp. 271-276

14. G. J. Myers, Composite/Structured Design. New York: Van Nostrand Reinhold Co., 1976

15. University of Maryland, TR-936, A Metal-Model for Software Development Resource Expenditures, J. W. Bailey and V. R. Basili, August 1980

16. R. C. Lingen, H. D. Mills, and B. I. Witt, Structured Programming: Theory and Practice. Reading, Mass.: Addison-Wesley Pub. Co., 1979

17. G. M. Weinberg, The Psychology of Computer Programming. New York: Van Nostrand Reinhold Co., 1971

18. G. J. Myers, The Art of Software Testing. New York: John Wiley & Sons, Inc., 1979

19. Software Engineering Laboratory, SEL-78-004, Structured FORTRAN Preprocessor (SFORT) PDP-11/70 User's Guide, D. S. Wilson and B. Chu, September 1978

20. Computer Sciences Corporation, CSC/TM-78/6296, Acceptance Test Methods, J. Niblack, October 1978

21. Software Engineering Laboratory, SEL-81-101, Guide to Data Collection, V. E. Church, D. N. Card, F. E. McGarry, et al., August 1982

22. --, Guide to Software Cost Estimation (to be published)

ORIGINAL PAGE IS
OF POOR QUALITY

BIBLIOGRAPHY OF SEL LITERATURE

The technical papers, memorandums, and documents listed in this bibliography are organized into two groups. The first group is composed of documents issued by the Software Engineering Laboratory (SEL) during its research and development activities. The second group includes materials that were published elsewhere but pertain to SEL activities.

SEL-Originated Documents

SEL-76-001, Proceedings From the First Summer Software Engineering Workshop, August 1976

SEL-77-001, The Software Engineering Laboratory, V. R. Basili, M. V. Zelkowitz, F. E. McGarry, et al., May 1977

SEL-77-002, Proceedings From the Second Summer Software Engineering Workshop, September 1977

SEL-77-003, Structured FORTRAN Preprocessor (SFORT), B. Chu and D. S. Wilson, September 1977

SEL-77-004, GSFC NAVPAK Design Specifications Languages Study, P. A. Scheffer and C. E. Velez, October 1977

SEL-78-001, FORTRAN Static Source Code Analyzer (SAP) Design and Module Descriptions, E. M. O'Neill, S. R. Waligora, and C. E. Goorevich, February 1978

†SEL-78-002, FORTRAN Static Source Code Analyzer (SAP) User's Guide, E. M. O'Neill, S. R. Waligora, and C. E. Goorevich, February 1978

SEL-78-102, FORTRAN Static Source Code Analyzer Program (SAP) User's Guide (Revision 1), W. J. Decker and W. A. Taylor, September 1982

SEL-78-003, Evaluation of Draper NAVPAK Software Design, K. Tasaki and F. E. McGarry, June 1978

†This document superseded by revised document.

SEL-78-004, Structured FORTRAN Preprocessor (SFORT) PDP-11/70 User's Guide, D. S. Wilson and B. Chu, September 1978

SEL-78-005, Proceedings From the Third Summer Software Engineering Workshop, September 1978

SEL-78-006, GSFC Software Engineering Research Requirements Analysis Study, P. A. Scheffer and C. E. Velez, November 1978

SEL-78-007, Applicability of the Rayleigh Curve to the SEL Environment, T. E. Mapp, December 1978

SEL-79-001, SIMPL-D Data Base Reference Manual, M. V. Zelkowitz, July 1979

SEL-79-002, The Software Engineering Laboratory: Relationship Equations, K. Freburger and V. R. Basili, May 1979

SEL-79-003, Common Software Module Repository (CSMR) System Description and User's Guide, C. E. Goorevich, A. L. Green, and S. R. Waligora, August 1979

SEL-79-004, Evaluation of the Caine, Farber, and Gordon Program Design Language (PDL) in the Goddard Space Flight Center (GSFC) Code 580 Software Design Environment, C. E. Goorevich, A. L. Green, and W. J. Decker, September 1979

SEL-79-005, Proceedings From the Fourth Summer Software Engineering Workshop, November 1979

SEL-80-001, Functional Requirements/Specifications for Code 580 Configuration Analysis Tool (CAT), F. K. Banks, A. L. Green, and C. E. Goorevich, February 1980

SEL-80-002, Multi-Level Expression Design Language-Requirement Level (MEDL-R) System Evaluation, W. J. Decker and C. E. Goorevich, May 1980

SEL-80-003, Multimission Modular Spacecraft Ground Support Software System (MMS/GSSS) State-of-the-Art Computer Systems/Compatibility Study, T. Welden, M. McClellan, and P. Liebertz, May 1980

SEL-80-004, System Description and User's Guide for Code 580 Configuration Analysis Tool (CAT), F. K. Banks, W. J. Decker, J. G. Garrahan, et al., October 1980

SEL-80-005, A Study of the Musa Reliability Model, A. M. Miller, November 1980

ORIGINAL PAGE IS
OF POOR QUALITY

SEL-80-006, Proceedings From the Fifth Annual Software Engineering Workshop, November 1980

SEL-80-007, An Appraisal of Selected Cost/Resource Estimation Models for Software Systems, J. F. Cook and F. E. McGarry, December 1980

†SEL-81-001, Guide to Data Collection, V. E. Church, D. N. Card, F. E. McGarry, et al., September 1981

SEL-81-101, Guide to Data Collection, V. E. Church, D. N. Card, F. E. McGarry, et al., August 1982

†SEL-81-002, Software Engineering Laboratory (SEL) Data Base Organization and User's Guide, D. C. Wyckoff, G. Page, and F. E. McGarry, September 1981

SEL-81-102, Software Engineering Laboratory (SEL) Data Base Organization and User's Guide Revision 1, P. Lo and D. Wykoff, March 1983

†SEL-81-003, Data Base Maintenance System (DBAM) User's Guide and System Description, D. N. Card, D. C. Wyckoff, and G. Page, September 1981

SEL-81-003, Software Engineering Laboratory (SEL) Data Base Maintenance System (DBAM) User's Guide and System Description, P. Lo and D. Card, April 1983

SEL-81-103, Software Engineering Laboratory (SEL) Data Base Maintenance System (DBAM) User's Guide and System Description, P. Lo and D. N. Card, April 1983

†SEL-81-004, The Software Engineering Laboratory, D. N. Card, F. E. McGarry, G. Page, et al., September 1981

SEL-81-104, The Software Engineering Laboratory, D. N. Card, F. E. McGarry, G. Page, et al., February 1982

†SEL-81-005, Standard Approach to Software Development, V. E. Church, F. E. McGarry, G. Page, et al., September 1981

SEL-81-105, Recommended Approach to Software Development, S. Eslinger, F. E. McGarry, and G. Page, May 1982

†This document superseded by revised document.

8552

ORIGINAL PAGE IS
OF POOR QUALITY

SEL-81-205, Recommended Approach to Software Development, F. E. McGarry, G. Page, S. Eslinger et al., April 1983

SEL-81-006, Software Engineering Laboratory (SEL) Document Library (DOCLIB) System Description and User's Guide, W. Taylor and W. J. Decker, December 1981

¹SEL-81-007, Software Engineering Laboratory (SEL) Compendium of Tools, W. J. Decker, E. J. Smith, A. L. Green, et al., February 1981

SEL-81-107, Software Engineering Laboratory (SEL) Compendium of Tools, W. J. Decker, W. A. Taylor, and E. J. Smith, February 1982

SEL-81-008, Cost and Reliability Estimation Models (CAREM) User's Guide, J. F. Cook and E. Edwards, February 1981

SEL-81-009, Software Engineering Laboratory Programmer Workbench Phase 1 Evaluation, W. J. Decker and F. E. McGarry, March 1981

SEL-81-010, Performance and Evaluation of an Independent Software Verification and Integration Process, G. Page and F. E. McGarry, May 1981

SEL-81-011, Evaluating Software Development by Analysis of Change Data, D. M. Weiss, November 1981

SEL-81-012, The Rayleigh Curve As a Model for Effort Distribution Over the Life of Medium Scale Software Systems, G. O. Picasso, December 1981

SEL-81-013, Proceedings From the Sixth Annual Software Engineering Workshop, December 1981

SEL-81-014, Automated Collection of Software Engineering Data in the Software Engineering Laboratory (SEL), A. L. Green, W. J. Decker, and F. E. McGarry, September 1981

SEL-82-001, Evaluation of Management Measures of Software Development, G. Page, D. N. Card, and F. E. McGarry, September 1982, vols. 1 and 2

SEL-82-002, FORTRAN Static Source Code Analyzer Program (SAP) System Description, W. A. Taylor and W. J. Decker, August 1982

¹This document superseded by revised document.

9552

ORIGINAL PAGE IS OF POOR QUALITY

SEL-82-003, Software Engineering Laboratory (SEL) Data Base Reporting Software User's Guide and System Description, P. Lo, September 1982

SEL-82-004, Collected Software Engineering Papers: Volume 1, July 1982

SEL-82-005, Glossary of Software Engineering Laboratory Terms, M. G. Rohleder, December 1982

SEL-82-006, Annotated Bibliography of Software Engineering Laboratory (SEL) Literature, D. N. Card, November 1982

SEL-82-007, Proceedings From the Seventh Annual Software Engineering Workshop, December 1982

SEL-82-008, Evaluating Software Development by Analysis of Changes: The Data From the Software Engineering Laboratory, V. R. Basili and D. M. Weiss, December 1982

SEL-Related Literature

††Bailey, J. W., and V. R. Basili, "A Meta-Model for Software Development Resource Expenditures," Proceedings of the Fifth International Conference on Software Engineering. New York: Computer Societies Press, 1981

Banks, F. K., "Configuration Analysis Tool (CAT) Design," Computer Sciences Corporation, Technical Memorandum, March 1980

††Basili, V. R., "Models and Metrics for Software Management and Engineering," ASME Advances in Computer Technology, January 1980, vol. 1

Basili, V. R., "SEL Relationships for Programming Measurement and Estimation." University of Maryland, Technical Memorandum, October 1979

Basili, V. R., Tutorial on Models and Metrics for Software Management and Engineering. New York: Computer Societies Press, 1980 (also designated SEL-80-008)

††Basili, V. R., and J. Beane, "Can the Parr Curve Help With Manpower Distribution and Resource Estimation Problems?", Journal of Systems and Software, February 1981, vol. 2, no. 1

††This article also appears in SEL-82-004, Collected Software Engineering Papers: Volume 1, July 1982.

††Basili, V. R., and K. Freburger, "Programming Measurement and Estimation in the Software Engineering Laboratory," Journal of Systems and Software, February 1981, vol. 2, no. 1

Basili, V. R., and B. T. Perricone, Software Errors and Complexity: An Empirical Investigation, University of Maryland, Technical Report TR-1195, August 1982

††Basili, V. R., and T. Phillips, "Evaluating and Comparing Software Metrics in the Software Engineering Laboratory," Proceedings of the ACM SIGMETRICS Symposium/Workshop: Quality Metrics, March 1981

Basili, V. R., R. W. Selby, and T. Phillips, Metric Analysis and Data Validation Across FORTRAN Projects, University of Maryland, Technical Report, November 1982

Basili, V. R., and R. Reiter, "Evaluating Automatable Measures for Software Development," Proceedings of the Workshop on Quantitative Software Models for Reliability, Complexity and Cost, October 1979

Basili, V. R., and D. M. Weiss, A Methodology for Collecting Valid Software Engineering Data, University of Maryland, Technical Report TR-1235, December 1982

Basili, V. R., and M. V. Zelkowitz, "Designing a Software Measurement Experiment," Proceedings of the Software Life Cycle Management Workshop, September 1977

†Basili, V. R., and M. V. Zelkowitz, "Operation of the Software Engineering Laboratory," Proceedings of the Second Software Life Cycle Management Workshop, August 1978

††Basili, V. R., and M. V. Zelkowitz, "Measuring Software Development Characteristics in the Local Environment," Computers and Structures, August 1978, vol. 10

Basili, V. R., and M. V. Zelkowitz, "Analyzing Medium Scale Software Development," Proceedings of the Third International Conference on Software Engineering. New York: Computer Societies Press, 1978

††This article also appears in SEL-82-004, Collected Software Engineering Papers: Volume 1, July 1982.

††Basili, V. R., and M. V. Zelkowitz, "The Software Engineering Laboratory: Objectives," Proceedings of the Fifteenth Annual Conference on Computer Personnel Research, August 1977

Card, D. N., "Early Estimation of Resource Expenditures and Program Size," Computer Sciences Corporation, Technical Memorandum, June 1982

Card, D. N., "Comparison of Regression Modeling Techniques for Resource Estimation," Computer Sciences Corporation, Technical Memorandum, November 1982

Card, D. N., and V. E. Church, "Analysis Software Requirements for the Data Retrieval System," Computer Sciences Corporation, Technical Memorandum, March 1983

Card, D. N., and V. E. Church, "A Plan of Analysis for Software Engineering Laboratory (SEL) Data," Computer Sciences Corporation, Technical Memorandum, March 1983

Card, D. N., and M. G. Rohleder, "Report of Data Expansion Efforts," Computer Sciences Corporation, Technical Memorandum, September 1982

††Chen, E., and M. V. Zelkowitz, "Use of Cluster Analysis To Evaluate Software Engineering Methodologies," Proceedings of the Fifth International Conference on Software Engineering. New York: Computer Societies Press, 1981

Freburger, K., "A Model of the Software Life Cycle" (paper prepared for the University of Maryland, December 1978)

Higher Order Software, Inc., TR-9, A Demonstration of AXES for NAVPAK, M. Hamilton and S. Zeldin, September 1977 (also designated SEL-77-005)

Hislop, G., "Some Tests of Halstead Measures" (paper prepared for the University of Maryland, December 1978)

Lange, S. F., "A Child's Garden of Complexity Measures" (paper prepared for the University of Maryland, December 1978)

Miller, A. M., "A Survey of Several Reliability Models" (paper prepared for the University of Maryland, December 1978)

††This article also appears in SEL-82-004, Collected Software Engineering Papers: Volume 1, July 1982.

ORIGINAL PAGE IS
OF POOR QUALITY

National Aeronautics and Space Administration (NASA), NASA Software Research Technology Workshop (proceedings), March 1980

Page, G., "Software Engineering Course Evaluation," Computer Sciences Corporation, Technical Memorandum, December 1977

Parr, F., and D. Weiss, "Concepts Used in the Change Report Form," NASA, Goddard Space Flight Center, Technical Memorandum, May 1978

Reiter, R. W., "The Nature, Organization, Measurement, and Management of Software Complexity" (paper prepared for the University of Maryland, December 1976)

Scheffer, P. A., and C. E. Velez, "GSFC NAVPAK Design Higher Order Languages Study: Addendum," Martin Marietta Corporation, Technical Memorandum, September 1977

Turner, C., and G. Caron, A Comparison of RADC and NASA/SEL Software Development Data, Data and Analysis Center for Software, Special Publication, May 1981

Turner, C., G. Caron, and G. Brement, NASA/SEL Data Compendium, Data and Analysis Center for Software, Special Publication, April 1981

Weiss, D. M., "Error and Change Analysis," Naval Research Laboratory, Technical Memorandum, December 1977

Williamson, I. M., "Resource Model Testing and Information," Naval Research Laboratory, Technical Memorandum, July 1979

††Zelkowitz, M. V., "Resource Estimation for Medium Scale Software Projects," Proceedings of the Twelfth Conference on the Interface of Statistics and Computer Science. New York: Computer Societies Press, 1979

Zelkowitz, M. V., "Data Collection and Evaluation for Experimental Computer Science Research," Empirical Foundations for Computer and Information Science (proceedings), November 1982

Zelkowitz, M. V., and V. R. Basili, "Operational Aspects of a Software Measurement Facility," Proceedings of the Software Life Cycle Management Workshop, September 1977

††This article also appears in SEL-82-004, Collected Software Engineering Papers: Volume 1, July 1982.